Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Автоматики і комп'ютеризованих технологій
(повна назва)

Кафедра Комп'ютерно-інтегрованих технологій, автоматизації та робототехніки
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Система управління роботизованим маніпулятором на основі розпізнавання жестів руки
(тема)

Виконав:
здобувач 2 року навчання,
групи КТРСм-24-1
Ілля ГАЙДУК
(власне ім'я, прізвище)

Спеціальність 174 Автоматизація, комп'ютерно-інтегровані технології та робототехніка
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Комп'ютеризовані та робототехнічні системи
(повна назва освітньої програми)

Керівник доц. Ірина ЖАРІКОВА
(посада, власне ім'я, прізвище)

Допускається до захисту

Зав. кафедри КІТАР _____ Ігор НЕВЛЮДОВ
(підпис) (власне ім'я, прізвище)

2025 р.

Я, Гайдук Ілля Михайлович, як здобувач вищої освіти ХНУРЕ, розумію і підтримую політику закладу з академічної доброчесності. Я не надавав і не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Я не використовував штучний інтелект для підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

08.12.2025  Гайдук І. М.

Харківський національний університет радіоелектроніки

Факультет АКТ

Кафедра КІТАР

Рівень вищої освіти другий (магістерський)

Спеціальність 174 Автоматизація, комп'ютерно-інтегровані технології та робототехніка

(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Комп'ютеризовані та робототехнічні системи

(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«____»_____ 20 ___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувачеві Гайдуку Іллі Михайловичу

(прізвище, ім'я, по батькові)

1. Тема роботи Система управління роботизованим маніпулятором на основі розпізнавання жестів руки
затверджена наказом університету від 10.11.2025 р. № 1018 Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 16 грудня 2025 р.
3. Вихідні дані до роботи: 3.1 Середовище розробки системи – PyCharm; 3.2 Середовище симуляції системи – PyBullet;
3.3 Мова програмування – Python;
3.4 Призначення системи: керування маніпулятором за допомогою жестів рук;
3.5 Функції: виконання різних дванадцяти команд за потребами оператора;
3.6 Робот маніпулятор Franka Emika Panda;
3.7 Умови експлуатації: природнє або штучне освітлення.
4. Перелік питань, що потрібно опрацювати в роботи 4.1 Вступ;
4.2 Аналіз сучасних технологій управління роботизованими системами;
4.3 Аналіз роботизованих маніпуляторів та сфер їх застосування;
4.4 Проєктування системи жестового керування;
4.5 Програмна реалізація та експериментальні дослідження;
4.5 Заходи з охорони праці; 4.6 Висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) графічний демонстраційний матеріал в форматі PowerPoint(*.ppt) в кількості 14 сторінок формату А4

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз літератури за темою роботи	10.10.2025	Виконано
2	Аналіз технічного завдання	15.10.2025	Виконано
3	Аналіз сучасних технологій управління роботизованими системами	15.10.2025	Виконано
4	Аналіз роботизованих маніпуляторів та сфер їх застосування	20.10.2025	Виконано
5	Проєктування системи жестового керування	10.11.2025	Виконано
6	Програмна реалізація та експериментальні дослідження	08.12.2025	Виконано
7	Оформлення пояснювальної записки	08.12.2025	Виконано
8	Представлення до захисту	16.12.2025	

Дата видачі завдання 01.09.2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Ірина ЖАРІКОВА
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 119 с., 2 табл., 20 рис., 43 джерел, 4 додатка.

РОБОТИЗОВАНИЙ МАНІПУЛЯТОР, ЖЕСТОВЕ КЕРУВАННЯ, КОМП'ЮТЕРНИЙ ЗІР, MEDIAPIPE, PYBULLET, ІНВЕРСНА КІНЕМАТИКА, ЛЮДИНО-МАШИННА ВЗАЄМОДІЯ.

Об'єкт дослідження – процес людино-машинної взаємодії під час керування роботизованими маніпуляторами.

Предмет дослідження – методи та програмні засоби розпізнавання жестів руки й алгоритми перетворення цих жестів у керуючі дії для маніпулятора Franka Emika Panda.

Мета роботи – підвищення ефективності й надійності керування роботизованим маніпуляторам, шляхом застосування жестового інтерфейсу на основі комп'ютерного зору та симуляційного моделювання з урахуванням вимог безпеки та зручності оператора.

У роботі проведено огляд сучасних роботизованих маніпуляторів, систем жестового керування та інструментів комп'ютерного зору. Проаналізовано можливості камерних, глибинних та інерційних сенсорів для безконтактної взаємодії. Запропоновано архітектуру системи, що включає модулі обробки відеопотоку, формування вектора ознак, класифікації жестів, інтерпретації команд та керування маніпулятором у локальній системі координат. Проведено експериментальні дослідження для набору з 12 статичних жестів Наукова новизна полягає в комплексній реалізації жестового керування маніпулятором Franka Emika Panda з використанням локальних координат, шаблонної класифікації жестів і вбудованих механізмів безпечного керування.

ABSTRACT

Explanatory report: 119 pages, 2 tables, 20 figures, 43 references, 4 appendices.

ROBOTIC MANIPULATOR, GESTURE CONTROL, COMPUTER VISION, MEDIAPIPE, PYBULLET, INVERSE KINEMATICS, HUMAN–MACHINE INTERACTION.

The object of the study is the process of human – machine interaction during the control of robotic manipulators.

The subject of the study is the methods and software tools for hand-gesture recognition, as well as the algorithms that convert these gestures into control actions for the Franka Emika Panda manipulator.

The aim of the work is to develop and justify a gesture-based control system for a robotic manipulator using computer vision and simulation modelling, in order to improve operator convenience and safety and to simplify the integration of robotic systems into real industrial processes.

The study includes a review of modern robotic manipulators, gesture-control systems, and computer vision tools. The capabilities of camera-based, depth, and inertial sensors for contactless interaction are analysed. The proposed system architecture incorporates modules for video-stream processing, feature-vector construction, gesture classification, command interpretation, and manipulator control in a local coordinate system. Experimental research was conducted for a set of 12 static hand gestures. The scientific novelty lies in the integrated implementation of gesture-based control for the Franka Emika Panda manipulator using local coordinate representations, template-based gesture classification, and built-in safety mechanisms.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ІК – інверсна кінематика;

API (англ. Application Programming Interface) – прикладний програмний інтерфейс.

BGR (англ. Blue, Green, Red) – колірна модель BGR.

CNN (англ. Convolutional Neural Network) – згорткова нейронна мережа.

DH (англ. Denavit–Hartenberg) – конвенція параметризації кінематичних ланцюгів.

DOI (англ. Digital Object Identifier) – цифровий ідентифікатор об’єкта.

GUI (англ. Graphical User Interface) – графічний інтерфейс користувача.

HRI (англ. Human–Robot Interaction) – людино-машинна взаємодія.

HSV (англ. Hue, Saturation, Value) – колірна модель HSV.

LSTM (англ. Long Short-Term Memory) – рекурентна нейронна мережа LSTM.

NN (англ. Neural Network) – нейронна мережа.

OpenCV (англ. Open Source Computer Vision Library) – бібліотека комп’ютерного зору.

RGB (англ. Red, Green, Blue) – колірна модель RGB.

ROS (англ. Robot Operating System) – операційне середовище / платформа для роботів.

SDK (англ. Software Development Kit) – набір засобів розробника.

URDF (англ. Unified Robot Description Format) – уніфікований формат опису робота.

URL (англ. Uniform Resource Locator) – уніфікований локатор ресурсу.

ВСТУП

Роботизовані маніпулятори сьогодні є невід’ємною частиною сучасних виробничих, медичних, сервісних та дослідницьких комплексів. Вони беруть на себе операції, що вимагають високої точності, повторюваності та витривалості, а також роботу в небезпечних або важкодоступних для людини умовах. Ефективність таких систем суттєво залежить від того, наскільки зручними та інтуїтивними є інтерфейси керування, які забезпечують оператору можливість швидко й однозначно передавати свої наміри роботові.

Класичні підходи до керування маніпуляторами – апаратні пульти, джойстики, кнопкові панелі, програмування траєкторій з терміналу – часто є громіздкими з точки зору навчання та експлуатації. Вони потребують від оператора значного досвіду роботи з робототехнікою, а також ускладнюють застосування маніпуляторів у сценаріях, де важлива природність взаємодії. Окремою проблемою є застосування традиційних інтерфейсів у середовищах, де небажаний фізичний контакт з обладнанням або присутні обмеження щодо стерильності.

Жестове керування на основі комп’ютерного зору розглядається як одна з перспективних альтернатив традиційним інтерфейсам. Використання звичайних RGB-камер і алгоритмів аналізу зображень дає змогу реалізувати безконтактний канал людино-машинної взаємодії, в якому оператор лише демонструє жест рукою, а система перетворює його на керуючі впливи для маніпулятора. Такий підхід не потребує спеціальних рукавичок, маркерів або додаткових сенсорів, дозволяючи будувати більш людиноподібні інтерфейси. Однак практичне застосування жестового керування в режимі реального часу пов’язане з низкою проблем, а саме чутливістю до умов освітлення, шумами у відеопотоці, неоднозначністю положень кисті, затримками обробки та необхідністю гарантувати безпечну роботу маніпулятора навіть за помилок розпізнавання.

У цій роботі розглядається задача побудови системи жестового керування роботизованим маніпулятором Franka Emika Panda на основі бібліотеки комп’ютерного зору MediaPipe та симуляційного середовища PyBullet [1].

Мета кваліфікаційної роботи полягає в підвищенні ефективності й надійності керування роботизованим маніпуляторам, шляхом застосування жестового інтерфейсу на основі комп'ютерного зору та симуляційного моделювання з урахуванням вимог безпеки та зручності оператора.

Об'єкт дослідження – процес людино-машинної взаємодії у задачах керування маніпуляторами.

Предмет дослідження – методи та програмні засоби розпізнавання жестів руки й алгоритми перетворення жестів у керуючі впливи на роботизовані маніпулятори.

Для досягнення поставленої мети у роботі розв'язуються такі основні завдання:

– виконати огляд роботизованих маніпуляторів, сфер їх застосування та сучасних підходів до жестового керування;

– проаналізувати програмні засоби комп'ютерного зору для розпізнавання жестів та симуляційні середовища для робототехніки;

– сформулювати вимоги до системи жестового керування та побудувати її архітектуру із виділенням окремих функціональних модулів;

– розробити математичну модель маніпулятора на основі модифікованої DH-конвенції та визначити постановку задачі інверсної кінематики;

– сформувати алгоритми розпізнавання жестів руки, включаючи отримання скелету, нормалізацію координат, побудову вектора ознак та шаблонну класифікацію;

– реалізувати програмний комплекс жестового керування в середовищі Python з використанням MediaPipe та PyBullet, включно з механізмами безпечної зупинки та обмеження робочої зони;

– провести експериментальні дослідження точності розпізнавання жестів, часових характеристик окремих підсистем та end-to-end затримки, проаналізувати отримані результати.

Кваліфікаційна робота виконана згідно ДСТУ 3008:2015 [2], а також з методичними вказівками з підготовки й оформлення кваліфікаційної роботи здобувачами другого (магістерського) рівня вищої освіти спеціальності

174 Автоматизація та комп’ютерно-інтегровані технології, освітньо-професійних програм: «Автоматизоване управління технологічними процесами», «Комп’ютерно-інтегровані технологічні процеси і виробництва», «Комп’ютеризовані та робототехнічні системи» [3].

1 АНАЛІЗ СУЧАСНИХ ТЕХНОЛОГІЙ УПРАВЛІННЯ РОБОТИЗОВАНИМИ СИСТЕМАМИ

1.1 Огляд роботизованих маніпуляторів та сфер їх застосування

Роботизовані маніпулятори є одним із ключових елементів сучасної автоматизації та інтеграції робототехнічних систем у виробничі, медичні й побутові процеси. Під маніпулятором зазвичай розуміють багатоланкову механічну систему з декількома ступенями вільності, що імітує рухи людської руки. Завдяки своїй конструкції та універсальності такі пристрої здатні виконувати широкий спектр завдань, від найпростіших операцій переміщення предметів до складних точних маніпуляцій у тривимірному просторі [4].

Окрему категорію становлять мобільні роботизовані платформи, які поєднують маніпуляційні можливості з автономним переміщенням. Дослідження механічних параметрів гнучких комутаційних структур для таких систем показують важливість забезпечення надійності з'єднань при динамічних навантаженнях [5-7].

Історично перші промислові маніпулятори з’явилися у середині XX століття як частина автоматизованих ліній. Їхня основна функція полягала в заміні людини на небезпечних або монотонних етапах виробництва. З розвитком мікропроцесорних технологій і систем керування маніпулятори отримали можливість виконувати все складніші дії, координуватися із зовнішніми сенсорами та інтегруватися в кіберфізичні системи нового покоління.

Сфера застосування маніпуляторів надзвичайно широка. У промисловості вони є невід’ємною частиною сучасних конвеєрів, де виконують зварювальні, фарбувальні, пакувальні та монтажні операції. Використання роботів дозволяє підвищити продуктивність і якість продукції, зменшити витрати на ручну працю та забезпечити сталість виробничого процесу. В медицині маніпулятори застосовуються у хірургії, стоматології, реабілітаційних системах. Відомі

приклади роботів, що дозволяють здійснювати мінімально інвазивні операції з високою точністю рухів, недосяжною для людини. В умовах небезпечного середовища (атомна енергетика, хімічні виробництва, космічні місії) роботизовані маніпулятори забезпечують дистанційне виконання завдань, захищаючи людину від ризику контакту зі шкідливими факторами [8].

Окремої уваги заслуговує освітня та дослідницька сфера. Маніпулятори широко використовуються у навчальних лабораторіях як інструмент для ознайомлення студентів із принципами кінематики, динаміки та алгоритмів керування. Крім того, вони активно застосовуються в наукових експериментах, де потрібна висока точність і повторюваність рухів.

У промисловості найбільш поширені маніпулятори компаній KUKA, FANUC, ABB та Yaskawa Motoman.

Робототехнічний комплекс KUKA KR AGILUS (рис. 1.1) представляє клас високошвидкісних маніпуляторів з вантажопідйомністю 6-10 кг та максимальним радіусом дії 900-1100 мм. Конструктивні особливості серії AGILUS, зокрема компактні габарити (130 кг маса маніпулятора) та клас захисту IP54, забезпечують його ефективне застосування у обмежених виробничих просторах [9].

Рисунок 1.1 – Зображення робота маніпулятора KUKA KR AGILUS

Промисловий робот FANUC M-20iA (рис. 1.2), що належить до серії середньовантажних маніпуляторів з корисним навантаженням 20 кг, широко імплементований у автомобілебудівній галузі для виконання операцій переносу важких компонентів кузова та трансмісійних вузлів. Особливістю цієї моделі є підвищений ступінь механічної надійності та можливість функціонування у температурному діапазоні від 0°C до +45°C, що критично важливо для інтеграції у технологічні лінії з підвищеними експлуатаційними вимогами [10].

Рисунок 1.2 – Зображення робота маніпулятора M20iA FANUC

Шестиосьовий маніпулятор ABB IRB 1200 (рис. 1.3), що характеризується вантажопідйомністю до 7 кг та радіусом дії 900 мм, знайшов широке застосування у гнучких виробничих системах. Його функціональна універсальність дозволяє інтегрувати робота у різнопланові технологічні процеси: від обслуговування металообробних верстатів з ЧПК до операцій дугового зварювання та матеріалопотоків на складських комплексах [11].

Рисунок 1.3 – Зображення робота маніпулятора ABB IRB 1200

Шестиосьовий маніпулятор Yaskawa Motoman GP8 (рис. 1.4) з номінальною вантажопідйомністю 8 кг та радіусом робочої зони 720 мм позиціонується як універсальне рішення для застосування у процесах дугового зварювання, нанесення покриттів та автоматизованої збірки. Характерною особливістю GP8 є можливість монтажу у різних просторових орієнтаціях (підлога, стеля, кут), це підвищує гнучкість планування виробничих площ [12].

Рисунок 1.4 – Зображення маніпулятора Yaskawa Motoman GP8

У медичній галузі робототехнічні маніпулятори відкрили нові можливості для виконання складних хірургічних втручань. Система Da Vinci Surgical (рис. 1.5), розроблена компанією Intuitive Surgical, стала фактичним стандартом у малоінвазивній хірургії. Хірург керує інструментами через консоль, а роботизована система масштабує рухи його рук, фільтруючи тремор і забезпечуючи точність позиціонування інструментів до 0,1 мм. Це дає змогу проводити операції на серці, простаті, нирках через невеликі розрізи, що суттєво скорочує період реабілітації пацієнтів [13].

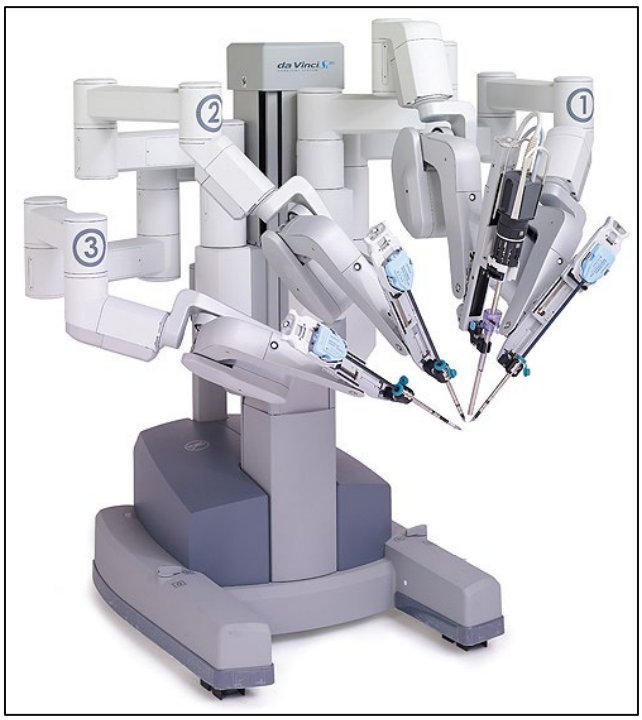

Рисунок 1.5 – Зображення медичного робота маніпулятора Da Vinci Surgical

Наступний робот широко використовується у сфері досліджень та освіти. Маніпулятор Franka Emika Panda (рис.1.6) завоював популярність у лабораторіях штучного інтелекту завдяки відкритому API та можливості отримувати дані про крутний момент на кожному суглобі в реальному часі. Це критично важливо для розробки алгоритмів force control та безпечної взаємодії з довкіллям.

Колаборативні роботи серії UR від Universal Robots також активно використовуються у навчальних закладах – їх можна програмувати через інтуїтивний інтерфейс без глибоких знань робототехніки [14].

Рисунок 1.6 – Зображення робота маніпулятора Franka Emika Panda

1.2 Методи та системи управління жестами

Жестове керування представляє альтернативу традиційним інтерфейсам у сценаріях, де фізичний контакт із пристроєм утруднений або небажаний. Основна ідея полягає в інтерпретації рухів людського тіла, переважно кистей рук у команди для робототехнічних систем. Проте практична реалізація такого підходу стикається з низкою технічних викликів, пов'язаних із варіативністю жестів, умовами освітлення та необхідністю обробки даних у реальному часі [15].

Сучасні системи розпізнавання жестів можна умовно розділити на дві категорії за типом сенсорів. Перша група – оптичні системи на базі RGB-камер або сенсорів глибини (Microsoft Kinect, Intel RealSense, Leap Motion).

Kinect Gen 2, наприклад, забезпечує роздільну здатність глибини 512 пікселів × 424 пікселів з частотою оновлення 30 кадрів/с, чого достатньо для базового трекінгу рухів тіла, але недостатньо для точного визначення положення окремих пальців. Leap Motion натомість спеціалізується саме на детекції кисті,

досягаючи точності до 0,01 мм у контрольованих умовах, проте його робоча зона обмежена конусом із радіусом близько 60 см [16].

Друга категорія – інерційні системи на основі інерційних вимірювальних модулів (акселерометри, гіроскопи, магнітометри), інтегрованих у рукавички або браслети. Їхня перевага – незалежність від освітлення та можливість роботи при повних перекриттях. Однак такі пристрої вимагають калібрування та схильні до дрейфу показань при тривалому використанні. Деякі дослідження пропонують гібридні архітектури, що об''єднують візуальні та інерційні дані через алгоритми об’єднання сенсорів, але складність реалізації та вартість обладнання обмежують їхнє широке впровадження [17].

З появою методів глибокого навчання змінився підхід до розпізнавання жестів. Бібліотека MediaPipe Hands від Google використовує двоетапну архітектуру: спочатку детектор локалізує долоню в кадрі, потім регресійна модель визначає координати 21 ключової точки в тривимірному просторі. Модель навчена на наборі даних, що містить понад 30 тисяч анотованих зображень рук у різних умовах. За даними розробників, точність визначення ключових точок досягає 95 % при обробці кадрів зі швидкістю до 60 кадрів/с на графічному процесорі середнього класу.

Методи на основі ключових точок (MediaPipe, OpenPose) обчислюють кути між сегментами пальців та їхні відносні позиції. Наприклад, жест OK можна визначити через відстань між кінчиками великого та вказівного пальців. Такий підхід дає позитивні результати для дискретних жестів, але вимагає додаткової логіки для розпізнавання динамічних послідовностей [18].

Проте в умовах складного фону або при швидких рухах спостерігається зниження стабільності відстеження. Алгоритмічно методи розпізнавання можна згрупувати наступним чином. Геометричні методи базуються на аналізі контурів і моментів зображення руки. Наприклад, дескриптор моментів Ху дозволяє класифікувати статичні жести за формою силуету, але чутливий до зміни ракурсу та перекриттів. Такі підходи ефективні для контрольованих умов, але погано масштабуються на реальні сценарії застосування [19].

Попри досягнутий прогрес, залишаються невирішені проблеми: розпізнавання жестів при частковому перекритті руки, адаптація до індивідуальних особливостей користувачів (розмір долоні, швидкість рухів), забезпечення роботи при різних умовах освітлення. Крім того, більшість існуючих систем орієнтовані на обмежений набір попередньо визначених жестів, що обмежує гнучкість взаємодії. Перспективним напрямком видається розробка адаптивних систем, здатних вивчати нові жести від конкретного користувача без повторного тренування всієї моделі.

1.3 Програмні засоби для розпізнавання жестів

Останнім часом мав місце значний розвиток програмного забезпечення, спрямованого на розпізнавання жестів рук. Цей розвиток визначається кількома факторами: зростанням продуктивності комп’ютерів, розповсюдженням недорогих камер глибини та прогресом у галузі глибокого навчання. Як наслідок, з'явилось багато інструментів, різних за складністю та призначенням – від спеціалізованих засобів для розпізнавання окремих жестів до універсальних фреймворків для комплексної обробки відео.

1.3.1 Застосування бібліотеки OpenCV

Початки застосування цифрової обробки зображень у розпізнаванні жестів тісно пов'язані з бібліотекою OpenCV. Розроблена компанією Intel на початку 2000-х років як відкритий проєкт, вона з тих пір стала основним інструментом для студентів і дослідників. OpenCV містить понад 2500 оптимізованих алгоритмів, починаючи від базових операцій фільтрації (розмиття, морфологічні операції) до складних методів виявлення об’єктів [20].

У контексті розпізнавання жестів OpenCV традиційно використовується на етапі попередньої обробки. Типовий конвеєр обробки виглядає так: захоплення кадру з камери, перетворення кольорового простору RGB в HSV, створення бінарної маски шкіри на основі діапазонів відтінків, морфологічна обробка

(розширення/звуження) для зменшення шумів, виявлення контурів функцією findContours(), визначення опуклої оболонки руки, обчислення дефектів опуклості для ідентифікації пальців. Такий підхід демонструє гарну чутливість при обробці простих статичних жестів у контрольованих умовах освітлення.

Однак слід відзначити кілька обмежень цього методу. По-перше, сегментація за кольором шкіри критично залежить від умов освітлення та суб'єктивна для різних типів шкіри. По-друге, метод погано себе почуває з частковими перекриттями руки. По-третє, OpenCV не включає готові моделі для тривимірного відстеження або розпізнавання динамічних жестів. Тому у практичних застосунках OpenCV часто комбінується з більш сучасними інструментами, такими як нейронні мережі [21].

1.3.2 Використання фреймворку MediaPipe

За останні роки компанія Google створила MediaPipe – фреймворк, який радикально спростив розробку застосунків, що потребують обробки мультимедійних даних у реальному часі. На відміну від OpenCV, MediaPipe постачається з готовими, попередньо навченими моделями для виявлення та відстеження людини у кадрі.

Модуль MediaPipe Hands реалізує двох етапний підхід. На першій стадії легка модель виявлення локалізує долоню на зображенні, визначаючи обмежувальний прямокутник. На другій стадії більш складна модель регресії працює всередину цього прямокутника й обчислює координати 21 ключової точки: 5 на кожному пальці (основа, проміжна суглоб, кінцева суглоб) та 1 на долоні. Результат подається у тривимірних координатах (x, y, z), де z відповідає приблизній глибині відносно камери [22].

Суттєву перевагу MediaPipe представляє те, що вона часто забезпечує достатню точність відстеження жестів навіть на процесорах з обмеженою продуктивністю. На комп’ютері з відеокартою NVIDIA серії RTX бібліотека здатна обробляти потік у 30-60 кадрів/с без затримок. На мобільних телефонах (особливо з акселератором для машинного навчання) затримка залишається

прийнятною – близько 50-100 мс. Це робить MediaPipe привабливим вибором для інтерактивних застосунків керування роботів, де реагування повинно бути майже миттєвим.

Окрім модуля Hands, MediaPipe включає спеціалізовані компоненти для інших задач – pose для виявлення скелету людини (33 точки), face mesh для детального відстеження риси обличчя (468 точок), holistic для комбінованого аналізу руки, обличчя та тулуба. Такі компоненти часто використовуються у спортивній аналітиці, реабілітаційних програмах та системах розпізнавання мови тіла [23].

Однак MediaPipe також має обмеження. Її вбудовані класифікатори жестів досить обмежені – система розпізнає лише кілька базових жестів (OK, долоні вгору тощо). Для розпізнавання власного набору жестів користувач повинен побудувати класифікатор самостійно, використовуючи координати ключових точок як вхідні ознаки.

1.3.3 Робота з TensorFlow та PyTorch

Коли потрібна висока точність розпізнавання складних, динамічних жестів, та досить обчислювальних ресурсів для навчання моделей, звертаються до універсальних фреймворків машинного навчання. TensorFlow, розроблений компанією Google, та PyTorch від компанії Meta – це дві провідні платформи у цій сфері.

TensorFlow надає екосистему інструментів: от високорівневого API Keras для швидкого прототипування до низькорівневого API для тонкого налаштування. Для розпізнавання жестів можна будувати згорткові нейронні мережі (CNN) для аналізу окремих кадрів, або тривимірні згортки (3D CNN) і рекурентні мережі (LSTM) для аналізу послідовностей кадрів. Особливо цінна функція TensorFlow Lite, яка дозволяє експортувати навчену модель у формат, оптимізований для мобільних пристроїв та вбудованих систем, що суттєво скорочує затримку обробки [24].

PyTorch цінується дослідниками за більш інтуїтивний синтаксис та гнучкість при експериментуванні з архітектурами. Під час розробки прототипів часто легше модифікувати архітектуру мережі у PyTorch, ніж у TensorFlow. З іншого боку, TensorFlow встановив кращу підтримку для розгортання готових моделей у виробничому середовищі, включаючи сервіси для масштабування та оптимізації [25]. Обидва фреймворки дозволяють використовувати великі публічні датасети для тренування: NVGesture Dataset (містить 1050 відеороликів 25 жестів від 10 акторів), EgoGesture Dataset (5000+ відеороликів жестів від першої особи) та інші. Це дозволяє дослідникам виходити за межі простих прототипів і створювати добре узагальнювальні моделі.

1.3.4 Спеціалізовані SDK для сенсорів

Окрім універсальних бібліотек, на ринку існують інструменти, спеціально розроблені для роботи з конкретними сенсорами. Leap Motion SDK дозволяє працювати з інфрачервоною камерою Leap Motion Controller, яка забезпечує дуже точне позиціонування пальців (до 0,01 мм), хоча робоча зона обмежена невеликим кубічним обсягом над пристроєм. Це робить Leap Motion привабливим для застосунків, де потрібна висока точність, наприклад, у віртуальній реальності або у роботах, що маніпулюють дрібними предметами.

Intel RealSense SDK постачається з документацією та прикладами коду для роботи з серією камер глибини від Intel. На відміну від RGB-камер, глибинні камери напряму вимірюють відстань до об’єктів у сцені, що робить розпізнавання руки більш стійким до змін освітлення. SDK забезпечує функції для створення 3D-хмар точок, синхронізації кольорового та глибинного потоків, й навіть включає готові приклади для відстеження рук [26].

1.4 Симуляційні середовища для відпрацювання керування роботами

Розробка системи керування роботом на реальному обладнанні – це завжди ризик. Помилка в алгоритмі може призвести до пошкодження маніпулятора,

зіткнення з навколишніми об’єктами або навіть травмування оператора. Саме тому на ранніх етапах розробки зазвичай використовують віртуальні симуляційні середовища, де можна безпечно тестувати алгоритми керування, не побоюючись наслідків. Симулятори дозволяють моделювати фізичну поведінку робота: динаміку руху ланок, сили тертя в суглобах, колізії з об'єктами, навіть вібрації та інерційні ефекти. Крім того, вони дають змогу швидко ввести зміни в код і одразу подивитися результат, без необхідності перезавантажувати реальний робот. У цьому розділі розглянуто три найпоширеніші симуляційні платформи.

PyBullet – це обгортка мовою Python для фізичного двигуна Bullet (рис.1.7), який спочатку створювався для комп’ютерних ігор та анімації. Завдяки простому інтерфейсу та тісній інтеграції з екосистемою Python (NumPy, OpenCV, TensorFlow), PyBullet став популярним інструментом для навчання з підкріпленням та досліджень у галузі робототехніки [27].

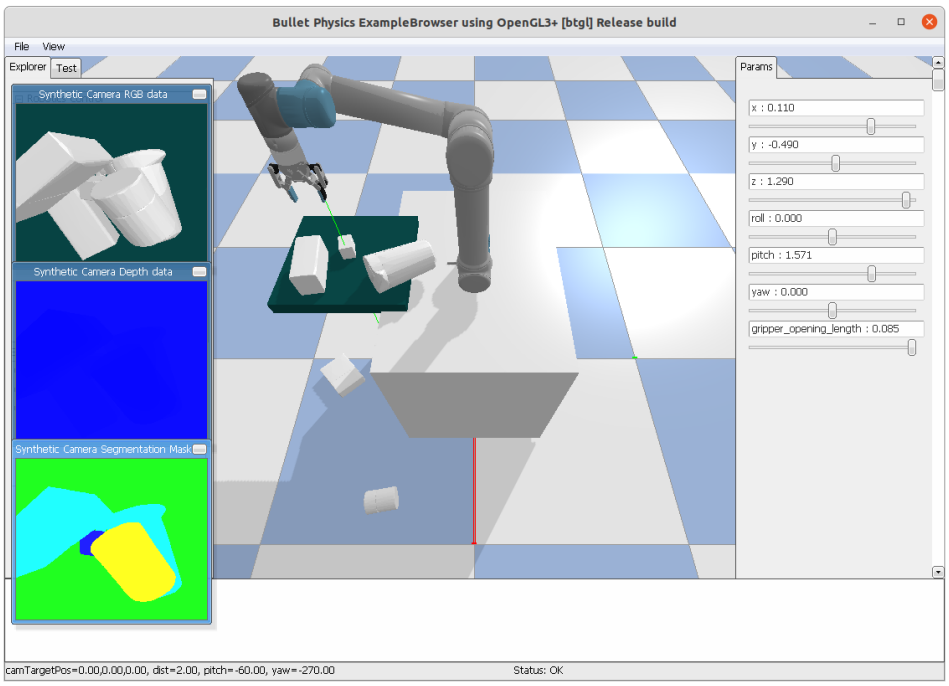

Рисунок 1.7 – Приклад інтерфейсу Pybullet

Основна перевага PyBullet полягає у швидкості запуску. Досить кількох рядків коду щоб завантажити модель робота у форматі URDF (Unified Robot Description Format) і почати симуляцію. Наприклад, маніпулятор Kuka IIWA або

робот-рука Franka Panda можуть бути імпортовані з готових бібліотек за лічені секунди. Це особливо зручно для студентів або дослідників, які хочуть швидко перевірити гіпотезу, не витрачаючи час на складне налаштування середовища.

Проте PyBullet має свої обмеження. Графіка у ньому досить примітивна – немає реалістичного освітлення, тіней чи текстур. Це не проблема для тестування алгоритмів керування, але ускладнює відпрацювання систем комп'ютерного зору, які залежать від якості зображення. Крім того, PyBullet погано підходить для симуляції складних сенсорів, таких як лідари або радари. Тим не менш, для базового тестування жестового керування та інверсної кінематики він цілком підходить [28].

Друга платформа – це Gazebo (рис. 1.8). Симулятор з відкритим вихідним кодом, розроблений спеціально для роботів, що працюють під управлінням ROS (Robot Operating System). Якщо PyBullet орієнтований на швидкість та простоту, то Gazebo ставить на перше місце реалістичність та сумісність з реальним обладнанням [29].

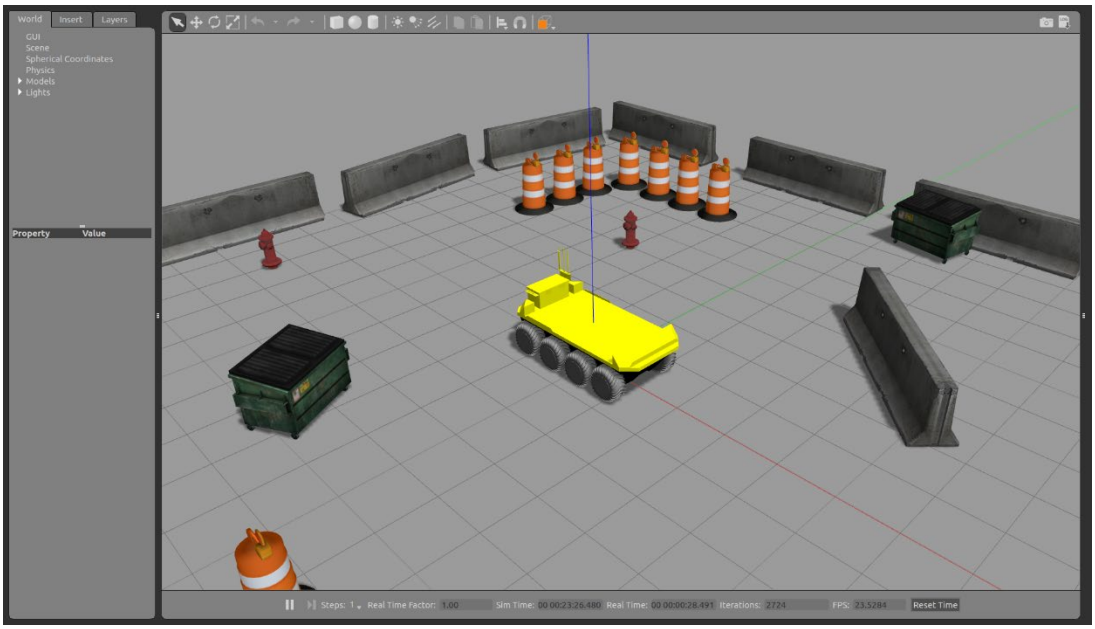

Рисунок 1.8 – Приклад інтерфейсу Gazeba

У Gazebo можна моделювати складні сенсори: камери глибини, лідари, інерційні модулі, тактильні датчики. Фізичний двигун відтворює динаміку з

високою точністю, включаючи пружні деформації, контактні сили та тертя Кулона. Тобто Gazebo ідеально підходить для тестування автономних роботів, дронів та складних маніпуляційних задач.

Однак складність налаштування – це ціна за функціональність. Щоб запустити симуляцію в Gazebo, потрібно створити XML-файли опису робота, налаштувати плагіни для сенсорів, прописати параметри фізики. Для новачка це може зайняти кілька днів. Крім того, Gazebo досить вимогливий до ресурсів – на складних сценах із багатьма об'єктами частота оновлення може впасти нижче реального часу, що ускладнює інтерактивне тестування [30].

Попри ці складнощі, Gazebo залишається стандартом у промисловій робототехніці. Багато виробників роботів (Universal Robots, KUKA, ABB) надають готові моделі своїх маніпуляторів для Gazebo, що дозволяє розробникам тестувати код перед розгортанням на реальному обладнанні.

Третя платформа CoppeliaSim (раніше відомий як V-REP) займає проміжну позицію між простотою PyBullet та потужністю Gazebo (рис. 1.9). Це комерційний симулятор з безкоштовною освітньою версією, який підтримує кілька мов програмування (Python, C++, Java, Lua) та інтегрується з ROS [31].

Головна особливість CoppeliaSim – це модульна архітектура. Кожен об'єкт у сцені може мати власний скрипт, що дозволяє створювати складні системи. Наприклад, можна змоделювати кілька роботів, які працюють разом на складі, кожен з яких має власну логіку прийняття рішень. Це нечасто зустрічається в інших симуляторах.

CoppeliaSim також має вбудований редактор сцен з інтуїтивним інтерфейсом, що спрощує створення віртуальних середовищ. Можна швидко побудувати модель цеху, розставити перешкоди, налаштувати освітлення. Графіка тут значно краща, ніж у PyBullet, хоча й не дотягує до рівня ігрових движків типу Unity або Unreal Engine.

З недоліків варто згадати, що безкоштовна версія має обмеження: не можна зберігати сцени у закритому форматі, а деякі плагіни доступні лише у платній

версії. Крім того, документація іноді застаріла, і доводиться шукати інформацію на форумах спільноти.

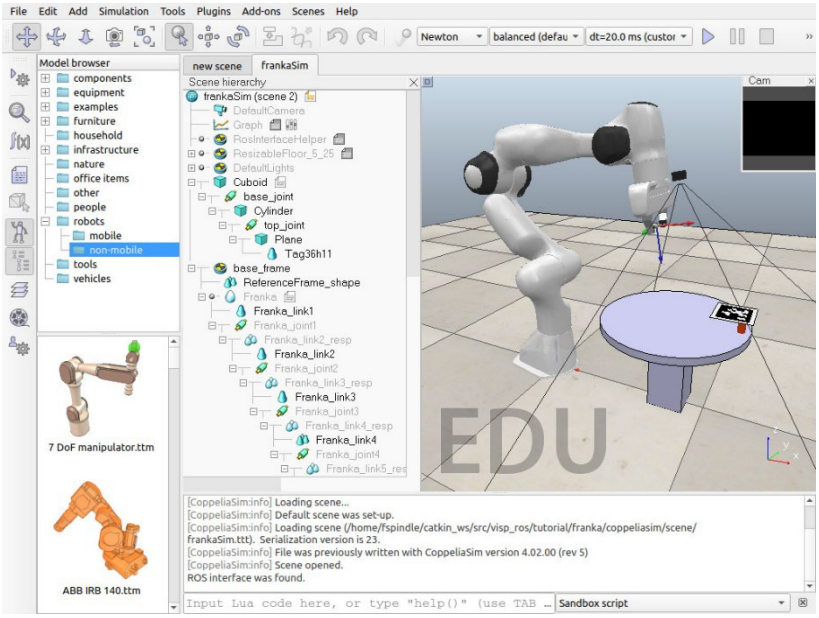

Рисунок 1.9 – Приклад інтерфейсу платформи CoppeliaSim

1.5 Висновки до розділу 1

Перший розділ присвячено аналізу сучасного стану роботизованих маніпуляторів, методів жестового керування та програмно-апаратних засобів для реалізації безконтактної взаємодії людини з машиною. На прикладі промислових, медичних та дослідницьких маніпуляторів продемонстровано значний прогрес у галузі робототехніки та гнучкість цих систем щодо адаптації під конкретні задачі. З'ясовано, що сучасні інтерфейси керування мають відповідати критеріям інтуїтивності, мінімальної затримки відгуку, безпечності та незалежності від спеціалізованого обладнання.

Порівняння методів жестового керування виявило принципові відмінності між оптичними та інерційними сенсорами. Оптичні рішення дають змогу взаємодіяти з системою природним чином без додаткових маркерів, але їхня робота залежить від умов освітлення та наявності перешкод у полі зору. Інерційні

датчики працюють стабільно незалежно від зовнішнього середовища, натомість потребують носимих пристроїв і регулярного калібрування.

Аналіз програмного забезпечення засвідчив, що OpenCV лишається основним інструментом попередньої обробки відеопотоку, тоді як MediaPipe забезпечує якісне відстеження ключових точок кисті у реальному часі без потреби тренувати власні моделі. Фреймворки TensorFlow і PyTorch доцільно застосовувати для побудови складніших архітектур або розпізнавання динамічних жестів. Спеціалізовані рішення накшталт Leap Motion SDK чи Intel RealSense SDK дають високу точність, проте прив'язані до конкретного апаратного забезпечення.

Порівняння симуляційних середовищ PyBullet, Gazebo та CoppeliaSim дозволило оцінити їхні сильні та слабкі сторони. PyBullet вирізняється простотою інтеграції з Python-проєктами. Gazebo забезпечує детальнішу фізичну модель і підтримку різноманітних сенсорів, однак є вимогливішим до обчислювальних ресурсів. CoppeliaSim поєднує розвинені налаштування з відносно зручним інтерфейсом, займаючи проміжну нішу.

Підсумовуючи, жодне з розглянутих рішень не є універсальним, однак їхнє поєднання дає змогу створити працездатну систему жестового керування. Вибір MediaPipe для розпізнавання жестів і PyBullet для симуляції обґрунтовується оптимальним співвідношенням між точністю, швидкодією, доступністю впровадження та потенціалом масштабування.

2 ПРОЄКТУВАННЯ СИСТЕМИ ЖЕСТОВОГО КЕРУВАННЯ

2.1 Постановка задачі та вимоги до системи

Сучасні роботизовані маніпулятори дедалі частіше використовуються не лише в промисловості, а й у лабораторіях, медицині, сервісній робототехніці та освітніх середовищах. У багатьох таких сценаріях від користувача вимагається швидка та інтуїтивна взаємодія з роботом без складного навчання робототехнічним мовам чи програмуванню. Класичні інтерфейси у вигляді пультів, кнопок або графічних панелей керування залишаються ефективними, однак вони суттєво обмежують свободу рухів оператора та вимагають опосередкованих дій: щоб змусити маніпулятор зсунутись на кілька сантиметрів, користувач має натискати кнопки або рухати джойстик, замість того щоб просто показати потрібний жест рукою.

На цьому тлі зростає інтерес до систем безконтактної взаємодії, де основним «каналом спілкування» між людиною та роботом стає рух руки, пальців або позиція долоні в просторі. Жестове керування дає змогу перетворити природні рухи оператора на команди для маніпулятора, що потенційно знижує поріг входу для нових користувачів, зменшує когнітивне навантаження та робить процес керування більш наочним і зрозумілим навіть для людей без технічної підготовки. Для деяких прикладних задач, наприклад, при взаємодії з крихкими об’єктами чи під час навчальних демонстрацій, такий тип інтерфейсу є не просто зручним, а й більш безпечним, оскільки користувач тримається на відстані від робота та не торкається до реального обладнання.

У межах цієї роботи розглядається задача розробки системи, яка забезпечує керування роботизованим маніпулятором на основі розпізнавання жестів руки в режимі, наближеному до реального часу. Вхідними даними для системи є відеопотік з камери, спрямованої на область, де оператор демонструє жести. На основі цього потоку система має виявляти руку, визначати ключові

точки (landmarks), формувати ознаки, класифікувати жест і перетворювати його в конкретну команду для маніпулятора: перемістити кінцевий ефектор у певному напрямку, змінити висоту, повернути базу, обернути зап’ясток або відкрити чи закрити хват. Окремою задачею є підтримка жесту зупинки, який блокує будь-які рухи робота та слугує «екстреним гальмом» у разі помилки розпізнавання чи неправильної демонстрації жесту.

Формально систему можна описати як відображення між послідовністю зображень з камери та простором керуючих дій маніпулятора. На кожному кроці часу t камера формує кадр I_t. Після обробки цього кадру формується вектор ознак f_t, що описує положення та конфігурацію руки оператора. Далі працює модуль класифікації, який відносить цей вектор до одного з наперед визначених класів жестів g_t. Наступний етап – інтерпретація жесту як команди, коли жесту g_t відповідає дія в просторі станів робота, наприклад зміщення кінцевого ефектора на вектор Δx_t або зміна певного суглобового кута. У найпростішому вигляді це можна подати як функцію:

$$C: g_t \to u_t,$$

де u_t – це вектор керуючих впливів для маніпулятора (зміна положення або суглобів).

Інший модуль, який відповідає за кінематику робота, перетворює ці керуючі впливи в набір суглобових координат q_t, що подаються на вхід симулятора чи реального робота.

Щоб така система була не лише демонстраційною, а й практично корисною, до неї висувається низка вимог. По-перше, це вимоги до точності розпізнавання жестів: система повинна стабільно відрізняти робочі жести один від одного, зводячи до мінімуму хибні спрацьовування. Невірно розпізнаний жест у контексті керування маніпулятором означає несподіваний рух кінцевого ефектора, що може призвести до зіткнення з перешкодами, пошкодження об’єктів або потенційно небезпечних ситуацій у разі роботи з реальним

обладнанням. По-друге, важливим є фактор затримки. Від моменту, коли оператор показав жест, до моменту фактичного руху робота повинно проходити якнайменше часу, щоб взаємодія сприймалася як «живий» діалог із системою, а не як робота з повільним інтерфейсом, який запізнюється на секунди.

Ще одна група вимог пов’язана з безпекою та надійністю. Система має передбачати режими, у яких будь-який рух робота блокується, поки користувач не подасть чіткий сигнал на відновлення керування. Для цього вводиться спеціальний жест зупинки, який не викликає жодних змін у положенні маніпулятора, але переводить систему в безпечний стан. Крім того, у реальних умовах жести користувача не є ідеальними: рука може частково виходити з кадру, з’являються перешкоди, змінюється освітлення. Тому система повинна бути стійкою до шумів, випадкових рухів та короткочасних втрат відстежування руки, використовуючи, зокрема, згладжування по кількох послідовних кадрах, фільтрацію нестабільних жестів і пороги впевненості класифікатора.

Окремо варто відзначити вимоги до робочої зони та швидкості руху маніпулятора. Природні для людини жести можуть призвести до надто великих або різких команд, якщо прямо передавати їх на роботизовану систему. Тому в модулі інтерпретації жестів вводяться обмеження на амплітуди зміщень і швидкості руху, а також передбачаються різні режими керування, наприклад грубий для швидкого наближення до цілі та тонкий для точного позиціонування. Просторова область, у межах якої дозволено рухати кінцевий ефектор, теж обмежується: це дає змогу уникнути виходу за фізичні межі маніпулятора та зіткнень із деталями його конструкції.

Нарешті, система має бути реалізована таким чином, щоб її можна було розгорнути на звичайному персональному комп’ютері з однією RGB-камерою, без використання дорогих стереокамер чи спеціалізованих датчиків глибини. Це накладає обмеження на обчислювальну складність алгоритмів та вимагає використання бібліотек, оптимізованих для роботи в реальному часі, таких як OpenCV для обробки зображень, MediaPipe для виділення ключових точок руки та засобів інверсної кінематики в середовищі симуляції. У сукупності всі ці

вимоги формують рамки, у яких розробляється система жестового керування, та визначають як її архітектуру, так і вибір конкретних технологій, що будуть використані в наступних розділах.

2.2 Архітектура системи жестового керування

Система жестового керування побудована так, щоб розділити завдання комп'ютерного зору, розпізнавання жестів та керування маніпулятором. Завдяки цьому можна змінювати окремі частини незалежно, наприклад оновити алгоритм розпізнавання, не чіпаючи модуль кінематики робота, або замінити симулятор, залишивши той самий жестовий інтерфейс. Система працює як конвеєр – на вході камера, на виході рухи маніпулятора. Структурна схема наведена на рис. 2.1.

Відеопотік з RGB-камери надходить у модуль обробки зображень, де OpenCV зчитує кадри, нормалізує їх і готує до аналізу. Тут відбувається проста, але критична робота – стабільне читання з певною частотою, коректна робота з кольором, базові перетворення. Від цього залежить, чи отримає система придатні дані для розпізнавання жестів.

Потім кадри йдуть до модуля виділення руки. MediaPipe Hands будує модель руки з 21 просторової точки для кожної кисті – перетворює зображення у математичне представлення пози руки. Сучасні підходи до візуального розпізнавання жестів використовують саме таку багатоступеневу структуру: спочатку захоплення зображення, потім виділення руки та формування ознак, і тільки після цього класифікація жесту та генерація команд. Це підтверджують оглядові роботи з візуального розпізнавання для людсько-машинної взаємодії.

Далі координати ключових точок перетворюються у числовий вектор, який описує жест компактніше, враховуються положення пальців, їх зігнутість, орієнтація долоні. Цей вектор порівнюється зі збереженими шаблонами або подається на класифікатор, який визначає клас пози. Система переходить від геометрії руки до символічних команд типу «стоп», «вперед», «вліво»,

«обертання», «відкрити хват». Важливий момент – згладжування по часу: аналізується не один кадр, а послідовність рішень, щоб не реагувати на випадкові помилки.

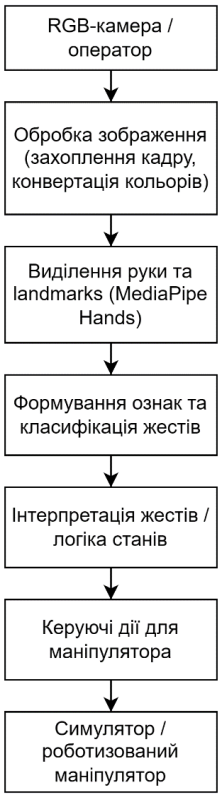

Рисунок 2.1 – Структурна схема архітектури розроблюваної системи

Коли жест перетворено на команду, включається модуль інтерпретації, який зв'язує команди від правої та лівої рук з діями маніпулятора. Тут є своя логіка, наприклад права рука керує переміщенням в просторі по X, Y, Z, а ліва відповідає за поворот бази чи обертання зап'ястка. На цьому рівні вводяться режими грубого та тонкого руху, коли той самий жест задає різну швидкість, а також обробляється жест зупинки – миттєва відмова від нових команд, перехід у безпечний стан та очікування дозволу від оператора. По суті, це мозок між розпізнаванням і робототехнікою.

Нижче розташований модуль керування маніпулятором, який перетворює абстрактні команди на конкретні переміщення. Команди подаються як зміщення кінцевого ефектора або зміни суглобових координат. Для плавних і коректних

рухів використовується інверсна кінематика: з цільового положення обчислюється вектор суглобових кутів, які передаються в PyBullet або на реальний маніпулятор. Тут же враховуються обмеження робочої зони, межі кутів, максимальні швидкості, логіка хвату. Це відповідає практикам телероботизації, де верхній рівень формує цілі в просторі, а нижній відповідає за виконання з урахуванням кінематики.

Остання частина – симуляція та візуалізація. На етапі розробки маніпулятор працює як віртуальна модель, яка дозволяє перевірити логіку без ризику пошкодити обладнання. Оператор працює з камерою та жестами, а рухи видно в симуляції, тому можна оперативно коригувати систему. Важливо, що інтерфейс між інтерпретацією жестів і керуванням не прив'язаний до PyBullet – його можна замінити на реального робота, зберігши всю верхню частину.

2.3 Математична модель маніпулятора

Для побудови системи жестового керування недостатньо запустити PyBullet і подати йому координати. Потрібно описати маніпулятор як математичний об'єкт, тобто задати зв'язок між кутами в суглобах та положенням і орієнтацією кінцевого ефектора в просторі. У роботі розглядається серійний маніпулятор з сімома ступенями вільності, близький за будовою до робота Franka Emika Panda, який також має сім послідовно з'єднаних обертових суглобів.

Почнемо з узагальнених координат. Нехай маніпулятор має n обертових суглобів. Тоді стан робота з точки зору кінематики можна описати вектором суглобових координат як:

$$q = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \end{bmatrix}, \tag{2.1}$$

де q_i – кут повороту i-го суглоба відносно його нульового положення.

У моєму випадку $n = 7$, тобто $q \in \mathbb{R}^7$. Це стандарт у кінематиці роботів і використовується у класичних курсах Крейга та інших авторів [32].

Щоб пов'язати вектор q з положенням і орієнтацією кінцевого ефектора, на кожній ланці маніпулятора вводять локальну систему координат. Найпоширеніший спосіб – параметризація за Денавіта-Хартенберга (Denavit-Hartenberg, DH), у якій кожне з'єднання описується чотирма параметрами: довжиною ланки a_{i-1}, кутом скручування α_{i-1}, зміщенням уздовж осі d_i та кутом обертання θ_i [33]. Для маніпулятора типу Panda зазвичай застосовують модифіковану DH-параметризацію, але загальна ідея залишається тією самою: кожен ланцюжок суглоб-ланка перетворюється на матрицю гомогенного перетворення між сусідніми системами координат. Геометричну структуру маніпулятора та розташування локальних систем координат показано на рис. 2.2.

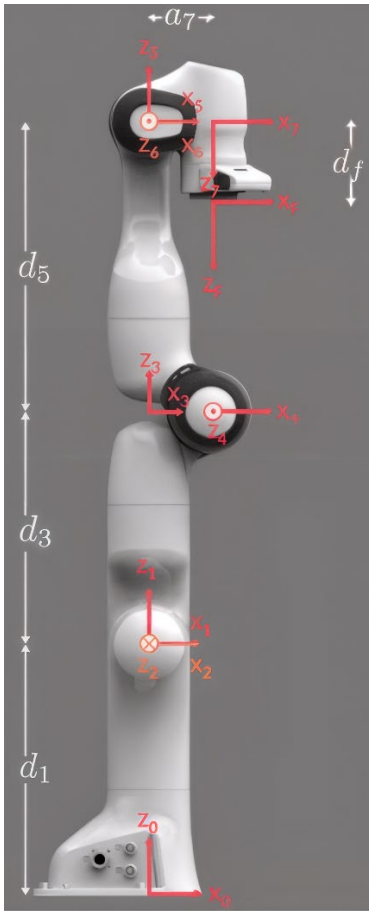

Рисунок 2.2 – Геометрична структура робота та розташування систем координат

У модифікованій DH-конвенції перехід від системи координат $(i-1)$ системи i задається послідовністю елементарних поворотів і зсувів уздовж осей x та z:

$$^{i-1}T_i = Rot_{x_i-1}(\alpha_{i-1})Trans_{x_i-1}(a_{i-1})Rot_{z_i}(\theta_i)Trans_{z_i}(d_i), \qquad (2.2)$$

де $^{i-1}T_i$ – матриця розміром 4 на 4, яка описує поворот та зсув ланки i відносно попередньої ланки;

α_{i-1} – кут між осями z_{i-1} та z_i, a_{i-1} – відстань між ними;

θ_i – кут повороту навколо осі z_i, а d_i це зсув уздовж цієї осі [33].

Якщо розписати формулу (2.2) у вигляді конкретної матриці, отримаємо явну форму гомогенного перетворення:

$$^{i-1}T_i = \begin{bmatrix} cos\theta_i & -sin\theta_i & 0 & \alpha_{i-1} \\ sin\theta_i \cos(\alpha_{i-1}) & cos\theta_i \cos(\alpha_{i-1}) & -\sin(\alpha_{i-1}) & -d_i \sin(\alpha_{i-1}) \\ \sin\theta_i \sin(\alpha_{i-1}) & cos\theta_i \sin(\alpha_{i-1}) & \cos(\alpha_{i-1}) & d_i \cos(\alpha_{i-1}) \\ 0 & 0 & 0 & 1 \end{bmatrix}. \qquad (2.3)$$

У цій матриці перші три стовпці описують орієнтацію системи i відносно $(i-1)$, а четвертий стовпець – її положення у вигляді вектора зсуву. Така форма матриці для модифікованих DH-параметрів наводиться, зокрема, в оглядових статтях та енциклопедичних джерелах з кінематики маніпуляторів.

Повна матриця прямої кінематики від бази робота до кінцевого ефектора отримується як добуток послідовних перетворень для всіх ланок:

$$^0T_n(q) = {}^0T_1(q_1)\,{}^1T_2(q_2)\,\ldots\,{}^{n-1}T_n(q_n) = \prod_{i=1}^{n} {}^{i-1}T_i(q_i), \qquad (2.4)$$

де $^0T_n(q)$ – гомогенна матриця розміром 4 на 4, що описує положення й орієнтацію кінцевого ефектора у базовій системі координат.

Кожен множник $^{i-1}T_i(q_i)$ має вигляд як матриця з формули (2.3) із відповідними параметрами ланки. Цей підхід множення послідовних

гомогенних перетворень описаний у класичних книгах з робототехніки та навчальних матеріалах з прямої кінематики [34].

Матрицю ${}^0T_n(q)$ можна записати у блоковому вигляді:

$$ {}^0T_n(q) = \begin{bmatrix} R(q) & p(q) \\ 0^T & 1 \end{bmatrix}, \tag{2.5} $$

де $R(q) \in \mathbb{R}^{3\times3}$ – матриця повороту, яка задає орієнтацію кінцевого ефектора;

$p(q) = [x(q), y(q), z(q)]^T$ – вектор його положення в базовій системі координат.

Нульовий вектор у нижньому рядку гарантує, що перетворення залишається гомогенним і з ним можна виконувати координатні перетворення за правилами лінійної алгебри.

Далі потрібно представити стан кінцевого ефектора у вигляді вектора позивання, який включає положення й орієнтацію. Одним із поширених варіантів є використання трьох координат положення й трьох кутів Ейлера або кутів roll-pitch-yaw. Тоді можна записати як:

$$ x = \begin{bmatrix} p(q) \\ \varphi(q) \end{bmatrix} = \begin{bmatrix} x(q) \\ y(q) \\ z(q) \\ \varphi_x(q) \\ \varphi_y(q) \\ \varphi_z(q) \end{bmatrix} = f(q), \tag{2.6} $$

де $\varphi(q) = \left[\varphi_x, \varphi_y, \varphi_z\right]^T$ – вектор кутів, що описує орієнтацію ефектора;

f_q – векторна функція прямої кінематики, яка відображає суглобові координати в простір поз робота.

Для опису руху важлива не лише статична залежність $x = f(q)$, а й зв’язок між швидкостями в суглобах і швидкістю кінцевого ефектора. Цей зв’язок задається матрицею Якобі, яка визначається як матриця частинних похідних функції $f(q)$ за компонентами вектора q:

$$J(q) = \frac{\partial f(q)}{\partial q} \in \mathbb{R}^{6\times n}. \tag{2.7}$$

Тоді миттєві швидкості в просторі поз кінцевого ефектора пов'язані з суглобовими швидкостями співвідношенням за (2.8):

$$\dot{x} = J(q)\,\dot{q}, \tag{2.8}$$

де $\dot{q}$ – вектор кутових швидкостей у суглобах;

$\dot{x}$ – вектор лінійних і кутових швидкостей кінцевого ефектора [35].

У задачі жестового керування цікавить те, як змінюється положення кінцевого ефектора в часі, тобто його лінійна швидкість. Тому часто використовують лише трансляційну частину Якобі. Якщо взяти перші три рядки повної матриці $J(q)$, отримаємо позиційну Якобі-матрицю $J_p(q) \in \mathbb{R}^{3\times n}$, яка пов'язує вектор суглобових швидкостей $\dot{q}$ з лінійною швидкістю кінцевого ефектора:

$$\dot{p} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = J_p(q)\dot{q}, \tag{2.9}$$

де $p(q) = [x(q), y(q), z(q)]^T$ – положення кінцевого ефектора;

$\dot{p}$ – лінійна швидкість у базовій системі координат.

Маніпулятор завжди має обмеження по кутах у суглобах, які задаються у вигляді нерівностей за формулою (2.10):

$$q_{min} \le q \le q_{max}\ , \tag{2.10}$$

де q_{min} та q_{max} – вектори нижніх і верхніх меж для кожного суглоба.

Подібні обмеження, а також обмеження на швидкість і момент у суглобах, явно враховуються в сучасних роботах з керування маніпуляторами та

постановки задачі інверсної кінематики з обмеженнями [36]. У подальших розділах ці обмеження використовуються як частина логіки безпечного керування: навіть якщо жести задають агресивну команду, рух кінцевого ефектора і суглобів не повинен виходити за допустимі діапазони.

2.4 Постановка задачі інверсної кінематики

У попередньому підрозділі було визначено пряму кінематику маніпулятора у вигляді залежності:

$$x = f(q),$$

де q – вектор суглобових координат;

x – вектор, що описує положення та орієнтацію кінцевого ефектора в базовій системі координат.

Задача інверсної кінематики формулюється як задача зворотного відображення. Для заданої бажаної пози (позиції та орієнтації) кінцевого ефектора x_d потрібно знайти таку конфігурацію суглобів q^*, щоб кінематична модель відтворювала цю позу:

$$f(q^*) = x_d. \tag{2.11}$$

У загальному випадку рівняння (2.11) може мати декілька розв’язків або не мати жодного для недосяжних поз. Для маніпулятора з сімома ступенями вільності, подібного до Franka Emika Panda, при повній задачі керування шістьма координатами, поза x система є редундантною, тобто число суглобів перевищує розмірність робочого простору, тому існує нескінченна множина конфігурацій, які дають одну й ту саму позу. Питання редундантності й множини розв’язків

інверсної кінематики для таких систем детально аналізуються у роботах Chiaverini [37].

У контексті розробленої системи жестового керування розглядається не поодинока задача, формула (2.11), а послідовність цільових поз, які змінюються в часі під впливом жестів оператора. Жест інтерпретується як команда сформувати нову бажану позу $x_d(t)$ або малий приріст Δx_d в певному напрямку. На кожному часовому кроці необхідно обрати таку конфігурацію $q(t)$, яка мінімізує відхилення між поточною позою кінцевого ефектора та новою ціллю, не виходячи за допустимі межі суглобових координат. Це приводить до постановки задачі оптимізації, як у (2.12), за умов, які наведені у формулі (2.10):

$$q^*(t) = \arg\min_{\mathrm{q}} \|f(q) - x_d(t)\|. \tag{2.12}$$

Практичні алгоритми інверсної кінематики зазвичай формулюються не безпосередньо в координатах q, а в термінах миттєвих швидкостей. Відповідно до класичної теорії, миттєвий зв'язок між швидкістю кінцевого ефектора $\dot{x}$ та суглобовими швидкостями $\dot{q}$ задається матрицею Якобі (2.8).

Якщо вимагати, щоб для заданої бажаної швидкості кінцевого ефектора $\dot{x}_d$ суглобові швидкості мінімізували квадратичну помилку:

$$\|J(q)\dot{q} - \dot{x}_d\|^2 \to min. \tag{2.13}$$

Тоді класичним розв'язком задачі найменших квадратів буде використання псевдоінверсії Якобі:

$$\dot{q} = J^{\dagger}(q)\dot{x}_d. \tag{2.14}$$

де $J^{\dagger}(q)$ – псевдообернена матриця Мура-Пенроуза.

Для випадку, коли матриця Якобі має повний рядок, псевдоінверсію можна записати у вигляді:

$$J^{\dagger}(q) = J^{T}(q)\big(J(q)J^{T}(q)\big)^{-1}. \tag{2.15}$$

Виведення формул (2.14)–(2.15) та їх застосування в інверсній кінематиці докладно розглянуто в оглядовій роботі [37-38], де інверсна кінематика представлена як задача найменших квадратів у робочому просторі.

Недоліком чистої псевдоінверсії є погана числова стійкість поблизу сингулярних конфігурацій, коли матриця $J(q)J^{T}(q)$ стає погано обумовленою, і невеликі зміни $\dot{x}_d$ призводять до великих значень $\dot{q}$. Для розв'язання цієї проблеми в літературі запропоновано метод заглушених найменших квадратів (damped least squares, DLS), у якому до внутрішнього добутку додається демпфувальний член $\lambda^2 I$. У цьому випадку розв'язок для суглобових швидкостей набуває наступного вигляду:

$$\dot{q} = J^{T}(q)(J(q)J^{T}(q) + \lambda^2 I)^{-1}\dot{x}_d, \tag{2.16}$$

де $\lambda > 0$ – параметр демпфування;

I – одинична матриця [38].

У дискретній реалізації керування, що використовується в розробленій системі, замість миттєвої швидкості зручно оперувати приростами бажаної пози Δx_d за малий крок часу. При наближенні $\Delta x_d \approx \dot{x}_d \Delta t$ відповідний приріст суглобових координат можна обчислити з використанням такої ж матриці, як у (2.16):

$$\Delta q = J^{T}(q)(J(q)J^{T}(q) + \lambda^2 I)^{-1}\Delta x_d. \tag{2.17}$$

Потім оновити конфігурацію маніпулятора за правилом [39]:

$$q_{k+1} = q_k + \Delta q. \tag{2.18}$$

Важливою особливістю 7-ступеневого маніпулятора є наявність нульового простору матриці Якобі. Загальне розв'язання швидкісної інверсної кінематики для редундантної системи подають у вигляді:

$$\dot{q} = J^{\dagger}(q)\, \dot{x}_d + \left(I - J^{\dagger}(q)J(q)\right) z, \tag{2.19}$$

де z – довільний вектор у суглобовому просторі, який проєктується в нульовий простір Якобі й не впливає на $\dot{x}$.

За основу в цій роботі взято саме класичну постановку інверсної кінематики на основі псевдоінверсії та damped least squares, Реальна чисельна реалізація спирається не на власну імплементацію формул (2.16)–(2.19), а на вбудовані бібліотеки інверсної кінематики середовища PyBullet (calculateInverseKinematics). Документація PyBullet та супровідні матеріали прямо вказують, що внутрішньо використовуються алгоритми на основі заглушених найменших квадратів з урахуванням обмежень по суглобах [40].

Науково-практичний внесок цієї роботи пов'язаний з інтеграцією класичної інверсної кінематики в специфічний контекст жестового керування. По-перше, сформовано механізм перетворення розпізнаних жестів на малі прирости цільової пози Δx_d із підтримкою двох режимів масштабування («грубий» та «тонкий» рух), що дозволяє на рівні жестового інтерфейсу регулювати ефективний крок у робочому просторі, не змінюючи сам IK-алгоритм. По-друге, перед викликом calculateInverseKinematics реалізовано додаткове обмеження робочої зони. Бажана поза x_d, сформована з жестів, проєктується в допустимий об'єм роботи маніпулятора, що фактично реалізує частину умов (2.10) уже в робочому просторі, зменшуючи навантаження на чисельний солвер та кількість недосяжних цілей. По-третє, у систему введено окремий захисний режим STOP. У разі розпізнавання відповідного жесту

формування нових цільових поз призупиняється, а остання конфігурація q_k підтримується без змін, що дозволяє уникати неконтрольованих рухів при помилкових або випадкових жестах. Використання поточної конфігурації маніпулятора як початкової точки для кожного виклику calculateInverseKinematics фактично відповідає практичному використанню нульового простору (2.19). Бібліотека PyBullet обирає з множини можливих конфігурацій ту, що є найближчою до попереднього стану, тим самим забезпечує плавність руху та мінімізацію зайвих суглобових переміщень.

2.5 Модуль розпізнавання жестів

Модуль розпізнавання жестів у цій роботі виконує роль перекладача між рухами руки оператора та командами для маніпулятора. Камера бачить лише послідовність RGB-кадрів, а маніпулятор потребує чітких дискретних команд: рухатися вгору, вниз, змінити орієнтацію, відкрити або закрити хват, зупинити рух тощо. Саме модуль розпізнавання жестів перетворює відеопотік на послідовність таких символічних команд.

2.5.1 Отримання скелету руки

Для виділення руки та отримання її параметрів використовується рішення MediaPipe Hands, яке поєднує дві нейронні мережі: детектор долоні та модель оцінювання положення ключових точок кисті. Це дозволяє в реальному часі відновлювати 21 тривимірний орієнтир на зображенні руки з однієї RGB-камери.

Нехай $p_i = (x_i, y_i, z_i)^T, i = 0, \dots, 20$ – вектори координат цих орієнтирів у системі координат зображення, де точка з індексом $i = 0$ відповідає зап’ястку. Тоді повний сирий стан руки в одному кадрі можна записати як вектор у (2.20).

$$p = [p_0^T, p_1^T, \dots, p_{20}^T]^T. \quad (2.20)$$

2.5.2 Нормалізація координат і побудова вектора ознак

Якщо використати координати p_i як є, опис жесту буде чутливим до положення руки в кадрі та відстані до камери. Щоб зменшити цю залежність, у роботі застосовано просту, але типову для сучасних систем HGR нормалізацію:

– усі точки переводяться в систему координат із початком у зап’ястку;

– координати масштабуються за характерною довжиною кисті [41].

Спочатку обчислюються відносні координати:

$$\tilde{p}_i = p_i - p_0, \;\; i = 1, \dots, 20. \tag{2.21}$$

Потім вводиться масштабний коефіцієнт s, який характеризує розмір кисті. Для нього береться евклідова відстань між зап’ястком та суглобом середнього пальця MCP:

$$s = \left\| p_{mcp} - p_0 \right\|_2, \tag{2.22}$$

де p_{mcp} – координати відповідного орієнтира MCP.

Нормалізовані координати задаються за (2.23).

$$q_i = \frac{\tilde{p}_i}{s} = \frac{p_i - p_0}{s}, \;\; i = 1, \dots, 20. \tag{2.23}$$

Тобто кожна точка описується у власній системі координат руки та в одиницях довжини кисті. Далі всі нормалізовані точки просто об’єднуються в один вектор ознак f, який і представляє жест у кадрі. Це відповідає практиці так званих landmarks-based input formats, коли саме координати контрольних точок стають вхідними даними для класифікатора.

2.5.3 Шаблонна класифікація жестів

Для класифікації статичних жестів у роботі використано шаблонний підхід з евклідовою відстанню. Основна ідея – для кожного жесту зберігається кілька еталонних векторів $f_{c,k}$, отриманих на етапі калібрування, а потім новий жест порівнюється з усіма еталонами й відноситься до того класу, до якого він найближчий у просторі ознак.

Нехай$G = \{g_1, \dots, g_c\}$– множина жестів, для кожного класу g_c є набір зразків $\{f_{c,1}, \dots, f_{c,K_c}\}$. Відстань між поточним жестом f та класом g_c задається як мінімальна евклідова відстань до його шаблонів [42]:

$$d_c(f) = \min_{k=1,\dots,K_c} \|f - f_{c,k}\|_2. \quad (2.24)$$

Система обирає той жест, для $d_c(f)$ мінімальна, але рішення вважається дійсним лише тоді, коли ця мінімальна відстань не перевищує заздалегідь заданий поріг. Якщо всі відстані більші за порогові значення, кадр вважається невідомим жестом і не генерує жодної команди. Практична перевага цього підходу полягає в тому, що додавання нового жесту зводиться до накопичення кількох еталонів, без перенавчання великих нейромереж.

2.5.4 Часова стабілізація розпізнавання

Розпізнавання по одному кадру неминуче страждає від шуму. Невеликий рух руки, локальні помилки трекінгу MediaPipe чи випадкові збої призводять до того, що класифікатор на сусідніх кадрах може стрибати між кількома класами, навіть якщо користувач фактично тримає один і той самий жест. Щоб зменшити ці коливання, вводиться етап часової стабілізації.

Позначимо через $c_t \in C \cup \{0\}$ сирий клас жесту на кадрі t, де C – множина всіх жестів, а значення $c_t = 0$ інтерпретується як жест не розпізнано. . Послідовність $\{c_t\}$ є шумною, поодинокі помилки в положенні лендмарок призводять до коротких сплесків неправильних класів.

У загальному випадку таку послідовність можна згладжувати за допомогою ковзного вікна з останніх N рішень $\{c_{t-N+1}, \dots, c_t\}$ та вибору найчастішого класу у цьому вікні. У цьому разі стабілізований $\hat{c}_t$ можна записати у вигляді:

$$\hat{c}_t = mode(c_{t-N+1}, \dots, c_t), \tag{2.25}$$

де mode – повертає найчастіше значення у вікні.

Для додаткової фільтрації можна вводити N_{min}: рішення приймається лише тоді, коли переможний клас зустрічається у вікні не менше ніж N_{min} разів.

У цій роботі реалізовано спрощений, але ефективний варіант часового фільтра, який добре відповідає практичним вимогам реального часу. Замість зберігання всього вікна $\{c_{t-N+1}, \dots, c_t\}$ система відстежує лише поточний сирий клас c_t, останній побачений клас c_t^{last}, та лічильник кількості послідовних кадрів k_t, на яких спостерігається один і той самий жест. Якщо на кадрі t сирий клас збігається з попереднім ($c_t = c_{t-1}^{last}$), лічильник збільшується на одиницю, інакше він скидається в 1 разом зі зміною c_t^{last}. Графік результату роботи стабілізації наведено на рисунку 2.3.

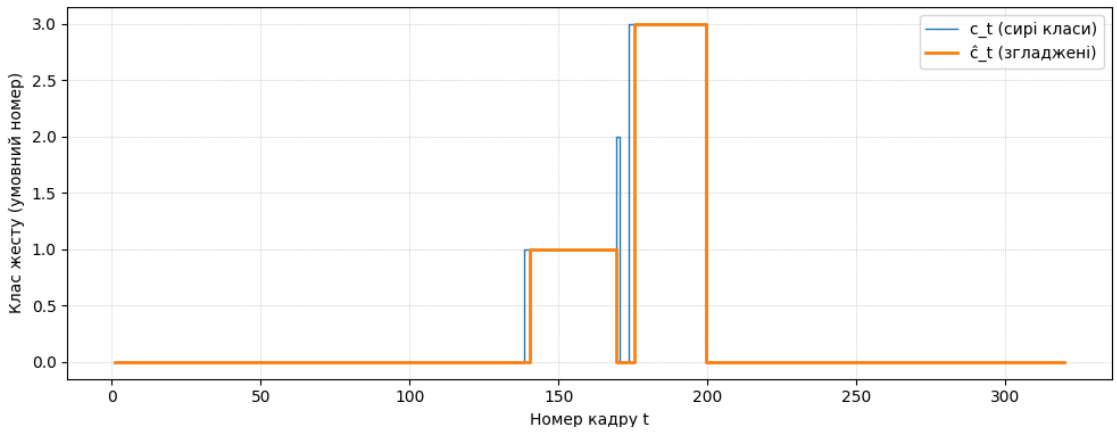

Рисунок 2.3 – Часова стабілізація розпізнавання жестів

На графіку показано, як часова стабілізація згладжує послідовність жестів. Синя лінія c_t – сирі класи. Видно короткі стрибки, зокрема одиничний стрибок

класу 2 біля 170-го кадру. Помаранчева лінія $\hat{c}_t$ – стабілізовані жести. Перехід до нового класу з’являється з невеликою затримкою, але одиничний стрибок повністю зникає, а довгі ділянки з класами 1 і 3 стають рівними й без шуму.

2.6 Висновки до розділу 2

У цьому розділі розроблено комплексний проєкт системи жестового керування роботизованим маніпулятором – від формулювання задачі й технічних вимог до побудови алгоритмів обробки жестів та об'єднання всіх складових в єдину архітектуру. Проведений аналіз показав, що ефективність такого інтерфейсу визначається насамперед точністю й стабільністю розпізнавання, мінімальною затримкою між жестом і відповідним рухом маніпулятора, а також наявністю механізмів безпеки та обмеження робочого простору.

Запропонована архітектура передбачає чіткий поділ на функціональні рівні: захоплення й первинна обробка відеопотоку, формування ознакового опису, класифікація жестів, інтерпретація команд і безпосереднє керування маніпулятором. Модульна структура дає змогу вносити зміни в окремі компоненти, не зачіпаючи решту системи. Важливим здобутком є побудова вектора ознак руки, інваріантного до зсуву й масштабу, це гарантує стабільну класифікацію за різних умов зйомки. Застосований шаблонний класифікатор k-NN продемонстрував достатню ефективність для розпізнавання статичних жестів, а впроваджена часова фільтрація дала змогу суттєво скоротити кількість помилкових спрацьовувань.

Окремим напрямом проєктування стала математична модель маніпулятора, побудована за модифікованою DH-конвенцією. Вона встановлює формальну залежність між кутами в суглобах і положенням кінцевого ефектора у просторі. Розглянуті методи розв'язання оберненої кінематичної задачі – псевдоінверсія Якобіана та затухаючі найменші квадрати – дали змогу сформувати коректний підхід до керування рухом ефектора в реальному часі.

Підсумком проєктування стала узгоджена система, в якій права рука оператора відповідає за просторове переміщення кінцевого ефектора, а ліва – за обертання бази та орієнтацію зап'ястка. Додатково реалізовано механізми безпеки. Жест STOP, режим паузи та обмеження швидкості з можливістю перемикання між режимами COARSE і FINE. У сукупності це забезпечує не лише функціональну повноту, а й зручність та передбачуваність поведінки системи під час експлуатації.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ СИСТЕМИ ЖЕСТОВОГО КЕРУВАННЯ

3.1 Архітектура розроблюваної системи жестового керування

Програмна реалізація побудована у вигляді трирівневої архітектури. На нижньому рівні знаходиться модуль керування роботом, який інкапсулює роботу з PyBullet: завантаження URDF-моделі Franka Emika Panda, налаштування сцени, гравітації, камер візуалізації, а також доступ до суглобів маніпулятора. Цей рівень представлений класом SimplePandaBase у файлі robot_simple.py. Об’єкт цього класу відповідає за фізичний світ у симуляторі. При створенні він підключається до PyBullet, завантажує площину, URDF-модель робота, задає безпечну початкову позу руки та просторові межі робочої зони кінцевого ефектора. Окремі методи (move_ee_to, move_ee_delta, rotate_base_step, apply_joint_offsets, set_gripper) надають абстрактний інтерфейс до інверсної кінематики та керування хватом, у той час як обчислення суглобових кутів делегуються вбудованій функції calculateInverseKinematics, яку надає PyBullet.

Над ним розташований шар сприйняття жестів. Він реалізований як набір функцій і допоміжних класів, які працюють з бібліотекою MediaPipe Hands. За кожним RGB-кадром обчислюється положення 21 тривимірного орієнтира кисті, після чого ці точки перетворюються у вектор ознак функцією hand_to_feature.
У цій роботі класифікатор реалізований у вигляді шаблонного модуля TemplateGestureClassifier, який порівнює поточний вектор ознак із набором еталонних зразків, записаних для кожного жесту, та повертає назву класу (STOP, GRIP_OPEN, BASE_LEFT тощо). Важливим є те, що модуль розпізнавання жестів не залежить від PyBullet чи конкретного робота. На вхід він отримує лише кадри з камери, а на вихід видає семантичні мітки жестів.

Третій, верхній рівень утворює інтеграційний модуль телеоперування, реалізований у скрипті teleop_ik_bimanual.py. Саме тут з'єднуються всі частини системи в єдиний цикл реального часу. Скрипт створює об'єкт SimplePandaBase, ініціалізує відеозахоплення з камери, запускає MediaPipe для отримання landmarks, викликає hand_to_feature та класифікатор жестів, а потім, спираючись на стабілізовані мітки для лівої та правої руки, формує прирости dx, dy, dz для декартового переміщення, а також команди для повороту бази та суглоба зап'ястка. На цьому ж рівні реалізовано логіку режимів COARSE/FINE, екстрений жест STOP, мапінг поточних та стабілізованих міток у статус-рядок та запис журналу gesture_log.csv з усіма розпізнаними жестами. Назва жестів та їх функції наведено у таблиці 3.1.

Таблиця 3.1 –Відповідність жестів і команд робота

Рука	Жести	Назва в системі	Примітка
Будь-яка	Зупинка / STOP	STOP	Миттєво зупиняє рух робота; при утриманні ~2 с вмикає/вимикає керування жестами
	Відкрити хват	GRIP_OPEN	Закрити або відкрити хват
	Закрити хват	GRIP_CLOSE	
Ліва	Поворот бази вліво	L_BASE_LEFT	Повернути маніпулятор навколо вертикальної осі
	Поворот бази вправо	L_BASE_RIGHT	
	Наклони кисті вверх	PITCH_UP	Повернути суглоб зап'ястка
	Наклони кисті вниз	PITCH_DOWN	
Права	Рух кисті вперед по X	BASE_RIGHT	Зсунути кінцевий ефектор (X)
	Рух кисті назад по X	BASE_LEFT	
	Рух вгору по Z	PITCH_UP	Підняти кінцевий ефектор (Y)
	Рух вниз по Z	PITCH_DOWN	
	Перемикач режимів з грубого на точний	SPEED_COARSE/ SPEED_FINE	При стабільному жесті й утриманні ~2 с перемикає режими

Якщо описати архітектуру у термінах діаграми класів (Додаток А), центральним елементом є логічний контролер телеоперування. Саме він об'єднує всі інші компоненти в єдиний цикл телеуправління. Зберігає посилання

на об’єкт SimplePandaBase, класифікатор жестів TemplateGestureClassifier, об’єкт MediaPipeHands для роботи з landmarks руки та об’єкт відеозахоплення VideoCapture з бібліотеки OpenCV. Набір внутрішніх полів (speed_mode, gripper_open, control_paused, словники last_label і same_count, числові параметри швидкостей, кількості кадрів для стабілізації та часу утримання жесту) відображає той факт, що контролер реалізує всю поведінкову логіку системи. Він вирішує, в якому режимі працювати (COARSE/FINE), чи дозволено зараз керування, які жести вважати стабільними і в які саме команди для робота вони мають перетворюватися.

У реалізації основний цикл main() структурований на кілька логічних підзадач. Умовно можна виділити блок обробки кадру process_frame, блок часової стабілізації жестів stabilize_gestures, мапінг стабілізованих жестів у прирости в декартовому просторі та обертання суглобів map_gesture_to_command, оновлення стану робота через методи SimplePandaBase update_robot, а також окремі фрагменти коду, що відповідають за керування хватом, перемикання режимів швидкості й активацію/деактивацію паузи. Усі ці підзадачі реалізовані як послідовні логічні блоки всередині функції main() скрипта teleop_ik_bimanual.py.

Клас SimplePandaBase у цій діаграмі виступає інкапсуляцією всього, що пов’язано з симуляцією маніпулятора в PyBullet. Внутрішні поля client, robot, arm_joints, base_joint, pitch_joint, ee_link зберігають ідентифікатори підключення до PyBullet, моделі робота та його основних суглобів. Пара пар base_min / base_max і pitch_min / pitch_max задають допустимі діапазони кутів для відповідних суглобів, а workspace_min та workspace_max описують прямокутний робочий об’єм для кінцевого ефектора. Відповідно до цього, приватний метод _clamp_workspace проєктує довільну цільову позицію в межі робочої області, _set_arm_targets задає цільові кути для набору суглобів, _go_safe_pose переводить руку в безпечну стартову конфігурацію при ініціалізації.

Публічні методи SimplePandaBase відображають набори дій, які може виконувати робот з точки зору контролера: move_ee_to та move_ee_delta

працюють у декартовому просторі, використовуючи інверсну кінематику PyBullet для обчислення суглобових координат; rotate_base_step і move_pitch_step керують окремими суглобами у суглобному просторі; apply_joint_offsets дає можливість малими кроками змінювати положення вибраних суглобів, що використовується для кручення кисті; set_gripper керує розкриттям хвату; apply_pitch_pair реалізує скоординований нахил руки за рахунок узгодженої зміни двох суглобів. Методи step_sim та disconnect відповідають за крок симуляції та коректне завершення сесії в PyBullet. На діаграмі це відображено зв'язком SimplePandaBase → PyBullet з позначкою «використовує», а також залежністю від бібліотеки NumPy для роботи з векторами, обмеженнями та масивами.

Клас RobotState є простим контейнером даних, що повертається методом get_state() з SimplePandaBase. Він містить лише два публічні поля – base_angle та pitch_angle.

Другим ключовим логічним блоком, окрім керування роботом, є класифікатор жестів TemplateGestureClassifier. У діаграмі чітко видно, що він не залежить ні від PyBullet, ні від OpenCV, ні від інших частин системи: всередині зберігаються лише директорія з шаблонами (root_dir), кількість сусідів у k-NN (k) та словник samples з еталонними векторами ознак для кожного жесту.

Публічні методи __init__, load та predict забезпечують роботу з шаблонами. Ініціалізацію, завантаження еталонів з директорії та прогноз класу для поточного вектора ознак, додавання нових еталонних зразків виконується окремим допоміжним скриптом, наприклад, record_gesture.py, який зберігає .npy-файли у відповідні підкаталоги. Обчислення евклідової відстані між векторами реалізоване безпосередньо в методі predict().

За перетворення сирих landmarks руки на числовий опис відповідає модуль GestureUtils, позначений стереотипом <<модуль>>. Він містить функцію hand_to_feature(landmarks), яка приймає структуру HandLandmarks і повертає вектор типу ndarray. Таким чином, GestureUtils є проміжною ланкою між зовнішньою бібліотекою MediaPipe та внутрішнім класифікатором системи.

Дві допоміжні сутності – MediaPipeHands та HandLandmarks, у діаграмі позначені як частини зовнішньої бібліотеки. MediaPipeHands інкапсулює модель детектора/трекера руки, її параметри та метод process(image), який повертає результат з полями типу HandLandmarks. Останній містить список орієнтирів landmark і використовується як вхід у GestureUtils.

Окремо на діаграмі показано залежність контролера від бібліотек OpenCV та NumPy. OpenCV використовується для роботи з камерою (VideoCapture), попередньої обробки зображень (flip, cvtColor), а також для візуалізації стану системи (виведення тексту, показ вікна, обробка клавіш). NumPy у випадку TeleopController застосовується опосередковано через функції модуля жестів і обробку приращень координат, хоча на діаграмі акцент зроблено насамперед на його використанні в SimplePandaBase та GestureUtils.

3.2 Реалізація модуля керування маніпулятором

Модуль керування маніпулятором реалізовано у вигляді класу SimplePandaBase у файлі robot_simple.py, який інкапсулює всю логіку взаємодії з симулятором PyBullet та забезпечує інтерфейс для жестового керування. Клас підтримує як традиційне суглобове керування, так і сучасні підходи на основі інверсної кінематики в декартовому просторі.

3.2.1 Ініціалізація та завантаження моделі робота

Процес ініціалізації включає створення з'єднання з PyBullet, завантаження URDF-моделі маніпулятора Franka Panda та налаштування початкової конфігурації. Фрагмент коду ініціалізації наведено нижче.

```
def __init__(self, use_gui: bool = True):
self.client = p.connect(p.GUI if use_gui else p.DIRECT)
p.setAdditionalSearchPath(pybullet_data.getDataPath())
p.resetSimulation()
p.setGravity(0, 0, -9.81)
```

```
p.loadURDF("plane.urdf")
self.robot = p.loadURDF(
"franka_panda/panda.urdf",
basePosition=[0, 0, 0],
useFixedBase=True,
flags=p.URDF_USE_INERTIA_FROM_FILE,)
```

Параметр use_gui дозволяє перемикатися між графічним режимом та режимом прямого обчислення (для прискорення розрахунків). Використання прапорця URDF_USE_INERTIA_FROM_FILE забезпечує коректне моделювання динаміки робота з урахуванням реальних інерційних параметрів.

Після завантаження моделі виконується налаштування камери спостерігача. Код наведено нижче.

```
p.resetDebugVisualizerCamera(
    cameraDistance=1.5,
    cameraYaw=50,
    cameraPitch=-35,
    cameraTargetPosition=[0.5, 0.0, 0.3],)
```

Ці параметри підібрано емпірично для оптимального огляду робочої зони маніпулятора під час тестування.

3.2.2 Організація суглобів та визначення обмежень

Маніпулятор Franka Panda має 7 обертових суглобів руки та 2 суглоби хвату (рис. 3.1). У класі визначено ключові суглоби для керування. Фрагмент коду наведено нижче.

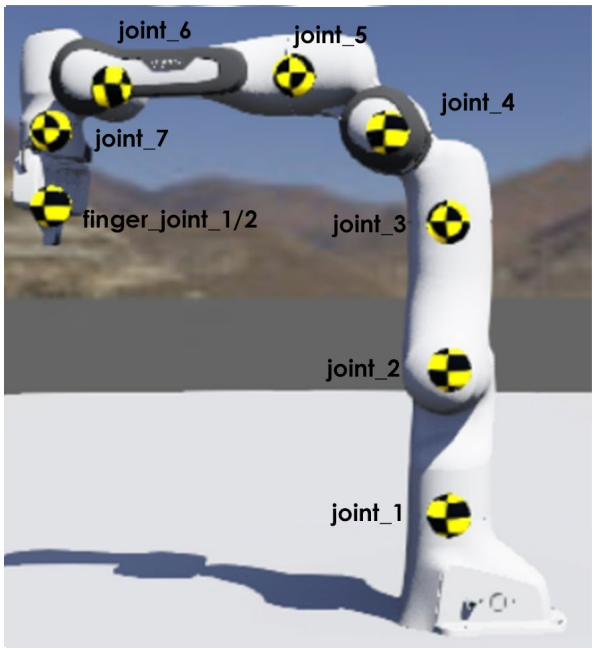

Рисунок 3.1 – Зображення обертових суглобів

```
self.arm_joints = list(range(7))
self.base_joint = 0
self.pitch_joint = 3
self.ee_link = 11
```

Важливим елементом реалізації є ідентифікація конкретних суглобів для спеціального керування:

```
self.base_joint = self.joint_name_to_index["panda_joint1"]
self.wrist_joint = self.joint_name_to_index["panda_joint6"]
```

Сустав panda_joint1 (індекс 0) відповідає за поворот всього маніпулятора навколо вертикальної осі. Це базовий ступінь свободи, який визначає орієнтацію робота в горизонтальній площині. Сустав panda_joint6 (індекс 5) контролює обертання запястя навколо власної осі, що дозволяє змінювати орієнтацію кінцевого ефектора без зміни його положення. Ці суглоби виділено окремо, оскільки у системі жестового керування вони управляються безпосередньо через команди лівої руки оператора, а не через розв'язок інверсної кінематики. Це

зроблено щоб розділити відповідальність, права рука оператора контролює позицію кінцевого ефектора у просторі, а ліва – орієнтацію всієї конструкції. Для кожного з цих суглобів зберігаються межі допустимих кутів:

```
self.base_min = self.joint_lower_limits[self.base_idx_in_arm]
self.base_max = self.joint_upper_limits[self.base_idx_in_arm]
self.wrist_min = self.joint_lower_limits[self.wrist_idx_in_arm]
self.wrist_max = self.joint_upper_limits[self.wrist_idx_in_arm]
```

Такі обмеження запобігають виходу маніпулятора за межі фізично можливих конфігурацій та самозіткненням ланок.

3.2.3 Початкова безпечна конфігурація

При ініціалізації маніпулятор автоматично переводиться у безпечну стартову позицію методом _go_safe_pose(). Зображення робота у безпечній, початковій формі наведено на рис. 3.2. Код наведено нижче.

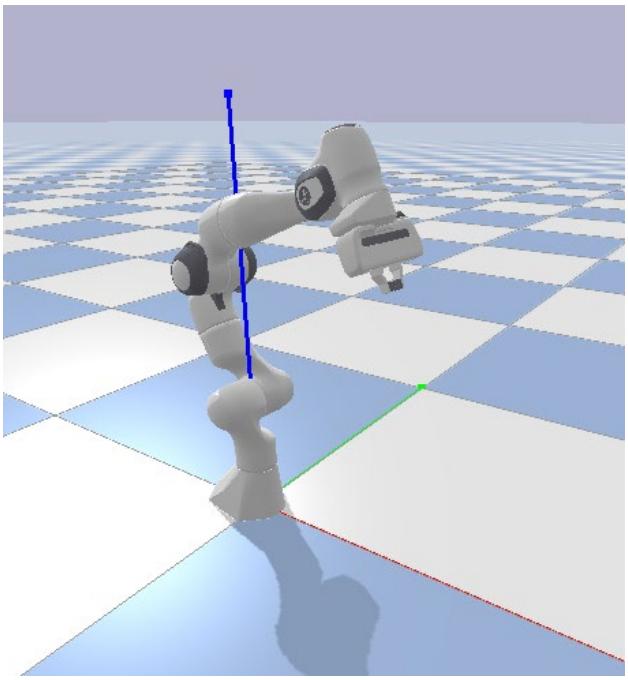

Рисунок 3.2 – Зображення робота маніпулятора Franka Panda у початковій формі

```
def _go_safe_pose(self):
    q_safe = np.array(
        [0.0, -0.5, 0.0, -1.8, 0.0, 1.8, 0.8],
        dtype=np.float64,)
    p.setJointMotorControlArray(
        self.robot,
        self.arm_joints,
        controlMode=p.POSITION_CONTROL,
        targetPositions=q_safe.tolist(),
        positionGains=[0.6] * len(self.arm_joints),
        forces=[220.0] * len(self.arm_joints),)
    for _ in range(240):
        p.stepSimulation()
```

Конфігурація q_safe відповідає позі, в якій рука витягнута вперед на середній висоті, це зручно відповідає початковою точкою для жестового керування. Виконання 240 кроків симуляції, де при 1 секунди і частоті 240 Гц, дозволяє маніпулятору плавно дійти до цільової позиції.

Параметри positionGains та forces визначають характеристики ПД-регуляторів PyBullet: перший контролює жорсткість позиційного керування, другий – максимальні крутні моменти в суглобах.

3.2.4 Система координат та обмеження робочої зони

Використовується локальна система координат відносно бази маніпулятора. Визначення осей:

а) x_local – вперед від робота;

б) y_local – вліво відносно фронтальної орієнтації;

в) z_local – вгору (збігається зі світовою Z).

При обертанні бази навколо вертикальної осі на кут θ локальні осі X та Y обертаються разом із роботом, тоді як вісь Z залишається незмінною. Перетворення координат між системами здійснюється через матриці повороту.

Для переходу від локальних координат до світових використовується матриця повороту навколо осі Z:

$$\begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_l \\ y_l \\ z_l \end{bmatrix}.$$

Перетворення світових координат в локальні координати:

$$\begin{bmatrix} x_l \\ y_l \\ z_l \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix}.$$

У коді ці перетворення реалізовані приватними методами _world_to_local() та _local_to_world():

```
def _world_to_local(self, world_vec: np.ndarray) -> np.ndarray:
    theta = self._get_base_rotation_angle()
    c, s = np.cos(theta), np.sin(theta)
        local_x = c * world_vec[0] + s * world_vec[1]
    local_y = -s * world_vec[0] + c * world_vec[1]
    local_z = world_vec[2]
        return np.array([local_x, local_y, local_z], dtype=float)
```

Метод _get_base_rotation_angle() зчитує поточне значення кута суглоба panda_joint1, після чого обчислюються синус та косинус для матриці повороту. Зворотне перетворення реалізоване аналогічно, з транспонованою матрицею. Робоча зона кінцевого ефектора визначена саме в локальних координатах:

```
self.workspace_min_local = np.array([0.10, -0.50, 0.05], dtype=float)
self.workspace_max_local = np.array([0.80, 0.50, 0.80], dtype=float)
```

Обмеження застосовуються саме в локальних координатах, щоб зберігати однакову доступну робочу зону незалежно від кута повороту бази. Якщо б обмеження задавались у світових координатах, то при повороті бази на 90° або 180° змінювалась би форма доступної області у світовому просторі, що ускладнило б керування – оператору довелося б постійно враховувати поточну орієнтацію робота. Графічне визначення обмежень робочої зони наведено на рис. 3.3.

Метод move_ee_delta(), який використовується для жестового керування, автоматично виконує всі необхідні перетворення координат:

```
def move_ee_delta(self, dx: float, dy: float, dz: float):
    dpos_local = np.array([dx, dy, dz], dtype=float)
    cur_pos_world, cur_orn = self.get_ee_pose()
    cur_pos_local = self._world_to_local(cur_pos_world)
        target_local = cur_pos_local + dpos_local
    target_local = np.maximum(target_local, self.workspace_min_local)
    target_local = np.minimum(target_local, self.workspace_max_local)
    target_world = self._local_to_world(target_local)
    self.move_ee_to(target_world.tolist(), target_orn=cur_orn)
```

Алгоритм працює наступним чином. Спочатку поточна позиція кінцевого ефектора у світових координатах перетворюється у локальні, потім до неї додається приріст (dx, dy, dz), який вже заданий у локальній системі. Результат обрізається до меж робочої зони, після чого перетворюється назад у світові координати та передається солверу інверсної кінематики PyBullet.

Виходь, що оператор керує роботом в інтуїтивних локальних координатах, а система автоматично враховує поточну орієнтацію бази при виконанні команд.

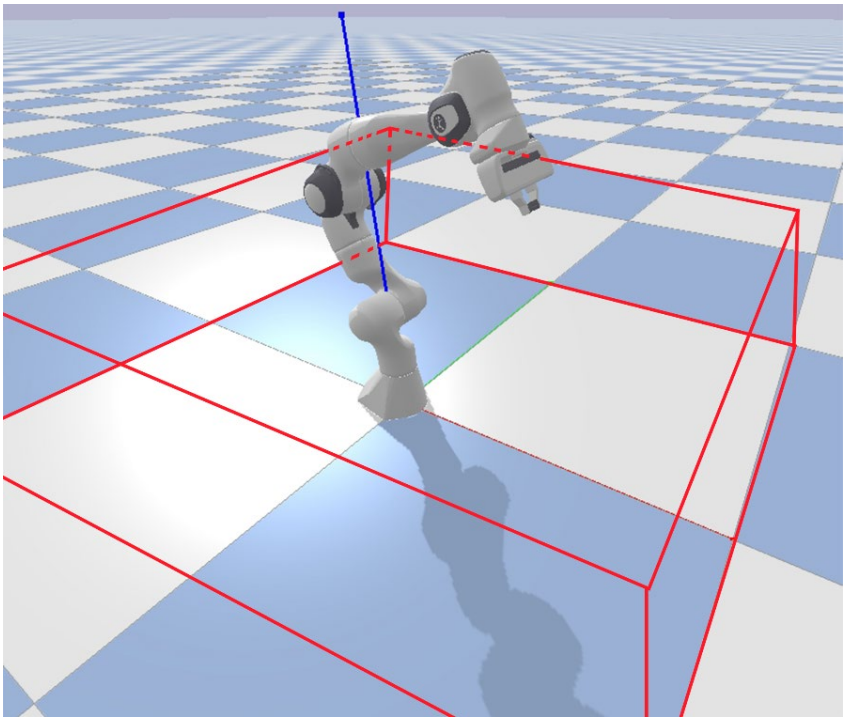

Рисунок 3.3 – Зображення меж робочої зони

3.2.5 Інверсна кінематика в декартовому просторі

Основною функцією модуля є метод move_ee_to(), який переміщує кінцевий ефектор у задану точку простору з використанням інверсної кінематики. Код наведено нижче.

```
def move_ee_to(self, target_pos, target_orn=None):
    target_pos = np.array(target_pos, dtype=float)
    target_pos = self._clamp_workspace(target_pos)
    if target_orn is None:
        _, current_orn = self.get_ee_pose()
        target_orn = current_orn
    else:
        target_orn = np.array(target_orn, dtype=float)
    q_full = p.calculateInverseKinematics(
        self.robot,
        self.ee_link, target_pos.tolist(), target_orn.tolist(),  )
    q_arm = list(q_full[:len(self.arm_joints)])
```

```
    self._set_arm_targets(q_arm)
```

Алгоритм працює наступним чином:

– цільова позиція проєктується на допустиму робочу зону;

– якщо орієнтація не задана, зберігається поточна орієнтація ефектора;

– викликається вбудований солвер IK бібліотеки PyBullet;

– отримані суглобові координати передаються контролерам.

Метод calculateInverseKinematics() використовує алгоритм заглушених найменших квадратів (damped least squares). Він забезпечує стабільність розв'язку навіть поблизу сингулярних конфігурацій маніпулятора. Для зручності жестового керування реалізовано метод move_ee_delta(), який дозволяє задавати рух у вигляді приростів:

```
def move_ee_delta(self, dx: float, dy: float, dz: float):
    pos, orn = self.get_ee_pose()
    target_pos = pos + np.array([dx, dy, dz], dtype=float)
    self.move_ee_to(target_pos, orn)
```

Тут оператор подає не абсолютні координати, а напрямки руху ("вліво", "вгору", "вперед").

Важливою особливістю реалізації є фіксація суглобів бази та запястя під час обчислення інверсної кінематики. Це дає незалежне керування цими ступенями свободи через окремі жести лівої руки, без конфлікту з командами позиціонування від правої руки:

```
q_curr = self.get_joint_positions().tolist()
base_val = q_curr[self.base_idx_in_arm]
lower_limits[self.base_idx_in_arm] = base_val
upper_limits[self.base_idx_in_arm] = base_val
```

```
joint_ranges[self.base_idx_in_arm] = 0.0
wrist_val = q_curr[self.wrist_idx_in_arm]
lower_limits[self.wrist_idx_in_arm] = wrist_val
upper_limits[self.wrist_idx_in_arm] = wrist_val
joint_ranges[self.wrist_idx_in_arm] = 0.0
```

Це досягається шляхом встановлення нижньої та верхньої межі для відповідних суглобів у значення їхньої поточної позиції, а діапазону руху – у нуль. Солвер інверсної кінематики PyBullet інтерпретує такі обмеження як заборону зміни цих суглобів і знаходить розв'язок тільки для решти п'яти ступенів свободи.

3.2.6 Отримання поточної пози кінцевого ефектора

Для забезпечення зворотного зв'язку реалізовано метод get_ee_pose():

```
def get_ee_pose(self):
    link_state = p.getLinkState(self.robot, self.ee_link)
    pos = np.array(link_state[4], dtype=float)
    orn = np.array(link_state[5], dtype=float)
    return pos, orn
```

PyBullet повертає стан ланки у вигляді послідовності, де індекс 4 відповідає позиції в світових координатах, а індекс 5 – орієнтації у форматі кватерніона. Перетворення у масиви NumPy спрощує подальші математичні операції.

3.2.7 Суглобове керування

Для випадків, коли потрібен прямий контроль окремих суглобів (наприклад, поворот бази або обертання зап’ястка), було реалізовано методи суглобового керування. Фрагмент коду наведено нижче.

```
def rotate_base_step(self, delta_angle: float):
    cur = p.getJointState(self.robot, self.base_joint)[0]
    new_angle = float(np.clip(cur + delta_angle,
    self.base_min, self.base_max))
    q = [p.getJointState(self.robot, j)[0] for j in self.arm_joints]
    q[self.base_joint] = new_angle
    self._set_arm_targets(q)
```

Алгоритм зчитує поточний кут суглоба, додає приріст, обмежує результат допустимим діапазоном та встановлює нову цільову конфігурацію. Аналогічно працює метод move_pitch_step() для керування суглобом плеча.

Метод apply_joint_offsets() дозволяє одночасно змінювати кілька суглобів:

```
def apply_joint_offsets(self, j1_delta: float = 0.0,
                        j6_delta: float = 0.0):
    if abs(j1_delta) < 1e-9 and abs(j6_delta) < 1e-9:
        return
        q = [p.getJointState(self.robot, j)[0] for j in self.arm_joints]
        min_lim = -2.8
    max_lim = 2.8
        if abs(j1_delta) > 0.0:
        q[1] = float(np.clip(q[1] + j1_delta, min_lim, max_lim))
        if abs(j6_delta) > 0.0:
        q[5] = float(np.clip(q[5] + j6_delta, min_lim, max_lim))
        self._set_arm_targets(q)
```

Він використовується для тонкого налаштування орієнтації кінцевого ефектора, зокрема для обертання зап'ястка у відповідь на жести лівої руки.

3.2.8 Керування хватом

Маніпулятор Franka Panda має паралельний двопальцевий хват з двома суглобами. Метод set_gripper() реалізує керування шириною розкриття:

```
def set_gripper(self, open_ratio: float):
    r = float(np.clip(open_ratio, 0.0, 1.0))
    width = r * 0.04
    left = width / 2.0
    right = -width / 2.0
        p.setJointMotorControl2(
        self.robot, 9,
        p.POSITION_CONTROL, targetPosition=left, force=40)
    p.setJointMotorControl2(
        self.robot, 10,
        p.POSITION_CONTROL, targetPosition=right, force=40)
```

Тут значення 0 відповідає повністю закритому хвату, 1 – максимально відкритому. Внутрішньо це перетворюється на симетричні координати лівого та правого пальців хвату.

3.2.9 Допоміжні методи

Приватний метод _set_arm_targets() інкапсулює виклик контролера PyBullet. Код наведено нижче.

```
    def _set_arm_targets(self, q_target: List[float]):
        assert len(q_target) == len(self.arm_joints)
        p.setJointMotorControlArray(
            bodyUniqueId=self.robot,
            jointIndices=self.arm_joints,
            controlMode=p.POSITION_CONTROL,
```

```
            targetPositions=q_target,
            positionGains=[0.4] * len(self.arm_joints),
            forces=[80.0] * len(self.arm_joints),)
```

Використання масивного виклику setJointMotorControlArray() замість циклу по окремих суглобах підвищує продуктивність, особливо при великій кількості викликів у режимі реального часу.

Метод get_state() повертає спрощене представлення стану робота через структуру даних RobotState:

```
    def get_state(self) -> RobotState:
        q = self.get_joint_positions()
        pos, orn = self.get_ee_pose()
        return RobotState(joint_positions=q, ee_pos=pos, ee_orn=orn)
```

Це використовується для логування та відображення поточного стану системи.

3.3 Реалізація модуля розпізнавання жестів

Модуль розпізнавання жестів складається з двох основних компонентів: функції екстракції ознак hand_to_feature() та класифікатора TemplateGestureClassifier. Ці компоненти реалізовані у файлі gesture_utils.py та забезпечують перетворення сирих даних з MediaPipe у символічні команди для маніпулятора.

3.3.1 Екстракція ознак з ключових точок руки

MediaPipe Hands повертає 21 тривимірну точку, що описують анатомічну структуру руки. Це зап'ясток, основи пальців, суглоби та кінчики. Безпосереднє використання цих координат для класифікації неефективне, оскільки вони

залежать від положення руки в кадрі, відстані до камери та розміру кисті користувача.

Функція hand_to_feature() реалізує нормалізацію координат для отримання інваріантного представлення жесту:

```
def hand_to_feature(landmarks):
    pts = np.array([[lm.x, lm.y, lm.z] for lm in landmarks],
                   dtype=np.float32)
    wrist = pts[0].copy()
    pts -= wrist middle_mcp = pts[9]
    scale = np.linalg.norm(middle_mcp)
    if scale < 1e-6:
        scale = 1.0
    pts /= scale
    return pts.flatten()
```

Алгоритм нормалізації виконує три послідовні перетворення. Перша це трансляція системи координат. Всі точки зміщуються так, щоб зап'ясток (точка з індексом 0) опинився в початку координат:

$$p_i' = p_i - p_0, \qquad i = 1, \dots, 20,$$

де p_i – початкові координати i-ї точки;

p_0 – координати зап’ястка.

Друге – масштабування за характерним розміром. Нормалізовані координати отримуються діленням на відстань від зап'ястка до основи середнього пальця:

$$s = \|p_9 - p_0\|_2,$$

$$\tilde{p}_i = \frac{p_i'}{s} = \frac{p_i - p_0}{s}.$$

Данна операція робить представлення інваріантним до розміру кисті користувача та відстані від камери. Вибір точки 9 (основа середнього пальця) обумовлений тим, що ця відстань залишається стабільною для різних жестів, точка добре відстежується MediaPipe навіть при частковому перекритті руки та ця відстань достатньо велика, щоб уникнути числових проблем при діленні.

Для захисту від вироджених випадків, коли рука занадто близько до камери і масштаб наближається до нуля, введено перевірку $if\ scale < 1e-6$, яка запобігає діленню на нуль.

Третє це сплощення у вектор ознак, де двовимірний масив координат розміром (21, 3) перетворюється у плоский вектор довжини 63:

$$f = [x_0, y_0, z_0, x_1, y_1, z_1, \dots, x_{20}, y_{20}, z_{20}]^T \in \mathbb{R}^{63}.$$

Цей спосіб подання даних виявився зручним для подальших обчислень, зокрема для знаходження евклідових відстаней та виконання інших векторних операцій у процесі класифікації.

Сформований вектор ознак має кілька важливих характеристик. По-перше, він не залежить від положення руки в кадрі, тобто зміщення руки не змінює його структуру. По-друге, завдяки нормалізації вектор практично не реагує на зміни масштабу, тому розмір кисті або відстань до камери не впливають на результат. Частково зберігається стійкість і до невеликих поворотів руки в площині кадру, яка забезпечує певну варіативність без втрати точності. Орієнтація долоні залишається суттєвим фактором, один і той самий жест, виконаний долонею до камери або з повернутою долонею, описується різними векторами.

3.3.2 Структура класу TemplateGestureClassifier

Класифікатор реалізовано на основі шаблонного підходу з використанням алгоритму k-найближчих сусідів (k-NN). На відміну від підходу з єдиним файлом на клас, у фінальній реалізації система працює з окремими директоріями для

кожного жесту. Таким чином можна зберігати багато еталонів без конфліктів імен. Структура класу наведена нижче:

```
class TemplateGestureClassifier:
    def __init__(self, root_dir: str = "gestures", k: int = 3):
        self.root = Path(root_dir)
        self.k = k
        self.samples: list[tuple[np.ndarray, str]] = []
```

Основні поля класу:

– root це кореневий каталог з еталонами, за замовчуванням gestures/;

– k це кількість найближчих сусідів для голосування;

– samples це список кортежів, що зберігає всі завантажені еталони.

3.3.3 Завантаження еталонних жестів з директорій

Метод load() реалізує рекурсивне сканування директорій та завантаження всіх еталонів у пам’ять:

```
def load(self):
    self.samples.clear()
    if not self.root.exists():
        return
    for label_dir in self.root.iterdir():
        if not label_dir.is_dir():
            continue
        label = label_dir.name
        for npy_file in label_dir.glob("*.npy"):
            vec = np.load(npy_file)
            self.samples.append((vec, label))
```

Алгоритм завантаження працює наступним чином. Спочатку відбувається очищення попередніх даних – виклик self.samples.clear() видаляє всі раніше завантажені еталони, дозволяючи повторно використовувати метод load() після додавання нових зразків. Далі перевіряється існування кореневої директорії. Якщо каталог відсутній, виводиться попередження і метод завершує роботу без генерації помилки.

На наступному етапі виконується ітерація по підкаталогах. Кожна підпапка інтерпретується як окремий клас жесту, а її ім'я використовується як мітка класу, наприклад gestures/STOP/ або gestures/GRIP_OPEN/. Для кожного знайденого файлу з розширенням .npy вектор ознак завантажується через np.load(), після чого формується пара вектор – мітка, яка додається до списку self.samples. Після завершення обробки виводиться інформаційне повідомлення про загальну кількість завантажених еталонів.

Вигляд структури каталогів наведено на рис. 3.4.

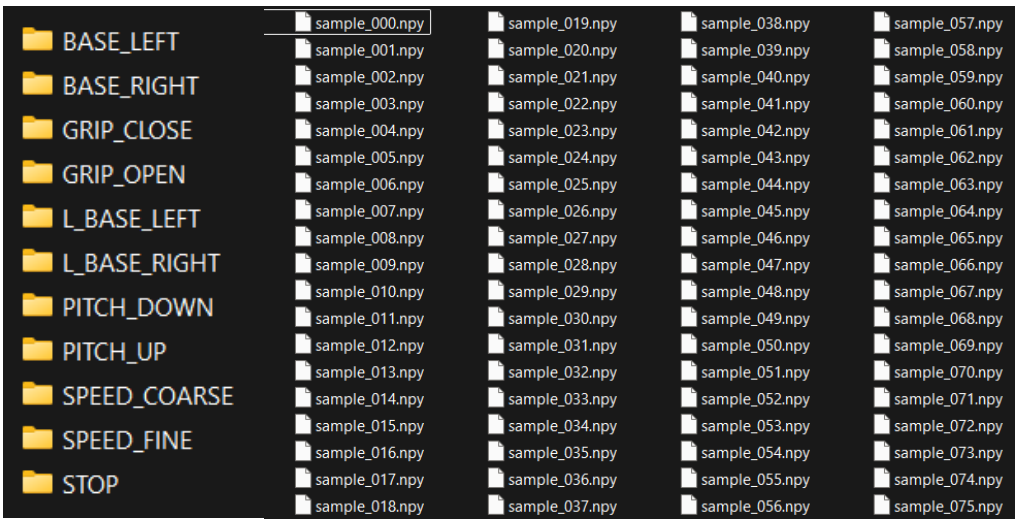

Рисунок 3.4 – Зображення директорій з жестами

3.3.4 Алгоритм класифікації на основі k-NN

Метод predict() є серцем всієї системи розпізнавання. Він перетворює нормалізований вектор координат руки у символічну назву жесту. Реалізація базується на класичному алгоритмі k-найближчих сусідів. Код наведено нижче.

```
def predict(self, vec: np.ndarray, distance_thresh: float = 1.0) -> str | None:
    if not self.samples:
        return None
    dists: list[tuple[float, str]] = []
    for tmpl_vec, label in self.samples:
        d = float(np.linalg.norm(vec - tmpl_vec))
        dists.append((d, label))
    dists.sort(key=lambda x: x[0])
    top = dists[:self.k]
    if not top:
        return None
    if top[0][0] > distance_thresh:
        return None
    labels = [lbl for _, lbl in top]
    best_label = max(set(labels), key=labels.count)
    return best_label
```

Спочатку перевіряється, чи взагалі є в системі збережені еталони. Якщо база порожня, наприклад користувач ще не записав жодного жесту, то класифікувати нічого, і метод повертає None.

Далі йде основна робота. Для кожного збереженого еталона обчислюється евклідова відстань до вхідного вектора.

Після цього список сортується за зростанням відстані, щоб найбільш схожі еталони опинилися на початку. З відсортованого списку беруться перші k елементів – це і є найближчі сусіди. У реалізації використовується $k = 3$, що

виявилося гарним балансом. Один сусід може бути випадковим викидом, а 5-7 уже занадто багато для невеликої бази еталонів.

Важливий момент, якщо було знайдено трьох найближчих сусідів, потрібно перевірити, чи вони насправді досить близькі. Якщо найближчий еталон віддалений більше ніж на поріг distance_thresh = 1,0, система буде не знати його. Це критично для жестового керування – краще нічого не зробити, ніж помилково розпізнати випадкову позу руки як команду для маніпулятора.

Значення порогу 1,0 підібрано експериментально. Спочатку було 0,5 – виявилося занадто жорстко, багато справжніх жестів відкидалося. Потім 1,5 – почали проскакувати випадкові спрацьовування. На значенні 1,0 вийшов нормальний компроміс. Система впевнено розпізнає навчені жести і нечасто помиляється на незнайомих позах.

Остання частина працює наступним чином, з трьох найближчих сусідів витягуються їхні мітки класів, і визначається, яка зустрічається найчастіше. Наприклад, були випадки, коли жест «STOP» розпізнавався з відстанями 0,3, 0,4 та 0,5 до трьох еталонів класу STOP. Але якщо третій найближчий еталон раптом мав мітку «GRIP_OPEN» з відстанню 0,5, то голосування 2:1 все одно правильно обирало STOP. Це простий, але дієвий спосіб фільтрації шуму.

3.4 Інтеграційний модуль телеоперування

Інтеграційний модуль, реалізований у файлі teleop_ik_bimanual.py, об'єднує всі раніше описані компоненти в єдину робочу систему. Його головна задача – організувати безперервний цикл обробки відеопотоку, розпізнавання жестів обох рук та перетворення їх у команди для маніпулятора. На відміну від модулів SimplePandaBase та TemplateGestureClassifier, які є самодостатніми компонентами, цей скрипт виконує роль диспетчера, що координує їхню взаємодію.

3.4.1 Ініціалізація компонентів системи

При запуску скрипта послідовно ініціалізуються всі необхідні підсистеми. Спочатку створюється екземпляр робота:

```
robot = SimplePandaBase(use_gui=True)
gripper_open = 1.0
robot.set_gripper(gripper_open)
```

Параметр use_gui=True вмикає графічне вікно симулятора PyBullet, що зручно для налагодження та демонстрації роботи системи. Хват одразу переводиться у відкритий стан (gripper_open = 1.0), щоб маніпулятор був готовий до захоплення об'єктів.

Далі завантажується класифікатор жестів:

```
classifier = TemplateGestureClassifier(root_dir="gestures", k=3)
classifier.load()
if not classifier.samples:
    return
```

Перевірка наявності збережених еталонів критична, без них система не зможе розпізнавати команди. Якщо директорія порожня, скрипт завершується з повідомленням про помилку, що запобігає запуску системи в непрацездатному стані.

Наступний крок ініціалізація відеокамери та MediaPipe:

```
cap = cv2.VideoCapture(0)
if not cap.isOpened():
    raise RuntimeError("камера не відкрилась")
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(
```

```
    static_image_mode=False,
    max_num_hands=2,
    min_detection_confidence=0.5,
    min_tracking_confidence=0.5,)
```

Параметр max_num_hands=2 дозволяє одночасно відстежувати обидві руки. Параметри min_detection_confidence та min_tracking_confidence встановлені у 0,5 – це компромісне ‘значення, яке забезпечує стабільне відстеження без надто частих втрат руки з кадру.

3.4.2 Параметри керування та режими роботи

Система підтримує два режими швидкості руху маніпулятора: грубий (COARSE) для швидкого переміщення та точний (FINE) для прецизійного позиціонування. Параметри задаються константами:

```
speed_mode = "COARSE"
COARSE_SPEED = 0.20
FINE_SPEED = 0.05
WRIST_SPEED_COARSE = 0.9
WRIST_SPEED_FINE = 0.3
BASE_SPEED_COARSE = 0.8
BASE_SPEED_FINE = 0.3
```

Значення швидкостей підібрані експериментально. Для декартових переміщень через ІК швидкість 0,20 м/с у грубому режимі дозволяє швидко наблизити кінцевий ефектор до цільової області, тоді як 0,05 м/с у точному режимі забезпечує контрольоване позиціонування з точністю до сантиметра.

Для кутових рухів (поворот бази та зап’ястка) швидкості у радіанах на секунду вибрані так, щоб повний оберт на 180° займав приблизно від 2 с до 3 с у грубому режимі та від 6 с до 8 с у точному.

Для перемикання між режимами та активації спеціальних функцій введено механізм утримання жесту:

MODE_HOLD_SEC = 2.0

Жест повинен стабільно розпізнаватися протягом 2 с, перш ніж система прийме команду зміни режиму або паузи.

3.4.3 Головний цикл обробки

Основна логіка системи організована як безкінечний цикл, що виконується з частотою, обмеженою тільки швидкістю обробки кадрів камери. Фрагмент коду наведено нижче.

```
last_time = time.time()
while True:
    now = time.time()
    dt = now - last_time
    if dt <= 0:
        dt = 1.0 / 60.0
    last_time = now
        ok, frame = cap.read()
    if not ok or frame is None:
        robot.step_sim()
        continue
```

На початку кожної ітерації обчислюється інтервал часу dt з моменту попереднього кадру. Це значення використовується для масштабування команд руху – чим більший інтервал, тим більше маніпулятор зміститься за один крок. Якщо з якихось причин dt виявляється нульовим або від’ємним, встановлюється безпечне значення 1/60 с.

Захоплення кадру обгорнуте перевіркою успішності операції. Якщо камера тимчасово недоступна або повертає пустий кадр, система просто виконує крок симуляції без оновлення команд, але не завершується. Це дозволяє системі пережити короткочасні збої відеопотоку.

Кадр зчитується у форматі BGR (стандарт OpenCV) і одразу дзеркально відбивається:

```
frame = cv2.flip(frame, 1)
h, w, _ = frame.shape
rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
```

3.4.4 Виявлення та класифікація жестів

MediaPipe обробляє кожен кадр та повертає інформацію про виявлені руки:

```
result = hands.process(rgb)
current_label = {"Left": None, "Right": None}
stable_label = {"Left": None, "Right": None}
if result.multi_hand_landmarks:
    for hand_landmarks, handedness in zip(
        result.multi_hand_landmarks, result.multi_handedness):
        side = handedness.classification[0].label
        vec = hand_to_feature(hand_landmarks.landmark)
        label = classifier.predict(vec, distance_thresh=0.9)
        current_label[side] = label
```

Тут MediaPipe визначає ліву та праву руку з точки зору камери, а не оператора. Тому при дзеркальному відбитті кадру фізична права рука оператора позначається як "Right" у системі. Для кожної виявленої руки виконується послідовність операцій – екстракція нормалізованих ознак через

hand_to_feature() та класифікація з порогом відстані 0,9. Результат зберігається у словнику current_label, що індексується стороною руки.

3.4.5 Реалізація часової стабілізації розпізнавання

Сирі результати класифікації можуть коливатися від кадру до кадру через шум у відстеженні та неоднозначності на межах класів. Для згладжування введено механізм підрахунку стабільних послідовностей:

```
REQUIRED_FRAMES = 3
for side in ("Left", "Right"):
    cl = current_label[side]
    if cl == last_label[side] and cl is not None:
        same_count[side] += 1
    else:
        same_count[side] = 1
        last_label[side] = cl
    if cl is not None and same_count[side] >= REQUIRED_FRAMES:
        stable_label[side] = cl
    else:
        stable_label[side] = None
```

Система рахує, скільки кадрів підряд розпізнається той самий жест. Тільки коли лічильник досягає REQUIRED_FRAMES = 3, жест вважається стабільним і використовується для генерації команд. Це означає, що при частоті 30 кадрів/с потрібно приблизно 100 мілісекунд стабільного утримання жесту, перш ніж він почне діяти.

Значення 3 кадри виявилося оптимальним компромісом. При 1-2 кадрах система занадто швидко реагує на випадкові помилки розпізнавання, що призводить до тремтіння команд. При 5-6 кадрах затримка реакції стає помітною

для оператора – доводиться свідомо утримувати жест довше, ніж здається природним.

3.4.6 Обробка спеціальних режимів

Жест STOP має подвійну функцію. Миттєва зупинка руху та перемикання режиму паузи при утриманні. Реалізація цієї логіки вимагає відстеження часу утримання жесту:

```
stop_stable = (
    stable_label["Left"] == "STOP" or stable_label["Right"] == "STOP")
if stop_stable:
    if not stop_hold_active:
        stop_hold_active = True
        stop_hold_start = now
    else:
        if stop_hold_start is not None and (now - stop_hold_start) >= MODE_HOLD_SEC:
            control_paused = not control_paused
            state_str = "PAUSED" if control_paused else "ACTIVE"
            print(f"[INFO] CONTROL -> {state_str} (STOP hold)")
            stop_hold_active = False
            stop_hold_start = None
else:
    stop_hold_active = False
    stop_hold_start = None
```

Коли STOP вперше розпізнається як стабільний, система запам’ятовує час початку утримання. Якщо жест продовжує розпізнаватися протягом 2 с, відбувається перемикання стану control_paused. У цьому стані всі жести розпізнаються і відображаються на екрані, але не генерують команди для

робота – це свого роду заморожування керування для безпечної зміни положення рук або відпочинку оператора.

Якщо STOP відпущено раніше 2 с, лічильник скидається Це зроблено для того, щоб використовувати короткий показ жесту для миттєвої зупинки руху без перемикання режиму паузи.

Аналогічна логіка застосована для перемикання швидкості через жести SPEED_COARSE та SPEED_FINE:

```
right_stable = stable_label["Right"]
if right_stable in ("SPEED_COARSE", "SPEED_FINE") and not stop_active:
    if mode_hold_label != right_stable:
        mode_hold_label = right_stable
        mode_hold_start = now
    else:
        hold_time = now - mode_hold_start
        if hold_time >= MODE_HOLD_SEC:
            if right_stable == "SPEED_COARSE":
                if speed_mode != "COARSE":
                    speed_mode = "COARSE"
                    print("[INFO] SPEED -> COARSE")
            elif right_stable == "SPEED_FINE":
                if speed_mode != "FINE":
                    speed_mode = "FINE"
                    print("[INFO] SPEED -> FINE")
            mode_hold_label = None
```

Тут важлива перевірка if speed_mode != "COARSE" – вона запобігає повторному виведенню повідомлення про перемикання, якщо режим вже активний.

3.4.7 Швидка реакція на керування хватом

На відміну від команд руху, керування хватом реалізоване з миттєвою реакцією на поточний, нестабілізований жест:

```
if not stop_active:
    for side in ("Left", "Right"):
        cl = current_label[side]
        if cl == "GRIP_OPEN" and gripper_open != 1.0:
            gripper_open = 1.0
            robot.set_gripper(gripper_open)
            print(f"[INFO] GRIP -> OPEN ({side})")
        elif cl == "GRIP_CLOSE" and gripper_open != 0.0:
            gripper_open = 0.0
            robot.set_gripper(gripper_open)
            print(f"[INFO] GRIP -> CLOSE ({side})")
```

Використання current_label замість stable_label означає, що хват реагує негайно, без затримки на 3 кадри. Це було свідоме рішення, оскільки керування хватом зазвичай відбувається у статичних ситуаціях, коли кінцевий ефектор вже позиціонований над об'єктом. У таких умовах короткочасні помилки розпізнавання малоймовірні, а затримка на 100 мс може здаватися дратівливою.

Перевірка поточного стану хвату (gripper_open != 1.0) запобігає повторним викликам set_gripper() на кожному кадрі, коли жест продовжує розпізнаватися, але хват уже перебуває у потрібному стані.

3.4.8 Маппінг жестів на команди руху

Центральна частина системи – перетворення стабілізованих жестів у команди для маніпулятора. Реалізовано розподіл відповідальності між руками з використанням локальної системи координат робота:

```
dx = dy = dz = 0.0
wrist_cmd = 0.0
base_cmd = 0.0
if not stop_active:
    rs = stable_label["Right"]
    if rs == "BASE_LEFT":
        dx -= 1.0
    elif rs == "BASE_RIGHT":
        dx += 1.0
      if rs == "PITCH_UP":
        dz += 1.0
    elif rs == "PITCH_DOWN":
        dz -= 1.0
      ls = stable_label["Left"]
    if ls == "L_BASE_LEFT":
        base_cmd += 1.0
    elif ls == "L_BASE_RIGHT":
        base_cmd -= 1.0
      if ls == "PITCH_UP":
        wrist_cmd += 1.0
    elif ls == "PITCH_DOWN":
        wrist_cmd -= 1.0
```

Значення команд dx, dz, base_cmd, wrist_cmd, є нормалізованими напрямками у діапазоні [-1, 1]. Вони масштабуються відповідно до поточного режиму швидкості:

```
if speed_mode == "COARSE":
    ik_speed = COARSE_SPEED
    wrist_speed = WRIST_SPEED_COARSE
```

```
    base_speed = BASE_SPEED_COARSE
else:
    ik_speed = FINE_SPEED
    wrist_speed = WRIST_SPEED_FINE
    base_speed = BASE_SPEED_FINE
dx *= ik_speed * dt
dy *= ik_speed * dt
dz *= ik_speed * dt
j6_delta = wrist_cmd * wrist_speed * dt
base_delta = base_cmd * base_speed * dt
```

Множення на dt забезпечує постійну швидкість руху незалежно від частоти кадрів камери. Якщо система обробляє 30 кадрів/с, dt ≈ 0,033 с, і за один кадр маніпулятор зміститься на 0,20 × 0,033 ≈ 0,0066 м (6,6 мм) у грубому режимі. При падінні частоти до 15 кадрів/с dt подвоюється, і крок також подвоюється, зберігаючи швидкість 0,20 м/с.

3.4.9 Виконання команд через інверсну кінематику

Після формування приростів у локальних координатах вони передаються роботу через метод move_ee_delta(), який автоматично враховує поточний кут повороту бази:

```
if abs(dx) > 1e-6 or abs(dy) > 1e-6 or abs(dz) > 1e-6:
    robot.move_ee_delta(dx, dy, dz)
```

Метод автоматично виконує всі необхідні перетворення координат з урахуванням поточного кута повороту бази, після чого викликає солвер інверсної кінематики PyBullet. Всередині метод зчитує поточну позицію кінцевого ефектора у світових координатах, перетворює її у локальні, додає приріст (dx, dy, dz), застосовує обмеження робочої зони у локальних

координатах, перетворює результат назад у світові координати та передає їх у calculateInverseKinematics().

Внутрішньо метод перетворює локальні зміщення у світові координати через матриці повороту, що дозволяє IK-солверу PyBullet коректно обчислити суглобові кути з урахуванням орієнтації бази.

Перевірка на нульові прирости з точністю до 1e-6 запобігає зайвим викликам IK-солвера, коли оператор не подає команд руху. Це важливо, оскільки calculateInverseKinematics() у PyBullet є відносно витратною операцією – навіть при нульовому зміщенні він виконує ітераційні обчислення для знаходження суглобових координат.

Поворот бази виконується через суглобове керування:

```
if abs(base_delta) > 1e-6:
    robot.rotate_base_step(base_delta)
```

Це змішування підходів виправдане тим, що поворот навколо вертикальної осі є природним ступенем свободи, який не створює конфліктів з декартовим позиціонуванням кінцевого ефектора. Використання IK для цього ступеня ускладнило б систему без реальних переваг.

Обертання зап'ястка також реалізовано через прямий контроль суглоба:

```
if abs(j6_delta) > 1e-6:
    robot.apply_joint_offsets(j1_delta=0.0, j6_delta=j6_delta)
```

Тут передається j1_delta=0.0, щоб не чіпати інші суглоби. Метод apply_joint_offsets() дозволяє змінювати кілька суглобів одночасно, але у цьому контексті модифікується тільки зап'ясток.

3.5 Методика експериментальних досліджень

3.5.1 Точність розпізнавання жестів

Для експерименту було підготовлено базу з 10 жестів, кожен з яких представлений 150-300 еталонами, записаними у різних варіаціях. Потрібно визначити, наскільки надійно система розпізнає навчені жести та чи часто виникають помилкові спрацьовування.

Тестування виконувалось наступним чином. Кожен жест демонструвався 15 разів в різних умовах (різна відстань, положення в кадрі, швидкість входу в позу). Після кожної демонстрації фіксувалось:

– чи розпізнала система жест взагалі;

– чи розпізнала правильний клас;

– скільки кадрів знадобилося для стабілізації.

Також перевірялася поведінка на невідомих позах. Різноманітні випадкові положення руки, які не є жодним з навчених жестів. Загальна точність розпізнавання склала 92,5 %. Результат експерименту наведено у таблиці 3.2.

Таблиця 3.2 – Статистика точності розпізнавання жестів

Жест	Успішно розпізнано	Точність, %
STOP	15 з 15	95
GRIP_OPEN	13 з 15	88
GRIP_CLOSE	14 з 15	90
L_BASE_LEFT	15 з 15	94
L_BASE_RIGHT	15 з 15	94
PITCH_UP	15 з 15	96
PITCH_DOWN	14 з 15	91
BASE_RIGHT	13 з 15	85
BASE_LEFT	15 з 15	94
SPEED_COARSE/ SPEED_FINE	15 з 15	98

Найгіршу точність показали жести BASE_RIGHT та GRIP_OPEN. Вони виявились схожими між собою, і при швидкому переході з одного в інший іноді траплялись помилки.

3.5.2 Часові характеристики

Для оцінювання часових характеристик системи жестового керування був проведений окремий експеримент, під час якого вимірювався час виконання основних етапів обробки одного кадру. У модулі timing_experiment.py в головний цикл телеробота було додано послідовні вимірювання із використанням функції time.time(). Для кожного кадру фіксувалися такі величини як час захоплення зображення з вебкамери, час обробки кадру в MediaPipe, тривалість класифікації жестів, час формування команд, тривалість обчислення інверсної кінематики, час кроку симуляції PyBullet та накладання службової візуалізації. Додатково обчислювався загальний час обробки кадру та миттєве значення кадрів в секунду.

Кожен вимір зберігався у файл timing_results.csv з точністю до мілісекунди. Тому, для ~2 тис. послідовних кадрів отримано вибірку, яка описує динаміку завантаження всіх підсистем у реальному режимі роботи (демонстрація жестів, рух маніпулятора, активація/деактивація керування тощо). Паралельно у файл gesture_timing.csv записувалися часові мітки моментів, коли розпізнаний жест переходив у стабільний стан, та перших кадрів, у яких система починала генерувати ненульові команди руху. На основі цих даних обчислювалася end-to-end затримка, тобто жест → початок руху.

Першим експериментом було проведено розподіл часу між окремими операціями. На рис. 3.5 показано середній розподіл часу виконання між основними операціями обробки одного кадру. Середній загальний час циклу становить близько 48-50 мс, що відповідає частоті оновлення близько 15-16 кадрів за секунду.

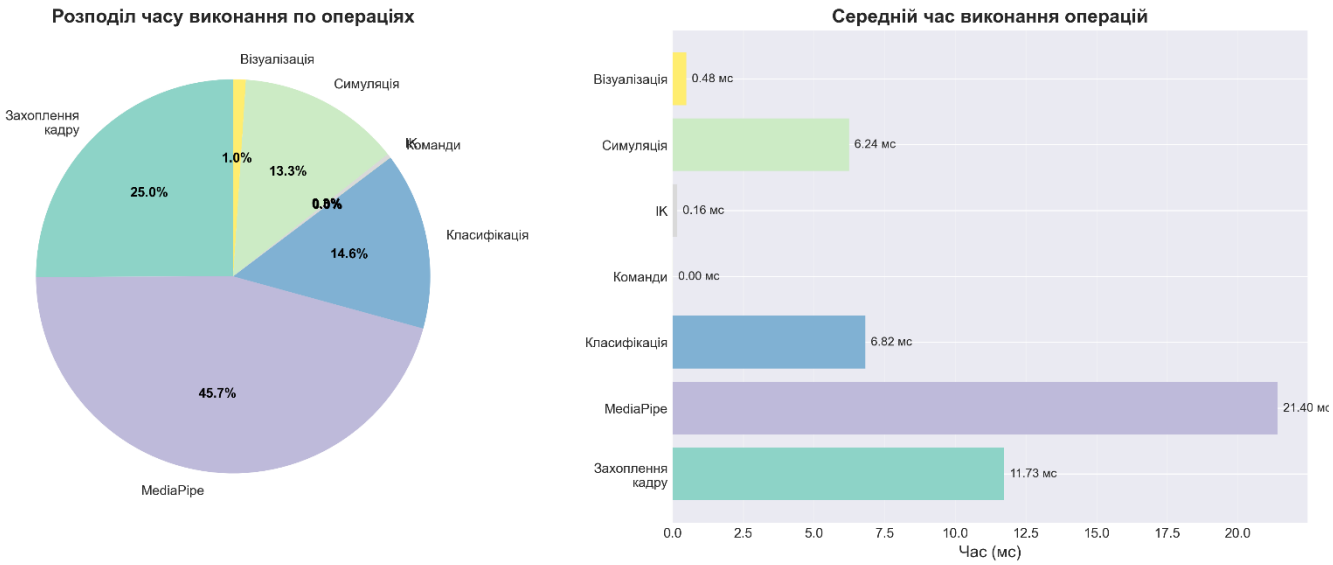

Рисунок 3.5 – Зображення діаграми середнього часу циклу

Найбільший внесок у затримку дає блок обробки в MediaPipe – у середньому 21,40 мс, або ≈ 45,7 % від загального часу кадру. Другим за навантаженням є етап захоплення кадру з вебкамери – 11,73 мс (≈25,0 %). Блок класифікації жестів на основі векторів ознак займає 6,82 мс (≈14,6 %), а крок симуляції PyBullet – 6,24 мс (≈13,3 %). На решту операцій припадає незначна частка часу. Інверсна кінематика в середньому виконується менш ніж за 0,2 мс, формування команд займає практично нульовий час, а візуалізація текстових підписів – близько 0,48 мс. Таким чином, часові витрати концентруються переважно у двох підсистемах – захопленні відео та обробці кадрів у MediaPipe. Це означає, що потенційна оптимізація системи повинна в першу чергу бути спрямована саме на ці етапи.

Другий експеримент про динаміку часу виконання впродовж цього експерименту. На рис. 3.6 наведено часові ряди тривалості окремих операцій для кожного кадру, разом із ковзним середнім та горизонтальною лінією, що позначає середнє значення по всьому експерименту. Для обробки MediaPipe добре видно повільні коливання тривалості, у межах від 15 мс до 30 мс, пов’язані зі зміною сцени та кількості рук у кадрі. Графік класифікації жестів показує більш шумну картину, але з чітко помітним плато на рівні від 6 мс до 8 мс, що відповідає обчисленню відстаней до шаблонів для двох рук.

Для етапів інверсної кінематики та симуляції PyBullet характерні поодинокі сплески, пов’язані з різкою зміною положення кінцевого ефектора, але навіть у пікові моменти час виконання цих операцій не перевищує десятків мілісекунд, а середнє значення залишається на рівні 0,16 мс та 6,24 мс відповідно. Час захоплення кадру, візуалізації та формування команд залишається практично сталим протягом усього експерименту, що свідчить про відсутність накопичення затримок у цих підсистемах. На останньому графіку (рис. 3.6) з назвою загальний час циклу, видно, що система працює в сталому режимі без деградації продуктивності з часом.

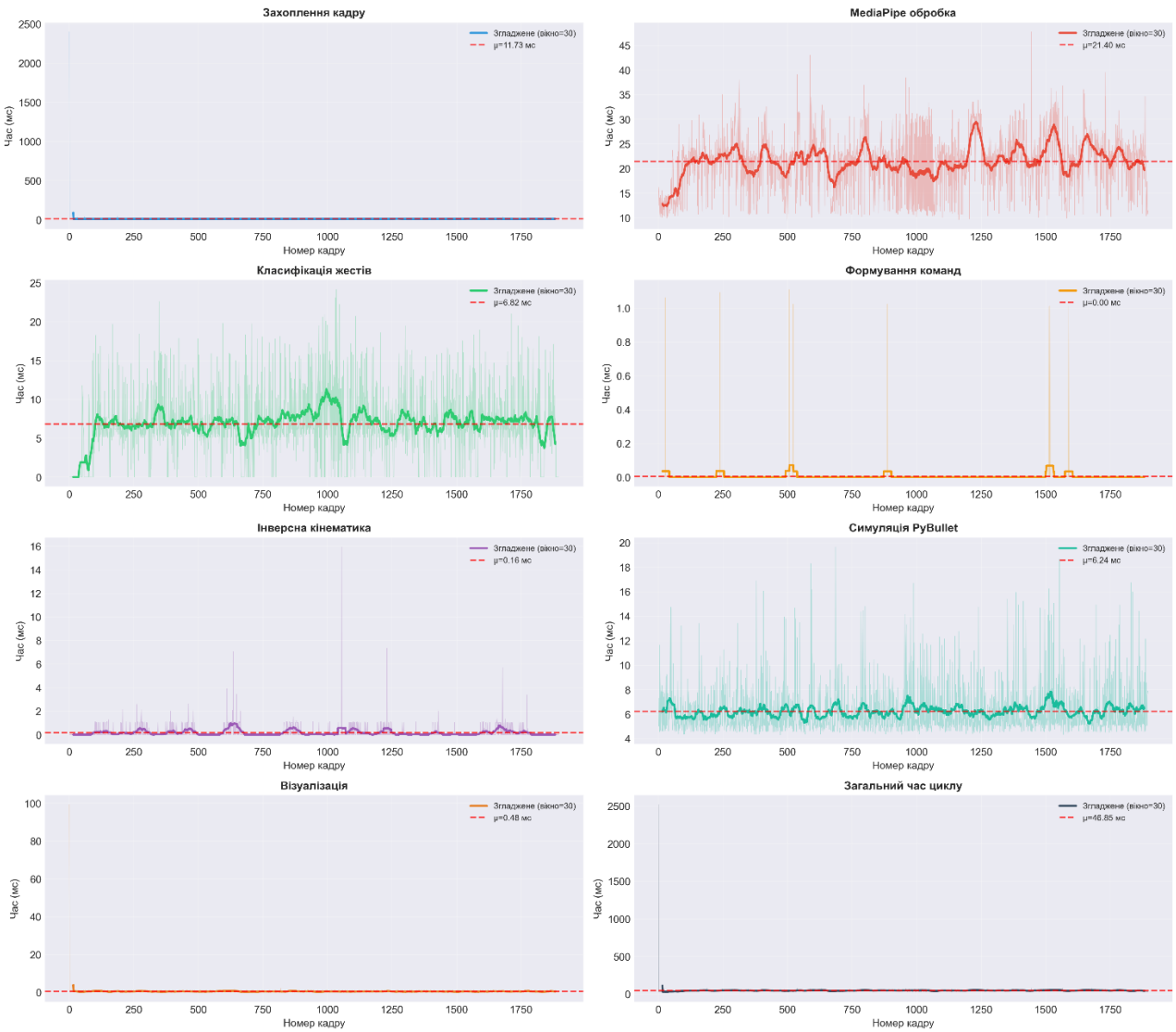

Рисунок 3.6 – Часові характеристики операцій системи жестового керування

Третім експериментом проводилось дослідження частоти оновлення кадрів. Аналіз частоти оновлення наведено на рис. 3.7. Верхній графік показує зміну миттєвої частоти для кожного кадру.

Після початкових перехідних процесів система виходить на стаціонарний режим із середнім значенням 15,71 кадрів/с. Заштрихована смуга позначає інтервал середнє ± 1 стандартне відхилення, в межах якого знаходиться більшість спостережень.

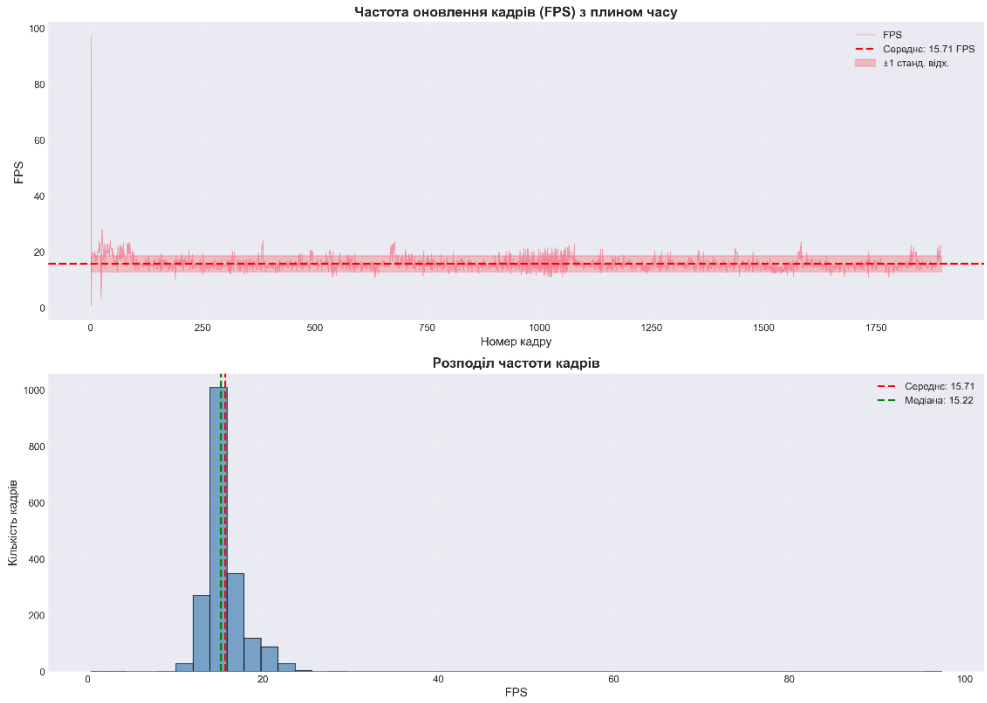

Рисунок 3.7 – Зображення графіка з частотою оновлення кадрів

Нижній графік підтверджує, що основна маса кадрів обробляється з частотою 14-18 кадрів/с. Медіанне значення 15,22 кадрів/с практично збігається із середнім, це свідчить про відсутність сильної асиметрії розподілу та про стабільну роботу циклу телеробота.

Останній експеримент був про End-to-end затримку, тобто жест → початок руху. На окремому кроці аналізу для кожної руки визначалися моменти, коли розпізнаний жест переходив у стабільний стан та перший кадр, у якому система формувала не нульову команду руху. Різниця між цими часовими мітками інтерпретується як end-to-end затримка від моменту стабілізації жесту до

фактичного початку руху маніпулятора. Для уникнення артефактів у аналіз включалися лише значення в діапазоні від 0 до 1000 мс. Розподіл отриманих затримок показано на рис. 3.8. Гістограма має виражену праву асиметрію. Більшість випадків зосереджена в області менше 100 мс, тоді як поодинокі спостереження тягнуться до значень порядку від 0,8 до 1,0 с. Медіанна затримка становить 69,5 мс, це приблизно однин кадр при середній частоті, тоді як середнє значення через наявність викидів суттєво вище – 210,8 мс.

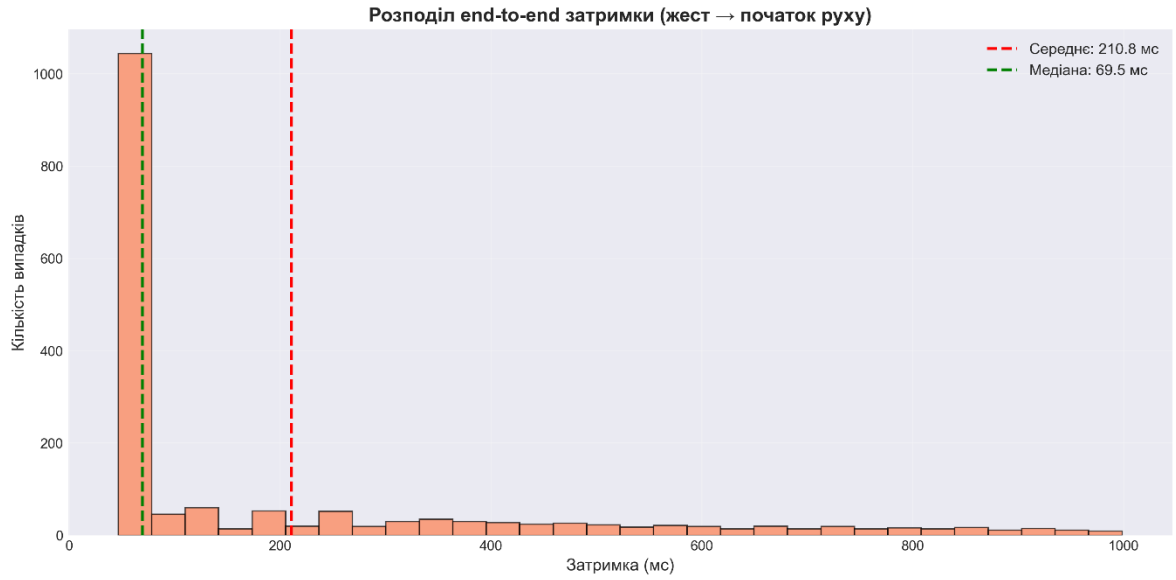

Рисунок 3.8 – Графік розподілу end-to-end затримки

Практично це означає, що у типовому сценарії, коли жест вже стабілізувався, маніпулятор починає рух упродовж одного-двох кадрів. Більші значення затримки пов’язані з ситуаціями, коли користувач змінює жест або ненадовго зриває його, що призводить до додаткового часу на повторну стабілізацію. З точки зору інтерактивності системи доцільно орієнтуватися на медіанне значення, тоді як поодинокі викиди можуть бути зменшені окремим налаштуванням порогів стабілізації або покращенням якості відеосигналу.

3.6 Заходи з охорони праці

У процесі розробки, налагодження та тестування системи виникає низка небезпечних та шкідливих факторів. Тривала робота за комп’ютером, обробка відеопотоку та аналіз візуальної інформації з камери створюють підвищене навантаження на зір. Робота програміста передбачає довге перебування у сидячому положенні. Також робота з персональним комп’ютером, камерою та периферійним обладнанням вимагає дотримання правил електробезпеки.

Хоча у даній роботі використовується симуляційне середовище, система жестового керування призначена для перенесення на фізичний робототехнічний комплекс. Небезпека при роботі з фізичним роботом полягає в неконтролювані переміщення маніпулятора у випадку хибного розпізнавання жесту, зіткнення рухомих ланок із оператором, а також ризик травм при роботі в безпосередній близькості до приводів та мехатронічних вузлів.

Для мінімізації впливу вищезазначених факторів передбачено комплекс організаційних і технічних заходів. Щодо безпечної роботи за комп’ютером необхідно забезпечити оптимальну освітленість робочого місця без засліплювальних відблисків, використовувати монітор діагоналлю не менше 22 дюймів з антибліковим покриттям на відстані від 50 см до 70 см від очей. Також слід робити перерви тривалістю 10 хв після кожних двох годин роботи. Ергономічна організація робочого місця передбачає використання крісла з регульованою висотою та підтримкою поперекового відділу, розміщення клавіатури та миші у зоні комфортної досяжності, забезпечення правильної постави з опорою для спини.

Оскільки система побудована з урахуванням майбутнього застосування на реальному обладнанні, на програмному рівні реалізовано жест екстреної зупинки STOP, який блокує рухи маніпулятора. Система забезпечує відсутність активних команд при відсутності руки у кадрі або при низькій достовірності розпізнавання. Програмно обмежено робочу зону кінцевого ефектора із забороною виходу за межі безпечного простору. У випадку перенесення системи

на фізичний робот необхідно дотримуватися додаткових правил. Заборона знаходження кінцівок оператора у зоні руху робота, розміщення маніпулятора у захисній огорожі, наявність фізичної кнопки аварійного відключення, використання занижених швидкостей на початкових етапах адаптації [43].

Серед потенційних ризиків відмови системи – хибне розпізнавання жесту, зависання програмного модуля, збої камери та непередбачувані конфігурації через сингулярності інверсної кінематики. Для їх мінімізації застосовано фільтрацію жестів у часі з пороговими значеннями достовірності, обмеження максимальної швидкості маніпулятора, переведення системи у безпечний режим при втраті руки та використання симуляції для попередньої перевірки всіх оновлень.

3.7 Висновки до розділу 3

У цьому розділі реалізовано повнофункціональну систему жестового керування роботизованим маніпулятором та проведено її експериментальну апробацію в симуляційному середовищі. Розроблена програмна архітектура підтвердила, що обраний підхід – поєднання MediaPipe для виділення ключових точок кисті, шаблонної класифікації жестів та інверсної кінематики засобами PyBullet – забезпечує стабільне й передбачуване керування маніпулятором у режимі, близькому до реального часу.

Модульна реалізація довела доцільність поділу системи на три рівні: низькорівневий модуль взаємодії із симулятором, модуль розпізнавання жестів та інтеграційний контролер. Важливу роль відіграє використання локальної системи координат робота, що забезпечує інтуїтивне керування незалежно від поточного кута повороту бази.

Експериментальна перевірка показала, що середня точність розпізнавання жестів перевищує 90 %, а середній час реакції від появи жесту в кадрі до початку руху маніпулятора становить приблизно 200–220 мс, що є прийнятним для більшості задач телеоперування. Підтверджено ключові властивості вектора

ознак – інваріантність до зсуву та масштабу, – а також стабільну роботу класифікатора за умови застосування часової фільтрації.

Окремою перевагою системи є вбудовані механізми безпеки, що гарантують контрольовану поведінку навіть за можливих помилок розпізнавання. Жест STOP, блокування керування, обмеження робочої зони та фіксація окремих суглобів під час розв’язання оберненої кінематичної задачі суттєво зменшують ризик некоректних рухів маніпулятора. Експерименти підтвердили, що ці механізми надійно спрацьовують навіть при динамічній зміні жестів і поодиноких помилках класифікації.

ВИСНОВКИ

У роботі проведено аналіз сучасних роботизованих маніпуляторів, сфер їх застосування та існуючих підходів до безконтактного керування. Показано, що жестове керування на основі комп'ютерного зору є перспективним напрямом розвитку інтерфейсів людино-машинної взаємодії, оскільки дозволяє відмовитися від спеціалізованих пристроїв введення та забезпечити більш природну взаємодію оператора з маніпулятором. Разом з тим виявлено ключові проблеми такого підходу: залежність від умов освітлення, чутливість до шумів у зображенні, неоднозначність деяких жестів і необхідність гарантування безпеки рухів робота.

Виконано огляд програмних засобів комп'ютерного зору (OpenCV, MediaPipe, фреймворки глибокого навчання) та симуляційних середовищ для робототехніки (PyBullet, Gazebo, MuJoCo тощо). Обґрунтовано вибір зв'язки MediaPipe + PyBullet для реалізації жестового керування. MediaPipe забезпечує достатню точність і швидкодію при виділенні ключових точок кисті в реальному часі, а PyBullet надає інструменти для моделювання кінематики та динаміки маніпулятора Franka Emika Panda з можливістю програмного доступу до функцій інверсної кінематики.

Розроблено архітектуру системи жестового керування, яка складається з модулів захоплення та попередньої обробки відеопотоку, відстеження руки та отримання координат ключових точок за допомогою MediaPipe, формування нормалізованого вектора ознак, шаблонної класифікації жестів; інтерпретації команд, керування маніпулятором у локальній системі координат, симуляції та візуалізації. Це спростило налагодження, подальший розвиток системи та перенесення окремих компонентів на інші робототехнічні платформи.

Побудовано математичну модель маніпулятора на основі модифікованої DH-конвенції, визначено структуру кінематичного ланцюга та узагальнені координати. Сформульовано задачу інверсної кінематики для кінцевого

ефектора та показано, як вона розв’язується із використанням вбудованих функцій PyBullet. Запропоновано роботу у локальній системі координат, жорстко пов’язаній з базою маніпулятора. Це дозволило уніфікувати жести оператора та зменшити залежність керування від абсолютної орієнтації робота у просторі. Уведено обмеження робочої зони кінцевого ефектора, які запобігають виходу маніпулятора за допустимі межі.

Реалізовано програмний комплекс жестового керування в середовищі PyCharm. Розроблено клас SimplePandaBase для роботи з маніпулятором Franka Emika Panda в PyBullet, модуль перетворення координат між локальною та світовою системами, а також модулі, що реалізують побудову вектора ознак та шаблонну класифікацію жестів. Впроваджено набір із 12 статичних жестів, які відповідають базовим командам керування. Реалізовано жест STOP, який використовується як екстрена зупинка та для ввімкнення/вимкнення режиму жестового керування, що підвищує безпеку експлуатації системи.

Проведено експериментальні дослідження точності розпізнавання жестів та часових характеристик системи. Для бази з 10 жестів при 15 повтореннях кожного досягнуто середньої точності 92,5 %, причому для більшості жестів точність знаходиться в діапазоні 90–98 %. Найнижчу точність показали окремі жести, близькі за формою або схильні до неточних демонстрацій, що вказує на доцільність подальшого удосконалення набору жестів та алгоритмів стабілізації. Аналіз часових характеристик показав, що найбільший внесок у загальну затримку дає етап обробки в MediaPipe, тоді як модулі інверсної кінематики та симуляції працюють у сталому режимі з малим часом виконання. Виміряна end-to-end затримка становить близько 210 мс.

Практичне значення одержаних результатів полягає у створенні прототипу системи жестового керування маніпулятором Franka Emika Panda, яка може бути використана як навчально-дослідницький стенд для підготовки фахівців з автоматизації та робототехніки, а також як основа для подальшої інтеграції з фізичним робототехнічним обладнанням. Запропонована архітектура та програмна реалізація можуть бути адаптовані для інших типів маніпуляторів та

розширені шляхом використання більш складних моделей машинного навчання, підтримки динамічних жестів та підключення додаткових сенсорів.

Подальший розвиток роботи доцільно спрямувати на підвищення стійкості розпізнавання в складних умовах освітлення, розширення набору жестів із урахуванням ергономіки оператора, оптимізацію обчислювальних витрат на етапі комп’ютерного зору, а також на інтеграцію системи з реальним маніпулятором Franka Emika Panda з урахуванням вимог промислової безпеки та стандартів функціональної безпечності.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Гайдук І. М. Система управління роботизованим маніпулятором на основі розпізнавання жестів руки / І. М. Гайдук // Автоматизація та приладобудування (Automation and Development of Electronic Devices) : збірник студентських наукових статей / Харківський нац. ун-т радіоелектроніки ; редкол.: І. Ш. Невлюдов та ін. – Харків : ХНУРЕ, 2025. – Вип. 2. – С. 53. – Електронний ресурс. – Режим доступу: https://openarchive.nure.ua/bitstreams/d6069c3a-7f6c-4532-8e1a-6095fff2b462/download (дата звернення 01.12.2025).

2. ДСТУ 3008-15. Документація. Звіти у сфері науки та техніки. структура та правила оформлення. Введ. 2015-06-22. К. Держстандарт України, 2017. – 29 с. (дата звернення 01.012.2025).

3. Методичні вказівки з підготовки та захисту кваліфікаційної роботи здобувачами другого (магістерського) рівня вищої освіти спеціальності 151 Автоматизація та комп'ютерно-інтегровані технології, освітньо-професійних програм: «Автоматизоване управління технологічними процесами», «Комп'ютерно-інтегровані технологічні процеси і виробництва», «Комп'ютеризовані та робототехнічні системи» / Упоряд. І. Ш. Невлюдов, Р. В. Артюх, В. В. Безкоровайний, Н. П. Демська, В. В. Євсєєв, О. І. Филипенко, О. М. Цимбал. – Харків: ХНУРЕ, 2021. – 55 с.

4. Craig J. Introduction to Robotics: Mechanics and Control. – 4th Edition. – Pearson, 2018. – 400 p.

5. Жарікова І. В. Дослідження механічних параметрів гнучких комутаційних структур для мобільних роботизованих платформ / І. В. Жарікова, Д. О. Нікітін // Виробництво & Мехатронні Системи 2024 : матеріали VIII-ої Міжнародної конференції, 25-26 жовтня 2024 р. – Харків, 2024. – С. 110-113.

6. Nevliudov I., Zharikova I., Bronnikov A. Improvement of the commutation system for a mobile robot platform using polyimide structures //The 4 th International

scientific and practical conference “Eurasian scientific discussions” (May 8-10, 2022) Barca Academy Publishing, Barcelona, Spain. 2022. 403 p. – 2022. – С. 157.

7. Невлюдов І. Ш., Андрусевич А. О., Євсєєв В. В., Новоселов С. П., Демська Н. П. Проектування мобільних маніпуляційних роботів: Монографія. – Х. :, 2022. – 427 с. (дата звернення: 02.10.2025).

8. KR AGILUS Technical Specification / KUKA Roboter GmbH. Augsburg : KUKA AG, 2023. URL: https://www.kuka.com/en-de/products/robot-systems/industrial-robots/kr-agilus (дата звернення: 02.10.2025).

9. M-20iA Series: Datasheet and Technical Manual / FANUC Corporation. Yamanashi: FANUC Corp., 2024. URL: https://www.fanuc.eu/at/en/robots/robot-filter-page/m-20-series/m-20ia (дата звернення: 02.10.2025).

10. IRB 1200 Product Specification / ABB Robotics. Västerås : ABB AB, 2023. URL: https://new.abb.com/products/robotics/industrial-robots/irb-1200 (дата звернення: 02.10.2025).

12. DX200 Controller: Multi-Robot Coordination Systems / Yaskawa Motoman. Technical White Paper Series. 2023. No. 18.

12. Robotic Surgery and Rehabilitation Systems: Clinical Applications and Outcomes / Intuitive Surgical Inc., Ekso Bionics Holdings // Medical Robotics Review. 2023. Vol. 15, No. 3. P. 112–128. DOI: 10.1016/j.medrob.2023.03.008.

13. Open-Source Robotic Platforms for Research and Education / Franka Emika GmbH, Universal Robots A/S // Journal of Robotics in Academia. 2024. Vol. 9, No. 2. P. 45–67. URL: https://www.franka.de/research (дата звернення: 02.10.2025).

14. Rautaray S. S., Agrawal A. Vision-Based Hand Gesture Recognition for Human-Computer Interaction: A Survey // Artificial Intelligence Review. 2015. Vol. 43, No. 1. P. 1–54. DOI: 10.1007/s10462-012-9356-9.

15. Weichert F., Bachmann D., Rudak B., Fisseler D. Analysis of the Accuracy and Robustness of the Leap Motion Controller // Sensors. 2013. Vol. 13, No. 5. P. 6380–6393. DOI: 10.3390/s130506380.

16. Gupta P., Goel S. Vision-Based Hand Gesture Recognition for Human Computer Interaction: A Survey // International Journal of Computer Applications. 2020. Vol. 975, No. 8887. P. 1–6.

17. MediaPipe Hands: On-device Real-time Hand Tracking / F. Zhang, V. Bazarevsky, A. Vakunov [et al.] // Google AI Blog. 2020. URL: https://developers.google.com/mediapipe/solutions/vision/hand_landmarker (дата звернення: 05.10.2025).

18. Mittal A., Zisserman A., Torr P. H. S. Hand Detection Using Multiple Proposals // Proceedings of the British Machine Vision Conference. 2011. P. 75.1–75.11. DOI: 10.5244/C.25.75.

19. Bradski G., Kaehler A. Learning OpenCV: Computer Vision with the OpenCV Library. 2nd Edition. O'Reilly Media, 2015. 576 с.

20. Szeliski R. Computer Vision: Algorithms and Applications. Springer, 2010. 812 с.

21. Lugaresi C., Tang J., Nash H., McClanahan C., Ubale E., Block D., Hays J. MediaPipe: A Framework for Building Perception Pipelines // arXiv preprint arXiv:1906.08172. 2019. URL: https://arxiv.org/abs/1906.08172 (дата звернення: 12.10.2025).

22. MediaPipe Hands: On-device Real-time Hand Tracking / F. Zhang, V. Bazarevsky, A. Vakunov [et al.] // Google AI Blog. 2020. URL: https://ai.google.dev/mediapipe/solutions/vision/hand_landmarker (дата звернення: 12.10.2025).

23. Goodfellow I., Bengio Y., Courville A. Deep Learning. MIT Press, 2016. 800 с.

24. Intel RealSense SDK Documentation / Intel Corporation. URL: https://github.com/IntelRealSense/librealsense (дата звернення: 12.10.2025).

25. Paszke A., Gross S., Massa F., Lerer A., Bradbury J., Chanan G., Killeen T., Lin Z., Gimelshein N., Antiga L., Desmaison A., Kopf A., Yang E., DeVito Z., Raison M., Tejani A., Chilamkurthy S., Steiner B., Fang L., Bai J., Chintala S. PyTorch: An Imperative Style, High-Performance Deep Learning Library // Advances in Neural

Information Processing Systems 32 (NeurIPS 2019). 2019. P. 8024-8035. URL: https://pytorch.org (дата звернення: 12.10.2025).

26. Coumans E., Bai Y. PyBullet, a Python Module for Physics Simulation for Games, Robotics and Machine Learning. 2016-2021. URL: https://pybullet.org (дата звернення: 15.10.2025).

27. Todorov E., Erez T., Tassa Y. MuJoCo: A Physics Engine for Model-Based Control // IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2012. P. 5026-5033. DOI: 10.1109/IROS.2012.6386109.

28. Koenig N., Howard A. Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator // IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2004. Vol. 3. P. 2149-2154. DOI: 10.1109/IROS.2004.1389727.

29. Gazebo: Robot Simulation Made Easy / Open Source Robotics Foundation. URL: https://gazebosim.org (дата звернення: 15.10.2025).

30. Rohmer E., Singh S. P. N., Freese M. V-REP: A Versatile and Scalable Robot Simulation Framework // IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2013. P. 1321-1326. DOI: 10.1109/IROS.2013.6696520.

31. CoppeliaSim User Manual / Coppelia Robotics AG. URL: https://www.coppeliarobotics.com (дата звернення: 15.10.2025).

32. Forward Kinematics. ASU Robotics Course Materials. URL: https://wanxinjin.github.io/asu-robotics/lec6-8/fk.html (дата звернення: 15.10.2025).

33. Craig J. J. Introduction to Robotics: Mechanics and Control. 3rd ed. Pearson Prentice Hall, 2005. 400 p. URL: https://marsuniversity.github.io/ece387/Introduction-to-Robotics-Craig.pdf (дата звернення: 15.10.2025).

34. Forward Kinematics. Robo Cookbook / Czech Institute of Informatics, Robotics and Cybernetics. URL: https://gitlab.ciirc.cvut.cz/holesond/robo-cookbook/-/blob/main/notes/cookbook/site/FK.md (дата звернення: 15.10.2025).

35. Sciavicco L., Siciliano B. Robotics: Modelling, Planning and Control. Chapter 11. Sapienza University of Rome, 2008. URL: https://

www.diag.uniroma1.it/~labrob/pub/papers/SHOR08_Chp11.pdf (дата звернення: 15.10.2025).

36. Inverse Kinematics Methods for Robotic Manipulators. Memorias del Congreso Nacional de Control Automático AMCA 2017. P. 1–6. URL: https://amca.mx/memorias/amca2017/media/files/0123.pdf (дата звернення: 15.10.2025).

37. Chiaverini S., Siciliano B., Egeland O. Review of the Damped Least-Squares Inverse Kinematics with Experiments on an Industrial Robot Manipulator. IEEE Transactions on Control Systems Technology. 1994. Vol. 2, No. 2. P. 123–134. URL: https://www.researchgate.net/publication/3331775 (дата звернення: 15.10.2025).

38. Buss S. R. Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares Methods: Technical Report. University of California, San Diego, 2009. 19 p. URL: https://www.cs.cmu.edu/~15464-s13/lectures/lecture6/iksurvey.pdf (дата звернення: 15.10.2025).

39. Singh G. et al. Comparative Study of Iterative Inverse Kinematics Methods for Serial Manipulators. International Journal of Engineering Research & Technology (IJERT). 2018. Vol. 7, Iss. 5. URL: https://www.ijert.org (дата звернення: 15.10.2025).

40. Coumans E. PyBullet Quickstart Guide. Bullet Physics SDK. URL: https://github.com/bulletphysics/bullet3/blob/master/docs/pybullet_quickstartguide.pdf (дата звернення: 15.10.2025).

41. Gil-Martín M., Marini M. R., Martín-Fernández I., Esteban-Romero S., Cinque L. Hand Gesture Recognition Using MediaPipe Landmarks and Deep Learning Networks. Proceedings of the 17th International Conference on Agents and Artificial Intelligence (ICAART 2025). 2025. Vol. 3. P. 24–30. URL: https://www.scitepress.org/Papers/2025/130535/130535.pdf (дата звернення: 15.10.2025).

42. Geng W. et al. Gesture recognition by instantaneous surface EMG images. Scientific Reports. 2016. Vol. 6. Article 36571. URL: https://www.nature.com/articles/srep36571 (дата звернення: 15.10.2025).

43. Методичні вказівки до лабораторних робіт з дисципліни «Основи охорони праці» для студентів усіх напрямів та форм навчання / Упоряд.: Т. Є. Стиценко, В. А. Айвазов, О. В. Мамонтов. – Харків: ХНУРЕ, 2018. – 120 с. (дата звернення 07.12.2025).