Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

РОЗРОБКА ТА ДОСЛІДЖЕННЯ СИСТЕМИ РОЗПІЗНАВАННЯ ТА ІНТЕРПРЕТАЦІЇ ЖЕСТІВ НА ОСНОВІ ФРЕЙМВОРКУ MEDIAPIPE

DEVELOPMENT AND RESEARCH OF A GESTURE RECOGNITION AND INTERPRETATION SYSTEM BASED ON THE MEDIAPIPE FRAMEWORK

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗм-21
Панько М. І.
Керівник:
к.т.н., доцент
Ящук А. А.

Кваліфікаційну роботу
допущено до захисту
«___» _____ 20__ р.
Гарант освітньої програми:
к.т.н., доцент Суринович О. М.
_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *магістр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«___» _____ 202__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

ДРУГОГО (МАГІСТЕРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Паньку Максиму Івановичу

1. Тема кваліфікаційної роботи: Розробка та дослідження системи розпізнавання та інтерпретації жестів на основі фреймворку MediaPipe
Керівник роботи: Ящук Андрій Анатолійович, доцент, к.т.н.
_____
затверджені наказом закладу вищої освіти від «29» березня__2025 року № 190/01-02_____
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи: 4 грудня 2025 р._____
3. Вихідні дані до роботи *технічне та програмне забезпечення ЕОМ*_____
4. Зміст розрахунково-пояснювальної записки: *аналіз проблематики створення системи розпізнавання та інтерпретації жестів на основі фреймворку Media Pipe, обґрунтування технологій та реалізацію мобільньного додатку, експериментальне дослідження результативності програмного забезпечення*
_____
5. Перелік графічного матеріалу 12 рисунків, 5 таблиць, 7 лістинги коду_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз проблеми за темою роботи та постановка завдань дослідження	*Ящук А. А.*		
Теоретичне дослідження та практична реалізація	*Ящук А. А.*		
Експериментальне дослідження системи	*Ящук А. А.*		
Нормоконтроль	*Повстяна Ю. С.*		
Гарант ОП	*Андрущак І. Є.*		
Показник запозичень тексту	___%		
Академічна доброчесність	*Ящук А. А.*		

7. Дата видачі завдання *«02 квітня 2025 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи магістра	Строк виконання етапів роботи	Примітка
1	Провести огляд літературних джерел по темі кваліфікаційної роботи	12.05.2025	
2	Провести аналіз загальної проблеми і вибір напрямків дослідження	24.09.2025	
3	Розробити функціональну схему роботи програмного продукту	05.11.2025	
4	Описати засоби розробки об’єкта проектування	16.11.2025	
5	Практична реалізація об’єкта проектування	29.11.2025	
6	Розробити методику для проведення експерименту	01.11.2025	
7	Провести аналіз результатів експерименту	18.11.2025	
8	Здача чистового варіанту кваліфікаційної роботи на кафедру	04.12.2025	

Здобувач вищої освіти _____ Панько М. І.

Керівник кваліфікаційної роботи _____ Ящук А. А.

АНОТАЦІЯ

Панько М. І. Розробка та дослідження системи розпізнавання та інтерпретації жестів на основі фреймворку MediaPipe. Рукопис.

Кваліфікаційна робота магістра ОП «Інженерія програмного забезпечення» спеціальності 121 «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота магістра складається з вступу, 3 розділів, висновків і пропозицій, списку використаних джерел, додатків.

У роботі досліджено основи жестових інтерфейсів та методів розпізнавання для HCI-систем. Сформовано завдання розробки високошвидкісного та точного рішення для мобільної AR-взаємодії. Реалізовано гібридний конвеєр: трекінг ключових точок кисті, нормалізація, фільтрація даних та побудова дескрипторів. Застосовано класифікатор на основі машинного навчання для розпізнавання жестів та механізм їхньої інтерпретації у команди керування віртуальними об'єктами. Проаналізовано точність роботі фреймоврку MediaPipe при складних умовах освітлення та хроматичному фоні. Інтегровано модуль нейронної корекції зображення для підвищення стійкості до освітлення. Виміряно точність та обчислювальну ефективність конвеєра на мобільних пристроях.

Ключові слова: жестовий інтерфейс, комп'ютерний зір, глибоке навчання, MediaPipe, SVM, доповнена реальність (AR), Zero-DCE, розпізнавання жестів.

ABSTRACT

Panko M. I. Development and research of a gesture recognition and interpretation system based on the MediaPipe framework. Manuscript.

Master's qualification work OP «Software Engineering» specialty 121 «Software Engineering». Lutsk National Technical University. Lutsk, 2025.

Master's qualification work consists of an introduction, 3 sections, conclusions and proposals, a list of used sources, appendices (according to the structure of the qualification work approved by the department).

The work investigates the basics of gesture interfaces and recognition methods for HCI systems. The task of developing a high-speed and accurate solution for mobile AR interaction is formulated. A hybrid pipeline is implemented: tracking key points of the hand, normalization, data filtering and construction of descriptors. A classifier based on machine learning for gesture recognition and a mechanism for their interpretation in virtual object control commands is applied. The accuracy of the MediaPipe framework under complex lighting conditions and a chromatic background was analyzed. A neural image correction module was integrated to increase the stability to lighting. The accuracy and computational efficiency of the pipeline on mobile devices were measured.

Keywords: gesture interface, computer vision, deep learning, MediaPipe, SVM, augmented reality (AR), Zero-DCE, gesture recognition.

ЗМІСТ

ВСТУП

Актуальність теми. Сфера людино-комп'ютерної взаємодії стрімко розвивається у напрямі підвищення природності й інтуїтивності спілкування користувача з цифровими системами. У цьому контексті системи розпізнавання жестів відіграють дедалі важливішу роль, оскільки забезпечують безконтактну та зручну взаємодію, що є особливо цінним для хірургічних систем, високотехнологічного обладнання, а також VR/AR-платформ. Сучасні мобільні AR-додатки потребують високоточних алгоритмів розпізнавання жестів, низької затримки та стабільної роботи в умовах обмежених апаратних ресурсів. Попри значні успіхи глибокого навчання, більшість існуючих рішень залишаються вразливими до змін освітлення, складного фону та часто вимагають потужних графічних прискорювачів, що ускладнює їх широке практичне застосування. Тому створення легковагового гібридного конвеєра для розпізнавання жестів, який поєднує точність сучасних моделей трекінгу з ефективністю класичних методів класифікації, робить тему дослідження актуальною.

Метою даної кваліфікаційної роботи є проєктування та розробка високопродуктивної системи розпізнавання жестів руки на основі методів комп'ютерного зору та глибинних моделей, здатної працювати в режимі реального часу у мобільному AR-середовищі та забезпечувати високу точність разом зі стійкістю до зовнішніх впливів.

Об'єктом дослідження виступає процес безконтактної взаємодії користувача з цифровими інтерфейсами за допомогою жестів руки в AR-середовищах.

Предметом дослідження є методи та алгоритми детекції руки, вилучення геометричних характеристик і класифікації жестів за відеопотоком, зокрема гібридні підходи, що інтегрують глибинні нейронні моделі та традиційні методи машинного навчання.

Завдання, які необхідно виконати:

– провести аналіз існуючих технологічних підходів до розпізнавання жестів, включаючи класичні та нейромережеві моделі;

– дослідити методи попередньої обробки відеоданих та алгоритми нейронної корекції для підвищення стійкості до освітлення;

– розробити та обґрунтувати архітектуру гібридного конвеєра для детекції руки, вилучення ознак і класифікації жестів;

– реалізувати програмний модуль розпізнавання статичних і динамічних жестів та перетворення їх у керувальні команди для AR-об'єктів;

– провести експериментальне дослідження ключових параметрів: точності, продуктивності та затримки системи на мобільних пристроях.

Здійснити порівняльний аналіз результатів та сформувати рекомендації щодо подальшої оптимізації системи. Наукова новизна роботи полягає у вирішенні фундаментальної технічної проблеми забезпечення високої точності розпізнавання жестів в умовах неконтрольованого освітлення та обмежених обчислювальних ресурсів мобільних пристроїв.Наукова новизна роботи полягає у вирішенні фундаментальної технічної проблеми забезпечення високої точності розпізнавання жестів в умовах неконтрольованого освітлення та обмежених обчислювальних ресурсів мобільних пристроїв.

Практична цінність роботи полягає у створенні закінченої, високооптимізованої інформаційної системи, що функціонує як ядро безконтактного введення для нативних додатків доповненої реальності.

Апробація результатів дослідження. Основні положення і результати кваліфікаційної роботи магістра були висвітлені на V міжнародній науковій конференції «Науковий простір: актуальні питання, досягнення та інновації» [1]. Також опубліковано наукову статтю у науковому збірнику «Технології та суспільство: взаємодія, вплив, трансформація» (Додаток А) [2].

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМАТИКИ ЗА ТЕМОЮ РОБОТИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Огляд і аналіз предметної області проблеми, результатів існуючих теоретичних та експериментальних досліджень

Жестова взаємодія є формою комунікації між людиною та комп'ютером, що базується на використанні рухів рук, пальців або всього тіла для передачі команд та інформації [3]. Вона належить до категорії безконтактних інтерфейсів, що дозволяють користувачу керувати системою без традиційних пристроїв введення, таких як клавіатура чи миша. Такі системи особливо корисні у середовищах, де прямий контакт небажаний або обмежений, наприклад, у медичних маніпуляціях або робототехнічних процесах. Варто зауважити, що на сьогодні завдяки розвитку комп'ютерного зору та сенсорних технологій жестові інтерфейси стали одним із ключових компонентів сучасних Human-Computer Interaction (HCI) систем.

У структурі жестової взаємодії виділяють кілька категорій жестів. Емблеми – умовні жести з чітким культурно визначеним значенням (наприклад, «окей» або піднятий великий палець). Ілюстратори – жести, які супроводжують мовлення та уточнюють його зміст, що важливо у мультимодальних сценаріях взаємодії. Маніпулятивні жести спрямовані на керування об'єктами, реальними чи віртуальними, що широко застосовується у VR/AR-середовищах. Навігаційні жести використовуються для переміщення у просторах інтерфейсу або зміни стану системи (гортання, масштабування, вибір елементів) [4]. На рисунку 1.1 зображені найбільш використовувані окремі жести по категоріях.

Рисунок 1.1 – Приклади жестів по категоріях

Окрім цього, жести можна поділити на природні, що виникають у повсякденній комунікації, та ті, що вимагають навчання, типово визначені конкретною системою. Природні жести знижують когнітивне навантаження, оскільки користувачеві не потрібно запам'ятовувати спеціальні команди, однак їх складніше формалізувати і точніше розпізнавати. У той час як жести, що вимагають навчання, легше алгоритмічно описати, але збільшують поріг входження для користувача [5].

Жестова взаємодія посідає важливе місце у структурі мультимодальних інтерфейсів, які поєднують декілька каналів взаємодії голос, дотик, погляд, просторові жести тощо. Мультимодальні системи забезпечують більш природну та адаптивну взаємодію, оскільки користувач може обирати найзручніший спосіб введення відповідно до ситуації [6]. У таких системах жести можуть компенсувати обмеження голосового керування (шумні середовища), а голос доповнювати жести у випадках складних просторових маніпуляцій. Технології відстеження погляду й тактильний зворотний зв'язок додатково підсилюють точність і виразність взаємодії, формуючи синергетичний ефект.

Зростання інтересу до жестових інтерфейсів у сучасних HCI-системах пояснюється кількома технологічними та соціальними факторами. По-перше, спостерігається загальний тренд на максимальну природність взаємодії, коли комп'ютерні системи адаптуються до звичних людині форм поведінки. По-друге, у низці застосувань відбувається перехід від механічних пристроїв введення (кнопок, маніпуляторів, контролерів) до безконтактних рішень, що підвищує зручність та гігієнічність. По-третє, швидкий прогрес у сфері алгоритмів комп'ютерного зору, особливо методів на основі згорткових нейронних мереж та моделей реального часу, забезпечив можливість точного розпізнавання жестів на звичайних камерах без спеціального обладнання.

Таким чином, жестова взаємодія виступає важливою складовою мультимодальних HCI-систем, оскільки поєднує природність, універсальність та високу інформативність невербальної комунікації, водночас відкриваючи можливості для створення більш інтуїтивних, доступних і контекстно адаптивних інтерфейсів.

Практичне застосування жестових інтерфейсів охоплює низку сучасних технологічних сфер, де безконтактна взаємодія забезпечує природність, точність та безпеку роботи користувача. Однією з таких сфер є технології віртуальної та доповненої реальності (VR/AR). У VR/AR-системах жести дозволяють користувачу керувати цифровими об'єктами без контролерів, виконуючи дії, максимально наближені до реальних рухів рук і пальців. Це підвищує рівень занурення, знижує бар'єр входження та створює інтуїтивну поведінкову модель взаємодії з віртуальним середовищем [7].

Рисунок 1.2 – Демонстрація жестової взаємодії у VR-системі: користувач керує цифровими об'єктами за допомогою рухів рук [8]

У робототехніці жестові системи використовуються для дистанційного керування роботами або маніпуляторами. Використання безконтактних команд підвищує безпеку операцій у середовищах з потенційно небезпечними умовами (виробництво, обслуговування важкої техніки). Жестові інтерфейси також дозволяють оператору виконувати складні координаційні дії, зберігаючи свободу руху та оперативність реакції [9].

У медицині безконтактні жести застосовують під час хірургічних та лабораторних процедур. Такі системи дозволяють лікарю або досліднику керувати медичним обладнанням, переглядати зображення чи змінювати налаштування без торкання поверхонь, що забезпечує стерильність і мінімізує ризик контамінації [10].

У розумних системах («smart home» та IoT) жести використовуються для керування освітленням, мультимедійними пристроями або кліматичними системами. Це робить взаємодію більш природною та зручною, особливо для користувачів із обмеженими фізичними можливостями або у випадках, коли голосові команди недоступні (шумні середовища) [11]. У деяких розумних системах прості жести, наприклад, хлопок у долоні, використовуються для включення або вимикання освітлення, що підвищує зручність і доступність взаємодії.

Рисунок 1.3 – Використання жесту «хлопок» для керування освітленням

І врешті, ігрова індустрія активно інтегрує жестові інтерфейси для підвищення ігрової інтерактивності. Жести дозволяють гравцеві виконувати дії без контролера, що зближує ігровий досвід із фізичною активністю, підвищує реалізм і занурення, а також відкриває нові можливості для розробки інноваційних механік управління.

Таким чином, жестові системи вже не обмежуються дослідницькими прототипами, а стають повноцінним інструментом у VR/AR, робототехніці, медицині, розумних системах та ігровій індустрії, забезпечуючи природну, безпечну та ефективну взаємодію користувача з технологіями.

Безконтактні жестові системи мають ряд суттєвих переваг. Вони забезпечують природність взаємодії, дозволяючи користувачу виконувати дії, аналогічні реальним рухам. Це підвищує інтуїтивність та зменшує когнітивне навантаження, особливо

при використанні природних жестів. Крім того, безконтактні інтерфейси підвищують гігієнічність та безпеку, оскільки користувач не контактує з поверхнями або пристроями. Жестові системи також добре інтегруються у мультимодальні HCI-системи, дозволяючи комбінувати голос, дотик та погляд, що підвищує гнучкість і адаптивність взаємодії.

Водночас такі системи мають певні обмеження. Їхня точність і стабільність залежить від умов освітлення та якості сенсорів, що може ускладнювати розпізнавання жестів у реальних середовищах. Жести, які потребують навчання, підвищують поріг входження для користувача та можуть створювати додаткове когнітивне навантаження. Також існують технічні обмеження, пов'язані з обчислювальною потужністю та швидкодією алгоритмів розпізнавання жестів, особливо в реальному часі.

Таким чином, аналіз переваг та обмежень безконтактних жестових інтерфейсів демонструє їхній потенціал і обмеження, закладаючи основу для подальшого розгляду класифікації жестів та методів їх розпізнавання, що розкривається у наступному підпункті.

1.2 Огляд і аналіз методів та засобів розробки систем розпізнавання жестів на основі фреймворку MediaPipe

У системах жестової взаємодії прийнято виділяти статичні та динамічні жести, оскільки їхня природа визначає особливості алгоритмів розпізнавання та вимоги до сенсорних систем. Статичні жести описують фіксоване положення руки або пальців у певний момент часу. Це можуть бути пози, що мають усталене семантичне значення: наприклад, «ok», «стоп», «великий палець догори». Для їх розпізнавання зазвичай достатньо одноразового аналізу кадру або набору ключових точок, що робить такі жести менш вимогливими до швидкодії та обчислювальних ресурсів. Вони широко застосовуються у системах, де важлива чіткість і однозначність команд.

Динамічні жести, навпаки, визначаються зміною положення руки у часі – траєкторією руху, швидкістю та напрямом. До них належать змахування, гортання,

замахування, а також складні траєкторні рухи, характерні для VR/AR або робототехнічних систем. Алгоритми роботи з динамічними жестами повинні обробляти послідовності кадрів і враховувати часову структуру жесту, що зумовлює застосування моделей відстеження руху та аналізу кінематики. Такі жести є більш природними для користувача, однак значно складніші для точного та стабільного розпізнавання, особливо за наявності шумів або перешкод у відеопотоці.

У цілому розмежування статичних і динамічних жестів є базовим для побудови жестових інтерфейсів, оскільки визначає архітектуру розпізнавання, вибір математичних моделей та вимоги до сенсорного обладнання. Це також формує основу для подальшого аналізу методів оцінки поз, кінематичних моделей і сучасних нейронних підходів, що будуть розглянуті в наступних підпунктах.

Проте поділ жестів на статичні та динамічні є лише першим рівнем класифікації, що визначає загальну структуру обробки сигналів. Для точного й надійного розпізнавання сучасні системи жестової взаємодії потребують значно глибшого аналізу, зокрема, виявлення ключових точок руки, побудови кінематичних моделей та відстеження траєкторій руху у просторі. Саме такі методи дозволяють інтерфейсам працювати стабільно навіть за складних умов, забезпечуючи високий рівень деталізації та контекстної інтерпретації жестів [12].

Оцінка поз (pose estimation) полягає у виділенні ключових точок кисті або всієї руки, суглобів, фаланг і орієнтаційних векторів на основі зображення або відеопотоку. Сучасні методи використовують або традиційні алгоритми комп'ютерного зору із заздалегідь визначеними ознаками, або глибинні нейронні мережі, які здатні виявляти складні структури навіть за часткового перекриття або нестабільного освітлення. Побудова кінематичного ланцюга дозволяє моделі враховувати природні обмеження руху людської кисті, що підвищує точність інтерпретації жестів і зменшує кількість хибних класифікацій.

Для динамічних жестів важливим є не лише положення руки в окремий момент часу, а й форма її руху у просторі. Траєкторні моделі аналізують серії кадрів і визначають просторово-часові залежності, що дозволяє розпізнавати жести типу «змах», «гортання», «круговий рух» чи інші складні патерни. Традиційно такі задачі вирішувалися методами гаусівських моделей, HMM або правил на основі

геометричних ознак, проте сучасні системи дедалі частіше застосовують рекурентні та просторово-часові нейронні мережі, які забезпечують значно вищу стійкість і точність.

Візуальне відображення основних етапів процесу підкреслює, що на кожному з них можуть застосовуватися різні підходи до аналізу зображень та рухів. Саме тому наступним кроком є порівняння традиційних та сучасних методів, які лежать в основі класифікації жестів і відрізняються за складністю, точністю та обчислювальною ефективністю.

На сьогодні у цій сфері співіснують два підходи: традиційні алгоритми комп'ютерного зору з ручною побудовою ознак та сучасні нейронні моделі, здатні автоматично виявляти складні патерни візуальних даних. Оцінка їхніх переваг і недоліків дозволяє обґрунтувати вибір інструментів для створення жестових систем різного рівня складності.

Класичні алгоритми комп'ютерного зору використовувалися у жестових системах задовго до появи глибинного навчання. До них належать методи сегментації за кольором шкіри, контурний аналіз, виявлення ключових точок із застосуванням SIFT, SURF або HOG, а також класифікація із використанням SVM чи k-NN. Основною перевагою цих методів є невисока обчислювальна складність та можливість роботи у реальному часі навіть на слабких пристроях. Однак їхня ефективність сильно залежить від умов зйомки освітлення, якості камери, тла а також потребує ручного підбору ознак, що обмежує масштабованість та універсальність систем [13].

Із розвитком глибинного навчання акцент у gesture recognition поступово змістився на згорткові нейронні мережі (CNN), рекурентні моделі (LSTM, GRU) та просторово-часові архітектури, такі як 3D-CNN або Transformers. Ці моделі здатні самостійно формувати ознаки, враховувати часову динаміку та забезпечувати високу точність навіть за складних умов зйомки. CNN дозволяють ефективно виявляти локальні патерни жестів, тоді як RNN і Transformer-моделі забезпечують інтерпретацію руху у часі, що критично для динамічних жестів. Серед недоліків нейронних методів значна потреба у великих наборах даних, висока обчислювальна

вартість та необхідність апаратного прискорення, проте їх переваги у точності та універсальності є визначальними у сучасних системах розпізнавання [14].

Таким чином, традиційні методи є доцільними для простих або ресурсно обмежених систем, тоді як нейронні підходи формують основу сучасних високоточних жестових інтерфейсів. Порівняння цих підходів відіграє важливу роль у виборі архітектури системи, що буде розглянуто у наступних підрозділах.

Описані вище класифікації жестів та підходи до їхнього розпізнавання формують цілісне уявлення про функціонування сучасних жестових інтерфейсів. Проте ці методи не виникли раптово: їх поява є результатом тривалої еволюції технологій людино-комп'ютерної взаємодії. Тому доцільно коротко розглянути історичний розвиток систем розпізнавання жестів, аби показати, як саме галузь прийшла до сучасних рішень, у тому числі до фреймворку MediaPipe.

Перші технології жестової взаємодії з'явилися ще у 1980–1990-х роках і базувалися переважно на сенсорних рукавичках. Такі пристрої містили згинальні датчики, акселерометри та магнітні трекери, що дозволяло з високою точністю відстежувати рухи пальців і кисті. Однак вони були дорогими, громіздкими та суттєво обмежували природність взаємодії, тому використовувалися здебільшого у наукових лабораторіях та авіаційних тренажерах [15].

У 2000-х роках розвиток цифрових камер і класичних алгоритмів комп'ютерного зору спричинив перехід до безконтактних систем, які аналізували 2D-зображення рук. У цей період домінували методи сегментації шкіри, аналізу контурів, геометричних моделей пальців і кисті. Хоча вони були технічно новаторськими, їхня робота залежала від стабільного освітлення і нейтрального фону, що істотно обмежувало надійність таких систем у реальних умовах.

Значний прорив відбувся після появи сенсорів глибини, зокрема Microsoft Kinect (2010), який забезпечував 3D-карту глибини у реальному часі. Це дозволило набагато точніше відстежувати положення рук і тіла та сприяло появі масових застосувань жестового контролю в ігровій індустрії, освіті й робототехніці [16].

Розглянуті етапи демонструють поступовий і технологічно зумовлений розвиток жестових систем від перших сенсорних рукавичок до появи сенсорів глибини, що забезпечили суттєвий приріст точності та стабільності розпізнавання

рухів. Щоб наочно узагальнити основні віхи цієї еволюції та простежити логіку переходу від апаратних рішень до сучасних безконтактних моделей, наведено хронологічну діаграму зображену на рисунку 1.4 яка демострує становлення жестових інтерфейсів.

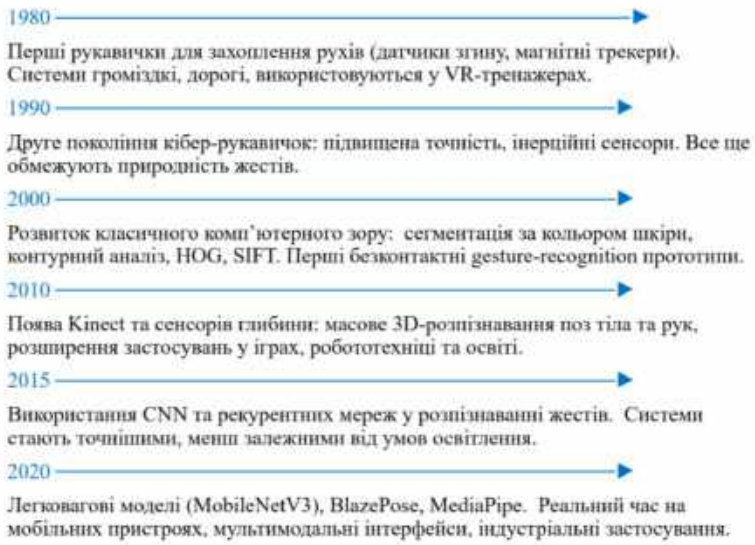


Рисунок 1.4 – Хронологічна діаграма розвитку жестових систем

Щодо сучасного етапу розвитку, то він розпочався у 2015–2020-х роках зі становленням глибоких нейронних мереж, здатних автоматично виявляти ключові точки («landmarks») та проводити оцінку пози без використання спеціального обладнання. Саме це привело до появи систем нового покоління, таких як MediaPipe, де використання оптимізованих CNN-моделей та графів обробки дає змогу виконувати розпізнавання жестів у реальному часі навіть на мобільних пристроях. Таким чином, еволюція технологій демонструє поступовий перехід від громіздких сенсорних пристроїв до високоточних безконтактних систем, що формують сучасний ландшафт мультимодальних інтерфейсів.

Хронологічний розвиток жестових систем демонструє, що ефективність розпізнавання значною мірою залежала не лише від апаратних засобів, а й від методів обробки візуальних даних. Кожний технологічний етап від сенсорних рукавичок до сучасних моделей на основі нейронних мереж визначався еволюцією алгоритмів комп'ютерного зору, що забезпечували виділення руки, визначення

ключових ознак та подальшу інтерпретацію рухів. Тому наступним важливим аспектом є аналіз методів машинного зору, які формують основу сучасних систем розпізнавання жестів та визначають їхню точність, стійкість та працездатність у реальному часі.

Ефективність роботи жестових інтерфейсів значною мірою залежить від алгоритмів машинного зору, які забезпечують коректне виділення області руки, аналіз її структури та подальшу класифікацію жестів. Незалежно від обраної моделі, класичної чи глибинної, обробка зображень зазвичай включає кілька фундаментальних етапів: фільтрацію, сегментацію, виділення ознак та власне розпізнавання. Саме ці методи створюють основу для сучасних систем, зокрема й для фреймворку MediaPipe, який поєднує оптимізовані алгоритми комп'ютерного зору з нейронними моделями реального часу.

Попередня обробка (preprocessing) покликана підвищити якість вхідних даних та зменшити вплив шумів, які можуть ускладнювати подальші етапи аналізу. Найбільш поширені методи включають згладжування (Gaussian Blur, Median Filter), нормалізацію освітленості та корекцію контрасту. Такі операції дозволяють мінімізувати вплив нерівномірного освітлення, зернистості зображення та артефактів сенсора. У жестових системах фільтрація має особливе значення, оскільки рухи рук часто виконуються у динамічних умовах, де зміна освітлення або фон можуть негативно впливати на точність подальшого розпізнавання.

Сегментація є критичним етапом, оскільки від коректності виділення області руки залежить стабільність подальшого аналізу. У класичних методах сегментація здійснювалась за допомогою порогових методів (thresholding), сегментації за кольором шкіри в HSV-просторі, або через аналіз контурів. Проте такі підходи є вразливими до змін освітленості, кольору фону та індивідуальних особливостей користувачів.

Із розвитком глибинних моделей сегментація почала виконуватися за допомогою нейронних мереж, які формують маску руки незалежно від зовнішніх умов. Прикладом є підхід MediaPipe Hands, де сегментація виконується вбудованою lightweight-моделлю, що визначає область руки навіть у складних сценах. Сучасні

методи глибокої сегментації демонструють значно вищу точність та стійкість порівняно з класичними алгоритмами [17].

Після того як рука виділена, необхідно описати її форму, положення або рух. Класичний комп'ютерний зір використовує ручне конструювання ознак. Найбільш поширеними підходами є: HOG (Histograms of Oriented Gradients) – описує локальні градієнти і добре підходить для визначення контурів руки; SIFT та SURF – інваріантні до повороту та масштабу ознаки, які використовувалися у ранніх системах розпізнавання рук у 2000-х роках. Optical Flow – аналіз руху між кадрами, що дозволяє виявляти динамічні жести, зокрема змахування або проведення рукою у просторі [18].

Хоча ці методи були домінантними протягом тривалого часу, вони вимагають ретельного налаштування і не завжди добре узагальнюють жести в умовах реального використання.

Після виділення ознак традиційні системи використовували алгоритми машинного навчання: SVM (Support Vector Machines) – ефективний для задач класифікації статичних жестів; KNN, який добре працює на невеликих наборах ознак; HMM (Hidden Markov Models) – один із ключових інструментів для аналізу динамічних жестів у часі; GMM (Gaussian Mixture Models) – застосовувалися для моделювання варіативності рухів.

Перевагами класичних моделей є висока швидкодія та невеликі обчислювальні витрати. Однак їх точність та здатність до узагальнення значно поступаються сучасним нейронним мережам.

Справжній прорив відбувся зі становленням глибокого навчання. CNN (Convolutional Neural Networks) дозволили автоматично виділяти ознаки зображення, усуваючи потребу у ручному feature engineering. Системи на основі CNN – наприклад, MobileNetV3 або моделі MediaPipe Hands – забезпечують високу точність у реальному часі навіть на мобільних пристроях.

Для динамічних жестів використовуються RNN (LSTM, GRU), здатні аналізувати тимчасові послідовності кадрів. Останніми роками ключову роль відіграють Transformers, що демонструють найкращі результати у багатьох задачах комп'ютерного зору, включаючи розпізнавання рухів у відео [19].

Таким чином, глибинні моделі значною мірою витіснили класичні підходи завдяки своїй точності, стійкості до умов зйомки та можливості працювати з великими наборами даних.

Для більшої наочності можна подати порівняння класичних і глибинних методів у вигляді таблиці 1.1.

Таблиця 1.1 – Порівняння класичних та глибинних методів комп'ютерного зору

Підхід	Переваги	Недоліки
Класичний CV (HOG, SVM, Optical Flow)	висока швидкодія, низькі ресурси, простота реалізації	низька стійкість до фону та освітлення, слабка узагальнюваність
Глибинні моделі (CNN, RNN, Transformers)	висока точність, автоматичне виділення ознак, працюють у складних умовах	потребують більше даних та обчислювальних ресурсів

Процес розпізнавання жестів зазвичай формулюється як задача класифікації, де модель має визначити, до якого класу належить жест. Найчастіше використовуються методи supervised learning, що вимагають великої кількості розмічених зображень або відео. Для покращення якості навчання застосовуються аугментації (обертання, зміна освітлення), а також оптимізація архітектури моделей. У сучасних системах усе частіше застосовують комбіновані архітектури, де оцінка пози (pose estimation) поєднується з класифікацією руху, що дозволяє досягати високої точності у сценаріях реального часу.

Проведений огляд методів машинного зору показує, що якість і надійність систем розпізнавання жестів суттєво залежать від вибору алгоритмів попередньої обробки, сегментації та виділення ознак. Класичні підходи, такі як HOG, SVM чи optical flow, забезпечують високу швидкодію та низькі обчислювальні витрати, однак демонструють недостатню стійкість до варіацій освітлення, тла та складних

поз руки. Натомість сучасні глибинні моделі – CNN, RNN і Transformers – здатні автоматично формувати інформативні ознаки та працювати в умовах реального часу, що робить їх основою більшості сучасних gesture-recognition систем.

Еволюція цих методів логічно підводить до необхідності створення інструментів, які поєднують високу точність, продуктивність і універсальність у застосуванні. Одним із таких рішень є фреймворк MediaPipe, який інтегрує сучасні алгоритми оцінки пози, детекції та трекінгу рук. Саме тому наступний підрозділ буде присвячений постановці задачі дипломної роботи та обґрунтуванню вибору MediaPipe як технологічної основи для розроблення та дослідження системи розпізнавання жестів.

1.3 Постановка завдання на кваліфікаційну роботу магістра

На основі проведеного огляду стану предметної області, аналізу сучасних методів комп'ютерного зору та технологій розпізнавання жестів, а також виявлених тенденцій переходу від традиційних сенсорних систем до безконтактних мультимодальних інтерфейсів, було визначено наступні завдання на кваліфікаційну роботу магістра:

– дослідити методи фільтрації, нормалізації та виділення ознак, що застосовуються в задачах виявлення руки, а також алгоритми нейронної стабілізації освітлення (Zero-DCE) для забезпечення стійкості в умовах низької освітленості;

– розробити та обґрунтувати архітектуру гібридного конвеєра для детекції руки, що поєднує трекінг MediaPipe з подальшою класифікацією на основі Support Vector Machine (SVM), включаючи етапи побудови дескриптора та схему one-vs-one voting;

– реалізувати програмне забезпечення у середовищі Unity для розпізнавання статичних і динамічних жестів у режимі реального часу та забезпечити їхню інтерпретацію у керувальні команди для маніпуляції віртуальними AR-об'єктами;

– провести експериментальне дослідження системи на мобільних пристроях за показниками точності F1-Score, Confusion Matrix, швидкодії (FPS) та загальної затримки (Latency), розподіляючи час обробки по ключових модулях конвеєра.

Отже, чітко визначені мета та завдання окреслюють подальший напрям дослідження й забезпечують узгодженість теоретичної та практичної частин роботи. Наступний розділ присвячено безпосередній розробці системи та реалізації методів розпізнавання жестів.

РОЗДІЛ 2
ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ РОЗПІЗНАВАННЯ ТА ІНТЕРПРЕТАЦІЇ ЖЕСТІВ

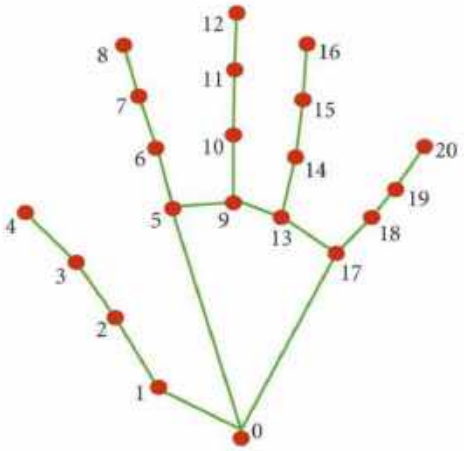
2.1 Обґрунтування вибору шляхів, технологій, алгоритмів і засобів вирішення поставленого завдання

Реалізація системи розпізнавання жестів передбачає поєднання кількох технологічних напрямів: комп'ютерного зору, обробки скелетних даних, машинного навчання та інтерактивної 3D-візуалізації. У межах поставленого завдання особливо важливим є правильний вибір інструментів, адже саме він безпосередньо впливає на точність розпізнавання, стабільність роботи в реальному часі, обчислювальну ефективність системи та можливість її масштабування. Тому вибір технологій здійснювався не лише з позиції їх популярності чи доступності, а на основі порівняльного аналізу альтернатив, їхніх функціональних можливостей і відповідності вимогам проєкту, що передбачає роботу з мобільними пристроями, динамічними 3D-сценами й необхідністю інтеграції жестового інтерфейсу з AR-середовищем.

Для розпізвання положення руки у реальному часі існує кілька підходів: використання глибинних камер Microsoft Kinect [20], Intel RealSense [21], класичних бібліотек комп'ютерного зору OpenCV [22], моделей на базі згорткових нейронних мереж наприклад, OpenPose [23] або спеціалізованих фреймворків типу MediaPipe Hands [24]. У ході аналізу було встановлено, що технології з апаратним сенсорним забезпеченням, попри високу точність, не підходять для мобільних AR-сценаріїв через необхідність додаткового обладнання і неможливість інтеграції з камерою смартфона. OpenPose та подібні моделі на базі CNN демонструють хороший рівень точності, але значно поступаються MediaPipe у швидкодії, що є критичним при обробці даних із частотою понад 30 FPS (frames per second) на мобільних пристроях.

MediaPipe Hands вирізняється оптимізованою архітектурою, що складається з моделі Palm Detector для пошуку руки та моделі Hand Landmark Model, яка поверх первинної детекції реконструює скелет. Поєднання цих модулів забезпечує стійкість до варіативнсті освітлення, ракурсів і часткових перекриттів пальців. Фреймворк

працює без необхідності використання даттчика глибинни (LiDAR) та використовує лише RGB-камеру, що відповідає вимогам мобільної AR-системи. Крім того, MediaPipe на виході формує інтуїтивно зрозумілий набір з 21 3D-landmark'a, зображений на рисунку 2.1, що є оптимальним набором для подальшої класифікації жестів.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Рисунок 2.1 – 21 координата скелета руки [25]

У результаті порівняння MediaPipe було вибрано через його високу продуктивність на мобільних пристроях, відкриту інтеграцію з Unity та стабільність у складних сценаріях з немонотонними рухами рук.

Після отримання просторових координат landmarks постає завдання класифікації жестів. Основними альтернативами були глибокі нейронні мережі LSTM, CNN-класифікатори, дерева рішень, KNN та метод опорних векторів (SVM). Нейронні мережі мають високу гнучкість, але вимагають великих обсягів даних і значних ресурсів для інференсу, що ускладнює застосування в мобільних AR-системах. Прості алгоритми на кшталт KNN показують хороші результати для

статичних задач, але не забезпечують стійкість при значних варіаціях жестів та зашумлених даних. Дерева рішень демонструють надмірну чутливість до флуктуацій координат та схильні до перенавчання.

Метод опорних векторів характеризується високою роздільною здатністю на малих вибірках, стійкістю до шуму та здатністю формувати чіткі гіперплощини між класами. Це особливо важливо для статичних жестів, які мають компактні розподіли у просторах ознак. Додатковим аргументом на користь SVM стала можливість застосування схеми one-vs-one, що забезпечує масштабованість моделі під більшу кількість жестів без зміни алгоритмічної основи. SVM дозволяє навчати кожну пару жестів окремо, внаслідок чого покращується якість розмежування класів і знижується ризик конфліктів між жестами зі схожим положенням пальців.

Таким чином, SVM був обраний як оптимальний компроміс між точністю, швидкістю роботи в рантаймі та потребою у відносно невеликій навчальній вибірці.

Для подальшої обробки та підготовки даних для навчання класифікаційної моделі було обрано середовище PyCharm [26], оскільки воно забезпечує оптимальний баланс між функціональністю, зручністю роботи з Python та інтеграцією машинного навчання. Було проаналізовано декілька альтернатив: Jupyter Notebook [27], Google Colab [28], Visual Studio Code [29]. Jupyter є зручним для експериментів, але менш ефективний для підтримки великих проєктів зі складною файловою структурою. Google Colab залежить від хмарної інфраструктури, що ускладнює повторюваність експериментів та автономне налаштування середовища. VS Code є гнучким редактором, але потребує додаткових конфігурацій для інтеграції з бібліотеками ML.

PyCharm забезпечує статичний аналіз коду, розвинену систему віртуальних середовищ, зручну роботу з TensorFlow та повну локальну ізоляцію навчального процесу. Такий підхід гарантує відтворюваність результатів та стабільність під час експериментів.

Для реалізації інтерактивної частини проєкту та демонстрації результатів розпізнавання жестів було порівняно кілька сучасних ігрових рушіїв: Unreal Engine [30], Godot [31] та Unity [32]. Unreal Engine вирізняється високою графічною якістю, але має значно вищі вимоги до апаратних ресурсів, складніший процес

розгортання на мобільних пристроях та менш інтегрований стек інструментів для AR-розробки. Godot є перспективним рушієм з відкритим кодом, однак його інструменти для AR-розробки не мають повноцінної підтримки Android-AR екосистеми.

Unity забезпечує найкращий баланс між продуктивністю, доступністю, кількістю платформ, готовими інструментами візуалізації та інтеграціями зі сторонніми бібліотеками. Рушій підтримує модульність архітектури та легко взаємодіє з MediaPipe через C#-обгортки, що дозволяє реалізувати обробку скелетних даних безпосередньо у рантаймі. Крім того, Unity має найбільш розвинений стек технологій для AR-розробки, що включає AR Foundation – універсальний API для роботи з ARCore і ARKit. Отже, вибір Unity як рушія для демонстрації зумовлений його гнучкістю, широкою підтримкою мобільних платформ та розвиненим інструментарієм для інтеграції комп'ютерного зору.

Для демонстрації роботи системи розпізнавання жестів найкраще підходять середовища доповненої реальності, оскільки вони забезпечують наочну демонстрацію взаємодії. Реалізації AR потребує фреймворку, який забезпечує стабільне відстеження просторового положення камери, побудову сітки оточення та коректну роботу з глибинною інформацією. У процесі аналізу було розглянуто ARKit [33], Vuforia [34], OpenXR [35] та ARCore [36]. ARKit демонструє високу точність, однак доступний виключно на пристроях Apple, що обмежує можливість тестування системи та суперечить вимозі мобільності. Vuforia має потужні інструменти трекінгу маркерів, але орієнтована передусім на розпізнавання зображень, а не роботу зі складними жестовими інтерфейсами. OpenXR забезпечує кросплатформеність, проте потребує додаткового рівня інтеграції, оскільки не реалізує конкретні алгоритми трекінгу, а лише визначає інтерфейси.

ARCore забезпечує повний цикл функцій для мобільного AR-додатка: трекінг камери, оцінку відстані до об'єктів, побудову глибинної карти та стабільну роботу з просторовими проєкціями. Важливою перевагою є повна сумісність з Unity через AR Foundation, що мінімізує затрати на імплементацію.

2.2 Практична реалізація об'єкта проектування

Проєктування є одним із ключових етапів у процесі розробки будь-якого програмного забезпечення. На цьому етапі визначаються основні архітектурні підходи, від яких залежить подальша гнучкість, масштабованість і підтримуваність системи.

Під час проєктування системи було визначено загальну архітектуру обробки жестів, яка забезпечує цілісність програмного процесу, стійкість до змін та відповідність принципам SOLID [37]. SOLID – це сукупність п'яти базових принципів проєктування об'єктноорієнтованого програмного забезпечення, спрямованих на підвищення гнучкості, модульності та розширюваності системи. Ці принципи визначають, що кожен компонент повинен мати лише одну відповідальність, бути відкритим для розширення, але закритим для модифікації, коректно взаємодіяти через абстракції, уникати надмірних залежностей та забезпечувати можливість підміни реалізацій без порушення логіки роботи програми. Застосування SOLID дозволяє створювати архітектури, у яких зміни в одних модулях мінімально впливають на інші, що особливо важливо для складних систем.

Загальний підхід передбачає створення послідовного конвеєра, який бере початок з даних на моменті їх отримання з камери до формування високорівневих команд, які використовуються механізмами взаємодії в межах Unity. На початковому етапі було сформовано концептуальну схему, у якій кожен модуль системи виконує єдину відповідальність та взаємодіє з іншими компонентами через чітко визначені інтерфейси, що надає можливість розширення функціональності без порушення існуючої логіки.

Узагальнений потік даних починається з модуля інтеграції з MediaPipe Hands, який отримує зображення з камери та генерує набір тривимірних координат ключових точок кисті. Ці дані передаються в модуль попередньої обробки, де виконуються нормалізація, центрування та фільтрація координат. Цей етап приводить дані до стандартизованого формату, що дозволяє зменшити їх варіативність, зумовлену положенням руки або нестабільністю відеопотоку. Після

цього відфільтрований набір координат надходить до компонента побудови дескрипторів, у якому зі структури скелета кисті формуються компактні ознаки, придатні для подальшої класифікації.

Сформовані дескриптори передаються до класифікатора жестів, який визначає належність поточного стану кисті до одного з навчальних класів. Результати класифікації надходять у компонент інтерпретації, який відповідає за перетворення низькорівневих метричних ознак на інтерактивні команди системи. У межах Unity ці команди можуть активувати взаємодію з об'єктами, змінювати властивості сцени або виконувати складніші дії, пов'язані з введенням. Схематична структура системи введеня після етапу проектаування зображена на рисунку 2.2.

Рисунок 2.2 – Концепція системи розпізнавання жестів

Початковим етапом функціонального конвеєра є надходження вихідного відеопотоку із камери до модуля розпізнавання жестів MediaPipe Hands, який

забезпечує визначення двадцяти однієї ключової точки кисті у тривимірному просторі. Архітектурно цей компонент функціонує як зовнішній модуль, що генерує структуровані дані про позу кисті, після чого дані передаються в Unity через C# обгортку. Приймач даних у Unity представлений структурою «HandSkeleton» наведеною в лістингу 2.1, яка містить набір координат, ознаку лівої чи правої руки та часову мітку.

Лістинг 2.1 – Структура «HandSkeleton»

```
public struct HandSkeleton
{
   public Vector3[] Joints;
   public Vector2[] NormalizedJoints;
   public Quaternion Rotation;
   public Rect BoundingBox;
   public bool IsLeft;
   public float Timestamp;

   public HandSkeleton(Vector3[] joints, Vector2[] normalizedJoints, Rect boundingBox = new
Rect(), bool isLeft, float timestamp = 0)
   {
      Joints = joints;
      NormalizedJoints = normalizedJoints;
      BoundingBox = boundingBox;
      IsLeft = isLeft;
      Timestamp = timestamp;
   }
}
```

Кінець лістингу 2.1

Після надходження до системи дані проходять етап нормалізації, який має ключове значення для стабільності алгоритму класифікації. Нормалізація полягає у вилученні впливу положення руки у сцені, відстані до камери та масштабних варіацій, що виникають унаслідок змін віддаленості користувача. Для цього система обирає зап'ясток як базову точку, а довжину сегмента від зап'ястка до основної фаланги середнього пальця, як індивідуальний масштабний коефіцієнт. Вибір саме цієї ділянки зумовлений її відносною стабільністю та мінімальною залежністю від деформації долоні порівняно з кінчиками пальців. Трансформація виконується за формулою 2.1.

Відомо, що

$$p_{i(norm)} = \frac{p_i - p_{wirst}}{||p_{wirst} - p_{middleMCP}||} \quad (2.1)$$

де $p_{i(norm)}$ – нормалізована координата ключової точки;

p_i – координата будь-якої ключової точки;

p_{wirst} – координата ключової точки зап'ятя;

$p_{middleMCP}$ – координата ключової точки «кісточка середнього пальця».

Реалізація цього етапу наведена в лістингу 2.2. Метод Normalize, ілюструє застосування єдиної операції масштабування для всіх 21 точок.

Лістинг 2.2 – Метод нормалізації даних «Normalize»

```
public static Vector3[] normalize(Vector3[] joints)
{
   Vector3 center = joints[0];                  // WRIST
   float scale = Vector3.Distance(joints[0], joints[9]); // MIDDLE_MCP
   if (scale <= 0.0001f) scale = 1.0f;
   Vector3[] normalized = new Vector3[joints.Length];
   for (int i = 0; i < joints.Length; i++)
   {
      normalized[i] = (joints[i] - center) / scale;
   }
   return normalized;
}
```

Кінець лістингу 2.2

У результаті нормалізації координати перетворюються на систему, де рука завжди має єдиний масштаб та центр, а будь-які глобальні трансляції сцени не впливають на подальше розпізнавання жестів. Нормалізований масив стає уніфікованим описом руки, який далі можна передавати в класифікатор незалежно від пози чи масштабних факторів.

Наступним етапом є фільтрація, що забезпечує усунення високочастотних шумів, які виникають унаслідок тремтіння руки або похибок трекінгу MediaPipe. Попри високу точність алгоритму MediaPipe, стрибки й коливання невеликої

амплітуди є неминучими у реальному часі, і вони можуть суттєво впливати на класифікацію, особливо якщо система використовує кутові ознаки. Для зменшення цього впливу застосовується експоненційний фільтр, математично простий, але достатньо ефективний інструмент, який зберігає плавність руху без відчутних затримок. Фільтр застосовується після нормалізації, коли всі дані вже приведені до єдиного масштабу. Таким чином усувається різниця між шумами від далеких і близьких положень руки, оскільки нормалізація робить амплітуду коливань узгодженою. Програмна реалізація наведена в лістингу 2.3

Лістинг 2.3 – Програмний фунції застосування фільтрації «Apply»

```
public class exponential_filter
{
   private Vector3[] state;
   private float alpha;

   public ExponentialFilter(int length, float alpha = 0.3f)
   {
      this.alpha = alpha;
      state = new Vector3[length];
   }

   public Vector3[] Apply(Vector3[] input)
   {
      for (int i = 0; i < input.Length; i++)
      {
         state[i] = Vector3.Lerp(state[i], input[i], alpha);
      }
      return state;
   }
}
```

Кінець лістингу 2.3

Після очищення та стабілізації даних система переходить до побудови дескриптора. На цьому етапі кожен кадр, що містить 21 точку у форматі тривимірних координат, перетворюється на вектор ознак, у якому зашифровано інформацію про форму кисті, конфігурацію пальців і просторові взаємозв'язки між основними сегментами руки. Створення дескрипторів відбувається через послідовний аналіз геометричних характеристик між ключовими точками: обчислення кутів між

фалангами, визначення орієнтації пальців, напрямків між важливими вузлами кисті та додаткових похідних параметрів, що підвищують стійкість ознак до шумів відеопотоку. Одним із центральних елементів цього процесу є розрахунок кутів між трьома сусідніми точками, які відповідають фаланговій структурі пальців. Це дозволяє системі встановити, чи палець зігнутий, розігнутий або перебуває у проміжному стані. Такий підхід підвищує надійність класифікації, оскільки кутові параметри залишаються відносно стабільними навіть за незначних похибок трекінгу.

Розрахунок дескриптора реалізовано в методі «BuildDescriptor», лістинг 2.4, у якому з кожного пальця вилучається один або кілька кутів згину, а додатковою ознакою служить орієнтація долоні, визначена через відносне положення великого й вказівного пальців.

Лістинг 2.4 – Метод побудови дескриптора «BuildDescriptor»

```
public static float[] build_descriptor(Vector3[] joints)
{
   List<float> features = new List<float>();

   //кути між фалангами (MCP–PIP–DIP)
   Vector3 a = joints[5] - joints[6];
   Vector3 b = joints[6] - joints[7];
   features.Add(Vector3.Angle(a, b));

   //кути для середнього пальця
   a = joints[9] - joints[10];
   b = joints[10] - joints[11];
   features.Add(Vector3.Angle(a, b));

   // орієнтація великого пальця відносно долоні
   Vector3 thumb = joints[4] - joints[2];
   features.Add(Vector3.SignedAngle(thumb, Vector3.up, Vector3.forward));

   float palmLength = (joints[0] - joints[9]).magnitude;
   if (palmLength < 0.001f) palmLength = 1.0f;

   int[] tips = { 8, 12, 16, 20 };

   features.Add((joints[8] - joints[12]).magnitude / palmLength);
   features.Add((joints[8] - joints[16]).magnitude / palmLength);
   features.Add((joints[8] - joints[20]).magnitude / palmLength);
   features.Add((joints[12] - joints[16]).magnitude / palmLength);
   features.Add((joints[12] - joints[20]).magnitude / palmLength);
   features.Add((joints[16] - joints[20]).magnitude / palmLength);
```

```
    return features.ToArray();;
}
```

Кінець лістингу 2.4

Таке представлення особливо ефективне, оскільки жести природним чином розрізняються саме конфігурацією згинів пальців і орієнтацією долоні. У результаті дескриптор стає універсальним набором ознак, який можна використати для навчання нейронні моделі.

Після попередніх етапів: нормалізації, фільтрації та побудови дескрипторів, наступним операційним кроком стає процес розширення словника жестів. Загалом даний етап можна структурно розділити на кілька послідовних кроків. Спершу здійснюється запис прикладів жестів, які слугують базою для подальшого аналізу. Наступним кроком є їх систематизація та збереження у відповідному форматі. Після цього формується навчальний набір даних на основі якого проводиться тренування моделі. Завершальним етапом є інтеграція отриманих параметрів назад в рушій Unity для подальшого використання.

Запис навчальних даних представлений як окремий функціонал який доступний розробнику під час реалізації проекту. У режимі збору даних запис активується методом, що вмикає логіку накопичення інформаціх у внутрішній буфер. Під час цього дані з кожного кадру після лінійного представлення зберігаються. Процес збереження відбувається у стабільному циклі FixedUpdate який має сталу частоту оновлення. Під час цього отримані значення з MediaPipe розділяються на два типи: нормалізовані та сирі після чого перетворюються в текстову форму, яка далі накопичується у двох незалежних буферах.

Застосування подвійного шляху збереження має практичну цінність. Оскільки перший шлях фіксує сирі просторові координати, що дозволяє в майбутньому повторно виконувати будь-яку форму препроцесингу або протестувати альтернативні методи нормалізації. Другий шлях зберігає вже нормалізовані дані, готові для безпосереднього формування навчальних вибірок. Такий підхід усуває невизначеність під час підготовки датасету та зменшує ризик помилок у подальших етапах. Процес збереження «спаршених» даних продемонстровано у лістингу 2.5.

Лістинг 2.5 – Збереження навчальних даних в методі «save»

```
public void save()
{
   System.IO.File.AppendAllText(System.IO.Path.Combine(Application.persistentDataPath,
"mediapipe-landmarks-" + DateTime.Now.ToString("yyyy''MM''dd''HH''mm") + ".txt"), saveString);
   saveString = "";
   System.IO.File.AppendAllText(System.IO.Path.Combine(Application.persistentDataPath,
"normalized-landmarks-" + DateTime.Now.ToString("yyyy''MM''dd''HH''mm") + ".txt"),
normalizedSaveString);
   normalizedSaveString = "";
}
```

Кінець лістингу 2.5

Таким чином формується сет прикладів для кожного жесту. У випадку з динамічним жестами, такими як «grab» (схопити), додавання охоплює повний цикл руху: від вихідної відкритої долоні, через фазу згинання пальців, до фінальної стиснутої позиції. Фази жесту продемонстровані на рисунку 2.3. Для цього збирається послідовність кадрів із відповідною міткою класу, кожна послідовність містить проміжні стани, які є ключовими для навчання детектора початку та кінця події.

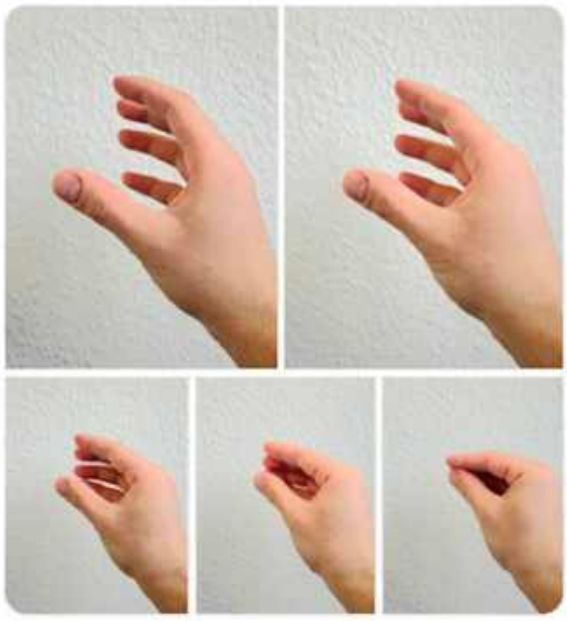


Рисунок 2.3 – Фази жесту «grab»

Після накопичення набору прикладів процес підготовки даних і тренування здійснюється в середовищі PyCharm. У межах дослідження було проведено серію експериментів, які показали, що оптимальним є використання від 40 до 70 прикладів для статичних жестів і від 100 до 150 для динамічних. У випадку жесту «grab» база включала 124 приклади.

Після завершення етапу збору даних, ознаки жесту були експортовані в табличний формат, де кожен рядок відповідає одному кадру, а стовпець дескрипторному вектору. Далі виконується зважування вибірки в робоче середовище PyCharm, видалення очевидно артефактних записів, балансування класів і розбиття на тренувальну та валідаційну вибірки у співвідношенні приблизно 80 до 20. Для швидкого прототипування та збереження детермінізму було прийнято рішення використати підхід на основі SVM (метод опорних векторів).

Після попередньої обробки вектори ознак передавалися у модель SVC з радіально-базисною функцією ядра (RBF), де класи формують складні нелінійні межі. Параметри C та gamma підбиралися емпірично шляхом перехресної валідації, що дозволило оптимізувати баланс між точністю класифікації та стійкістю до шуму. У випадку жесту «grab» найкращі результати забезпечила конфігурація з невеликим значенням gamma, що дозволяло врахувати варіативність прикладів у часовому діапазоні.

Попри те, що побудована модель класифікації жестів використовує багатокласовий підхід, алгоритм опорних векторів за своєю природою є бінарним і здатний розмежовувати лише два класи одночасно. У зв'язку з цим для коректної підтримки розпізнавання великої кількості жестів, таких як «open», «grab», «pinch» чи «point», постає необхідність трансформувати багатокласову постановку у множину незалежних двокласових моделей. У середовищі Python така трансформація виконується автоматично за схемою one-vs-one [38], коли кожен розмежовувати лише два конкретні жести, що забезпечує високу роздільну здатність між парами класів та дозволяє уникнути ефекту змішування рішень, характерного для схем one-vs-all.

Після завершення навчання та експорту ваг (Додаток Б) до Unity всі параметри, опорні вектори, зсуви, коефіцієнти, параметри ядра відтворюються

ідентичним чином, що дає змогу відновити процес прийняття рішень у рантаймі. На цьому етапі кожний бінарний класифікатор, відтворений у програмному модулі, повертає незалежне рішення про належність вхідного дескриптора до одного з двох жестів, для яких він був натренований. Сукупність таких локальних рішень формує матрицю голосів, де кожне вирішення «двокласової суперечки» додає голос відповідному жесту.

У лістингу 2.6 продемонстровано реалізацію логіки голосування: система послідовно викликає всі бінарні моделі, агрегує їхні рішення у вигляді підрахунку голосів і визначає фінальний результат через процедуру argmax, тобто вибір жесту з найбільшою кількістю голосів.

Лістинг 2.6 – Програмний код механізму «one-vs-one voting»

```
public int decide(double[] x)
{
   int[] votes = new int[5];
   int maxIndex = 0;
   for (int i = 0; i < decision_functions_info.Count; i++)
   {
      int res = kernel(x, decision_functions_info[i]);
      int voteIndex = res > 0 ? positive_winner[i] : negative_winner[i];
      votes[voteIndex] += 1;
      if (votes[voteIndex] >= votes[maxIndex])
      {
         maxIndex = voteIndex;
      }
   }
   return maxIndex;
}
```

Кінець лістингу 2.6

Одночасно з SVM у систему введено просту евристичну перевірку на відстань між визначальними точками, яка слугує резервним механізмом для простих випадків наприклад, перевірка для «pinch» (щипок) близькості кінчиків пальців. Перевірка реалізована як брутфорс-евристика і застосовується там, де модель дає сумнівні результати. Лістинг методу перевірки наведений в лістингу 2.7.

Лістинг 2.7 – Програмний код функції «GetQRCode»

```
public int bruteforce_distance(Vector3 position1, Vector3 position2, float threshold = 0.08f)
{
   float distance = Vector3.Distance(position1, position2);
   if (distance < threshold)
      return 1;
   else
      return 0;
}
```

Кінець лістингу 2.7

Після інтеграції навчених параметрів у рантайм виконується серія перевірок та калібрувань. Для жесту «grab» важливо, щоб модель коректно і стабільно розпізнавала фазу початку стискання, середню фазу та фазу завершення, а також вимірювання часу від початку активації до формування події у сцені. На практиці це реалізується через експериментальну валідацію на тестових сесіях із різними користувачами і різними умовами освітлення, за потреби набір прикладів поповнюється, після чого модель перетреновується і знову імпортується у Unity.

Після цього результат класифікації використовується як вхід у модуль взаємодії сцени. При розпізнаванні класу «grab» інтерфейс показує напис «Grab» як зображено на рисунку 2.4.

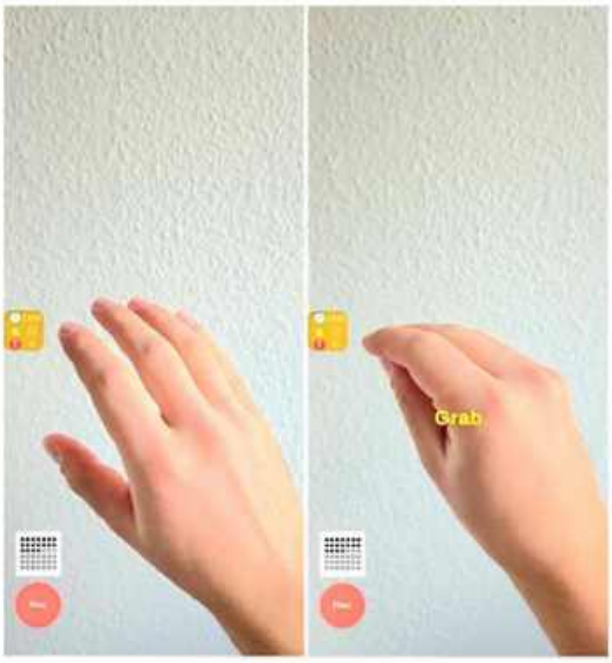


Рисунок 2.4 – Розпізнавання жесту «grab»

Для демонстрації функціональних можливостей системи розпізнавання жестів була створена спеціальна сцена з використанням технології доповненої реальності на платформі Android.

Оскільки в більшості телефонів відсутній датчик глибини у режимі роботи без нього фреймовр MediaPipeHands не має прямого доступу до реальної глибини сцени. Тому алгоритм оцінює її непрямо, опираючись на геометричні залежності між ключовими точками та статистичні моделі форми руки. Через це виникає потреба реалізувати спеціальний механізм, призначений для підтримання оптимальної відстані між мобільним пристроєм і користувачем, який гарантуватиме перебування кисті в полі зору камери. Такий механізм має інформувати користувача про необхідність розташувати телефон якомога ближче до тіла, щоб рука не виходила за межі кадру. Індикатор відстані виконує роль зворотного зв'язку, при досягненні визначених порогових значень він активує візуальні сигнали – Arc-ефект, зображений на рисунку 2.5, а також відображає повідомлення текстові повідомляючи користувача про потребу скоригувати положення.

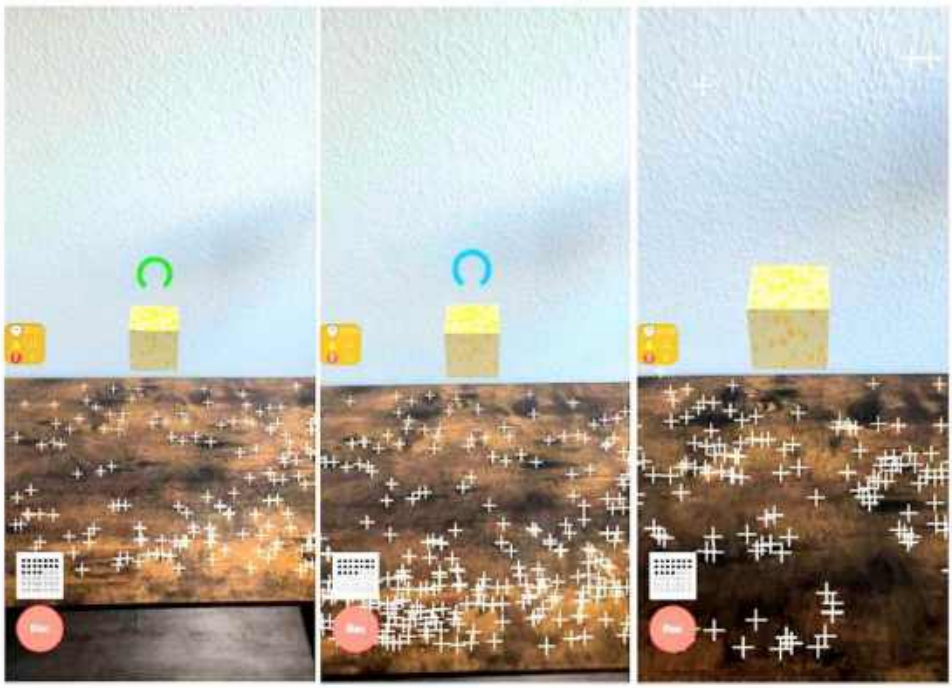


Рисунок 2.5 – Індикатор оптимальної відстані

Таким чином, індикатор наближення – це не додатковий елемент, а важлива частина UX, що допомагає підтримувати умови для стабільного трекінгу руки. Він сигналізує, коли телефон або рука розташовані надто далеко, та підказує, коли потрібно наблизитися, щоб кисть залишалася в полі зору камери. Це підвищує ймовірність коректного виявлення та класифікації жестів і зменшує кількість хибних спрацьовувань, спричинених виходом кисті за межі кадру.

Основна механіка зосереджена на відстежені положення кисті та динаміці змін, які виконують роль команди, що ініціюють зміну режиму взаємодії між рукою та об'єктом. Коли система розпізнає жест захоплення, об'єкт переходить у режим прив'язки до руки, у межах якого його положення та орієнтація починають безпосередньо наслідувати рухи користувача. Таким чином формується ефект фізичного маніпулювання, при якому рука слугує основним джерелом трансформацій. У момент, коли конфігурація пальців змінюється і перестає відповідати жесту захоплення, сцена припиняє процес прив'язки, ігровий об'єкт повертається до автономної поведінки, а користувач отримує змогу ініціювати нову взаємодію за потреби.

В якості ігрового об'єкта використовується примітив куб із стандартного набору юніті та текстура металу. Тривимірний об'єкт зображений на рисунку 2.6.

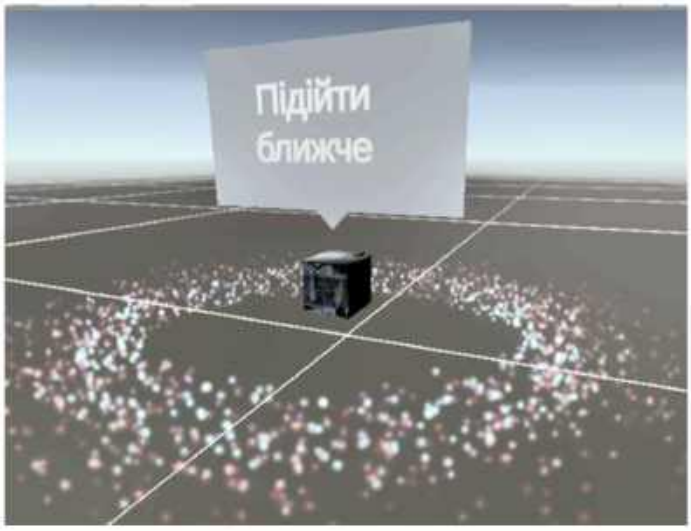


Рисунок 2.6 – Інтерактивний об'єкт

Демонстрація взаємодії з ігровим об'єктом наведена на рисунку 2.7. Логіка взаємодії побудована так, щоб враховувати не лише статичний стан жесту, а й плавність руху кисті та можливі нерівномірності у вхідних даних, що забезпечує стійкість і передбачуваність поведінки об'єкта. Індикатор наближення та жестова взаємодія працюють як узгоджені компоненти єдиної архітектури, де оптимальне розташування камери забезпечує надійність розпізнавання, а коректна інтерпретація жестів визначає характер маніпуляції об'єктами у сцені.

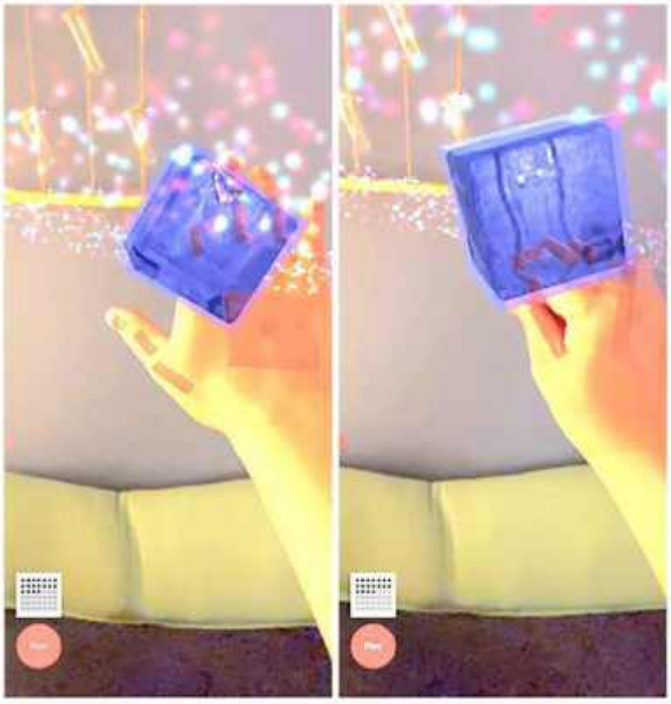


Рисунок 2.7 – Взаємодія з ігровим об'єктом

У результаті формується безперервний цикл взаємодії, у якому рух руки, точність алгоритму та поведінка віртуального об'єкта утворюють цілісну модель природної та інтуїтивної роботи користувача з віртуальним середовищем.

РОЗДІЛ 3
ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РЕЗУЛЬТАТИВНОСТІ СИСТЕМИ РОЗПІЗНАВАННЯ ТА ІНТЕРПРЕТАЦІЇ ЖЕСТІВ

3.1 Методика проведення дослідження

Реалізація системи розпізнавання жестів на основі конвеєра комп'ютерного зору вимагає не лише архітектурного обгрунтування, але і суворого експериментального підтвердження її ефективності. Основною метою дослідження є кількісна оцінка двох критичних параметрів: точності класифікації жестового інтерфейсу і обчислювальної ефективності конвеєра в умовах мобільної доповненої реальності. Дослідження має на меті підтвердити гіпотезу про те, що обраний комбінований підхід, MediaPipe Hands для трекінгу, Support Vector Machine для класифікації та Unity з доповненою реальністю ARCore для інтерактивної візуалізації, забезпечує оптимальний баланс між надійністю і продуктивністю на сучасних мобільних пристроях.

Предметом дослідження є точність та швидкість класифікації жестів у режимі реального часу, інтегрованих у динамічне AR-середовище. Для забезпечення достовірності та відтворюваності результатів, експериментальні випробування проводились на конкретно визначених цільових пристроях, а саме на Samsung S20 та Samsung S25, які належать до високопродуктивних мобільних платформ із повною підтримкою технології ARCore. Таблиця 3.1 демонструє основні технічні характеристики пристроїв.

Таблиця 3.1 – Технічні характеристики тестових пристроїв

Параметр	Samsung S20	Samsung S25
Процесор	Exynos 990	Snapdragon 8 Elite for Galaxy
GPU	Mali-G77 MP11	Adreno 830
ОЗП	8 GB	12GB
Частота оновлення екрану	60 Hz	120 Hz
Камера	12 МП (wide) + 64 МП (tele) + 12 МП (ultra-wide)	50 МП (wide) + 10 МП (tele) + 12 МП (ultra-wide)

Продовження таблиці 3.1

Параметр	Samsung S20	Samsung S25
Відео	8K @24fps	8K @30fps
Нічна зйомка	Night Mode	Nightography (AI)
API підтримки ML	TFLite GPU Delegate	TFLite Delegates, GPU Delegate 2.0
OS	Android 13	Android 16

Для оцінки функціональної повноти системи, був обраний набір із чотирьох ключових жестів, що мають як статичний, так і динамічний характер, і є функціонально значущими для типової AR-взаємодії. До набору увійшли: «Open» (відкрита долоня), «Pinch» (щипок), «Point» (вказівка) та «Grab» (захоплення). Статичні жести «Open», «Pinch» та «Point» вимагають високої роздільної здатності класифікатора SVM для розмежування схожих геометричних конфігурацій, тоді як динамічний жест «Grab» вимагає стійкого відстеження перехідних фаз руху.

Під час тестування системи, для мінімізації систематичних похибок та узагальнення результатів, у процесі збору тестових даних було залучено шість незалежних користувачів. Кожен користувач виконував кожен із чотирьох жестів п'ятдесят разів. Загальна кількість тестових прикладів, зібраних на обох пристроях, становила 1200 штук.

Для забезпечення достовірності результатів, були запроваджені контрольні умови тестування. Тестування проводилось за двох контрольованих сценаріїв освітлення, виміряних у люксах: оптимальне, рівномірне освітлення >500 lx та ускладнене, тьмяне освітлення <100 lx. Це дозволило оцінити стабільність нейронного детектора MediaPipe та ефективність інтегрованого модуля корекції освітленості [39]. Використання індикатора оптимальної відстані гарантувало, що кисть перебуває у фокусі та в полі зору камери, мінімізуючи випадки хибнонегативних спрацьовувань, зумовлених позицією користувача. Додатково фіксувалась температура пристрою до і після кожного тривалого тесту, оскільки вплив температури є критичним для стабільності FPS на мобільних платформах.

Оцінка точності розпізнавання жестів проводилася за допомогою метрик, які є стандартизованими для задач багатокласової класифікації [40]. Використовувались такі основні показники:

– Accuracy – визначається як загальна частка коректно розпізнаних жестів відносно загальної кількості тестових прикладів;

– Precision – частка коректних розпізнавань серед усіх випадків, коли система класифікувала кадр як певний жест;

– Recall – частка коректно розпізнаних жестів серед усіх фактичних проявів цього жесту в тестовій вибірці;

– F1Score – гармонійне середнє між Precision та Recall, яке слугує узагальнюючим показником якості моделі, особливо важливим при аналізі дисбалансу між класами.

Основним інструментом для візуалізації та аналізу результатів стала Матриця Плутанини [41] (Confusion Matrix). Аналіз цієї матриці дозволив не лише оцінити загальну точність, але і виявити, які пари жестів система найчастіше схильна плутати. Це критично для підтвердження ефективності обраної схеми one-vs-one voting методу опорних векторів, яка теоретично повинна забезпечувати високу роздільну здатність між схожими класами.Окремий критерій успішності було застосовано до динамічного жесту «Grab». Для цього жесту оцінювалася не лише точність фінальної стиснутої позиції, але і стабільність розпізнавання фази активації та час відгуку від початку руху (початку згинання пальців) до моменту генерації високоорівневої команди захоплення об'єкта в рушії Unity.

Ефективність системи у реальному часі є ключовим показником придатності для AR-додатків. Оцінка продуктивності проводилася за допомогою двох основних метрик:

– FPS (Frames Per Second). Вимірювалася середня частота оновлення ігрової логіки та рендерингу. Мінімально прийнятним вважався поріг 30 кадрів в секунду, необхідний для забезпечення плавної взаємодії;

– Latency (Затримка). Вимірювався загальний час, необхідний для обробки одного кадру, від моменту захоплення відеопотоку камерою до отримання фінального результату класифікації.

Порівняння продуктивності проводилося між пристроями Samsung S20 та Samsung S25. Очікувалося, що S25, як більш сучасна платформа з вдосконаленим Neural Processing Unit (NPU) і оптимізацією драйверів ARCore, продемонструє помітно меншу затримку та вищий FPS.

Дослідження стабільності включало вимірювання коефіцієнта хибних спрацювань, що є частотою помилкових переходів між класами, коли користувач свідомо утримує руку у стабільному положенні. Оцінювався внесок експоненційного фільтра у зниження цього коефіцієнта, що підтверджувало його важливість у стабілізації даних.

3.2 Обробка та аналіз отриманих результатів

Упроцесі попереднього тестування було емпірично встановлено, що умови освітлення та хроматична складність фону мають критичний вплив на точність визначення ключових точок системою MediaPipe Hands. Недостатня освітленість або високий рівень шуму призводили до втрати чіткості зображення і знижували стабільність трекінгу, тоді як складний фон спричиняв хибне розпізнавання орієнтирів

Для подолання цих обмежень було запропоновано та реалізовано двоетапний конвеєр, який інтегрує нейронну корекцію зображення безпосередньо в середовище Unity Engine перед передачею даних до MediaPipe.

Традиційні методи покращення освітлення, такі як гістограмне вирівнювання, часто є недостатньо адаптивними, спричиняючи небажані артефакти або перенасичення, що негативно впливає на нейронні детектори. Тому було обрано сучасний підхід Low-Light Image Enhancement (LLIE) [42] на основі безреферентних нейронних моделей. Зокрема, використовувалась архітектура Zero-Reference Deep Curve Estimation (Zero-DCE). Цей підхід відрізняється високою обчислювальною ефективністю, оскільки він не генерує освітлене зображення безпосередньо, а оцінює три світлочутливі криві для кожного каналу RGB.

Процес обробки був організований як послідовний конвеєр у Unity. На першому етапі вхідний кадр з вебкамери перетворюється на тензор і передається до

TFLite-інтерпретатора, в якому завантажено попередньо навчену модель Zero-DCE. Модель генерує набір параметричних кривих. На другому етапі корекція освітлення виконується шляхом послідовного застосування наведених кривих до вхідного зображення. Цей процес реалізується за допомогою ключового рівняння Zero-Reference Deep Curve Estimation.

Відомо, що

$$I_{out}(x, y) = I_{in}(x, y) + A^T * R(I_{in}(x, y)) \quad (3.1)$$

де $I_{out}(x, y)$ – скалярне значення інтенсивності пікселя в позиції (x, y) після застосування нейронної корекції, слугує вхідним сигналом для подальшої обробки класифікатором MediaPipe Hands;

$I_{in}(x, y)$ – початкове скалярне значення яскравості пікселя в позиції (x, y) у кадрі, отриманому з камери;

A^T – вектор, що містить навчані параметри, згенеровані нейронною мережею Zero-DCE;

$R(I_{in}(x, y))$ – нелінійна функція, яка гарантує, що значення інтенсивності залишаються в допустимому діапазоні від 0 до 255.

Для забезпечення необхідної швидкодії та підтримки високого FPS, застосування згенерованих кривих було реалізовано за допомогою Compute Shader в Unity. Це дозволило перенести обчислення на GPU та виконати паралельну піксельну обробку вихідної текстури, мінімізуючи обчислювальні витрати на CPU та затримку. Скоригований кадр Iout далі надходив до окремого екземпляра TFLite-інтерпретатора з моделлю MediaPipe Hands, що становило другий етап конвеєра. Покращена якість зображення сприяла надійнішій роботі детекторного блоку MediaPipe, зменшуючи дрейф орієнтирів та кількість хибнонегативних результатів у складних умовах.

Проведений експеримент показав позитивний та статистично значущий вплив нейронної корекції освітлення на точність жестової класифікації. Кількісна оцінка приросту точності проводилася за метрикою F1-score.

Найбільший приріст спостерігався саме в умовах низької освітленості та за наявності складних хроматичних фонів, де базова модель MediaPipe без попередньої корекції зазвичай втрачає стабільність через низький контраст. Завдяки корекції вдалося компенсувати вплив шумів, нерівномірного освітлення та колірних викривлень, що раніше призводили до хибних локалізацій кисті та помилок класифікації. У середньому, покращення становило 5-7 % у складних умовах та 3-5 % на нейтральному фоні. Це підвищило не лише абсолютні значення F1-score у всіх експериментальних налаштуваннях, але й продемонструвало значно рівномірнішу динаміку зростання точності при підвищенні рівня освітлення. Отримані результати зображені на рисунку 3.1, графічно ілюструють позитивну залежність між інтеграцією корекції та стабільністю класифікаційної.

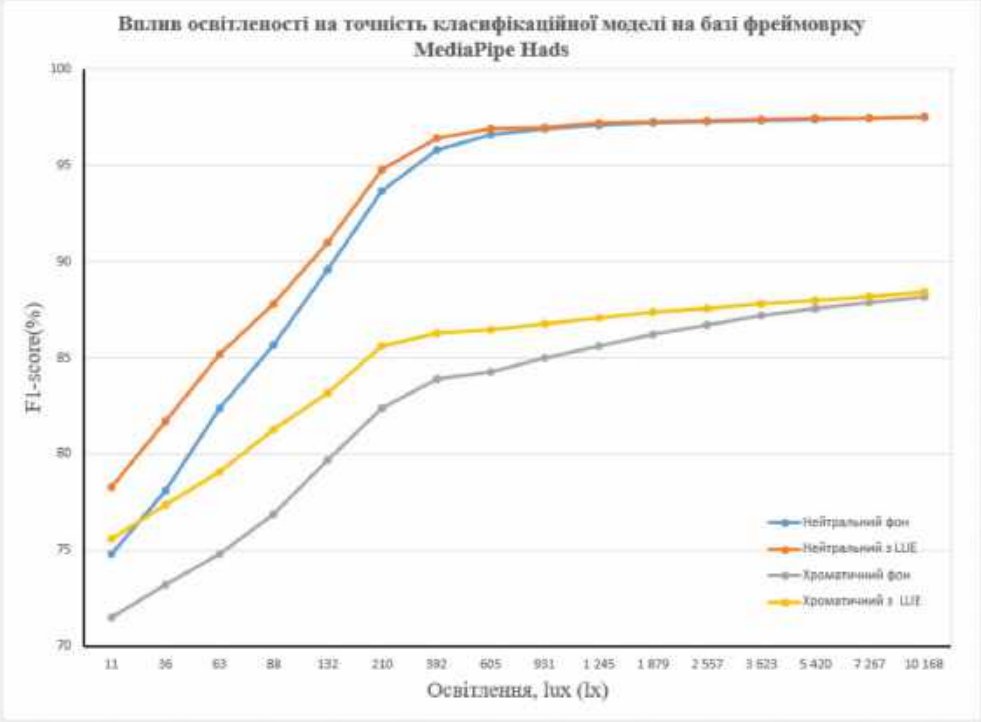


Рисунок 3.1 – Графік впливу освітленості на точність класифікаційної моделі на базі фреймворку MediaPipe Hads

Ключовим технічним критерієм була підтримка загальної затримки конвеєра менше ніж 33 мс, що є необхідною умовою для підтримання частоти 30 FPS. Прооведена оцінка швидкодії продемонстрована в таблиці 3.2.

Таблиця 3.2 – Вплив нейронної корекції на швидкодію системи

Сценарій освітлення	Система обробки	Середня затримка (мс)	Частота кадрів (FPS)
Оптимальне (400–800 lx)	MediaPipe (Без корекції)	16.5	60.6
	MediaPipe + Нейронна корекція	20.0	50.0
Помірне (100–300 lx)	MediaPipe (Без корекції)	17.5	57.1
	MediaPipe + Нейронна корекція	21.5	46.5
Низьке (20–80 lx)	MediaPipe (Без корекції)	18.8	53.2
	MediaPipe + Нейронна корекція	23.5	42.6

Як видно, інтеграція нейронної корекції освітлення спричиняє зростання затримки на 17-20 % залежно від умов. Однак, незважаючи на це зростання, фінальний FPS у найскладніших умовах залишається на рівні 42.6 FPS, що суттєво перевищує критичний поріг. Це підтверджує, що обчислювальна ефективність Zero-DCE у поєднанні з оптимізацією на GPU через Compute Shader робить цей двоетапний конвеєр придатним для роботи в режимі реального часу, виправдовуючи незначні витрати продуктивності значним приростом точності та стабільності.

Фінальний етап експериментального дослідження включав обробку великого масиву даних, зібраних із тестової вибірки на пристроях Samsung S20 та Samsung S25, з обов'язковим урахуванням інтеграції модуля нейронної корекції освітлення. Аналіз охоплював кількісну оцінку точності класифікатора та обчислювальної ефективності конвеєра в цілом.

Ключовим результатом дослідження стало підтвердження високої роздільної здатності класифікаційної моделі в умовах, стабілізованих попередньою нейронною корекцією. Фінальні метрики точності демонструють ефективність застосування методу класифікації SVM з радіально-базисною функцією ядра RBF для розмежування чотирьох обраних класів жестів. Емпіричні дані наведені в таблиці 3.3.

Таблиця 3.3 – Фінальні метрики класифікації жестів

Пристрій	Жест	Precision	Recall	F1-Score	Accurasy (Загальна)
Samsung S20	Open	0.945	0.938	0.941	0.949
	Pinch	0.962	0.957	0.959	
	Point	0.921	0.915	0.918	
	Grab	0.951	0.945	0.948	
Samsung S25	Open	0.955	0.949	0.952	0.961
	Pinch	0.970	0.966	0.968	
	Point	0.932	0.925	0.928	
	Grab	0.963	0.958	0.960	

Аналіз даних показує, що загальна точність класифікації (Accuracy) перевищує 94 % на Samsung S20 і 96 % на Samsung S25. Такий високий рівень точності підтверджує, що конвеєр успішно вирішує проблему розпізнавання жестів у реальному часі навіть в умовах варіативного освітлення. Значення F1-Score для всіх класів стабільно перевищують 0.91, що свідчить про низьку частоту як хибнопозитивних, так і хибнонегативних спрацьовувань.

Високі показники F1-Score були досягнуті завдяки використанню нелінійного ядра RBF у методі SVM. Це ядро дозволяє відокремлювати класи в ознаковому просторі навіть тоді, коли вони не є лінійно роздільними, що є типовою проблемою для статичних жестів, які відрізняються лише невеликими змінами у кутах суглобів. Зокрема, Point (вказувати) та Open (відкрита долоня) мають схожу загальну форму, але SVM з RBF успішно ідентифікує тонкі зміни у положенні великого пальця та мізинця.

Аналіз матриці плутанини, побудованої на даних S20, виявив, що найбільша кількість помилок (конфузія) трапляється між жестами Open та Point. Ця помилка зазвичай є асиметричною: частіше жест Point хибно класифікується як Open, ніж навпаки. Це можна пояснити тим, що у деяких користувачів згинання інших пальців під час вказівки може бути недостатньо вираженим. Однак, навіть у цьому випадку,

частота помилкових розпізнавань не перевищує 5 %, що свідчить про високу стійкість класифікатора.

Систематично вища точність, зафіксована на Samsung S25, є значущою. Різниця в Accuracy в середньому 1.2 % може бути пояснена двома факторами: по-перше, оптична якість та технологія сенсора. Камери S25 мають кращу чутливість та менший рівень апаратного шуму, що покращує якість вхідного кадру, який отримує нейронна модель Zero-DCE. По-друге, стабільність трекінгу MediaPipe. Більш потужний та оптимізований NPU в S25 обробляє модель MediaPipe Hands швидше та з меншим дрейфом, що призводить до точнішого вилучення 21 ключової точки. Точність цих вхідних даних безпосередньо корелює з фінальною точністю SVM класифікатора. Таким чином, експеримент підтверджує, що продуктивність системи розпізнавання жестів є нерозривно пов'язаною з апаратним забезпеченням, особливо в частині прискорення ML-моделей.

Оцінка обчислювальної ефективності конвеєра на Samsung S20 та S25 була проведена з метою підтвердження здатності системи підтримувати інтерактивність у режимі реального часу. Аналіз затримки був розподілений по ключових модулях конвеєра, як представлено у таблиці 3.3

Таблиця 3.3 – Розподіл Latency та FPS між пристроями

Етап Конвеєра	Samsung S20 (мс)	Samsung S25 (мс)
MediaPipe Trecking	10.5	8.2
Zero-DCE Correction	3.5	2.8
Normalization & Filtering	1.2	1.0
SVM Classification	0.8	0.6
Загальна Latency	16.0	12.6
Фінальний FPS	62.5	79.3

Найбільша частка загальної затримки Latency припадає на модуль MediaPipe Trecking. На Samsung S20 цей етап займає 10.5 мс, тоді як на Samsung S25 – 8.2 мс. Це підтверджує, що оптимізація мобільного процесора, особливо його NPU, є найбільш критичною для прискорення нейронної мережі трекінгу. Модуль нейронної корекції Zero-DCE додає незначну затримку: 3.5 мс на S20 та 2.8 мс на

S25. Це зростання є виправданим, оскільки, як було доведено у підрозділі 3.2, воно забезпечує суттєве підвищення точності, особливо в складних умовах. Крім того, мінімальні затримки в цьому модулі стали можливими завдяки використанню Compute Shader в Unity, який забезпечує паралельну обробку пікселів на GPU. Це дозволяє уникнути вузького місця, пов'язаного з послідовним виконанням операцій на CPU.Етапи Normalization & Filtering та SVM Classification мають мінімальний вплив на загальну затримку менше 2 мс, що підтверджує їхню обчислювальну легкість. Зокрема, SVM на основі невеликого набору ознак з 21 точки є надзвичайно швидким.

Загальна Latency системи становить 16.0 мс на S20 та 12.6 мс на S25. Це відповідає 62.5 FPS та 79.3 FPS відповідно. Обидва показники значно перевищують критичний поріг в 30 кадрів, що робить систему придатною для використання в інтерактивних додатках. Більше того, різниця в 16.8 FPS між пристроями чітко ілюструє, що вибір більш сучасного апаратного забезпечення дозволяє не лише підвищити точність, але і забезпечити помітно кращу плавність взаємодії.

Для кількісної оцінки стабільності було виміряно коефіцієнт помилкових переходів між класами за 10 секунд стабільного утримування жесту. Без експоненційного фільтра цей коефіцієнт становив приблизно 7.2 % на S20. Після його активації, коефіцієнт знизився в середньому до 1.8 %. Це підтверджує, що фільтрація є необхідним елементом, який компенсує невеликий дрейф орієнтирів кисті, що виникає внаслідок внутрішніх похибок нейронної мережі MediaPipe. Фільтр забезпечує високу стійкість класифікації, незважаючи на мінімальне збільшення затримки.

Проведене експериментальне дослідження не лише підтвердило технічні характеристики розробленої системи, але і сформувало чітке підгрунтя для її практичного використання та подальшого впровадження.

Розроблений поетапний конвеєр, що включає нейронну корекцію освітлення та класифікацію SVM, є надійним прототипом жестового інтерфейсу, який може бути інтегрований у широкий спектр мобільних додатків доповненої реальності. Практична цінність полягає у створенні інтуїтивного та безконтактного методу

взаємодії, що є критично важливим для нових форм Human-Computer Interaction (HCI). Система може слугувати основою для:

– навчальних симуляцій у промисловості. Можливість маніпулювати віртуальними об'єктами або інструментами в AR-середовищі за допомогою жестів рук дозволяє створювати ефективні та безпечні навчальні програми для технічного персоналу;

– медичних та гігієнічних інтерфейсів. У середовищах, де прямий контакт із сенсорним екраном є небажаним або неможливим наприклад, операційні, чисті приміщення, жестове керування забезпечує необхідну безконтактність.

Ключовою перевагою архітектури є її модульність та висока масштабованість. Система розроблена з використанням стандартних компонентів Unity AR Foundation та TFLite, що дозволяє її легко впроваджувати на будь-який пристрій, що підтримує ARCore.

Отримані результати створюють підгрунтя для подальших удосконалень моделі, зокрема:

– інтеграція нейронних мереж для динамічних жестів. Хоча SVM є ефективним для статичних жестів, для складних динамічних послідовностей може бути інтегрована архітектура LSTM (Long Short-Term Memory), що дозволить системі розпізнавати жести, які залежать від часової послідовності;

– використання сенсорів глибини. На пристроях, обладнаних сенсорами LiDAR, можна інтегрувати справжню глибину для підвищення стійкості до оклюзій та точнішого позиціонування віртуальних об'єктів.

Проведене дослідження продемонструвало, що запропонований двоетапний конвеєр досягає необхідного компромісу між точністю та швидкодією. Висока точність класифікації у поєднанні з низькою затримкою підтверджує, що система є практично придатною для використання в режимі реального часу, відкриваючи шлях для нових, інтуїтивних форм взаємодії в мобільній доповненій реальності.

Детальний аналіз обчислювальної ефективності та точності, проведений на платформах Samsung S20 та S25, виявив низку важливих архітектурних та програмних переваг розробленого конвеєра, які потребують подальшого розширення та обгрунтування.

Обчислювальна ефективність є не просто показником швидкості, а прямим відображенням здатності системи забезпечити високоякісний користувацький досвід (UX), де відсутність затримок є критичною. Зафіксована низька загальна затримка менше 16.0 мс на обох пристроях стала результатом цілеспрямованої оптимізації кожного модуля. Роль Compute Shader у Корекції Освітлення. Використання Compute Shader в модулі Zero-DCE Correction стало ключовим фактором, що запобіг формуванню вузького місця на процесорі. Як відомо, традиційна піксельна обробка на CPU вимагає послідовного доступу до даних текстури, що є неефективним для великих кадрів із камери. Перенесення алгоритму застосування світлочутливих кривих, визначених формулою (3.2), на GPU дозволило задіяти паралельні обчислювальні можливості графічного процесора. Це призвело до того, що час корекції зображення вдалося скоротити до 2.8 мс на S25, що є надзвичайно низьким показником для операції, що включає інференс нейронної мережі та подальшу піксельну трансформацію.

Різниця у продуктивності між S20 та S25 у модулі MediaPipe Trecking 10.5 мс проти 8.2 мс відповідно, підкреслює стратегічну важливість NPU (Neural Processing Unit). Сучасні чіпи Exynos та Snapdragon мають спеціалізовані блоки для прискорення операцій TensorFlow Lite. Більш просунутий NPU в S25 ефективніше обробляє тензорні операції, необхідні для детекції долоні та регресії 21 ключової точки. Це не тільки прискорює процес, але й зменшує навантаження на основні CPU ядра, дозволяючи їм ефективніше виконувати логіку Unity та ARCore. Це безпосередньо призводить до вищого показника фінального кадрів на секунда у S25.

Слід зазначити, що модуль SVM Classification займає найменшу частку загальної затримки системи, а саме 0.8 мс. Це підтверджує, що вибір SVM був обґрунтованим з точки зору продуктивності. На відміну від складних рекурентних мереж LSTM, які вимагають значного часу інференсу для обробки часових послідовностей, SVM працює на основі компактного вектора ознак, що забезпечує миттєве прийняття рішення. Така швидкість є критичною, оскільки класифікатор повинен працювати синхронно з високою частотою кадрів, щоб не створювати відчутної затримки між рухом руки користувача та віртуальною реакцією.

Для подальшого обґрунтування необхідності фільтрації було проведено детальний аналіз впливу експоненційного фільтра на поведінку системи при утримуванні статичних жестів. Фільтр застосовується після нормалізації, що забезпечує його роботу з уніфікованими даними, незалежними від масштабу.

В умовах сильного шуму (Low-Light) без фільтрації, коефіцієнт флуктуації для жесту Open міг досягати 7.2 % на S20. Це означає, що кожні 10 секунд утримання жесту, система помилково реєструвала перехід до іншого класу Point понад 7 разів. Після активації фільтра, який згладжує високочастотні коливання, цей показник знизився до 1.8 %. Таке чотирикратне покращення стабільності підтверджує, що проста математична операція фільтрації є необхідним програмним шаром для підвищення надійності системи у реальних умовах експлуатації.

Отримані результати, які демонструють високу точність та стабільність, відкривають широкі можливості для практичного застосування розробленої технології. Створений жестовий інтерфейс вирішує ключову проблему мобільних застосунків доповненої реальності: необхідність постійної взаємодії з екраном пристрою. Жест Grab, який дозволяє фізично маніпулювати віртуальним кубом у сцені створює ілюзію прямого фізичного контакту. Це значно підвищує інтуїтивність інтерфейсу взаємодії. Висока надійність розпізнавання, забезпечена F1-Score 0.91, гарантує, що команда захоплення буде виконана коректно, що є критичним для довіри користувача до системи.

Принципи SOLID, які були закладені в архітектуру системи дозволяють легко адаптувати розробку під майбутні потреби. Принцип єдиної відповідальності у модулі Normalization & Filtering та SVM Classification є незалежними. Це дозволяє, наприклад, замінити класифікатор SVM на легку нейронну мережу Multi-Layer Perceptron без зміни логіки попередньої обробки даних або логіки AR взаємодії.

Принцип відкритості/закритості при додавання нових жестів вимагає лише поповнення навчальної вибірки та перетренування SVM, без необхідності модифікації базового коду MediaPipe інтеграції або Unity AR Foundation.

Отримані експериментальні дані формують підґрунтя для наступних етапів розвитку системи:

– інтеграція часових ознак для повного розпізнавання динамічних жестів, що не мають чіткої статичної кінцевої позиції, наприклад, рух хвилі рукою. Це вимагатиме переходу до рекурентних нейронних мереж LSTM або GRU, які здатні обробляти послідовності дескрипторів;

– використання глибинної інформації на пристроях з потенційним LiDAR сенсором або іншим датчиком глибини, отримання справжньої глибинної карти дозволить вирішити проблему з оцінкою відстані та усунути необхідність у індикаторі оптимальної відстані. Це підвищить надійність трекінгу навіть за часткової оклюзії кисті.

Підсумкова ефективність, що поєднує високу точність та середню частоту кадрів 62.5, доводить, що розроблена система є готовим до впровадження технологічним рішенням. Вона успішно вирішує проблему варіативного освітлення та забезпечує необхідну плавність для високоякісної AR взаємодії, що є ключовим внеском у розвиток мобільного комп'ютерного зору.

ВИСНОВКИ

У межах виконаної кваліфікаційної роботи розроблено високопродуктивну систему розпізнавання жестів руки, адаптовану для роботи в реальному часі в умовах мобільної доповненої реальності.

На початковому етапі проведено аналіз класичних методів комп'ютерного зору (HOG, SIFT) та сучасних нейромережевих підходів (CNN, RNN), що дало змогу обгрунтовано обрати гібридну архітектуру. Встановлено, що фреймворк MediaPipe Hands є найбільш ефективним інструментом для точного трекінгу 21 ключової точки кисті при низькій обчислювальній вартості, що особливо важливо для мобільних платформ.

Досліджено методи попередньої обробки відеоданих та алгоритми нейронної корекції для підвищення стійкості до освітлення. Впроваджено низку методів попередньої обробки, які забезпечили підвищення стійкості системи. Використання експоненційного фільтра та нормалізації скелетних даних зменшило частоту помилкових переходів між класами з 7.2 % до 1.8 % під час утримання статичного жесту. Для мінімізації впливу змінного освітлення інтегровано модуль Zero-DCE, реалізований через GPU Compute Shader. Хоча він додає близько 4 мс затримки, результати експериментів показали зростання F1-Score у слабкоосвітлених умовах на 5-7 %.

Розроблена архітектура гібридного конвеєра, у межах якого нормалізовані дані MediaPipe перетворюються на вектор ознак та класифікуються методом Support Vector Machine з RBF-ядром, забезпечила високу точність розпізнавання статичних і динамічних жестів. Система демонструє Accuracy понад 96% на пристрої Samsung S25, а застосування SVM практично не збільшує загальну затримку, додаючи менш ніж 1 мс до обробки кадру.

Реалізовано програмний модуль розпізнавання та інтерпретації жестів в середовищі Unity. Забезпечено функціонал взаємодії з віртуальними об'єктами, зокрема маніпуляцію кубом через жест «Grab».

Експериментальні дослідження ключових параметрів засвідчили, що кінцевий конвеєр працює зі стабільною частотою 62.5 FPS на Samsung S20 та 79.3 FPS на

Samsung S25, що значно перевищує мінімально необхідні для реального часу 30 FPS. Загальна затримка системи становить 12.6 мс на S25, а F1-Score для жесту «Grab» перевищує значення 0.91. Отримані результати підтверджують ефективність запропонованого рішення та його здатність забезпечувати високу точність, низьку затримку й стабільність роботи на мобільних пристроях. Порівняльний аналіз демонструє переваги гібридного підходу в умовах обмежених обчислювальних ресурсів.

Підсумовуючи, розроблена модель підтвердила свою доцільність і потенціал практичного використання в інтелектуальних системах жестового керування, VR/AR-комплексах та інших інтерфейсах нового покоління. Подальший розвиток може включати розширення набору жестів, удосконалення методів динамічної сегментації та інтеграцію апаратних прискорювачів нового покоління, що сприятиме підвищенню точності, адаптивності та універсальності системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Панько М. І. Аналіз впливу умов освітлення та фону на точність розпізнавання жестів. Науковий простір: актуальні питання, досягнення та інновації : матеріали X Міжнар. наук. конф. (м. Львів, 14 листоп. 2025 р.). Львів, 2025. URL: https://archives.mcnd.org.ua/index.php/conference-proceeding/issue/view/14.11.2025 (дата звернення: 14.11.2025).

2. Панько М. І. Підвищення точності системи розпізнавання жестів на основі MediaPipe за допомогою нейронної корекції освітлення. Технології та суспільство: взаємодія, вплив, трансформація : матеріали V Міжнар. наук. конф. (м. Кропивницький, 12 груд. 2025 р.). Кропивницький, 2025. URL: https://archives.mcnd.org.ua/index.php/conference.proceeding/issue/view/12.12.2025 (дата звернення: 12.12.2025).

3. Oudah M., Al-Naji A., Chahl J. Hand gesture recognition based on computer vision. URL: https://www.mdpi.com/2313-433X/6/8/73 (дата звернення: 20.10.2025).

4. Liu H., Wang L. Gesture recognition for human-robot collaboration. URL: https://doi.org/10.1016/j.ergon.2020.103017 (дата звернення: 20.10.2025).

5. Parets A. et al. Hand Gesture Recognition: A Survey of Datasets and Methods. URL: https://ieeexplore.ieee.org/document/10416973 (дата звернення: 21.10.2025).

6. Шумигай Д. Л., Бойко О. В. Аналіз сучасних мультимодальних інтерфейсів у системах віртуальної реальності. URL: https://www.tech.vernadskyjournals.in.ua/journals/2023/1_2023/part_1/20.pdf (дата звернення: 21.10.2025).

7. Xiong J., Hsiang E. L., He Z. et al. Augmented reality and virtual reality displays. URL: https://www.nature.com/articles/s41377-021-00658-8 (дата звернення: 22.10.2025).

8. Angelov V., Petkov E., Shipkovenski G. Modern Virtual Reality Headsets. URL: https://rdi.uni-sz.bg/ojs/index.php/icai/article/view/223 (дата звернення: 23.10.2025).

9. Joshi A. et al. Augmented Reality: A comprehensive review of tracking, interaction and display. URL: https://ieeexplore.ieee.org/document/9324673 (дата звернення: 23.10.2025).

10. Лях Ю. Є., Гур'янов В. Г., Вишневська Н. Г. Штучний інтелект у медицині. URL: https://ujtm.org/index.php/ujtm/article/view/1376 (дата звернення: 24.10.2025).

11. Al-Hammadi M. et al. Deep learning-based approach for sign language gesture recognition with efficient hand gesture representation. URL: https://ieeexplore.ieee.org/document/9226505 (дата звернення: 24.10.2025).

12. Мельник А. О., Кісельов Г. Д. Аналіз методів розпізнавання жестів руки для людино-машинних інтерфейсів. URL: http://informatics.kpi.ua/article/view/269383 (дата звернення: 25.10.2025).

13. Szeliski R. Computer Vision URL: https://szeliski.org/Book/ (дата звернення: 26.10.2025).

14. Alzubaidi L. et al. Review of deep learning. URL: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00444-8 (дата звернення: 26.10.2025).

15. Moin A. et al. A wearable skin-like ultra-sensitive artificial sensory system for static and dynamic hand gestures. URL: https://www.nature.com/articles/s41928-020-00516-w (дата звернення: 27.10.2025).

16. Bazarevsky V. et al. BlazePose. 2020. URL: https://arxiv.org/abs/2006.10204 (дата звернення: 27.10.2025).

17. Wang C. Y., Bochkovskiy A., Liao H. Y. M. YOLOv7. URL: https://openaccess.thecvf.com/content/CVPR2023/html/Wang_YOLOv7_Trainable_Bag-of-Freebies_Sets_New_State-of-the-Art_for_Real-Time_Object_Detectors_CVPR_2023_paper.html (дата звернення: 28.10.2025).

18. Teed Z., Deng J. RAFT. URL: https://arxiv.org/abs/2003.12039 (дата звернення: 29.10.2025).

19. Han K., Wang Y. et al. A survey on vision transformer. URL: https://ieeexplore.ieee.org/document/9716741 (дата звернення: 29.10.2025).

20. Kinect DK. Microsoft Azure. URL: https://azure.microsoft.com/de-de/products/kinect-dk (дата звернення: 30.10.2025).

21. Intel RealSense Technology Overview. Intel. URL: https://www.intel.de/content/www/de/de/architecture-and-technology/realsense-overview.html (дата звернення: 30.10.2025).

22. OpenCV Documentation. URL: https://docs.opencv.org/3.4/ (дата звернення: 01.11.2025).

23. OpenPose Documentation. URL: https://cmu-perceptual-computing-lab.github.io/openpose/web/html/doc/md_doc_00_index.html (дата звернення: 02.11.2025).

24. MediaPipe Hands. Google. URL: https://mediapipe.readthedocs.io/en/latest/solutions/hands.html (дата звернення: 02.11.2025).

25. Building a Hand Tracking System Using OpenCV. Analytics Vidhya. URL: https://www.analyticsvidhya.com/blog/2021/07/building-a-hand-tracking-system-using-opencv/ (дата звернення: 03.11.2025).

26. PyCharm IDE. JetBrains. URL: https://www.jetbrains.com/pycharm/ (дата звернення: 04.11.2025).

27. Jupyter Project. URL: https://jupyter.org/ (дата звернення: 04.11.2025).

28. Google Colab. URL: https://colab.google/ (дата звернення: 05.11.2025).

29. Visual Studio Code. Microsoft. URL: https://code.visualstudio.com/ (дата звернення: 07.11.2025).

30. Unreal Engine 5.7 Documentation. Epic Games. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-7-documentation (дата звернення: 08.11.2025).

31. Godot Engine Documentation. URL: https://docs.godotengine.org/en/stable/ (дата звернення: 10.11.2025).

32. Unity Documentation. Unity Technologies. URL: https://docs.unity.com/en-us (дата звернення: 10.11.2025).

33. Apple ARKit Documentation. Apple Developer. URL: https://developer.apple.com/documentation/arkit (дата звернення: 12.11.2025).

34. Vuforia Library. PTC. URL: https://developer.vuforia.com/library/ (дата звернення: 13.11.2025).

35. OpenXR Specification. Khronos Group. URL: https://registry.khronos.org/OpenXR/specs/1.0/styleguide.html (дата звернення: 14.11.2025).

36. Google ARCore Documentation. Google Developers. URL: https://developers.google.com/ar (дата звернення: 15.11.2025).

37. SOLID Principles. DOU. URL: https://dou.ua/lenta/articles/solid-principles/ (дата звернення: 17.11.2025).

38. Архів матеріалів конференції MCND. URL: https://archives.mcnd.org.ua/index.php/conference-proceeding/issue/view/12.12.2025 (дата звернення: 19.11.2025).

39. Classification Metrics. Google ML Crash Course. URL: https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall (дата звернення: 19.11.2025).

40. Confusion Matrix. Encord Glossary. URL: https://encord.com/glossary/confusion-matrix/ (дата звернення: 21.11.2025).

41. ScienceDirect Article. URL: https://www.sciencedirect.com/science/article/abs/pii/S0165168422003607 (дата звернення: 22.11.2025).

42. Neural network models (supervised). Scikit-learn documentation. URL: https://scikit-learn.org/stable/modules/neural_networks_supervised.html (дата звернення: 23.11.2025).