Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБЛЕННЯ ІНТЕРАКТИВНОЇ СИСТЕМИ КЕРУВАННЯ ПРЕЗЕНТАЦІЄЮ АБО КОМП'ЮТЕРОМ ЗА ДОПОМОГОЮ ЖЕСТІВ
(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-22-2
Аксьонова В. Р.
(прізвище, ініціали)
Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Інформатика
(повна назва освітньої програми)
Керівник проф. Гороховатський В. О.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О. А.
(підпис) (прізвище, ініціали)

2026 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Аксьоновій Вікторії Романівні
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення інтерактивної системи керування презентацією або комп'ютером за допомогою жестів

затверджена наказом університету від 26 травня 2026 року № 435Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 20 червня 2026 р.
3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, бібліотека комп'ютерного зору з відкритим кодом OpenCV, мова програмування Python, середовище розробки Microsoft Visual Studio Code.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз сучасних рішень для управління комп'ютером та презентацією за допомогою жестів.
2. Розроблення алгоритмів керування курсором, скролінгу та виконання системних команд.
3. Програмна реалізація системи безконтактного керування комп'ютером та презентацією.
4. Аналіз результатів розробленої системи.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність проблеми, постановка задачі, етапи розроблення застосунку, тестування застосунку, висновки, апробація результатів роботи.

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

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

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	13.04.2026	
2	Аналіз завдання, підбір літератури	14.04.26-19.04.26	
3	Аналіз літератури з проблеми, що вирішується	19.04.26-25.04.26	
4	Аналіз існуючих методів розпізнавання	25.04.26-27.04.26	
5	Формування кроків вирішення проблеми	27.04.26-09.05.26	
6	Програмна реалізація	30.04.26-25.05.26	
7	Аналіз отриманих результатів	25.05.26-04.06.26	
8	Оформлення пояснювальної записки	05.06.26-09.06.26	
9	Перевірка на нормоконтроль	10.06.26-19.06.26	
10	Перевірка на плагіат	11.06.26-30.06.26	
11	Рецензування	20.06.26-30.06.26	
12	Підготовка презентації та доповіді	20.06.26-30.06.26	
13	Занесення роботи в електронний архів	30.06.26-09.07.26	
14	Попередній захист кваліфікаційної роботи	30.06.26-09.07.26	

Дата видачі завдання 13 квітня 2026 р.

Здобувач _____
(підпис)

Керівник роботи _____ проф. Гороховатський В. О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 74 с., 37 рис., 40 джерел.

БЕЗКОНТАКТНЕ КЕРУВАННЯ, КОМП'ЮТЕРНИЙ ЗІР, РОЗПІЗНАВАННЯ ЖЕСТІВ, MEDIAPIPE, OPENCV, PYTHON.

Об'єктом роботи є програмна модель для здійснення керування комп'ютером та презентацією за допомогою жестів руки.

Метою роботи є розроблення програми, яка забезпечує інтуїтивну взаємодію користувача з пристроєм без використання традиційних засобів введення, таких як миша та клавіатура.

Під час роботи використано: методи комп'ютерного зору (розпізнавання жестів руки, визначення ключових точок руки), методи обробки даних (алгоритми згладжування координат, фільтрація шумів), архітектурний підхід Data-Driven Design (використання JSON для гнучкого налаштування команд).

У результаті розроблено програмний застосунок, що забезпечує керування презентацією або комп'ютером за допомогою жестів.

Область застосування: системи керування презентаціями, безконтактні інтерфейси взаємодії, освітні технології, розумні системи керування, допоміжні технології для людей з обмеженими можливостями.

CONTACTLESS CONTROL, COMPUTER VISION, GESTURE RECOGNITION, MEDIAPIPE, OPENCV, PYTHON.

The objective of this work is a software model for controlling a computer and a presentation using hand gestures.

The goal of this work is to develop an application that ensures intuitive user interaction with the device without using traditional input methods, such as a mouse and a keyboard.

The following were used during the work: computer vision methods (hand gesture recognition, hand landmarks tracking), data processing methods (coordinate smoothing algorithms, noise filtering), and the Data-Driven Design architectural approach (using JSON for flexible command configuration).

As a result, a software application has been developed that enables the control of a presentation or a computer using gestures.

Applications: presentation control systems, contactless interaction interfaces, educational technologies, smart control systems, assistive technologies for people with disabilities.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 6
Вступ 7
1 Інтерактивне керування комп'ютером за допомогою жестів 9
1.1 Аналіз предметної області 9
1.2 Огляд існуючих прикладних рішень 12
1.3 Розпізнавання жестів за набором ключових точок руки 17
1.4 Постановка задачі 19
2 Методи розпізнавання жестів та керування системою 21
2.1 Моделі подання цифрового зображення з використанням ключових точок 21
2.2 Нейромережева архітектура та принципи роботи системи розпізнавання жестів Gesture Recognizer 25
2.3 Обґрунтування алгоритму розпізнавання руки та виконання кліка 28
2.4 Аналіз вертикального зміщення для скролінгу 32
2.5 Модель роботи системи на основі станів 35
3 Результати програмного моделювання інтерактивної системи 40
3.1 Обґрунтування вибору програмного середовища 40
3.2 Етапи розроблення інтерактивної системи 41
3.3 Тестування та аналіз результатів 56
3.4 Перспективи подальшої роботи 66
Висновки 68
Перелік джерел посилання 70

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CV – Computer Vision (комп’ютерний зір)

FPS – Frames Per Second (кадри в секунду)

Gesture control – керування за допомогою жестів

Landmarks – ключові точки руки

OpenCV – Open Source Computer Vision Library (бібліотека комп’ютерного зору з відкритим кодом)

ВСТУП

Одним із перспективних напрямків розвитку сучасних інформаційних технологій є комп’ютерний зір та системи розпізнавання образів. Дані технології активно використовуються у сферах автоматизації, безконтактної взаємодії людини з комп’ютером, робототехніки, мультимедійних систем та штучного інтелекту. Особливу увагу сьогодні приділяють системам розпізнавання жестів, які дозволяють здійснювати керування комп’ютером без використання традиційних пристроїв введення, таких як клавіатура чи миша [1–3].

Розпізнавання жестів є одним із напрямків комп’ютерного зору, основним завданням якого є аналіз відеопотоку або зображення для визначення положення руки та інтерпретації жестів користувача. На основі отриманих даних система може виконувати певні команди, наприклад перемикання слайдів презентації, керування гучністю, прокручування сторінок або керування курсором миші. У певному сенсі задача розпізнавання жестів є задачею аналізу та семантичної інтерпретації зображення.

Сучасні системи розпізнавання жестів базуються на використанні методів штучного інтелекту, нейромережевих моделей та алгоритмів обробки зображень. Завдяки розвитку бібліотек комп’ютерного зору та машинного навчання стало можливим створення інтерактивних систем, здатних працювати в режимі реального часу з використанням звичайної вебкамери [4–6].

Актуальність теми полягає у зростаючій потребі у безконтактних способах взаємодії з комп’ютером, особливо в умовах розвитку дистанційної освіти, інтерактивних презентацій, мультимедійних систем та допоміжних технологій для людей з обмеженими можливостями. Використання жестового керування дозволяє зробити взаємодію з комп’ютером більш природною, інтуїтивною та зручною.

Сьогодні стрімкий розвиток технологій штучного інтелекту та комп’ютерного зору активно впроваджуються у повсякденне життя. Системи розпізнавання жестів знаходять застосування у сфері освіти, медицини, розумних будинків, інтерактивних інформаційних систем, ігрової індустрії та автоматизації робочих процесів. Особливого значення такі технології набули після поширення дистанційного навчання та онлайн-презентацій, де виникла потреба у швидкому та зручному керуванні цифровими матеріалами без фізичного контакту з пристроями.

У межах кваліфікаційної роботи розробляється інтерактивна система керування комп’ютером та презентаціями за допомогою жестів руки. Система використовує технології комп’ютерного зору для захоплення та аналізу відео з вебкамери, розпізнає жести користувача та виконує відповідні системні команди.

1 ІНТЕРАКТИВНЕ КЕРУВАННЯ КОМП'ЮТЕРОМ ЗА ДОПОМОГОЮ ЖЕСТІВ

1.1 Аналіз предметної області

Gesture control (керування за допомогою жестів) – це спосіб взаємодії людини з комп'ютерною системою, при якому команди передаються не через традиційні пристрої введення (клавіатуру чи мишу), а за допомогою рухів рук, пальців або всього тіла. Такий підхід базується на технологіях комп'ютерного зору та аналізу зображень, які дозволяють системі розпізнавати положення та рухи користувача у реальному часі [1–4].

Основою gesture control є визначення та інтерпретація жестів – певних положень або рухів руки, яким заздалегідь призначено відповідні дії. Жести можуть бути статичними (фіксоване положення руки, наприклад відкрита долоня або кулак) та динамічними (рух у часі, наприклад проведення рукою вгору або вниз). Для їх розпізнавання використовуються спеціальні алгоритми, які аналізують відеопотік з камери та визначають landmarks (ключові точки руки).

Перевагою такого способу взаємодії є природність та інтуїтивність, оскільки користувач не прив'язаний до фізичних пристроїв і може керувати системою на відстані. Це особливо актуально для презентацій, інтерактивних систем, віртуальної та доповненої реальності, а також у випадках, коли використання традиційних засобів керування є незручним або неможливим.

Технологія керування за допомогою жестів (gesture control) знаходить широке застосування в різних сферах, де важлива інтуїтивна та безконтактна взаємодія з цифровими системами [1].

Однією з основних областей використання є керування презентаціями. У цьому випадку gesture control дозволяє доповідачу взаємодіяти зі слайдами без використання миші або клавіатури. За допомогою жестів можна

перемикати слайди, запускати демонстрацію, керувати вмістом на екрані або навіть взаємодіяти з курсором. Це значно підвищує мобільність та свободу рухів під час виступу, а також створює більш сучасний і динамічний формат подачі матеріалу.

Ще однією важливою сферою є системи віртуальної та доповненої реальності – Virtual Reality (рис. 1.1) та Augmented Reality (рис. 1.2). У таких середовищах жести виступають одним із ключових способів взаємодії з віртуальними об’єктами. Користувач може переміщувати, обертати або масштабувати об’єкти, використовуючи лише рухи рук. Це дозволяє створити ефект повного занурення у цифровий простір і зробити взаємодію максимально природною.

Рисонок 1.1 – Окуляри віртуальної реальності

Рисунок 1.2 – Приклад доповненої реальності на грі Pokemon Go

Також gesture control активно використовується у смарт-пристроях, таких як телевізори, розумні дисплеї, системи «розумного дому» та автомобільні інтерфейси. У цих випадках жести дозволяють керувати пристроями на відстані, наприклад перемикати канали, регулювати гучність, прокручувати меню або взаємодіяти з додатками. Це особливо зручно в ситуаціях, коли користувач не може або не хоче торкатися пристрою.

Традиційні засоби керування комп’ютером, такі як миша та клавіатура, протягом тривалого часу залишаються основними інструментами взаємодії користувача з системою. Проте вони мають низку обмежень, які стають особливо помітними в сучасних умовах використання інформаційних технологій.

По-перше, використання миші та клавіатури передбачає фізичний контакт із пристроями, що обмежує свободу рухів користувача. Це може бути незручно під час проведення презентацій, коли доповідач змушений постійно повертатися до комп’ютера для перемикання слайдів або виконання інших дій.

По-друге, традиційні засоби керування є стаціонарними, тобто користувач прив’язаний до конкретного місця. Це ускладнює взаємодію в ситуаціях, де потрібна мобільність, наприклад під час виступів, навчання або демонстрації матеріалів у великих приміщеннях.

Крім того, у певних умовах використання (наприклад, у медичних установах, на виробництві або під час роботи з забрудненими руками) застосування миші та клавіатури може бути незручним або небажаним з гігієнічної точки зору.

Актуальність розроблення систем керування комп’ютером за допомогою жестів зумовлена стрімким розвитком технологій комп’ютерного зору та зростаючими вимогами до більш природної та інтуїтивної взаємодії людини з цифровими системами. Сучасні користувачі очікують швидких, простих і безконтактних способів керування пристроями, що не потребують використання додаткових фізичних інструментів.

Таким чином, розробка систем керування жестами є актуальною та перспективною задачею, оскільки поєднує в собі сучасні технології, зручність використання та потребу у більш природних способах взаємодії людини з комп’ютером.

1.2 Огляд існуючих прикладних рішень

Сучасні системи керування комп’ютером за допомогою жестів базуються на використанні методів комп’ютерного зору та технологій машинного навчання, які дозволяють аналізувати відеопотік у реальному часі та інтерпретувати рухи користувача. Для реалізації таких систем застосовуються спеціалізовані програмні бібліотеки та фреймворки, що забезпечують обробку зображень, розпізнавання об’єктів та визначення ключових точок тіла людини [3].

Одним із найвідоміших готових рішень у сфері gesture control є система Microsoft Kinect, розроблена компанією Microsoft. Kinect являє собою пристрій, оснащений RGB-камерою (рис. 1.3), інфрачервоним сенсором глибини та мікрофонами, що дозволяють відстежувати положення тіла людини у просторі. Спочатку система була створена для ігрової консолі Xbox та орієнтувалася насамперед на геймерів. Основною ідеєю було забезпечення взаємодії з грою без використання класичного контролера за допомогою рухів тіла, жестів та голосових команд [2].

Рисунок 1.3 – Камера Kinect

На момент появи Kinect технологія виглядала революційною, оскільки пропонувала повноцінне body tracking, gesture control та голосове керування без необхідності використання додаткових пристроїв [3]. Проте на практиці система не змогла повністю замінити традиційні засоби керування. Багато користувачів відзначали, що постійні рухи руками швидко викликають втому, а для більшості ігор звичайний геймпад є значно зручнішим та точнішим. Особливо це стосувалося hardcore-геймерів та AAA-ігор, де потрібна висока швидкість реакції та точність керування. Найкраще Kinect проявив себе у танцювальних іграх, фітнес-застосунках та party games, де активна взаємодія рухами є частиною ігрового процесу.

Основною перевагою Kinect є можливість розпізнавання рухів у тривимірному просторі без необхідності використання додаткових датчиків на тілі користувача [7]. Система дозволяє відстежувати положення рук, голови та інших частин тіла, що робить її ефективною для реалізації жестового керування. Однак Kinect має і певні недоліки, серед яких висока вартість обладнання, залежність від спеціалізованого сенсора та обмежена мобільність системи. Крім того, з часом технологія Kinect втратила популярність, а її підтримка з боку Microsoft була фактично припинена. На відміну від Kinect, сучасні програмні рішення на основі веб-камери та бібліотек комп’ютерного зору дозволяють реалізувати gesture control без використання додаткового обладнання, що робить такі системи більш доступними для широкого кола користувачів.

Одним із сучасних прикладів gesture control є система Manitas, розроблена інженером Начо Мартіном [8]. Дана програма призначена для керування комп’ютером та взаємодії з вікнами операційної системи за допомогою жестів рук. Для роботи система використовує веб-камеру та алгоритми комп’ютерного зору, які аналізують рухи користувача у режимі реального часу.

Основною особливістю Manitas є те, що система не потребує спеціалізованого обладнання, такого як рукавички або сенсори глибини.

Програма дозволяє виконувати базові дії керування інтерфейсом, наприклад переміщення курсора, взаємодію з вікнами або виконання команд за допомогою жестів. Вихідний код проєкту був опублікований на платформі GitHub, що дозволяє розробникам модифікувати та вдосконалювати систему. Подібні рішення демонструють сучасну тенденцію переходу від дорогого спеціалізованого обладнання до програмних систем gesture control, які працюють лише за допомогою звичайної веб-камери та алгоритмів машинного навчання.

Таким чином, сучасні системи gesture control демонструють перехід від складних апаратних рішень зі спеціалізованими сенсорами до більш доступних програмних систем, що працюють на основі звичайної веб-камери та алгоритмів комп’ютерного зору. Реалізація подібних систем стала можливою завдяки розвитку сучасних бібліотек і фреймворків для обробки зображень, машинного навчання та розпізнавання жестів. Саме тому для створення ефективних систем керування комп’ютером за допомогою жестів важливим є вибір відповідних технологій, які забезпечують високу швидкість роботи, точність розпізнавання та можливість функціонування у режимі реального часу.

TensorFlow разом із високорівневим API Keras є одними з найпоширеніших інструментів для побудови систем глибокого навчання. Вони використовуються для створення та навчання нейронних мереж, які можуть розпізнавати складні патерни у даних, зокрема зображення та відео. У контексті розпізнавання жестів ці технології дозволяють будувати моделі, які аналізують положення руки або послідовність кадрів та визначають, який жест виконує користувач [9, 10].

Основною перевагою TensorFlow і Keras є висока точність розпізнавання та можливість навчати моделі під конкретні задачі. Це дозволяє реалізовувати дуже складні системи gesture control, які здатні розпізнавати не лише статичні жести, а й динамічні рухи, комбінації жестів та контекстні дії. Крім того, ці фреймворки підтримують GPU-обчислення, що значно прискорює навчання

моделей. Однак основним недоліком є висока складність розробки та потреба у великій кількості навчальних даних [11]. Для якісного результату необхідно формувати датасети жестів, проводити навчання та налаштовувати архітектуру нейронної мережі. Також такі системи потребують значних обчислювальних ресурсів, що ускладнює їх використання в простих або легких desktop-застосунках.

Ще одним популярним фреймворком для глибокого навчання, який активно використовується у дослідженнях та розробці систем штучного інтелекту є PyTorch. Він відомий своєю гнучкістю та більш «інтуїтивним» підходом до побудови нейронних мереж у порівнянні з TensorFlow [12]. PyTorch часто використовується для експериментів із моделями комп’ютерного зору та розпізнавання жестів.

Основною перевагою PyTorch є динамічна побудова обчислювального графа, що робить процес розробки більш зручним і зрозумілим. Це дозволяє швидко тестувати і змінювати архітектуру моделей, що особливо корисно при дослідженні нових підходів до gesture recognition. Крім того, PyTorch має сильну підтримку в науковій спільноті та широко використовується у сучасних AI-дослідженнях [13]. Недоліком PyTorch, як і інших deep learning рішень, є складність впровадження в реальні системи в режимі реального часу без додаткової оптимізації. Також для якісного навчання моделей потрібні великі набори даних та значні обчислювальні ресурси. Через це PyTorch найчастіше використовується для розробки та навчання моделей, які потім інтегруються у готові застосунки через більш легкі фреймворки або бібліотеки.

У данній роботі розглядаються основні технології, які використовуються при розробці систем керування жестами, зокрема бібліотеки OpenCV та MediaPipe [5, 14, 15] .

OpenCV (Open Source Computer Vision Library) – це одна з найпоширеніших бібліотек для обробки зображень і відео в реальному часі. Вона містить понад тисячу алгоритмів для аналізу зображень, включаючи

фільтрацію, трансформацію, виявлення об’єктів, трекінг руху та роботу з відеопотоком. OpenCV широко використовується як базовий інструмент у системах комп’ютерного зору.

Основними перевагами OpenCV є висока швидкість роботи, велика кількість реалізованих алгоритмів, підтримка різних мов програмування (C++, Python, Java) та можливість інтеграції з іншими бібліотеками машинного навчання. Завдяки цьому OpenCV є стандартом де-факто для задач обробки зображень у реальних застосунках. Однак OpenCV не має вбудованих високорівневих моделей для розпізнавання жестів або розуміння положення руки. Це означає, що для реалізації складних задач необхідно самостійно розробляти алгоритми або використовувати додаткові бібліотеки машинного навчання.

MediaPipe – це сучасний open-source фреймворк, розроблений компанією Google для побудови систем комп’ютерного зору та машинного навчання, що працюють у режимі реального часу. Основною особливістю MediaPipe є можливість роботи з відеопотоком у реальному часі та використання готових моделей для розпізнавання об’єктів, поз, облич і жестів рук [14, 15].

Фреймворк використовує модульну архітектуру, де кожен етап обробки даних представлений у вигляді окремих компонентів pipeline. Це дозволяє гнучко будувати системи різної складності, наприклад, поєднувати розпізнавання жестів із трекінгом руки або аналізом поз тіла. У контексті даної роботи MediaPipe використовується для визначення ключових точок руки (landmarks) та класифікації жестів користувача.

Перевагами MediaPipe є висока продуктивність, низька затримка, кросплатформеність (Windows, Linux, Android, iOS) та наявність готових рішень, зокрема для трекінгу рук і жестів, що значно прискорює розробку інтелектуальних систем взаємодії. Серед недоліків MediaPipe можна виділити обмежену кількість вбудованих жестів та залежність від якості вхідного відео.

Крім того, система використовує попередньо навчені моделі, що ускладнює глибоку кастомізацію поведінки без додаткових алгоритмів [14].

Основним обмеженням є те, що стандартний Gesture Recognizer підтримує лише фіксований набір жестів, що не дозволяє повністю адаптувати систему під специфічні сценарії взаємодії без розширення функціоналу власними методами.

1.3 Розпізнавання жестів за набором ключових точок руки

У сучасних системах керування жестами використовуються різні підходи до розпізнавання рухів людини, які можна умовно поділити на два основні напрями: класичні методи комп'ютерного зору та методи, що базуються на машинному та глибокому навчанні [16–20].

Традиційні методи комп'ютерного зору базуються на ручному виділенні ознак із зображення або відео. До таких ознак можуть належати контури, форма руки, колір шкіри, рух об'єктів або геометричні характеристики зображення. Після виділення ознак застосовуються алгоритми класифікації або порогові правила для визначення конкретного жесту.

Перевагою цього підходу є відносна простота реалізації та низькі вимоги до обчислювальних ресурсів. Однак класичні методи є чутливими до умов освітлення, фону та положення камери, що значно знижує їх точність у реальних умовах. Крім того, такі системи погано масштабуються та потребують ручного налаштування для кожного конкретного сценарію.

Більш сучасним підходом є використання методів машинного навчання (ML) та глибокого навчання (DL). У цьому випадку система навчається на великій кількості прикладів жестів і самостійно визначає закономірності у даних. Для цього застосовуються нейронні мережі, зокрема згорткові нейронні мережі (CNN), рекурентні мережі (RNN) або спеціалізовані моделі для аналізу поз людини.

Основною перевагою цього підходу є висока точність розпізнавання та здатність працювати зі складними сценами, де присутні зміни освітлення, фону або положення руки. Такі системи також краще узагальнюють дані та можуть розпізнавати більш складні та динамічні жести. Недоліками є висока обчислювальна складність, потреба у великій кількості навчальних даних та складність навчання моделей. Крім того, для реального часу часто потрібна оптимізація або використання спеціалізованих бібліотек.

Одним із ключових елементів сучасних систем розпізнавання жестів є використання так званих ключових точок руки (landmarks) [21]. Це набір координат, які описують анатомічно важливі точки кисті людини, такі як кінчики пальців, суглоби та основа долоні. На основі цих точок система може аналізувати положення руки у просторі та визначати виконуваний жест.

У сучасних бібліотеках комп'ютерного зору, зокрема MediaPipe, рука представляється у вигляді набору з 21 ключової точки (рис. 1.4). Кожна точка має координати x, y (а іноді і z), які описують її положення у відносній системі координат зображення. Значення координат нормалізовані, тобто знаходяться у діапазоні від 0 до 1, що дозволяє легко переносити модель між різними розмірами екранів та камер.

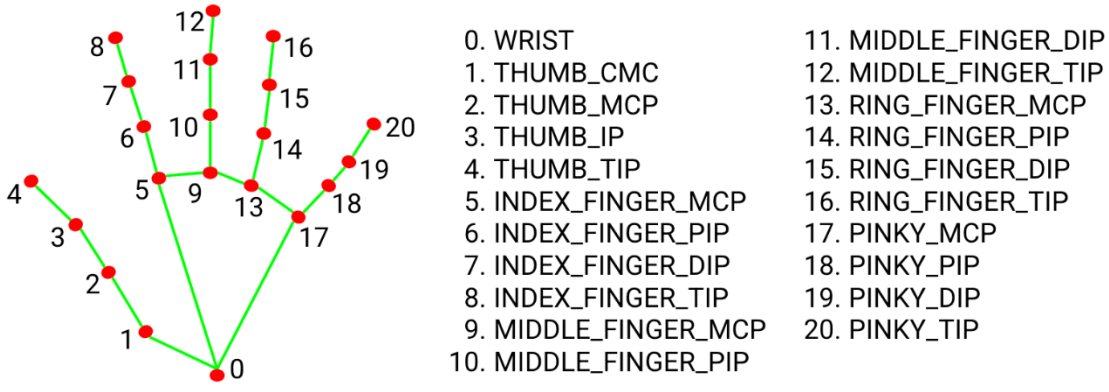

Рисунок 1.4 – Ключові точки долоні за бібліотекою MediaPipe

Ключові точки включають кінчики пальців, середні та основні суглоби кожного пальця, а також точку основи долоні. Наприклад, точка №8 зазвичай відповідає кінчику вказівного пальця, а точка №4 – великого пальця. Саме ці

точки часто використовуються для визначення додаткових жестів, таких як клік.

У системах керування за допомогою жестів усі рухи користувача можна умовно поділити на дві основні категорії: статичні та динамічні жести. Такий поділ використовується для підвищення точності розпізнавання та розширення функціональних можливостей системи.

Статичні жести – це жести, які визначаються за фіксованим положенням руки в певний момент часу. У цьому випадку система аналізує лише поточний кадр та положення ключових точок руки, не враховуючи зміну положення в часі. Прикладами статичних жестів є відкрита долоня (Open_Palm) або кулак (Closed_Fist). Такі жести є відносно простими для розпізнавання, оскільки базуються на геометричному розташуванні пальців.

Динамічні жести представляють собою послідовність змін положення руки або пальців у часі. На відміну від статичних, вони враховують рухи між кількома кадрами відеопотоку, що дозволяє аналізувати траєкторію руху. Прикладами динамічних жестів є прокрутка сторінки вгору або вниз, переміщення курсора, свайпи або інші рухи руки. Для їх реалізації використовується аналіз зміни координат ключових точок у часовому інтервалі, що дозволяє визначати напрямок і швидкість руху.

Отже, найбільш ефективним підходом до створення системи керування жестами є поєднання сучасних методів комп’ютерного зору з використанням landmarks та підтримкою як статичних, так і динамічних жестів. Це дозволяє створити універсальну, гнучку та зручну систему взаємодії користувача з комп’ютером.

1.4 Постановка задачі

Розроблення інтерактивної системи керування презентацією або комп’ютером за допомогою жестів є актуальним завданням у контексті

розвитку сучасних технологій комп’ютерного зору та людино-комп’ютерної взаємодії. Тому ставиться завдання створити програмну систему, яка працювати в режимі реального часу, аналізувати відеопотік з вебкамери та виконувати відповідні дії залежно від розпізнаних жестів.

Об’єктом роботи є програмна модель для здійснення керування комп’ютером та презентацією за допомогою жестів руки.

Метою роботи є розроблення програми, яка забезпечує інтуїтивну взаємодію користувача з пристроєм без використання традиційних засобів введення, таких як миша та клавіатура.

Для досягнення мети необхідно вирішити такі завдання:

– проаналізувати способи функціонування і побудови існуючих систем;

– реалізувати модуль розпізнавання жестів руки на основі відеопотоку з вебкамери;

– забезпечити керування презентацією за допомогою статичних жестів (перехід між слайдами, запуск презентації);

– розробити функціонал керування курсором миші на основі положення вказівного пальця;

– реалізувати механізм скролінгу сторінок за допомогою динамічних жестів;

– реалізувати функцію кліку шляхом визначення зближення великого та вказівного пальців;

– створити інформаційне UI-вікно, яке відображає поточний режим роботи системи, розпізнаний жест та виконану дію;

– реалізувати перемикання режимів роботи системи (режим презентації / режим керування комп’ютером), що забезпечує різні сценарії взаємодії;

– забезпечити можливість зміни жестів та посилання.

2 МЕТОДИ РОЗПІЗНАВАННЯ ЖЕСТІВ ТА КЕРУВАННЯ СИСТЕМОЮ

2.1 Моделі подання цифрового зображення з використанням ключових точок

У системах комп'ютерного зору відеопотік, що надходить із камери, розглядається як послідовність цифрових зображень (кадрів). Кожен кадр є дискретним представленням сцени в певний момент часу та може бути поданий у вигляді багатовимірної матриці.

Нехай один кадр позначається як *I*. Тоді його можна представити як матрицю пікселів [19]:

$$I = \{p(x, y)\}, \tag{2.1}$$

де *x,y* – просторові координати пікселі;

p(*x,y*) – значення пікселя в точці (*x,y*).

Кожне цифрове зображення можна уявити як сітку з маленьких точок, які називаються пікселями (picture element). У цифрових системах обробки зображень (наприклад, у комп'ютерному зорі) кожен піксель не є просто однією точкою, а містить інформацію про колір, а саме в форматі трикомпонентної теорії кольоросприйняття RGB. Позначимо його у вигляді вектора:

$$p(x, y) = (R, G, B), \tag{2.2}$$

де *R* – інтенсивність червоного каналу;

G – інтенсивність зеленого каналу;

B – інтенсивність синього каналу.

Кольоровий кадр можна подати як тривимірну матрицю:

$$I \in \mathbb{R}^{H*W*3}, \tag{2.3}$$

де H – висота зображення;

W – ширина зображення;

3 – кількість колірних каналів (RGB).

Таким чином, кожен кадр відеопотоку математично являє собою великий набір числових значень. Однак для розпізнавання жестів аналіз усіх пікселів є надмірним, оскільки основну інформацію про положення руки містять лише окремі характерні точки кисті. Для зменшення обсягу обчислень та підвищення швидкодії системи використовується метод виділення ключових точок руки (landmarks) [21–24]. У розробленій системі цю задачу виконує бібліотека MediaPipe.

Алгоритм MediaPipe Hands аналізує вхідне зображення та формує набір координат характерних точок кисті. Для кожної руки визначається 21 ключова точка (рис. 1.4). Кожна ключова точка описується координатами:

$$L_i = (x_i, y_i, z_i), \tag{2.4}$$

де x_i – горизонтальна координата;

y_i – вертикальна координата;

z_i – відносна глибина точки;

i – номер ключової точки.

У результаті замість аналізу мільйонів пікселів система працює лише з невеликим набором координат [25, 26]. Це значно зменшує складність обчислень та дозволяє виконувати розпізнавання жестів у реальному часі.

Після того як система розпізнає руку користувача, необхідно перетворити координати пальців, отримані з камери, у координати курсора на екрані комп’ютера. Саме цей етап забезпечує можливість керування мишкою

за допомогою жестів. Алгоритм MediaPipe повертає координати ключових точок руки у нормалізованому вигляді. Це означає, що значення координат не залежать від фактичної роздільної здатності камери та знаходяться у межах $0 \leq x, y \leq 1$. Завдяки цьому система може коректно працювати з різними вебкамерами без необхідності зміни математичної моделі або програмної реалізації. У такій системі координат:

– значення *x=0* відповідає крайній лівій межі кадру;

– значення *x=1* відповідає крайній правій межі кадру;

– значення *y=0* відповідає верхній межі кадру;

– значення *y=1* відповідає нижній межі кадру.

Тобто положення руки визначається не у пікселях, а у відносних координатах відносно розмірів зображення. Наприклад, координата $x = 0{,}5$ означає, що точка знаходиться приблизно посередині кадру по горизонталі незалежно від фактичної ширини зображення.

Для переміщення курсора потрібно перевести нормалізовані координати у координати екрана монітора. Якщо роздільна здатність екрана становить, наприклад, 1920×1080, то координати курсора повинні змінюватися саме у цих межах.

Перетворення здійснюється шляхом масштабування відносних координат відповідно до роздільної здатності екрану користувача. Для цього значення координат множаться на ширину та висоту екрану, що дозволяє отримати абсолютні координати положення курсора:

$$x_{screen} = x_{finger} * W_{screen}, \tag{2.5}$$

$$y_{screen} = y_{finger} * H_{screen}, \tag{2.6}$$

де x_{finger}, y_{finger} – нормалізовані координати пальця;

W_{screen}, H_{screen} – ширина та висота екрана;

x_{screen}, y_{screen} – координати курсора на моніторі.

Керування курсором реалізується за допомогою бібліотеки PyAutoGUI, яка забезпечує можливість програмного переміщення вказівника миші. Проте якщо напряму переміщати курсор у нові координати, його рух буде нестабільним. Причиною цього можуть бути шум відеокамери, мікрорухи руки, похибки розпізнавання або зміна освітлення.

У такому випадку курсор починає тремтіти або різко стрибати по екрану, що ускладнює керування системою. Оскільки безпосереднє відображення координат може призводити до різких та нестабільних рухів курсора, необхідно застосовувати механізм згладжування. Він базується на поступовому наближенні поточного положення курсора до цільового значення за допомогою коефіцієнта плавності. Це дозволяє зменшити тремтіння та зробити рух курсора більш природним і контрольованим.

Для цього можна скористатися методом лінійної інтерполяції. Його суть полягає у тому, що курсор переміщується не одразу у нову точку, а поступово наближається до неї. Між поточним положенням курсора і новою цільовою позицією обчислюється проміжне значення. Нове положення курсора визначається формулою:

$$x_{new} = x_{prev} + \alpha(x_{target} - x_{prev}), \tag{2.7}$$

$$y_{new} = y_{prev} + \alpha(y_{target} - y_{prev}), \tag{2.8}$$

де x_{prev}, y_{prev} – попереднє положення курсора;

x_{target}, y_{target} – нові координати пальця;

x_{new}, y_{new} – нові згладжені координати курсора;

α – коефіцієнт згладжування.

Коефіцієнт α задає швидкість реакції системи. Якщо взяти мале значення курсор рухається плавно, але із затримкою. Якщо велике значення коефіцієнта, то курсор реагує швидше, але може тремтіти. Використання

згладжування дозволяє зробити керування більш комфортним за рахунок усунення тремтіння та підвищення точності наведення.

2.2 Нейромережева архітектура та принципи роботи системи розпізнавання жестів Gesture Recognizer

Для стабільної роботи системи безконтактного керування в режимі реального часу важливо використовувати моделі машинного навчання, які можуть ефективно працювати навіть на пристроях з обмеженими обчислювальними ресурсами, без обов'язкового використання дискретного графічного процесора (GPU). Для цього можна використовувати спеціальний обчислювальний конвеєр MediaPipe Gesture Recognizer.

Система MediaPipe Gesture Recognizer реалізована у вигляді багатоступеневого конвеєра обробки зображень (pipeline), у якому кожен етап виконує окрему підзадачу [27]. Такий підхід дозволяє розділити складну задачу розпізнавання жестів на послідовність простіших операцій: детекцію долоні, нормалізацію області інтересу, побудову скелетної моделі руки та класифікацію жесту.

Важливим аспектом є забезпечення роботи системи в режимі реального часу. А саме, затримка обробки кожного кадру повинна бути мінімальною, а частота обробки відеопотоку – достатньо високою для комфортної взаємодії користувача. Саме тому використовуються легкі нейромережеві архітектури, оптимізовані для роботи на центральному процесорі без необхідності використання GPU.

Додатково система враховує можливі похибки відеоданих, такі як шум зображення, зміни освітлення та часткове перекриття руки. Для зменшення впливу цих факторів застосовуються механізми нормалізації вхідних даних, порогова фільтрація результатів класифікації та режим відстеження руки між кадрами, що підвищує стабільність роботи системи.

Першим етапом обробки кожного вхідного кадру є визначення області, в якій знаходиться долоня. Вхідне зображення згідно з (2.3) представляє кольоровий кадр розміром H×W, де кожен піксель описується трьома каналами (RGB). На цьому етапі система виконує задачу локалізації долоні, тобто визначає її положення на зображенні у вигляді обмежувальної рамки (bounding box) [28]. Для цього використовується спеціалізована нейромережа Palmdetector, яка побудована на основі архітектури Single Shot Detector (SSD), але значно спрощена та оптимізована для роботи на мобільних пристроях і вбудованих системах.

Ідея використання окремого детектора саме для долоні, а не для всієї руки, має як практичне, так і математичне обґрунтування. Долоня є більш стабільним об’єктом у порівнянні з пальцями, оскільки її форма змінюється менше під час руху. Це дозволяє зменшити складність моделі, тобто використовувати меншу кількість шарів і параметрів нейронної мережі без суттєвої втрати точності.

Крім того, спрощення задачі локалізації позитивно впливає на процес навчання моделі: функція втрат (loss function) стабільніше зменшується та швидше збігається під час тренування. Також така модель краще працює в умовах часткового перекриття об’єкта (оклюзії), коли окремі пальці можуть закривати один одного або бути частково невидимими.

Математично робота Palmdetector полягає в аналізі вхідного зображення як числової матриці пікселів та визначенні параметрів обмежувальної рамки, яка описує положення долоні на кадрі. У результаті роботи моделі формується вектор параметрів:

$$B = (x_{min}, y_{min}, w, h, \theta), \tag{2.9}$$

де $(x_{min},\ y_{min})$ – координати верхнього лівого кута рамки;

w, h – ширина і висота рамки;

θ – кут орієнтації (нахилу) долоні у просторі кадру.

Визначення кута θ дозволяє алгоритму виконати операцію нормалізації (повороту) вирізаного фрагмента зображення, щоб на наступний етап нейромережі рука завжди подавалася у вертикальному положенні. Це зменшує варіативність вхідних зображень і підвищує точність визначення ключових точок руки.

Після того як система Palmdetector знаходить та вирівнює область долоні, цей фрагмент зображення передається до другої нейромережі – Hand Landmark Model. На цьому етапі виконується більш детальний аналіз руки. Hand Landmark Model прогнозує координати, тобто не класифікує зображення, а обчислює положення ключових точок руки. Модель була навчена на великому наборі даних, що містить понад 30 000 зображень рук у різних положеннях, під різними кутами та при різному освітленні.

Основна задача цієї моделі – визначити точне просторове положення кисті користувача. У результаті роботи формується набір точок L, згідно з формулою (2.4).

Завершальним етапом є робота класифікатора жестів (Gesture Classifier). Він приймає набір координат L_i визначає, до якого жесту вони належать. Для цього використовується функція Softmax, яка перетворює результати моделі на ймовірності:

$$P(g_j \mid L) = \frac{e^{z_j}}{\sum_{k=1}^{M} e^{z_k}}, \tag{2.10}$$

де g_j– це конкретний жест (наприклад, Victory або Closed_Fist);

M – кількість усіх можливих жестів у системі.

Система обирає жест з найбільшою ймовірністю, але тільки якщо вона перевищує заданий поріг довіри (confidence threshold), який зберігається у конфігураційному файлі gestures.json. Якщо значення нижче порогу, результат ігнорується, щоб уникнути помилкових спрацьовувань через випадкові рухи.

Завдяки такій двоетапній структурі система працює точніше та краще відсіює зайві шуми зображення. Крім того, Palmdetector не запускається на кожному кадрі: якщо рука вже була знайдена раніше, наступні кадри обробляються через режим відстеження. Це зменшує навантаження на процесор і дозволяє системі працювати швидко та стабільно в реальному часі.

2.3 Обґрунтування алгоритму розпізнавання руки та виконання кліка

Наступним етапом після отримання, обробки зображення розпізнавання положення руки та визначення ключові точки руки необхідно проаналізувати форму кисті та положення пальців.

На відміну від класичних методів комп’ютерного зору, що працюють за рахунок аналізу окремих характеристик зображення: контурів, кольорів, меж об’єктів або простих геометричних форм, сучасні алгоритми розпізнавання руки використовують моделі машинного навчання [29–32].

Класичні методи мають ряд недоліків, оскільки ефективність їхньої роботи значною мірою залежить від зовнішніх умов середовища. На точність розпізнавання впливають рівень освітлення приміщення, колір та складність фону, а також якість відеокамери. Крім того, зміна положення руки або кута повороту кисті можуть призводити до помилок під час визначення форми руки та інтерпретації жестів. У таких випадках алгоритм може некоректно розпізнати жест або повністю втратити об’єкт на зображенні.

В свою чергу сучасні системи комп’ютерного зору використовують моделі машинного навчання, які працюють за іншим принципом. Такі моделі попередньо навчаються на великій кількості зображень рук у різних положеннях, при різному освітленні та на різних фонах. У процесі навчання модель визначає характерні особливості кисті людини та вчиться знаходити їх на нових зображеннях.

Завдяки використанню моделей машинного навчання вдається усунути більшість недоліків, характерних для класичних методів комп'ютерного зору. Зокрема, алгоритм забезпечує стабільне розпізнавання руки при зміні умов освітлення, а поворот кисті не призводить до втрати точності визначення положення пальців. Крім того, модель є менш чутливою до складного або динамічного фону, оскільки основна увага зосереджується безпосередньо на характерних ознаках руки. Це також дозволяє забезпечити коректне та стабільне розпізнавання жестів під час руху користувача у режимі реального часу.

Бібліотека MediaPipe містить головну модель машинного навчання для аналізу положення кисті у режимі реального часу. Модель визначивши необхідні ключові точки формує скелетну модель кисті – спрощене математичне представлення руки у вигляді набору точок та зв'язків між ними. Математично кисть руки можна представити як множину характерних точок:

$$\mathcal{L} = (L_1, L_2, \dots L_n), \tag{2.11}$$

де L_i – ключова точка кисті;

n – кількість визначених точок.

Використовуючи (2.4) та (2.11) отриманий набір координат дозволяє описати положення кисті у просторі та виконувати подальший математичний аналіз жестів. На основі координат ключових точок система може обчислювати відстані між пальцями, визначати кути нахилу кисті, аналізувати траєкторію руху руки та швидкість переміщення пальців, а також відстежувати зміну положення руки у часі. Саме аналіз геометричних співвідношень між ключовими точками кисті лежить в основі алгоритму розпізнавання жестів у розробленій системі.

Проте для коректного розпізнавання жестів недостатньо аналізувати лише окреме положення руки в одному кадрі. Жести це динамічні дії, тому система повинна враховувати зміну положення пальців та кисті у часі. Саме

послідовний аналіз декількох кадрів відеопотоку дозволяє відрізнити випадковий рух руки від цілеспрямованого жесту користувача.

Кожен кадр відеопотоку, отриманий з вебкамери, необхідно проаналізувати окремо, після чого порівняти координати ключових точок руки з координатами, отриманими у попередніх кадрах [33–36]. Такий підхід дозволяє системі не лише визначати поточне положення кисті, але й аналізувати зміну її положення у часі. Завдяки цьому алгоритм може відстежувати переміщення пальців, визначати напрямок руху руки, оцінювати швидкість її переміщення та тривалість виконання певного жесту. Крім того, порівняння координат у послідовності кадрів дозволяє оцінювати стабільність положення кисті та зменшувати вплив випадкових рухів або похибок розпізнавання.

Аналіз часової послідовності кадрів важливий для реалізації динамічних жестів. Наприклад, під час керування курсором система повинна безперервно відстежувати рух вказівного пальця та плавно змінювати положення курсора на екрані. Для виконання скролінгу необхідно аналізувати напрямок та величину вертикального переміщення руки протягом певного проміжку часу.

Аналогічно, для емуляції натискання кнопки миші система має визначити момент зближення пальців та перевірити, чи зберігається це положення достатньо стабільно для підтвердження жесту. Одним із найбільш інтуїтивно зрозумілих для користувача жестів для реалізації кліку є зближення кінчика великого пальця та кінчика вказівного пальця, що дозволяє природним чином імітувати дію фізичного натискання.

З математичної точки зору визначення даного жесту базується на аналізі просторового розташування ключових точок кисті. Після розпізнавання руки модель MediaPipe повертає координати всіх landmarks, серед яких:

– точка №4 відповідає кінчику великого пальця;

– точка №8 відповідає кінчику вказівного пальця.

Для визначення факту зближення пальців використовується обчислення евклідової відстані між цими двома точками. Евклідова відстань є

стандартною метричною характеристикою у двовимірному просторі та дозволяє визначити найкоротшу відстань між двома координатами.

$$d = \sqrt{(x_8 - x_4)^2 + (y_8 - y_4)^2}, \tag{2.12}$$

де x_8 та y_8 – координати кінчика вказівного пальця;

x_4 та y_4 – координати кінчика великого пальця;

d – евклідова відстань між двома пальцями.

У процесі роботи системи координати ключових точок руки мають оновлюватися для кожного нового кадру відеопотоку, який надходить з вебкамери у режимі реального часу. Оскільки камера безперервно формує послідовність кадрів, система постійно отримує нові значення координат кінчика великого та вказівного пальців і виконує повторне обчислення відстані між ними. Тобто, аналіз жесту виконується не одноразово, а безперервно протягом усього часу роботи програми.

Для кожного кадру визначається поточне значення евклідової відстані між двома пальцями. Отримане значення має порівнюватися із заздалегідь встановленим порогом – мінімально допустимою відстанню, при якій система вважатиме, що пальці достатньо наблизилися один до одного. Якщо обчислена відстань стає меншою за цей поріг, алгоритм інтерпретує таке положення пальців як жест натискання кнопки миші та виконує відповідну системну команду кліку.

Використання порогового значення дозволяє уникнути помилкових спрацювань у ситуаціях, коли пальці лише частково наближаються один до одного або випадково перетинаються під час руху руки. Значення порогу можна підбирати експериментально з урахуванням особливостей роботи камери, точності розпізнавання та зручності користування системою.

Вибір саме евклідової відстані як основного критерію визначення жесту пояснюється її математичними властивостями. Дана метрика дозволяє визначати реальну просторову відстань між двома точками незалежно від

напрямку їх переміщення у площині кадру. Це означає, що система однаково коректно працює як при горизонтальному, так і при вертикальному або діагональному зближенні пальців.

Крім того, використання евклідової відстані забезпечує стійкість алгоритму до зміни положення руки у кадрі. Навіть якщо користувач повертає кисть під різними кутами або переміщує руку у просторі, система продовжує коректно оцінювати ступінь зближення пальців. Це важливо для реалізації жестового керування у реальному часі, оскільки положення руки користувача постійно змінюється під час роботи.

Додатковою перевагою такого підходу є його обчислювальна простота. Формула евклідової відстані потребує мінімальної кількості математичних операцій, що дозволяє виконувати обчислення швидко навіть при безперервній обробці великої кількості кадрів. Завдяки цьому система здатна забезпечувати плавне та оперативне реагування на жести користувача без помітних затримок.

2.4 Аналіз вертикального зміщення для скролінгу

Однією з функцій системи, які необхідно розробити є реалізація прокручування сторінок або документів за допомогою жестів руки. Для цього використаєм аналіз вертикального переміщення вказівного пальця у послідовності кадрів відеопотоку. Такий підхід дозволить імітувати роботу колеса прокручування комп’ютерної миші та забезпечує безконтактне керування інтерфейсом операційної системи.

Основою алгоритму є відстеження зміни вертикальної координати пальця у часі. У системі використовується координата кінчика вказівного пальця, оскільки саме цей палець забезпечує найбільш точне та стабільне визначення напрямку руху руки. Для кожного нового кадру з вебкамери

зберігається значення координати y, після чого виконується порівняння поточного положення пальця з його положенням у попередніх кадрах.

Вертикальне зміщення визначається як різниця між початковою та кінцевою координатами пальця за певний проміжок часу:

$$\Delta y = y_{last} - y_{first}, \tag{2.13}$$

де y_{first} – початкове положення вказівного пальця;

y_{last} – поточне положення вказівного пальця;

Δy – величина вертикального зміщення.

Отримане значення вертикального зміщення дозволяє системі визначати напрямок руху руки користувача у просторі кадру. Для цього виконується аналіз різниці між поточним та початковим положенням вказівного пальця по вертикальній осі. Якщо значення вертикального зміщення Δy є додатним, це означає, що координата пальця збільшується, тобто палець переміщується вниз відносно зображення з вебкамери. Якщо ж значення Δy є від'ємним, координата зменшується, а отже рух пальця відбувається вгору.

Такий підхід дозволяє системі інтерпретувати напрямок руху руки як відповідну команду прокручування інтерфейсу. Рух руки вниз відповідатиме прокручуванню сторінки вниз, а рух вгору – прокручуванню вгору. Оскільки аналіз координат виконується послідовно для групи кадрів, система зможе отримувати можливість визначати не лише положення руки, а й характер її руху у часі.

Проте під час реальної роботи користувача положення руки ніколи не залишається абсолютно нерухомим. Навіть при спробі утримувати кисть у фіксованому положенні виникають природні мікрорухи пальців та незначні коливання руки. Крім фізичних особливостей руху людини, на результат розпізнавання можуть впливати шум відеокамери, зміна освітлення, неточності визначення координат алгоритмом або нестабільність відеопотоку.

Усі ці фактори можуть спричиняти незначні зміни координат пальця навіть тоді, коли користувач фактично не виконує жест скролінгу.

Якби система реагувала на будь-яке мінімальне зміщення координат, це призводило б до великої кількості помилкових спрацювань. Сторінка могла б починати прокручуватися навіть при випадковому русі руки або незначному тремтінні пальців, що суттєво погіршувало б зручність користування системою.

Для усунення цієї проблеми у алгоритмі необхідно використати порогове значення вертикального зміщення. Його суть полягає у тому, що команда скролінгу виконується лише тоді, коли величина переміщення пальця перевищує певне мінімальне значення. Математично ця умова записується як:

$$|\Delta y| > T_{scroll}, \tag{2.14}$$

де $|\Delta y|$ – абсолютне значення вертикального зміщення вказівного пальця;

T_{scroll} – встановлений поріг спрацювання скролінгу.

Використання абсолютного значення дозволяє оцінювати лише величину переміщення незалежно від його напрямку. Після цього система зможе окремо визначити, чи був рух спрямований вгору або вниз. Проте на етапі перевірки факту виконання жесту системі важливо визначити не напрямок руху, а саму наявність достатнього переміщення пальця. Це дозволяє оцінювати лише величину зміщення незалежно від того, у якому напрямку рухалася рука. Іншими словами, система спочатку перевіряє, чи був рух достатньо великим для того, щоб вважати його свідомим жестом користувача, а вже після цього окремо аналізує знак значення Δy для визначення напрямку прокручування.

Порогове значення спрацювання скролінгу визначає мінімально допустиму величину вертикального переміщення, при якій система буде виконувати команду прокручування. Правильний вибір порогового значення має критично важливе значення для стабільності роботи системи. Якщо поріг

буде встановлений надто малим, навіть незначні випадкові коливання руки або природні мікрорухи пальців будуть сприйматися як команда прокручування. У такому випадку система стане надмірно чутливою, а інтерфейс буде прокручуватися хаотично та неконтрольовано.

З іншого боку, надто велике значення порогу також негативно впливає на якість взаємодії із системою. У такому випадку користувачу доводиться виконувати значні та різкі рухи рукою для активації скролінгу. Це знижує природність керування, підвищує фізичне навантаження на користувача та погіршує точність роботи системи. Тобто порогове значення виступає своєрідним фільтром, який дозволяє відокремлювати цілеспрямовані жести від випадкових коливань руки. Завдяки цьому система реагує лише на достатньо виражені рухи користувача та ігнорує незначні зміни координат, що виникають під час нормальної роботи.

2.5 Модель роботи системи на основі станів

Для забезпечення коректної роботи системи жестового керування необхідно організувати логіку обробки жестів, перемикання режимів роботи та виконання системних команд. Оскільки система повинна реагувати на різні типи жестів залежно від поточного режиму роботи, її функціонування доцільно описати за допомогою математичної моделі скінченного множини станів.

Скінченна множина станів є математичною моделлю дискретної системи, яка у кожний момент часу перебуває лише в одному з можливих станів та може переходити між станами під впливом певних вхідних сигналів. У контексті системи такими сигналами виступають розпізнані жести користувача, а станами – режими роботи програми та режими керування покажчиком.

Використання моделі скінченного автомата дозволяє формалізувати логіку роботи системи, забезпечити контроль переходів між режимами та уникнути конфліктів між різними командами керування [37–39]. Крім того, такий підхід спрощує реалізацію алгоритму обробки жестів та підвищує стабільність функціонування системи у режимі реального часу. Математично скінченний автомат можна подати у вигляді множини:

$$A = (S, X, Y, \delta, \lambda, s_0), \tag{2.15}$$

де S – множина станів системи;

X – множина вхідних сигналів;

Y – множина вихідних дій;

δ – функція переходів між станами;

λ – функція вихідних дій;

s_0 – початковий стан системи.

У процесі функціонування система постійно отримує вхідні сигнали у вигляді розпізнаних жестів користувача. Однак однаковий жест може мати різне призначення залежно від поточного режиму роботи програми. Саме тому виникає необхідність у використанні механізму станів, який визначає контекст інтерпретації жестів.

Наприклад, у режимі керування комп’ютером певний жест може відповідати збільшенню гучності звуку, тоді як у режимі керування презентацією той самий жест може використовуватися для переходу до наступного слайда. Таким чином, кінцева дія системи визначається не лише самим жестом, але й поточним станом автомата.

Початковим станом системи є режим керування комп’ютером $S_0 = Computer$. У цьому стані система інтерпретує жести як команди керування операційною системою. Зокрема, користувач може виконувати переміщення курсора, прокручування сторінок, зміну рівня гучності або відкриття вебсайтів.

Іншим основним станом є режим керування презентацією $S_1 = Presentation$. У цьому режимі жести використовуються для керування показом презентації, зокрема для переходу між слайдами або запуску режиму демонстрації.

Крім основних режимів роботи система містить додаткові стани, пов’язані з типом керування покажчиком. До них належать:

– режим керування курсором;

– режим скролінгу.

У режимі керування курсором координати вказівного пальця безпосередньо перетворюються у координати курсора миші на екрані. У режимі скролінгу система аналізує вертикальне переміщення пальця та виконує прокручування сторінки.

Перехід між станами автомата здійснюється на основі аналізу розпізнаних жестів користувача та тривалості їх утримання. У системі використовуються два основні типи переходів:

– миттєві переходи після короткого виконання жесту;

– переходи після тривалого утримання жесту.

Наприклад, короткочасне виконання жесту перемикання режими використовується для переходу між режимами керування комп’ютером та презентацією. Якщо ж той самий жест утримується протягом більш тривалого часу, система переходить у режим налаштування жестів. Перехід між станами описушемо функцією переходів скінченного автомата:

$$\delta : S \times X \to S. \tag{2.16}$$

Ця функція визначає, у який стан повинна перейти система після отримання певного вхідного сигналу. Наприклад, перехід між режимами роботи можна описати $\delta(Computer, Gesture_{switch}) = Presentation.$ Це означає, що при виконанні жесту перемикання у режимі керування комп’ютером система переходить у режим керування презентацією.

Аналогічно зворотний перехід описується як $\delta(Presentation, Gesture_{switch}) = Computer$.

Важливою складовою логіки роботи системи жестового керування є використання таймерів утримання жестів та часових порогів. Їх застосування дозволяє системі розрізняти короткочасні та тривалі жести, а також запобігати помилковим спрацюванням, які можуть виникати через випадкові рухи руки або нестабільність розпізнавання.

Під час роботи системи кожен розпізнаний жест необхідно аналізувати не лише за його типом, але й за тривалістю виконання у часі. Для цього у момент першого виявлення певного жесту система має фіксувати початковий час його появи. Якщо жест продовжує розпізнаватися у наступних кадрах відеопотоку, система має обчислювати тривалість його утримання.

Для цього алгоритм аналізує часовий інтервал між моментом початку виконання жесту та поточним часом. Якщо тривалість утримання перевищує встановлений поріг, активується відповідний перехід стану. Наприклад, перехід у режим налаштування жестів виконується лише після утримання жесту протягом кількох секунд. Такий механізм дозволяє уникнути випадкового переходу між режимами через короткочасні або помилкові рухи руки.

Подібний принцип використовується і для завершення роботи програми. Якщо користувач утримує спеціальний жест виходу протягом заданого часу, система переходить у кінцевий стан завершення роботи. Подамо час утримання жесту як:

$$t_{hold} = t_{current} - t_{start}, \tag{2.17}$$

де t_{start}– момент початку виконання жесту*;*

$t_{current}$ – поточний момент часу;

t_{hold} – тривалість утримання жесту.

Отримане значення порівнюється із заздалегідь встановленими часовими порогами, які визначають умови активації певних команд або переходів між режимами роботи системи. Наприклад, коротке виконання жесту може використовуватися для перемикання між режимами керування, тоді як тривале утримання того самого жесту активує режим налаштування жестів або завершення роботи програми.

3 РЕЗУЛЬТАТИ ПРОГРАМНОГО МОДЕЛЮВАННЯ ІНТЕРАКТИВНОЇ СИСТЕМИ

3.1 Обґрунтування вибору програмного середовища

Для розроблення інтерактивної системи керування комп’ютером та презентацією за допомогою жестів було обрано мову програмування Python. Даний вибір обумовлений низкою переваг цієї мови у сфері комп’ютерного зору, штучного інтелекту та швидкої розробки програмного забезпечення. Важливою перевагою Python є велика кількість готових бібліотек для роботи зі комп’ютерним зором та автоматизацією взаємодії з операційною системою.

У межах даної роботи були використані такі бібліотеки:

– бібліотека OpenCV використовується для роботи з відеопотоком із вебкамери та обробки зображень;

– бібліотека MediaPipe застосовується для розпізнавання жестів руки та відстеження ключових точок кисті;

– бібліотека PyAutoGUI забезпечує емуляцію дій користувача, зокрема керування курсором, гучністю та клавіатурою;

– бібліотека Tkinter використовується для створення графічного інтерфейсу програми.

Усі необхідні бібліотеки встановлювалися у віртуальному середовищі Python 3.10. Такий підхід запобігає виникненню конфліктів між залежностями та забезпечує коректну роботу програми на різних пристроях. Поєднання OpenCV для обробки відеопотоку, MediaPipe для розпізнавання жестів, PyAutoGUI для системної автоматизації та JSON для збереження налаштування жестів дозволяє створити зручну та стабільну систему керування комп’ютером за допомогою жестів у реальному часі.

3.2 Етапи розроблення інтерактивної системи

Для забезпечення стабільної роботи та запобіганню конфліктам версій між бібліотеками необхідно підготувати робоче середовище. А саме створити віртуальне середовище Python (рис. 3.1), яке ізолює всі необхідні бібліотеки від інших проєктів. Скориставшись стандартним модулем Python *venv* створено окрему папку, в яку додамо власний інтерпретатор Python та всі необхідні для роботи бібліотеки (рис. 3.2).

```
PS D:\Computer_control> python -m venv venv
```

Рисунок 3.1 – Створення віртуального середовища

```
PS D:\Computer_control> .\venv\Scripts\activate
```

Рисунок 3.2 – Активація віртуального середовища

Далі необхідно встановити необхідні бібліотеки. Пакети та версії збережено у файлі requarements.txt.

Лістинг 3.1 Конфігурація файлу requarements.txt:

opencv-python== 4.12.0.88 # Обробка відеопотоку та комп'ютерний зір
mediapipe==0.10.9 # Розпізнавання жестів рук
PyAutoGUI ==0.9.54 # Емуляція системних команд миші та клавіатури

Для впорядкування програмних компонентів була розроблена така структура каталогів проєкту (рис. 3.3). Файли ініціалізації пакетів *__init__.py* вказують інтерпретатору, що директорія є повноцінним пакетом модулів, головний модуль отримує доступ до функцій напряму через ім'я пакета (рис. 3.4).

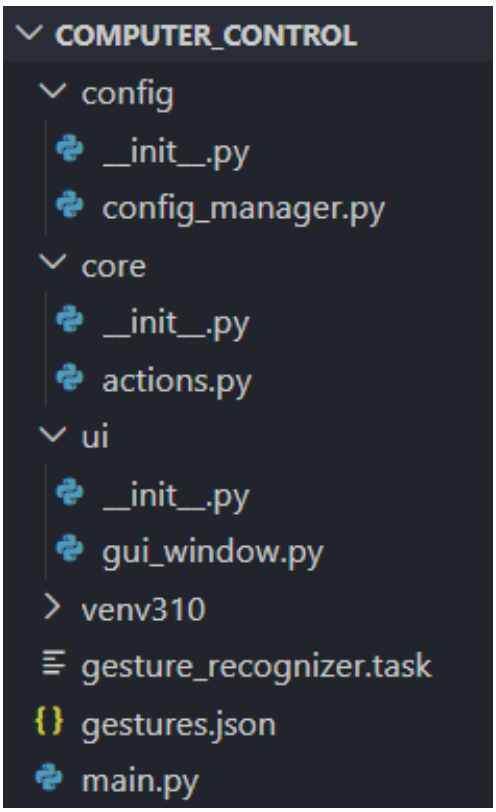

Рисунок 3.3 – Структура файлів проєкту

```
from config import load_config, save_config, DEFAULT_CONFIG
from core import execute_action
from ui import GestureGUI
```

Рисунок 3.4 – Підключення створених модулів

Модуль config_manager.py призначений для роботи з конфігураційними даними системи. Він забезпечує зчитування та збереження користувацьких налаштувань у файлі gestures.json, що дозволяє змінювати параметри роботи програми без внесення змін до програмного коду.

Модуль actions.py реалізує виконання системних команд операційної системи. Саме через нього здійснюється керування курсором миші, емуляція натискань клавіш клавіатури, регулювання гучності та виконання інших дій відповідно до розпізнаних жестів.

Модуль gui_window.py відповідає за графічний інтерфейс користувача. У вікні відображаються поточний режим роботи системи, розпізнаний жест та

виконана дія. Також через інтерфейс користувач може змінювати налаштування керування.

Головним модулем системи є файл main.py, який виконує роль координатора роботи програми. Він здійснює ініціалізацію камери, завантаження нейромережевої моделі розпізнавання жестів, отримання даних від модулів та організацію їх взаємодії.

Для розпізнавання жестів використовується попередньо навчена модель MediaPipe Gesture Recognizer, що зберігається у файлі gesture_recognizer.task. Налаштування користувача зберігаються у файлі gestures.json, який може змінюватися під час роботи програми.

Виконання системи можна поділити на такі кроки:

Крок 1. Запуск програми.

Крок 2. Завантаження конфігурації.

Крок 3. Ініціалізація графічного інтерфейсу.

Крок 4. Ініціалізація системи розпізнавання жестів.

Крок 5. Підключення вебкамери.

Крок 6. Отримання кадру з камери.

Крок 7. Розпізнавання жесту.

Крок 8. Перевірка режиму роботи.

Крок 9. Визначення команди.

Крок 10. Виконання дії.

Крок 11. Оновлення інтерфейсу користувача.

Крок 13. Повторення циклу (кроки 6 –12).

Крок 14. Завершення роботи.

Розглянемо кожен модуль більше детально. Основою модуля config_manager.py є функція load_config(), яка відповідає за завантаження конфігурації під час запуску програми (рис. 3.5). Спочатку виконується перевірка наявності файлу налаштувань. Якщо файл існує, його вміст зчитується та перетворюється у словник Python за допомогою функції json.load(). У випадку виникнення помилки під час читання або пошкодження

файлу автоматично завантажуються стандартні налаштування системи. Аналогічно, якщо файл конфігурації відсутній, програма також використовує налаштування за замовчуванням.

```
CONFIG_FILE = "gestures.json"

def load_config():
    if os.path.exists(CONFIG_FILE):
        try:
            with open(CONFIG_FILE, "r") as f:
                return json.load(f)
        except Exception:
            return dict(DEFAULT_CONFIG)
    return dict(DEFAULT_CONFIG)
```

Рисунок 3.5 – Завантаження конфігурації

Важливою особливістю цього модуля є забезпечення відмовостійкості під час операції введення/виведення під. Для читання конфігурації використовується перевірка помилок (try-except). Якщо файл налаштувань випадково видалено або пошкоджено, програма не завершує роботу з помилкою. У такому випадку автоматично завантажуються стандартні налаштування, які зберігаються в *DEFAULT_CONFIG*. Завдяки цьому система продовжує працювати стабільно навіть у разі проблем із конфігураційним файлом.

Для збереження змінених параметрів використовується функція save_config(config_data) (рис. 3.6). Вона приймає структуру даних із поточними налаштуваннями та записує її у файл gestures.json. Для цього використовується функція json.dump(), яка перетворює словник Python у JSON-представлення. Параметр *indent*=4 використовується для форматування файлу з відступами, що покращує його читабельність та спрощує ручне редагування користувачем за потреби.

```python
def save_config(config_data):
    with open(CONFIG_FILE, "w") as f:
        json.dump(config_data, f, indent=4)
```

Рисунок 3.6 – Збереження змінених параметрів

Модуль actions.py відповідає за виконання системних дій операційної системи відповідно до розпізнаних жестів (рис. 3.7). Його основною функцією є перетворення логічної команди (наприклад, «Next Slide» або «Volume Up») у конкретну дію на рівні операційної системи. Для реалізації взаємодії з операційною системою використовується бібліотека pyautogui, яка дозволяє емулювати натискання клавіш клавіатури, рух та клік миші. Для відкриття вебресурсів використовується стандартний модуль Python webbrowser.

```python
def execute_action(action_name, config_data):
    if action_name == "Next Slide":
        pyautogui.press('right')
    elif action_name == "Previous Slide":
        pyautogui.press('left')
    elif action_name == "Start Presentation":
        pyautogui.press('f5')
    elif action_name == "Open Site":
        webbrowser.open(config_data["website"]["default_url"])
    elif action_name == "Volume Up":
        pyautogui.press('volumeup')
    elif action_name == "Volume Down":
        pyautogui.press('volumedown')
    elif action_name == "Click":
        pyautogui.click()
```

Рисунок 3.7 – Функція виконання системних дій

Емуляція у режимі керування презентаціями (Режим «Presentation») адаптована під стандартні гарячі клавіші популярних офісних пакетів (Microsoft PowerPoint, Google Slides та інших):

– подія Next Slide та Previous Slide викликає надсилання віртуальних скан-кодів стрілок клавіатури right та left, що забезпечує кроспрограмне перегортання слайдів;

– подія Start Presentation здійснює емуляцію натискання клавіші F5, яка запускає режим показу презентації.

У режимі керування комп’ютером система дозволяє виконувати основні дії з операційною системою:

– регулювання звуку (Volume Up / Volume Down) передає системні команди керування звуковим мікшером (volumeup/volumedown), які перехоплюються безпосередньо ядром ОС, забезпечуючи лінійну зміну гучності незалежно від активного вікна;

– натискання миші (Click) надсилає натискання лівої кнопки миші (ЛКМ) у поточних координатах курсора, які розраховуються у головному циклі;

– відкриття сайту (Open Site) відкриває вебсайт у браузері. Адреса береться з налаштувань користувача, тому її можна змінювати через інтерфейс програми.

Модуль gui_window.py відповідає за реалізацію графічного інтерфейсу користувача системи керування жестами. Він створює вікно програми, у якому відображається поточний стан роботи системи та доступні елементи керування. Побудований з використанням бібліотеки tkinter, яка дозволяє створювати прості вікна та елементи управління.

Інтерфейс винесено в окремий клас GestureGUI, щоб розділити графічну частину та основну логіку програми. Це спрощує дотримання принципів чистої архітектури. Інтерфейс не виконує обробку жестів і не приймає системних рішень – він лише відображає інформацію та передає команди далі. Щоб система могла запускати потрібні дії, але не містити їх реалізацію середині себе скористаємося патерном Dependency Injection (впровадження залежностей). Взаємодія з основною програмою здійснюється через функції зворотного виклику (callbacks) (рис.3.8).

```
self.on_start_calibration = on_start_calibration
self.on_reset = on_reset
self.on_save_url = on_save_url
```

Рисунок 3.8 – Реєстрація функцій зворотного виклику

Сам інтерфейс містить (рис. 3.9):

– текстові поля для відображення інформації про систему;

– поле для введення URL-адреси сайту;

– кнопки для керування системою.

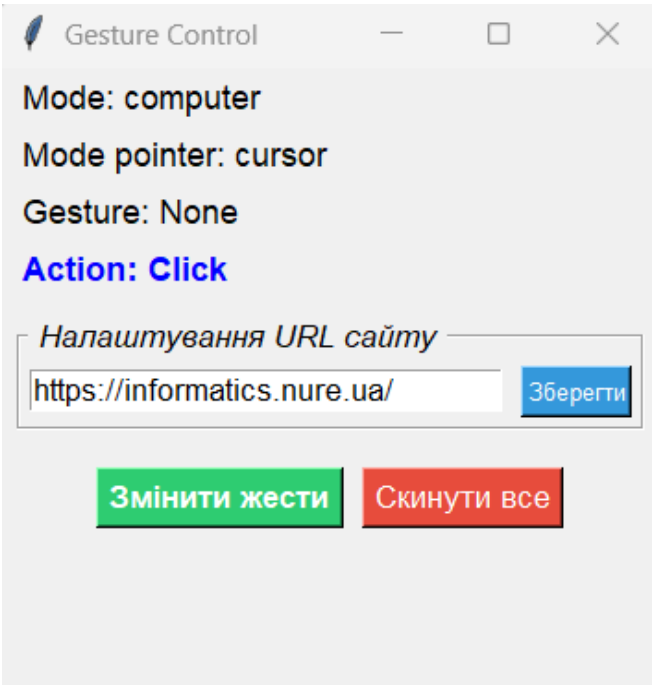

Рисунок 3.9 – Інтерфейс системи

Клас GestureGUI (рис. 3.10) містить кілька основних методів, які відповідають за створення та оновлення графічного інтерфейсу користувача.

```
class GestureGUI:
    def __init__(self, current_url, on_start_calibration, on_reset, on_save_url):
        self.root = tk.Tk()
        self.root.title("Gesture Control")
        self.root.attributes("-topmost", True)
        self.root.geometry("360x340+1000+50")

        self.on_start_calibration = on_start_calibration
        self.on_reset = on_reset
        self.on_save_url = on_save_url

        self.create_widgets(current_url)
```

Рисунок 3.10 – Конструктор класу GestureGUI

Для фіксації вікна програми поверх усіх інших вікон операційної системи (включаючи браузери та презентаційні плеєри) у конструкторі класу активовано прапорець *-topmost*. Це необхідно для UX, оскільки користувач завжди має бачити зворотний зв’язок від програми. Інтерфейс розміщується у правому верхньому куті монітора, ближче до центру. Це найбільш оптимальне розміщення вікна, яке не буде перекривати кнопки закриття програм та основні елементи робочого столу, а також дозволяє користувачу швидко отримувати інформацію про стан системи без відволікання від основної роботи.

Метод create_widgets відповідає за створення всіх елементів графічного інтерфейсу. У ньому формуються текстові поля для відображення стану системи (режим роботи, жест, виконана дія), а також елементи керування:

– поле для введення URL-адреси;

– кнопка збереження URL;

– кнопка запуску калібрування жестів;

– кнопка скидання налаштувань.

Метод update_labels використовується для оновлення інформації на екрані під час роботи системи. Він змінює текст у відповідних полях інтерфейсу, таких як режим роботи, поточний жест та виконувана дія. Викликається у головному циклі програми для відображення актуального стану системи в реальному часі.

Також у цьому методі використовується асинхронний рендеринг без блокування головного циклу . Зазвичай у Tkinter графічний інтерфейс працює через mainloop(), який повністю керує потоком виконання програми. Однак основний потік системи зайнятий безперервним циклом обробки відеопотоку з камери (while True). Тому використання mainloop() заблокує роботу алгоритму розпізнавання жестів.

Тому для оновлення інтерфейсу застосовано метод self.root.update(), який дозволяє оновлювати інтерфейс без зупинки основного циклу програми. Таким чином, графічне вікно оновлюється паралельно з обробкою

відеопотоку, що забезпечує плавну роботу системи та відсутність зависань інтерфейсу.

Лістинг 3.2 Реалізація методу оновлення інтерфейсу:

```
def update_labels(self, mode, pointer, gesture, action):
    self.label_mode.config(text=f"Mode: {mode}")
    self.label_mode_mouse.config(text=f"Mode pointer: {pointer}")
    self.label_gesture.config(text=f"Gesture: {gesture}")
    self.label_action.config(text=f"Action: {action}")
    self.root.update()
```

Метод update_colors відповідає за зміну кольорової теми інтерфейсу. Залежно від режиму роботи системи змінюється фон вікна та колір тексту. Це дозволяє візуально відрізняти різні режими роботи (наприклад, режим курсора та режим скролінгу) і підвищує зручність користування системою.

У модулі main.py об’єднуються всі ключові елементи системи: обробка відеопотоку з камери, розпізнавання жестів, виконання системних команд, а також оновлення графічного інтерфейсу користувача. Модуль відповідає за запуск і завершення програми: завантаження конфігураційних файлів на початку роботи та коректне закриття всіх ресурсів (камери, вікон інтерфейсу) після завершення. Також у цьому модулі організовано постійний потік даних. Програма безперервно отримує кадри з камери через OpenCV і передає їх у систему розпізнавання жестів MediaPipe для обробки в режимі реального часу. І main.py керує логікою роботи програми. Залежно від поточного стану система може переходити між різними режимами: калібрування жестів, керування курсором миші або режимом скролінгу. Таким чином реалізується перемикання між основними функціями програми.

Перед запуском головного циклу програми в модулі main.py виконується етап початкової ініціалізації. На цьому етапі встановлюються

основні змінні та налаштування (рис. 3.11), які визначають стартовий стан системи і дозволяють коректно розпочати її роботу.

```
config = load_config()
mode = "computer"
mode_pointer = "cursor"
current_gesture = "None"
current_action = "None"
```

Рисунок 3.11 – Ініціалізація початкових станів у файлі main.py

Процес перевизначення користувацьких команд реалізовано за допомогою функції зворотного виклику cb_start_calibration() (рис. 3.12). Головне завдання цієї функції – безпечне переведення системи зі стандартного режиму керування в режим калібрування, повне очищення поточних динамічних налаштувань жестів та тимчасове блокування елементів керування графічної форми для запобігання виникненню конфліктів інтерфейсу.

```
def cb_start_calibration():
    global is_calibrating, calibration_step, calibration_start_time, current_action
    is_calibrating = True
    calibration_step = 0
    calibration_start_time = None
    current_action = "Початок калібрування"
    # Скидаємо всі жести, окрім базових та режимів
    for m in ["computer", "presentation"]:
        for key in list(config[m].keys()):
            if key not in ["switch_mode"]:
                config[m][key] = "None"

    # Блокуємо кнопки під час калібрування
    gui.btn_calib.config(state=tk.DISABLED)
    gui.btn_reset.config(state=tk.DISABLED)
    gui.btn_save_url.config(state=tk.DISABLED)
```

Рисунок 3.12 – Функція початку калібрування

Спочатку встановлюються змінні стану калібрування: активується режим калібрування, скидається номер поточного кроку та час початку, а

також оновлюється текст поточної дії. Далі виконується очищення попередніх налаштувань жестів. Для цього система проходить по всіх режимах роботи (computer та presentation) і скидає призначені жести до значення "None". Винятком є лише команда перемикання режиму (switch_mode), яка не змінюється. Таким чином, перед початком нового налаштування всі попередні призначення жестів видаляються, щоб уникнути конфліктів під час повторного навчання. Після цього кнопки інтерфейсу блокуються, щоб користувач не міг змінювати налаштування або запускати інші дії під час процесу калібрування.

Функція cb_reset_config() (рис. 3.13) відповідає за скидання налаштувань системи до початкового стану. У першу чергу відбувається відновлення стандартної конфігурації: поточні налаштування замінюються на значення за замовчуванням (DEFAULT_CONFIG), після чого ці дані зберігаються у файл конфігурації. Далі оновлюється графічний інтерфейс користувача. Поле введення URL очищується і заповнюється стандартною адресою сайту, яка зберігається у конфігурації. У результаті виконання функції система повністю повертається до початкових налаштувань, а користувач отримує відповідне повідомлення про виконану дію.

```
def cb_reset_config():
    global config, current_action
    config = dict(DEFAULT_CONFIG)
    save_config(config)
    gui.entry_url.delete(0, 'end')
    gui.entry_url.insert(0, config["website"]["default_url"])
    current_action = "Скинуто до початкових"
```

Рисунок 3.13 – Функція скидання налаштувань

Функція cb_save_url() використовується для збереження нової вебадреси, яку вводить користувач у графічному інтерфейсі. Спочатку перевіряється, чи введений рядок не є порожнім. Якщо URL коректний, він записується у конфігураційний файл та оновлюється в основних

налаштуваннях системи. У випадку помилки (порожнього значення) система повідомляє користувача про некоректне введення.

Після цього створюється об'єкт графічного інтерфейсу GestureGUI, якому передаються поточні налаштування та функції зворотного виклику для взаємодії з основною логікою програми. Це дозволяє пов'язати інтерфейс користувача з обробкою жестів і системними налаштуваннями.

Наступним важливим етапом є ініціалізація модулю розпізнавання жестів MediaPipe. Для цього визначається функція result_callback, яка отримує результати роботи нейромережі. У ній обробляється розпізнаний жест, а також координати ключових точок руки (пальців), які використовуються для подальшого керування курсором та жестовими командами. Після цього виконується налаштування самої моделі розпізнавання жестів (рис. 3.14).

```
base_options = python.BaseOptions(model_asset_path='gesture_recognizer.task')
options = vision.GestureRecognizerOptions(
    base_options=base_options,
    running_mode=vision.RunningMode.LIVE_STREAM,
    result_callback=result_callback
)
recognizer = vision.GestureRecognizer.create_from_options(options)
cap = cv2.VideoCapture(0)
```

Рисунок 3.14 – Налаштування моделі розпізнавання жестів

В головному циклі програми, який на кожній ітерації забезпечує захоплення, перетворення та аналіз вхідних даних з вебкамери, відбувається зчитування матриці зображення (рис. 3.15). Також використовується функція cv2.flip(frame,1) віддзеркалює зображення по горизонталі. Це необхідно для того, щоб рухи ліворуч/праворуч на екрані збігалися з реальними рухами рук користувача. Кадр конвертується у формат mp.Image і разом із часовою міткою (timestamp_ms) передається в розпізнавач recognizer.recognize_async(), який працює в паралельному потоці.

```
while True:
    ret, frame = cap.read()
    if not ret:
        continue

    frame = cv2.flip(frame, 1)
    timestamp_ms = int(time.time() * 1000)
    mp_image = mp.Image(image_format=mp.ImageFormat.SRGB, data=frame)
    recognizer.recognize_async(mp_image, timestamp_ms)
```

Рисунок 3.15 – Зчитування зображення

Далі перевіряємо чи активний режим калібрування. Перед збереженням нового жесту формується список заборонених значень forbidden_gestures. До нього автоматично додаються базові жести системи (наприклад, Pointing_Up, Victory, Thumb_Down), які використовуються для службових функцій і не можуть змінюватися. Також у цей список включаються жести, які вже були призначені користувачем на попередніх етапах калібрування. Це дозволяє уникнути ситуації, коли один і той самий жест випадково використовується для кількох різних дій.

Для підтвердження вибору жесту недостатньо його короткочасного розпізнавання. Система вимагає стабільного утримання жесту протягом 3 секунд (*hold_duration* >= 3,0). Якщо користувач змінює жест або прибирає руку, таймер скидається, і відлік починається заново. Це дозволяє зменшити кількість помилкових спрацювань і підвищує точність калібрування.

Після проходження всіх етапів (*calibration_step* досягає кінця списку команд) режим калібрування автоматично вимикається (*is_calibrating* = *False*). Нові налаштування зберігаються у файл конфігурації за допомогою функції save_config(config). Після цього інтерфейс користувача повертається до нормального режиму роботи, а всі кнопки знову стають активними.

Далі програма перевіряє, який жест показано, і виконує відповідні функції. У стандартному використанні бібліотека MediaPipe здатна класифікувати лише статичні позиції руки у просторі. Для реалізації динамічних жестів типу «Scroll» (вертикальне прокручування сторінок) у

системі розроблено математичний алгоритм аналізу часових рядів координат. Логіка роботи цієї підсистеми базується на концепції ковзного вікна фіксованої довжини.

У режимі скролінгу система аналізує вертикальне переміщення вказівного пальця та перетворює його на команди прокручування сторінки (рис. 3.16). Спочатку перевіряється, чи доступні координати кінчика вказівного пальця та чи активний режим *scroll*. Якщо ці умови виконуються, координата пальця по осі *Y* додається до списку *history_y*, який зберігає історію останніх положень пальця. Для аналізу руху використовується ковзне вікно з кількох останніх кадрів. Якщо кількість збережених координат перевищує встановлений розмір вікна (*history_len*), найстаріше значення видаляється.

```
if mode_pointer == "scroll" and current_gesture == config["pointer"]["move_cursor"]:
    history_y.append(finger_y8)
    if len(history_y) > history_len: history_y.pop(0)
    if len(history_y) == history_len:
        movement = history_y[-1] - history_y[0]
        if time.time() - last_scroll_time > scroll_delay:
            if movement > swipe_threshold:
                pyautogui.scroll(-int(config["scroll"]["scroll_speed"]))
                current_action = "Scroll Down"
                history_y.clear()
                last_scroll_time = time.time()
            elif movement < -swipe_threshold:
                pyautogui.scroll(int(config["scroll"]["scroll_speed"]))
                current_action = "Scroll Up"
                history_y.clear()
                last_scroll_time = time.time()
```

Рисунок 3.16 – Обробки динамічного жесту скролінгу

Після накопичення достатньої кількості даних обчислюється величина переміщення *movement* згідно з формулою (2.13), порівнюється поточне положення пальця з його положенням на початку інтервалу спостереження. Для запобігання надто частому спрацьовуванню використовується затримка *scroll_delay*. Лише після завершення цього інтервалу система може виконати наступну команду прокручування.

Оскільки під час роботи можуть виникати незначні коливання руки або похибки розпізнавання, використовується порогове значення *swipe_threshold*. Якщо значення *movement* перевищує поріг *swipe_threshold*, вважається, що користувач виконав рух рукою вниз. Якщо ж значення *movement* менше за від'ємний поріг -*swipe_threshold*, система визначає рух руки вгору та виконує прокручування сторінки вгору.

Після виконання команди список координат очищується, а час останнього скролінгу оновлюється. Це дозволяє уникнути багаторазового виконання однієї й тієї ж команди через незначні коливання руки. У результаті система починає накопичувати нові дані про рух руки та очікує наступний жест для прокручування.

Для реалізації жесту натискання лівої кнопки миші використовується визначення відстані між кінчиком вказівного пальця та кінчиком великого пальця (рис. 3.17). Такий спосіб є інтуїтивно зрозумілим для користувача, оскільки імітує звичний рух стискання пальців.

```
if mode_pointer == "cursor" and finger_x4 is not None and finger_y4 is not None:
    distance = math.hypot(finger_x8 - finger_x4, finger_y8 - finger_y4)
    if time.time() - last_click_time > scroll_delay and distance < 0.05:
        execute_action("Click", config)
        current_action = "Click"
        last_click_time = time.time()
```

Рисунок 3.17 – Реалізація жесту кліка

Перед виконанням кліку система перевіряє, чи активний режим керування курсором та чи успішно визначені координати великого і вказівного пальців. Після цього обчислюється відстань між ними згідно з формулою (2.12), яка визначає довжину відрізка між двома точками на площині.

Якщо відстань між пальцями стає меншою за встановлений поріг (*distance* < 0,05), система вважає, що користувач виконав жест стискання пальців, який відповідає натисканню кнопки миші. Для запобігання

багаторазовому спрацюванню кліку використовується часова затримка. Програма перевіряє, чи минув заданий проміжок часу з моменту попереднього натискання. Це дозволяє уникнути ситуації, коли один жест викликає серію небажаних кліків через те, що пальці залишаються зведеними протягом кількох кадрів відеопотоку.

Після успішного розпізнавання жесту викликається функція execute_action("Click", config), яка передає команду до модуля actions.py. У цьому модулі за допомогою бібліотеки PyAutoGUI виконується емуляція натискання лівої кнопки миші. Після цього оновлюється інформація про поточну виконану дію та зберігається час останнього кліку для подальшого контролю повторних спрацювань.

Після обробки жестів система оновлює інтерфейс користувача. У вікні програми відображаються поточний режим роботи, активний режим керування, розпізнаний жест та виконана команда. Також оновлюється зображення з вебкамери, що дозволяє користувачу в режимі реального часу спостерігати за роботою системи та контролювати правильність розпізнавання жестів.

3.3 Тестування та аналіз результатів

Після запуску програми на екрані відображаються два основні вікна: графічний інтерфейс користувача та вікно відеопотоку з вебкамери (рис. 3.18). У вікні камери додатково виводиться інформація про розпізнаний жест у режимі реального часу. Далі виконується перевірка функціональності системи шляхом зміни параметрів конфігурації, зокрема оновлення посилання на вебресурс через інтерфейс користувача. У разі успішного оновлення URL система відображає відповідне повідомлення про коректне збереження змін (рис. 3.19).

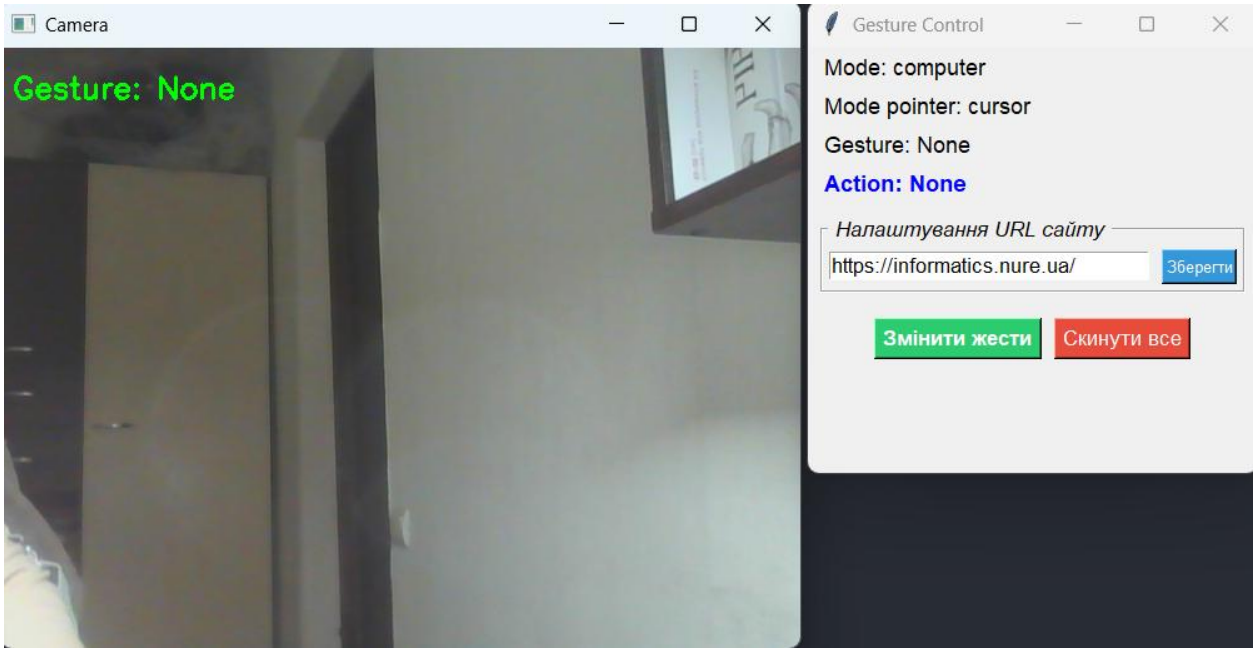

Рисунок 3.18 – Графічний інтерфейс користувача та вікно відеопотоку з вебкамери

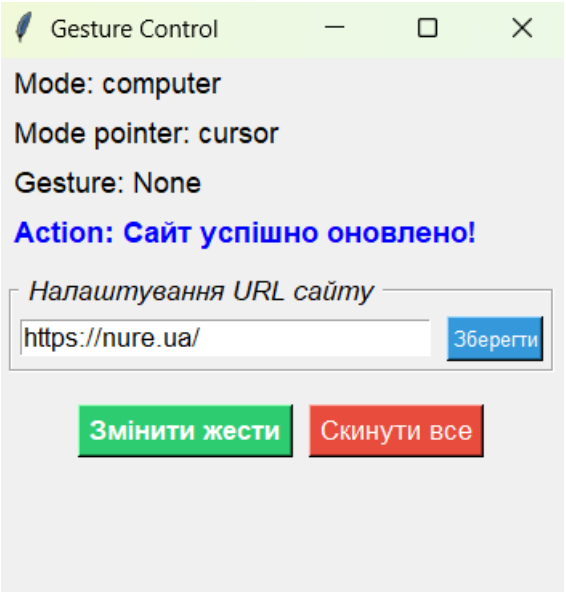

Рисунок 3.19 – Успішне збереження нового посилання

Для перевірки коректності роботи системи було проаналізовано вміст конфігураційного JSON-файлу (рис. 3.20). Після збереження нового вебпосилання відповідне значення у файлі конфігурації успішно оновилося та замінило попередній запис (рис. 3.21).

Рисунок 3.20 – Початковий вміст конфігураційного JSON-файлу

Рисунок 3.21 – Збереження нового вебпосилання

Якщо ж поле для введення залишається порожнім, програма формує повідомлення про помилку та зберігає попереднє значення посилання без змін, забезпечуючи цілісність конфігураційних даних (рис. 3.22).

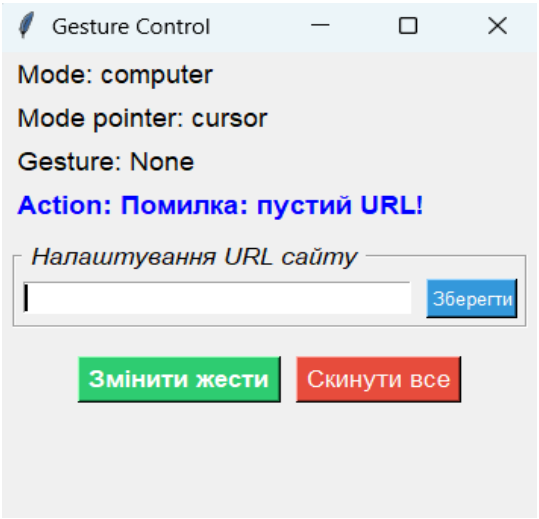

Рисунок 3.22 – Обробка некоректного (порожнього) значення URL

Натискання кнопки «Змінити жест» ініціює перехід системи у режим калібрування. У цей момент у вікні відеопотоку з вебкамери відображається повідомлення «CALIBRATION MODE», яке інформує користувача про активний стан налаштування (рис. 3.23). Одночасно в графічному інтерфейсі користувача виводиться інструкція із зазначенням необхідної дії – показати відповідний жест для призначеної функції керування.

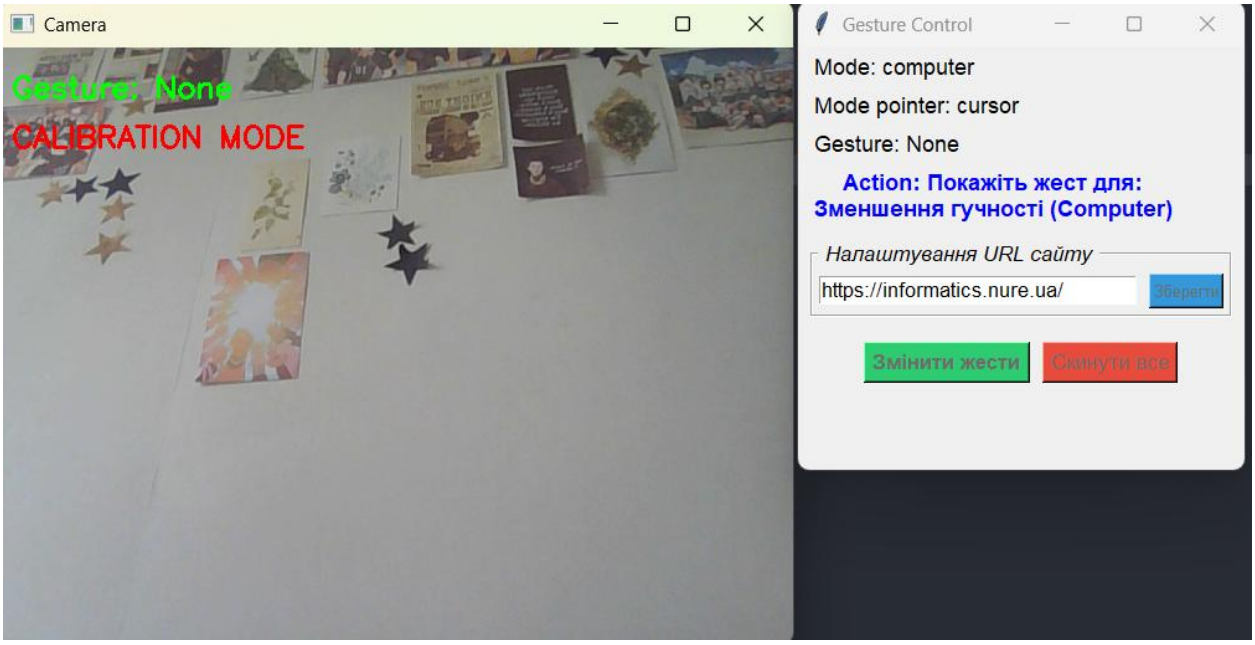

Рисунок 3.23 – Перехід системи у режим калібрування

Під час розпізнавання жесту системою в графічному інтерфейсі користувача відображається назва виявленого жесту, а також запускається таймер зворотного відліку, який показує час, необхідний для підтвердження цього жесту та його фіксації як команди керування (рис. 3.24).

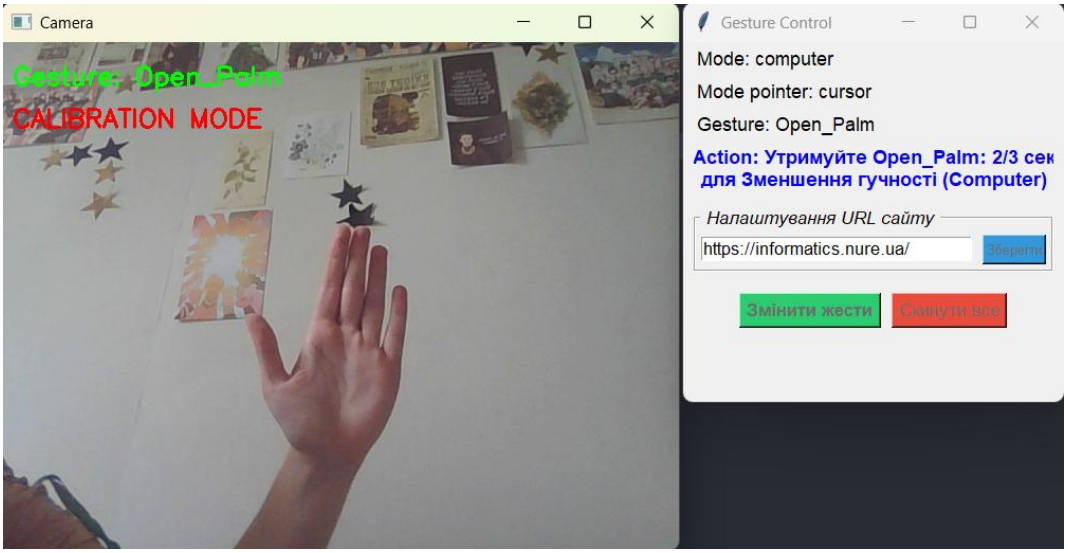

Рисунок 3.24 – Процес підтвердження жесту

У випадку, якщо розпізнаний жест вже використовується для іншої команди або належить до базових системних жестів, програма відображає відповідне повідомлення про неможливість його використання (рис. 3.25).

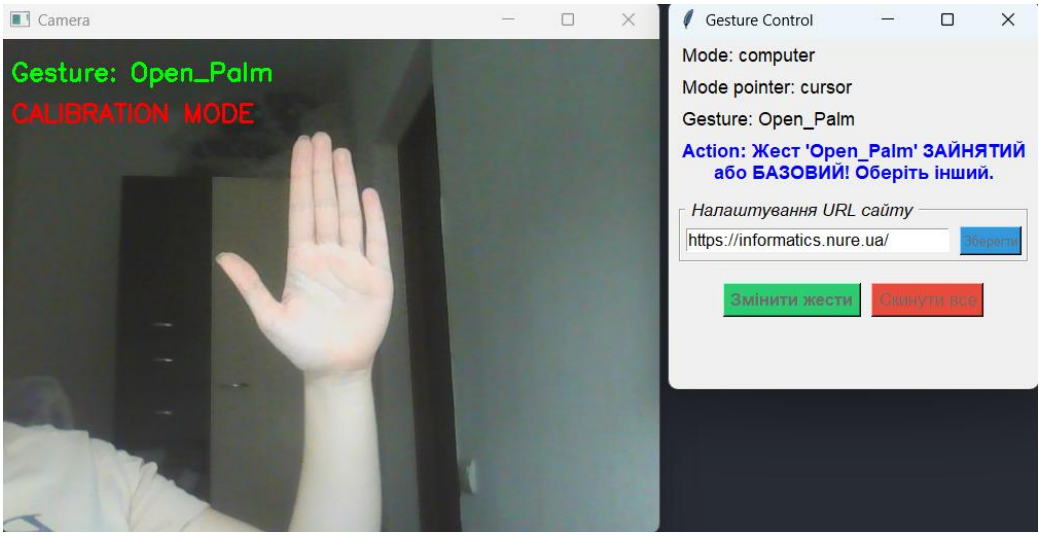

Рисунок 3.25 – Повідомлення про зайнятий або недопустимий жест під час калібрування

Після успішного завершення процесу налаштування жестів система відображає повідомлення про успішне збереження змін. Для перевірки коректності виконаних оновлень розглянемо оновлений файл конфігурації JSON, у якому зафіксовано нові призначення жестів (рис. 3.26).

Рисунок 3.26 – Оновлений вміст конфігураційного JSON-файлу

За потреби, після натискання кнопки «Скинути все» всі налаштування жестів та параметр вебпосилання повертаються до значень за замовчуванням (рис. 3.27), що забезпечує відновлення початкової конфігурації системи.

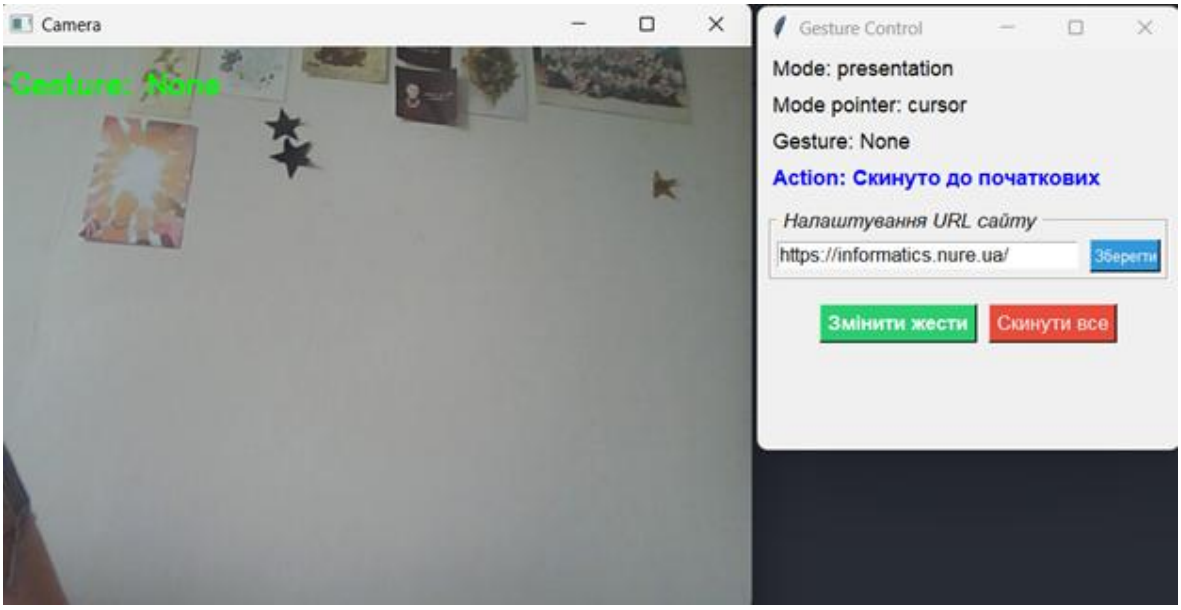

Рисунок 3.27 – Повернення до значень за замовчуванням жестів та вебпосилання

Перевіримо роботу жестів, що відповідають за регулювання гучності, зокрема «відкрита долоня» (Open_Palm) для збільшення гучності (рис. 3.28) та «кулак» (Closed_Fist) для зменшення гучності (рис. 3.29), для оцінки коректності їх розпізнавання та виконання системою.

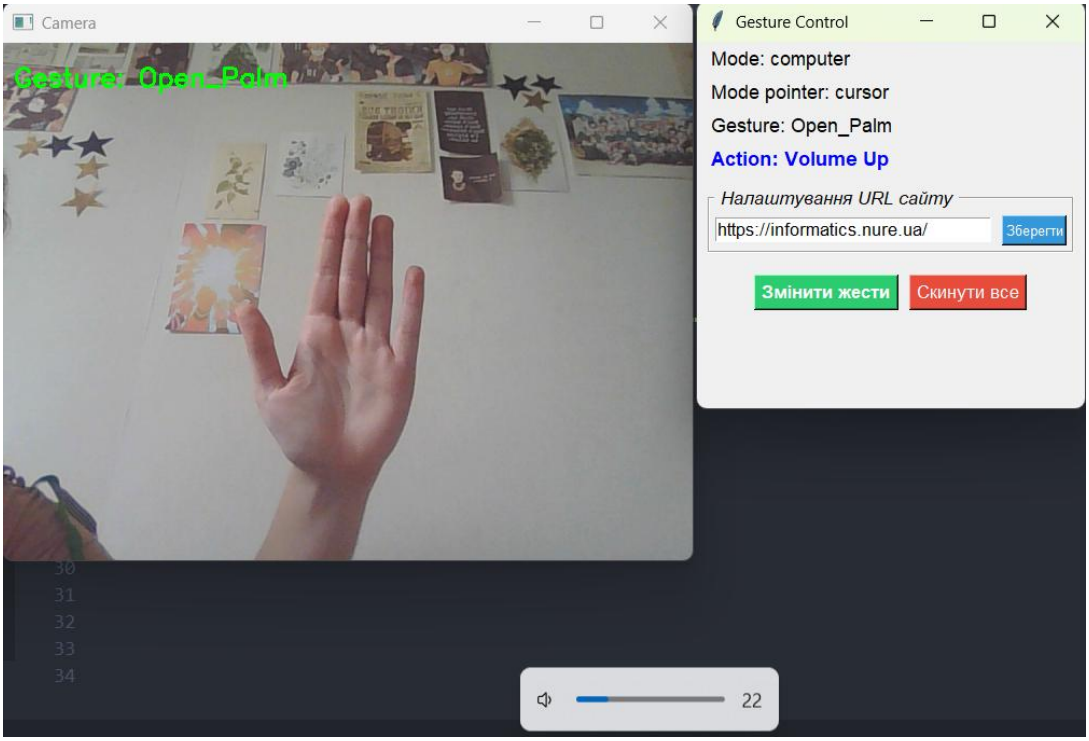

Рисунок 3.28 – Збільшення звуку

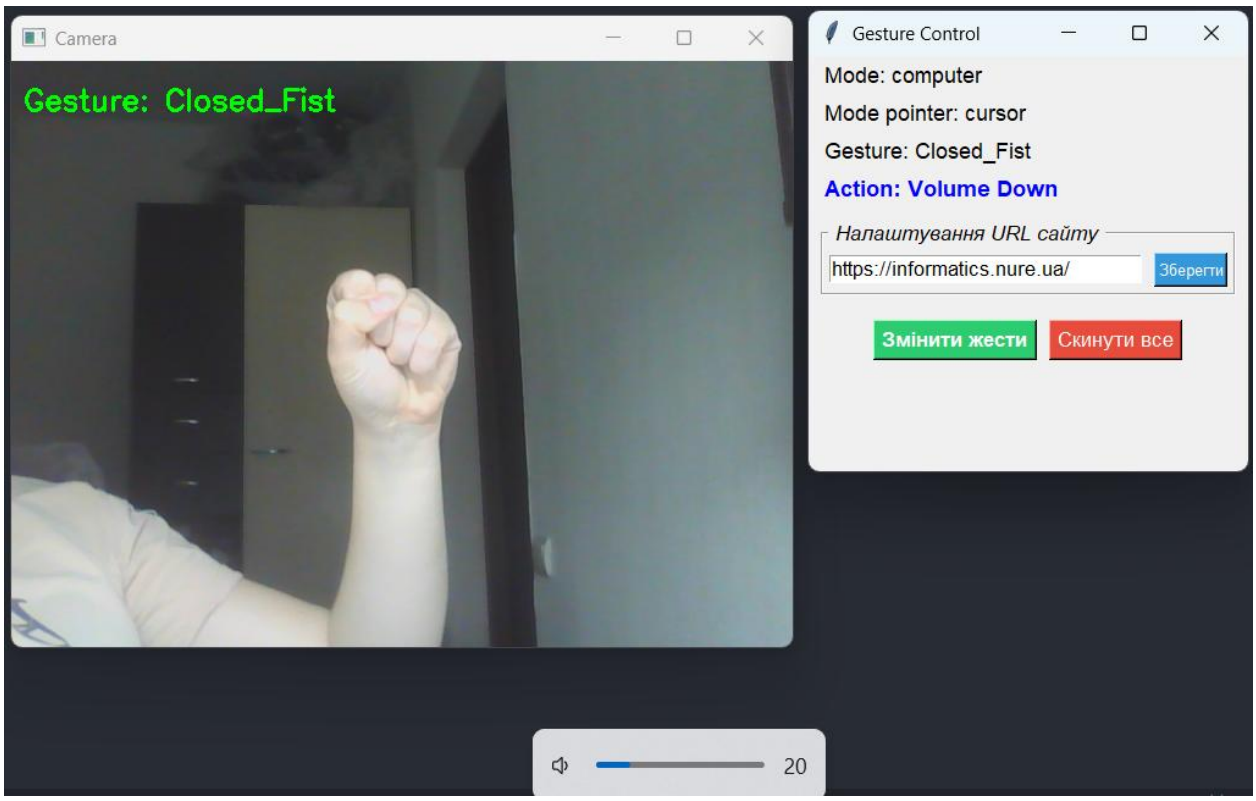

Рисунок 3.29 – Зменшення звуку

Для перемикання режиму з курсора на скрол необхідно виконати жест «палець вниз» (Thumb_Up) та утримувати його приблизно одну секунду, після чого припинити його виконання. У разі успішного розпізнавання та підтвердження жесту відбувається зміна режиму роботи системи (рис. 3.30). Перемикання реалізовано на основі логіки станів, що дозволяє уникнути випадкових спрацювань та забезпечує стабільність керування. При переході в режим скролінгу інтерфейс змінює колір на синій, що візуально показує активний стан. Після повернення до режиму керування курсором колір інтерфейсу знову стає білим, що сигналізує про відновлення стандартного режиму роботи.

Для переходу в режим презентації необхідно виконати жест «перемога» (Victory) (рис. 3.31). Після його розпізнавання система перемикається у відповідний режим роботи, призначений для керування презентацією. У цьому режимі користувач отримує доступ до функцій, пов’язаних із керуванням

мультимедійним контентом, зокрема зміною гучності та відкриттям URL-посилання.

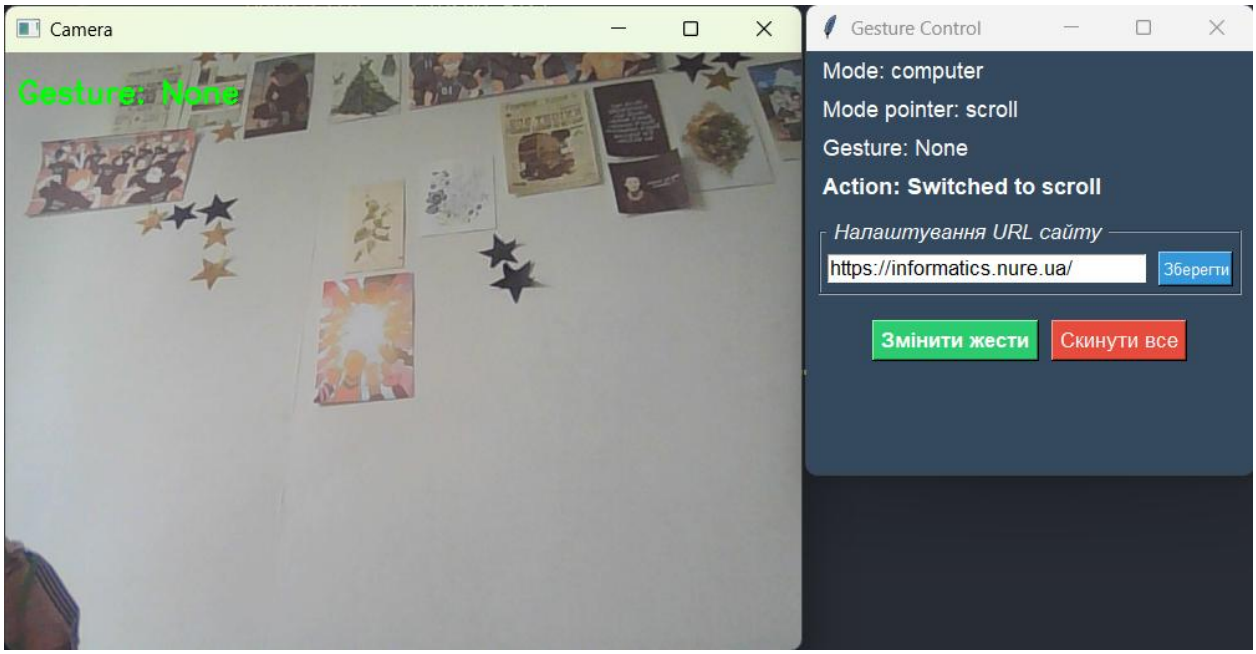

Рисунок 3.30 – Перемикання режиму з курсора на скрол

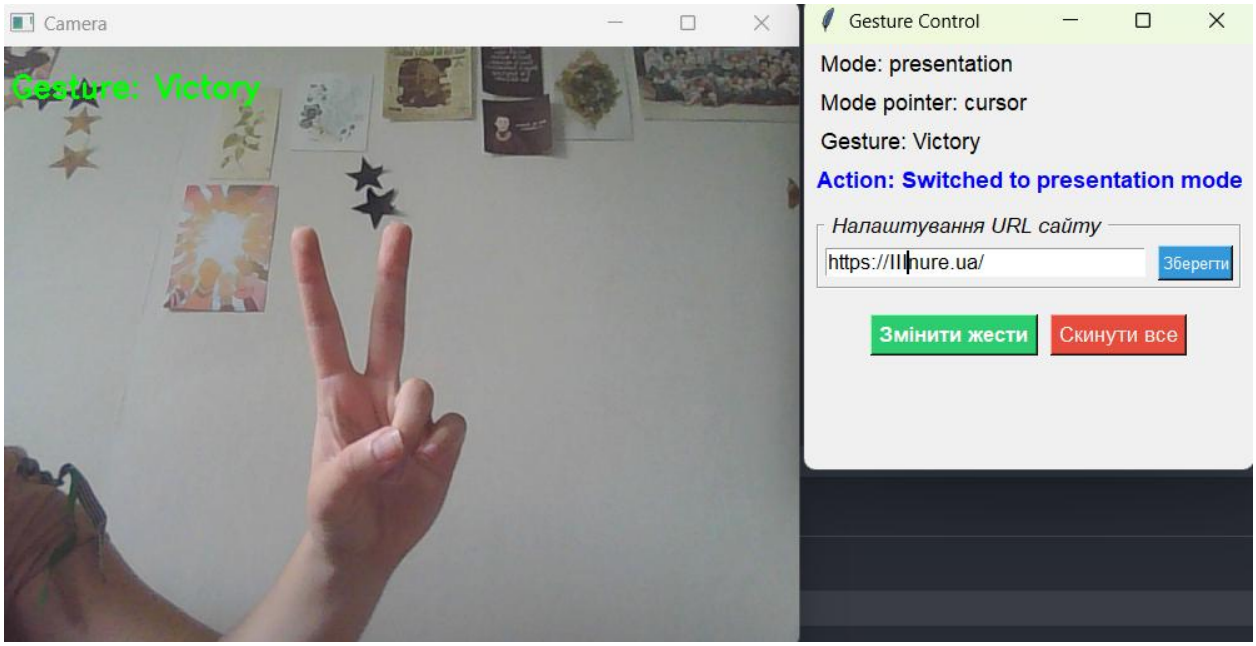

Рисунок 3.31 – Перемикання в режим презентації

Для переведення презентації у повноекранний режим використовується жест «палець вгору» (Pointing_Up) (рис. 3.32). Для гортання матеріалу вперед

застосовується жест «відкрита долоня» (Open_Palm) призначений для переходу до наступного слайду (рис. 3.33).

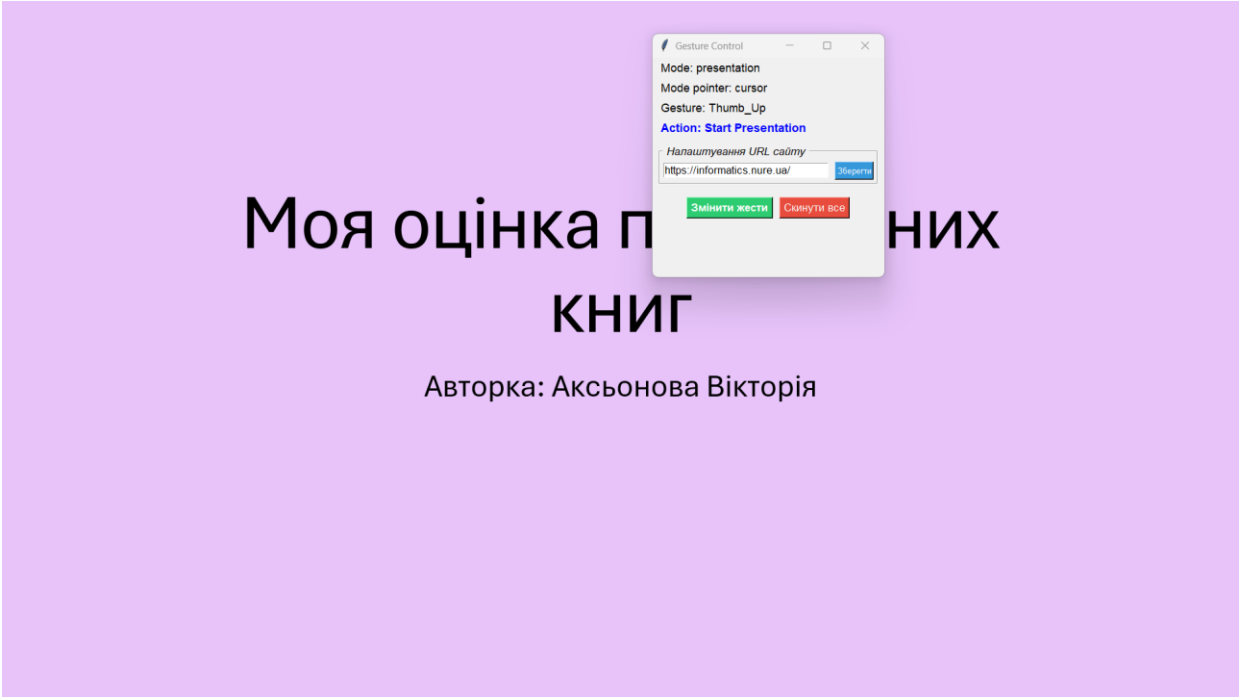

Рисунок 3.32 – Запуск презентації

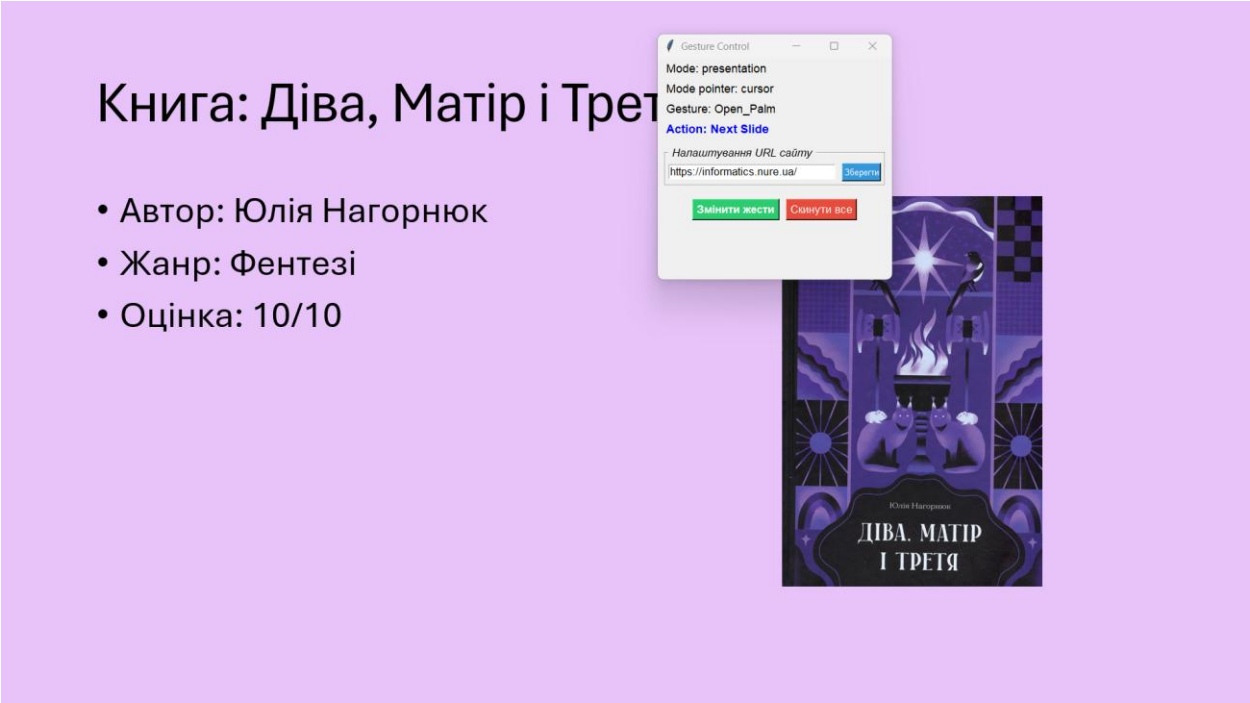

Рисунок 3.33 – Перемикання на наступний слайд

3.4 Перспективи подальшої роботи

Розроблена інтерактивна системи керування презентацією або комп’ютером за допомогою жестів у повному обсязі виконує поставлені завдання та забезпечує базову функціональність взаємодії користувача з операційною системою. Проте у процесі її розробки та тестування були визначені можливі напрями подальшого вдосконалення, які дозволять підвищити зручність, стабільність та функціональність програмного рішення.

Одним із основних напрямів розвитку є збільшення кількості підтримуваних жестів, а також розширення набору функцій, які можуть бути виконані за їх допомогою. Це дозволить реалізувати більш гнучку систему керування та адаптувати її під різні сценарії використання, включаючи роботу з мультимедійними програмами, офісними застосунками та іншими інтерактивними середовищами.

Ще одним перспективним напрямом є реалізація можливості вибору джерела відеопотоку, зокрема камери, з якою працює система. У поточній реалізації використовується стандартна камера пристрою, однак у реальних умовах можуть виникати ситуації, коли доступ до неї обмежений або вона зайнята іншими застосунками. Наприклад, під час змішаних онлайн-офлайн зустрічей або відеоконференцій камера може використовуватися одночасно декількома програмами, що призводить до конфліктів доступу.

Одним із можливих рішень цієї проблеми є підтримка декількох джерел відеопотоку або інтеграція із зовнішніми віртуальними камерами, подібними до тих, що використовуються у стрімінгових системах. Однак більш зручним та практичним підходом може бути використання додаткових пристроїв, зокрема мобільного телефону як зовнішньої камери, що є доступним і поширеним рішенням.

Також перспективою удосконалення системи є впровадження додаткових типів взаємодії, зокрема жесту подвійного кліку, який дозволить

реалізувати більш складні сценарії керування, наприклад відкриття файлів, запуск програм або підтвердження дій без використання клавіатури.

Ще одним важливим покращенням є впровадження перевірки коректності введеного вебпосилання. Зокрема, доцільно реалізувати автоматичну валідацію URL-адреси, яка перевірятиме, чи починається посилання з протоколу https://. У випадку відсутності коректного формату система може автоматично додавати необхідний префікс або виводити повідомлення про помилку. Це підвищить надійність роботи модуля відкриття вебресурсів та зменшить кількість помилок користувача.

Окремим напрямом розвитку є кросплатформна та мобільна інтеграція системи. Це дозволить використовувати систему не лише на персональних комп’ютерах, але й на мобільних пристроях або в гібридних середовищах, що значно розширить сферу її застосування.

Також важливим є подальше вдосконалення внутрішньої логіки програми, зокрема оптимізація алгоритмів обробки жестів, підвищення швидкодії та зменшення затримок при виконанні команд. Це дозволить забезпечити більш стабільну роботу системи в режимі реального часу.

Окрему увагу слід приділити покращенню користувацького інтерфейсу. Його модернізація може включати більш інформативне відображення стану системи, спрощення процесу налаштування жестів та підвищення загальної зручності використання програми.

ВИСНОВКИ

У рамках кваліфікаційної роботи розроблено систему для керування комп’ютером та презентаціями за допомогою жестів руки в режимі реального часу. Створена система забезпечує безконтактну взаємодію користувача з операційною системою за допомогою вебкамери та алгоритмів комп’ютерного зору.

У процесі виконання роботи було реалізовано всі поставлені завдання, зокрема: виконано аналіз існуючих підходів до побудови систем розпізнавання жестів; розроблено модуль обробки відеопотоку з вебкамери та розпізнавання положення руки; реалізовано керування презентацією за допомогою статичних жестів (перехід між слайдами, запуск демонстрації); створено функціонал керування курсором миші на основі координат вказівного пальця; реалізовано механізм скролінгу сторінок за допомогою динамічних жестів; впроваджено функцію емуляції кліку шляхом визначення зближення великого та вказівного пальців; розроблено інформаційне графічне інтерфейсне вікно для відображення поточного режиму роботи системи, розпізнаного жесту та виконаної дії; забезпечено перемикання між режимами роботи системи (режим керування комп’ютером та режим презентації); а також реалізовано можливість зміни налаштувань жестів і вебпосилання через конфігураційний файл та інтерфейс користувача.

У роботі було проаналізовано сучасні підходи до розпізнавання руки та жестів, зокрема методи, засновані на використанні моделей машинного навчання.

Практична значущість роботи полягає у створенні системи безконтактного керування комп’ютером, яка може бути використана для керування презентаціями, мультимедійними застосунками або інтерактивними системами.

Перспективи подальшого розвитку роботи можуть полягати у розширенні набору підтримуваних жестів, використанні нейронних мереж для класифікації складних динамічних жестів, підвищенні точності розпізнавання у складних умовах освітлення, а також адаптації системи для роботи з мобільними пристроями та системами доповненої реальності.

Результати роботи апробовано у вигляді тез доповіді під час Міжнародного молодіжного форуму «РАДІОЕЛЕКТРОНІКА І МОЛОДЬ У XXI СТОЛІТТІ» [40].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Wigdor, D., & Wixon, D. (2011). *Brave NUI world: Designing natural user interfaces for touch and gesture*. Morgan Kaufmann / Elsevier Science & Technology Books.

2. Wong, C. Y., Ng, G. W., & Ibrahim, A. A. A. (2022, March). Human motion recognition based on kinect sensor and leap motion controller. In *Proceedings of the 8th International Conference on Computational Science and Technology: ICCST 2021, Labuan, Malaysia, 28–29 August* (pp. 27-37). Singapore: Springer Singapore.

3. Pterneas, V. (2022). *Mastering the Microsoft Kinect: Body tracking, object detection, and the Azure cloud services*. Apress.

4. Dhabliya, D., Ugli, I. S. M., Murali, M. J., Abbas, A. H., & Gulbahor, U. (2023). Computer vision: Advances in image and video analysis. *E3S Web of Conferences*, *399*, Article 04045.

5. Yang, J. (2023, October). Real time object tracking using OpenCV. In *2023 IEEE 3rd International Conference on Data Science and Computer Application (ICDSCA)* (pp. 1472-1475). IEEE.

6. Гороховатський В., Творошенко І. (2026) Продуктивні моделі аналізу даних у методах розпізнавання зображень: монографія. Харків: ХНУРЕ. 153 с.

7. Kamble, T. U., & Mahajan, S. P. (2024). 3d vision using multiple structured light-based kinect depth cameras. *International Journal of Image and Graphics*, *24*(01), 2450001.

8. Nacho Martín. After 20 years I can finally build this interface How it works: human (me) moves fingers->webcam records fingers->AI (Google's. *X*. URL: https://x.com/nacmartin/status/1635224813894782976?s=20 (дата звернення: 08.05.2026).

9. Ünsalan, C., Höke, B., & Atmaca, E. (2024). The TensorFlow Platform and Keras API. In *Embedded Machine Learning with Microcontrollers: Applications on STM32 Development Boards* (pp. 215-228). Cham: Springer International Publishing.

10. Tvoroshenko I., Pomazan V., Gorokhovatskyi V., and Kobylin O. (2023) Application of video data classification models using convolutional neural networks, *International Journal of Academic and Applied Research*, 7(11), pp. 134-145.

11. Chollet, F. (2021). *Deep learning with Python* (2nd ed.). Manning Publications.

12. Bećirović, M., Kurtović, A., Smajlović, N., Kapo, M., & Akagić, A. (2025, September). Performance comparison of medical image classification systems using TensorFlow Keras, PyTorch, and JAX. In *International Conference on Medical and Biological Engineering* (pp. 137-146). Cham: Springer Nature Switzerland.

13. Liu, M. (2024). *Learn generative AI with PyTorch*. Simon and Schuster.

14. Sánchez-Brizuela, G., Cisnal, A., de la Fuente-Lopez, E., Fraile, J. C., & Perez-Turiel, J. (2023). Lightweight real-time hand segmentation leveraging MediaPipe landmark detection. *Virtual Reality*, *27*(4), 3125-3132.

15. Anand, A., Pandey, A., Mehndiratta, S., Goyal, H., Kaushik, P., & Rathore, R. (2024, May). Smart AI basedVolume Control System: Gesture Recognition with OpenCV & Mediapipe Integration. In *2023 International Conference on Smart Devices (ICSD)* (pp. 1-6). IEEE.

16. Mahadevkar, S. V., Khemani, B., Patil, S., Kotecha, K., Vora, D. R., Abraham, A., & Gabralla, L. A. (2022). A review on machine learning styles in computer vision – techniques and future directions. *Ieee Access*, *10*, 107293-107329.

17. Gorokhovatskyi, O., Peredrii, O., Gorokhovatskyi, V., Vlasenko, N. (2023) Explanation of CNN Image Classifiers with Hiding Parts. In: J. Benois-Pineau, R. Bourqui, D. Petkovic, G. Quenot (eds), Explainable Deep Learning Artificial Intelligence, pp. 125-146, Academic Press, 346 p.

18. Pomazan V., Tvoroshenko I., and Gorokhovatskyi V. (2023) Handwritten character recognition models based on convolutional neural networks, *International Journal of Academic Engineering Research*, 7(9), pp. 64-72.

19. Tvoroshenko I., Gorokhovatskyi V., Kobylin O., and Tvoroshenko A. (2023) Application of deep learning methods for recognizing and classifying culinary dishes in images, *International Journal of Academic and Applied Research*, 7(9), pp. 57-70.

20. Корякіна С., Творошенко І.С. (2024) Особливості методу SIAMESE NETWORKS щодо його застосування до задачі класифікації об'єктів на зображеннях, *Abstracts of XI International Scientific and Practical Conference «Modern generation: current problems, experience, development prospects», (November 12 – 15, 2024). Seville, Spain*, pp. 386-390.

21. Gorokhovatskyi, V. A. (2003). Recognition of images in conditions of incomplete information. KNURE, Kharkov.

22. Гороховатський В.О., Гадецька С.В., Стяглик Н.І. (2019) Вивчення статистичних властивостей моделі блочного подання для множини дескрипторів ключових точок зображень. Радіоелектроніка, інформатика, управління, №2, 100–107.

23. Гороховатський В.О., Творошенко І.С. (2025) Оцінювання значущості ознак для підвищення продуктивності структурних методів класифікації зображень. *Проблеми інформатики та моделювання (ПІМ-2025). Тези двадцять п`ятої міжнародної науково-технічної конференції (25 – 28 вересня 2025 року)*. *Харків: НТУ «ХПІ»*, С. 38-43.

24. Gorokhovatskyi V., Tvoroshenko I. (2024) An effective method for transforming an image description into a compact vector for classification. *Information Technology and Implementation (Satellite): Conference Proceedings, November 21, 2024, Kyiv, Ukraine / Ministry of Education and Science of Ukraine, Taras Shevchenko National University of Kyiv and [etc]; Vitaliy Snytyuk (Editor). – Kyiv: Publishing House «Caravela»*, pp. 25-28.

25. Li, Y. (2022, January). Research and application of deep learning in image recognition. In *2022 IEEE 2nd international conference on power, electronics and computer applications (ICPECA)* (pp. 994-999). IEEE.

26. Gorokhovatskyi V., Tvoroshenko I., Yakovleva O., and Hudáková M. (2026) Feature space quantization and application of metrics in structural methods of image recognition, *IEEE Access*, vol. 14, pp. 17812-17824.

27. Gorokhovatskyi V., Chmutov Y., Tvoroshenko I., and Kobylin O. (2025) Reducing computational costs by compressing the structural description in image classification methods, *Advanced Information Systems*, vol. 9, no. 1, pp. 5–12.

28. Cao, W., Lu, P., & Cao, W. (2024). Multimodal gesture recognition with spatio-temporal features fusion based on YOLOv5 and MediaPipe. *International Journal of Pattern Recognition and Artificial Intelligence*, *38*(08), 2455007.

29. Chandwani, L., Khilari, J., Gurjar, K., Maragale, P., Sonare, A., Kakade, S., ... & Kulkarni, R. (2023). Gesture based sign language recognition system using Mediapipe.

30. Gorokhovatsky, V. (2014), Structural Analysis and Intellectual Data Processing in Computer Vision, SMIT, Kharkiv.

31. Gorokhovatskyi V., Tvoroshenko I., Yakovleva O., and Hudáková M. (2025) Image description compression in classification structural methods, *IEEE Access*, vol. 13, pp. 43631-43641.

32. Gadetska, S., Gorokhovatskyi, V., & Stiahlyk, N. (2020). Image structural classification technologies based on statistical analysis of descriptions in the form of bit descriptor set. *CEUR Workshop Proceedings*, *2608*, 1027–1039.

33. Gorokhovatskyi, V., Gadetska, S., & Stiahlyk, N. (2023). Accelerating Image Classification based on a Model for Estimating Descriptor-to-Class Distance. International Journal of Computing, 22(4), 485-492.

34. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., and Zeghid M. (2022) Tools for fast metric data search in structural methods for image classification, *IEEE Access*, 10, pp. 124738-124746.

35. Гороховатський В., Передрій О., Творошенко І., Марков Т. (2023) Матриця відстаней для множини компонентів структурного опису як інструмент для створення класифікатора зображень, *Сучасні інформаційні системи*, 7(1), С. 5-13.

36. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., and Zeghid M. (2024) Improving the effectiveness of image classification structural methods by compressing the description according to the information content criterion, *Computers, Materials & Continua*, vol. 80, no. 2, pp. 3085-3106.

37. Гороховатський В.О., Гадецька С.В. (2020) Статистичне оброблення та аналіз даних у структурних методах класифікації зображень (монографія), Харків, ФОП Панов А.Н., 128 с.

38. Гороховатський В.О., Творошенко І.С. (2022) Аналіз багатовимірних даних за описом у формі множини компонент: монографія. Харків: ХНУРЕ. 124 с.

39. Gorokhovatskyi V., Tvoroshenko I., and Yakovleva O. (2024) Transforming image descriptions as a set of descriptors to construct classification features, *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 33, no. 1, pp. 113-125

40. Аксьонова В. Р. (2026) Розпізнавання жестів руки у системах безконтактного керування. Радіоелектроніка і молодь у XXI столітті: тези доповідей 30-го Міжнародного молодіжного форуму (Харків, 22–24 квітня 2026 р.). Харків: ХНУРЕ, 2025. Т.7. С. 22-24.