ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Система керування комп'ютером з використанням нейромережевої моделі розпізнавання жестів»

на здобуття освітнього ступеня бакалавра
зі спеціальності 122 Комп'ютерні науки
освітньо-професійної програми Штучний інтелект

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідні джерела

_____ Ярослав ЯРОШЕНКО
(підпис)

Виконав: здобувач вищої освіти група ШІД-42
Ярослав ЯРОШЕНКО

Керівник: Олександр ЗВЕНІГОРОДСЬКИЙ
к.т.н., доцент

Рецензент:
науковий ступінь, вчене звання _____
Ім'я, ПРІЗВИЩЕ

КИЇВ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут інформаційних технологій

Кафедра Штучного інтелекту
Ступінь вищої освіти Бакалавр
Спеціальність 122 Комп'ютерні науки
Освітньо-професійна програма Штучний інтелект

ЗАТВЕРДЖУЮ
Завідувач кафедрою Штучного інтелекту
_____ Ольга ЗІНЧЕНКО
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Ярошенко Ярославу Костянтиновичу

1. Тема кваліфікаційної роботи: Система керування комп'ютером з використанням нейромережевої моделі розпізнавання жестів ,
керівник кваліфікаційної роботи Звенігородський Олександр к.т.н., доцент,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56
2. Строк подання кваліфікаційної роботи «23» травня 2025р.
3. Вихідні дані до кваліфікаційної роботи: науково-технічна література.
4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

Огляд існуючих систем розпізнавання жестів та їх аналіз.
Вибір підходу до структури системи керування за допомогою жестів.
Аналіз і обґрунтування вибору програмного забезпечення та інструментів розробки.
Реалізація програмної системи.
Тестування точності та стабільності системи в реальних умовах.

5. Перелік графічного матеріалу: *презентація*
6. Дата видачі завдання «25 лютого» 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз технологій HCI та розпізнавання жестів.	25.02-03.03.2025	Виконано
2	Розробка структури системи та вибір інструментів розробки.	04.03-10.03.2025	Виконано
3	Формування датасету.	11.03-17.03.2025	Виконано
4	Навчання нейронної мережі для розпізнавання жестів.	18.03-31.03.2025	Виконано
5	Реалізація функціоналу системи.	01.04-21.04.2025	Виконано
6	Тестування системи в реальних умовах.	22.04-05.05.2025	Виконано
7	Оформлення, редагування та підготовка кваліфікаційної роботи до захисту.	06.05-23.05.2025	Виконано

Здобувач вищої освіти _____ Ярослав ЯРОШЕНКО

Керівник
кваліфікаційної роботи _____ Олександр ЗВЕНІГОРОДСЬКИЙ;к.т.н., доцент
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 55 стор., 44 рис., 5 табл., 17 джерел.

Мета роботи – збільшення ефективності HCI при наборі текстів і взаємодії з графічними інтерфейсами.

Об'єкт дослідження – людино-машинна взаємодія та її практичне застосування.

Предмет дослідження – алгоритми розпізнавання жестів.

Короткий зміст роботи – у роботі проведено теоретичний аналіз моделей людино-машинної взаємодії, а також сучасних апаратних і програмних рішень для розпізнавання жестів. Здійснено порівняння бібліотек MediaPipe Hands, YOLO та OpenPose, за результатами якого для реалізації трекінгу руки обрано MediaPipe Hands. Розроблено власний датасет української жестової абетки, та виконано навчання згорткової нейронної мережі.

В рамках реалізації створено систему, що дозволяє керувати мишею та вводити текст за допомогою жестів у режимі реального часу. Проведено тестування працездатності та точності моделі в реальних умовах.

Для функціонування системи необхідно мати персональний комп'ютер з веб-камерою та операційною системою Windows. Програмне забезпечення реалізовано мовою Python.

КЛЮЧОВІ СЛОВА: РОЗПІЗНАВАННЯ ЖЕСТІВ, ЛЮДИНО-МАШИННА ВЗАЄМОДІЯ, HCI, MEDIAPIPE, CNN, PYTHON, УПРАВЛІННЯ МИШЕЮ, ВВЕДЕННЯ ТЕКСТУ.

ABSTRACT

Text part of the bachelor's qualification work: 55 pages, 44 pictures, 5 table, 20 sources.

The purpose of the work – increase the efficiency of HCI when typing and interacting with graphical interfaces.

Object of research – human-machine interaction and its practical application.

Subject of research – is gesture recognition algorithms.

Summary of the work – the theoretical analysis of human-machine interaction models, as well as modern hardware and software solutions for gesture recognition is carried out. A comparison of the MediaPipe Hands, YOLO and OpenPose libraries was made, based on the results of which MediaPipe Hands was chosen to implement hand tracking. The authors developed their own dataset of the Ukrainian sign language alphabet and trained a convolutional neural network.

As part of the implementation, a system was created that allows you to control the mouse and enter text using gestures in real time. The model's performance and accuracy have been tested in real conditions.

To use the system, you need a personal computer with a webcam and Windows operating system. The software is implemented in Python.

KEYWORDS: GESTURE RECOGNITION, HUMAN-MACHINE INTERACTION, HCI, MEDIAPIPE, CNN, PYTHON, MOUSE CONTROL, TEXT INPUT.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

HCI (Human-Computer Interaction) – людино-машинна взаємодія;

GOMS (Goals, Operators, Methods, Selection rules) – модель цілей, операторів, методів і правил вибору;

CNN (Convolutional Neural Network) – згорткова нейронна мережа;

MSE (Mean Squared Error) – середньоквадратична помилка;

KLM (Keystroke-Level Model) – модель рівня натискань клавіш;

YOLO (You Only Look Once) – алгоритм для швидкого виявлення об'єктів на зображенні;

ReLU (Rectified Linear Unit) – випрямлена лінійна функція активації, яка використовується в нейронних мережах для запровадження нелінійності;

ВСТУП

Актуальність теми**.** У сучасному цифровому світі стрімкий розвиток технологій штучного інтелекту та людино-машинної взаємодії відкриває нові можливості для створення альтернативних інтерфейсів управління. Особливу актуальність набувають безконтактні методи взаємодії, зокрема жестове керування, яке дозволяє здійснювати інтуїтивну та природну комунікацію з цифровими пристроями без необхідності використання традиційних засобів введення.

У контексті зростання потреб у доступних та адаптивних рішеннях, системи керування на основі розпізнавання жестів стають перспективним напрямом розвитку. Поєднання глибинного навчання, згорткових нейронних мереж дозволяє розробляти точні, стабільні та масштабовані системи керування, які можуть бути ефективно впроваджені в різних сферах: від побуту до освіти, від інтерактивних презентацій до управління віртуальним середовищем.

Мета дослідження – підвищення ефективності людино-машинної взаємодії шляхом створення інтуїтивного та зручного способу керування комп'ютером без використання фізичних пристроїв введення.

Об'єкт дослідження – процес людино-машинної взаємодії, що охоплює комунікацію між користувачем і комп'ютерною системою. Особлива увага приділена безконтактним інтерфейсам керування, які забезпечують інтуїтивну та зручну взаємодію.

Предмет дослідження – алгоритми розпізнавання жестів.

Для досягнення поставленої мети необхідно вирішити такі завдання**:**

1. Провести аналіз існуючих систем для розпізнавання жестів
2. Розробити структуру програмної системи.
3. Вибрати програмні засоби для розробки системи.
4. Реалізувати програмну систему керування за допомогою жестів.
5. Провести тестування працездатності та точності системи в реальних умовах.

1 ЛЮДИНО-МАШИННА ВЗАЄМОДІЯ ТА ЇЇ ПРАКТИЧНЕ ЗАСТОСУВАННЯ

1.1 Поняття та типи взаємодії HCI

Людино-машинна взаємодія – це міждисциплінарна галузь, що вивчає способи взаємодії людини з комп'ютером та іншими цифровими пристроями. Його головна мета – розробити інтерфейси, які забезпечують зручний, ефективний, інтуїтивно зрозумілий та безпечний зв'язок між користувачем і системою. HCI складається з двох основних складових. З одного боку, він досліджує взаємодію між людини та технологіями в сучасному світі, а з іншого боку, передбачає використання технологій для розробки втручань, спрямованих на покращення якості життя людей.

HCI поєднує знання інформатики, психології, дизайну, ергономіки та лінгвістики (рис. 1.1). Це стосується не лише зовнішнього інтерфейсу, але й взаємодію в цілому.

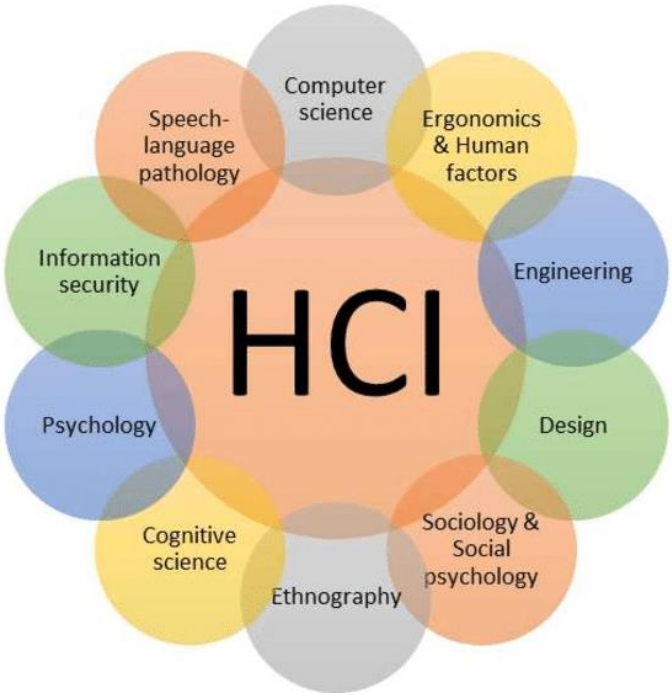


Рис. 1.1 Людино-машинна взаємодія та її галузі досліджень [1]

Тепер розглянемо способи взаємодії людини з комп'ютерними системами:

- Графічні інтерфейси користувача є найбільш поширеною формою взаємодії для більшості користувачів комп'ютерів. Графічні інтерфейси користувача

дозволяють користувачам взаємодіяти зі своїми пристроями за допомогою візуальних елементів.

- Сенсорні інтерфейси – дозволяє користувачам взаємодіяти із системою за допомогою сенсорних жестів, таких як проведення пальцями, торкання та зведення пальців. Прикладом є функція масштабування за допомогою пальців на смартфоні.
- Інтерфейси командного рядка – використовуйте введені команди для взаємодії з системою. Прикладом є вбудований термінальний додаток у більшості операційних систем з текстовим інтерфейсом, де можна вводити команди для взаємодії з системою
- Голосові інтерфейси користувача – дозволяють користувачам взаємодіяти із системою за допомогою голосових команд. Прикладами є Siri та Alexa. Вони особливо корисні для користувачів з обмеженими можливостями мобільності та тих, хто користується вільними руками
- Природні інтерфейси користувача – використовуйте жести та рухи тіла, щоб зробити взаємодію з системою більш природною та інтуїтивно зрозумілою. Прикладами є ігри віртуальної реальності.
- Мультимодальні інтерфейси – поєднують кілька режимів взаємодії, таких як зір, дотик і голос, для забезпечення гнучкішого користувацького досвіду. Прикладом є пошук елемента за допомогою голосової команди та вибір його за допомогою дотику

1.2 Сфери застосування HCI

Людино-машинна взаємодія має широке практичне застосування в різних галузях, де цифрові технології відіграють ключову роль у забезпеченні зручного та ефективного користувацького досвіду. Сучасні інтерфейси дозволяють не лише полегшити доступ до інформаційних систем, але й трансформують способи виконання повсякденних завдань. Розглянемо декілька основних сфер, у яких HCI демонструє найбільший вплив [12]:

- ***Охорона здоров'я***. Сьогодні пацієнти мають безліч можливостей: вони можуть купувати ліки онлайн, записуватись на прийом до лікаря за допомогою

мобільного додатку. Доповнена реальність і віртуальна реальність трансформують хірургічні втручання, що раніше були надзвичайно ризикованими. Тепер лікар може використовувати 3D-анімацію для візуалізації процесу. А також застосування їх у навчання майбутніх хірургів.

- ***Освіта.*** Сучасні технології дозволяють студентам легше засвоювати новий матеріал завдяки інтерактивним технологіям, онлайн-курсам та мультимедійним ресурсам. Заняття в класі стали дуже цікавими за допомогою розумних класів.
- ***Банківська справа***. Інтернет-банкінг і мобільні додатки дозволяють користувачам здійснювати фінансові операції без відвідування банку. Вони забезпечують швидкий доступ до рахунків, оплату послуг, перекази коштів тощо. Крім того, сучасні HCI-рішення впроваджують засоби захисту для зменшення ризиків кіберзлочинності.
- ***Соціальні та професійні мережі***. Взаємодія через цифрові платформи значно спростила процес налагодження комунікації – як у повсякденному, так і в професійному середовищі. Соціальні мережі, платформи для спільної роботи, а також сервіси з пошуку роботи стали невід'ємною частиною життя, що сприяє швидкому обміну інформацією та можливостям працевлаштування.

1.3 Моделі HCI

У процесі розробки систем людино-машинної взаємодії важливо враховувати не лише технічні аспекти, але й поведінку користувача, контекст використання та ефективність виконання завдань. Для цього застосовують моделі HCI, які слугують концептуальними рамками для аналізу, проєктування та оцінювання інтерфейсів. Вони допомагають краще зрозуміти, як саме відбувається взаємодія між людиною та комп’ютером.

1.3.1 Прогностичні та описові моделі

Описові моделі – це концептуальні схеми, які допомагають пояснити, як саме користувачі взаємодіють із системами. Вони не дають точного прогнозу але дозволяють краще зрозуміти структуру дій користувача, прийняття рішень, користувача, прийняття рішень, особливості виконання завдань та їх послідовність. Основна мета таких моделей – виявлення закономірностей,

типових стратегій та шаблонів поведінки, які можна використати при проєктуванні інтерфейсів.

Однією з найвідоміших описових моделей є GOMS (рис. 1.2). Вона розглядає взаємодію користувача як послідовність:

- Goals (цілі) – що саме користувач хоче досягти;
- Operators (оператори) – базові дії (наприклад, натискання клавіші, переміщення миші);
- Methods (методи) – способи досягнення цілей (різні комбінації операторів);
- Selection rules (правила вибору) – як користувач обирає конкретний метод з доступних.

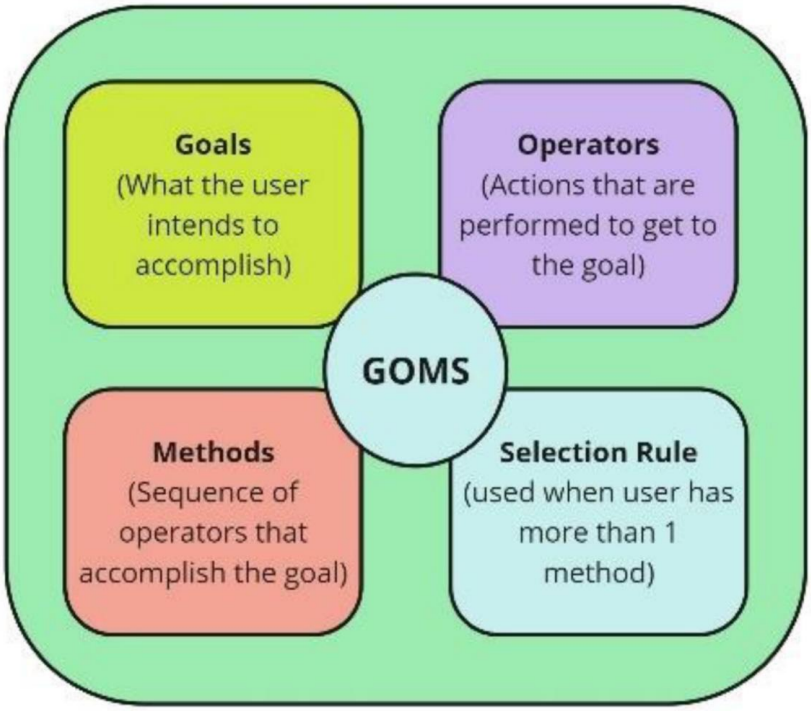


Рис. 1.2 Структура моделі GOMS [2]

Наприклад, якщо користувач хоче зберегти документ, він може це зробити кількома способами: через меню, поєднання клавіш або кнопку на панелі інструментів. GOMS дозволяє описати ці альтернативні сценарії і порівняти їхню ефективність.

Прогностичні моделі , на відміну від описових, мають на меті передбачити певні кількісні показники взаємодії: наприклад, скільки часу користувач витратить на виконання завдання, скільки помилок він зробить або яку продуктивність покаже інтерфейс у порівнянні з іншими. Ці моделі особливо

корисні на етапі проєктування, коли потрібно оцінити інтерфейс ще до його реалізації.

Класичним прикладом прогностичної моделі є закон Фіттса , який дозволяє передбачити час, необхідний для досягнення певної цілі на екрані (наприклад, клік мишкою по кнопці) залежно від її розміру та відстані до неї. Закон Фіттса широко застосовується при оптимізації розміщення елементів в інтерфейсах: кнопок, меню, посилань тощо.

Ще одним прикладом прогностичної моделі є модель KLM – спрощений варіант GOMS, що дозволяє оцінити час виконання завдання на основі конкретних дій користувача (натискань клавіш, рухів миші тощо). KLM активно використовується в UX-дизайні для оцінки ефективності різних варіантів інтерфейсів ще до створення прототипу.

1.3.2 Закон Фіттса

Закон Фіттса є однією з найнадійніших та найширше прийнятих моделей людського руху. Ця модель, можливо, є найуспішнішою з багатьох спроб моделювати людську поведінку як діяльність з обробки інформації. Використовується для моделювання часу, необхідної для навігації до елементів інтерфейсу. Математично закон подається у вигляді формули (1.1):

$$T = a + b * \log_2\left(1 + \frac{D}{W}\right) \quad (1.1)$$

де T – час руху;

D – відстань до об’єкта;

W – ширина об’єкта;

a і b – емпіричні коефіцієнти, що залежать від конкретного користувача та пристрою введення;

З практичної точки зору, закон Фіттса означає, що чим ближча і більша ціль, тим швидше користувач зможе її досягти (рис. 1.3). Це має велике значення при проєктуванні графічного інтерфейсу користувача. Наприклад:

- кнопки, що використовуються часто, мають бути великими та розташованими близько до зони фокусу користувача;

- меню або елементи керування, які важко досягти (далеко розташовані або маленькі), можуть негативно вплинути на продуктивність і задоволення від користування.

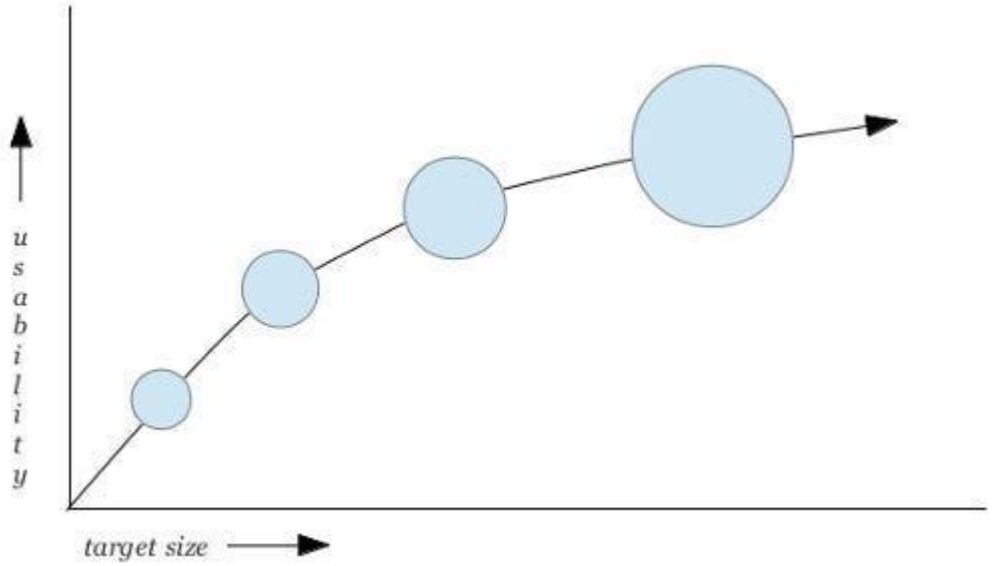


Рис. 1.3 Залежність зручності використання (usability) від розміру цілі (target size) згідно із законом Фіттса [3].

1.3.3 Модель бімануального вводу Гіара

Люди мають дві руки, і використовують їх асиметрично – кожна виконує окрему роль у взаємодії. Цей факт лежить в основі моделі бімануального вводу Гіара, яка досліджує координацію між домінантною та недомінантною рукою.

Основні положення моделі:

- Нерівномірність ролей – домінантна рука виконує точні, локалізовані дії, тоді як недомінантна відповідає за встановлення контексту або загальне позиціонування.
- Послідовна взаємозалежність – дії домінантної руки залежать від попередньої роботи недомінантної. Наприклад, при письмі недомінантна рука тримає зошит або позиціонує аркуш, а домінантна виконує дрібну моторну дію – письмо (рис. 1.4).
- Шкала просторової точності – дії недомінантної руки є грубішими і охоплюють ширший простір, натомість домінантна зосереджується на точності.

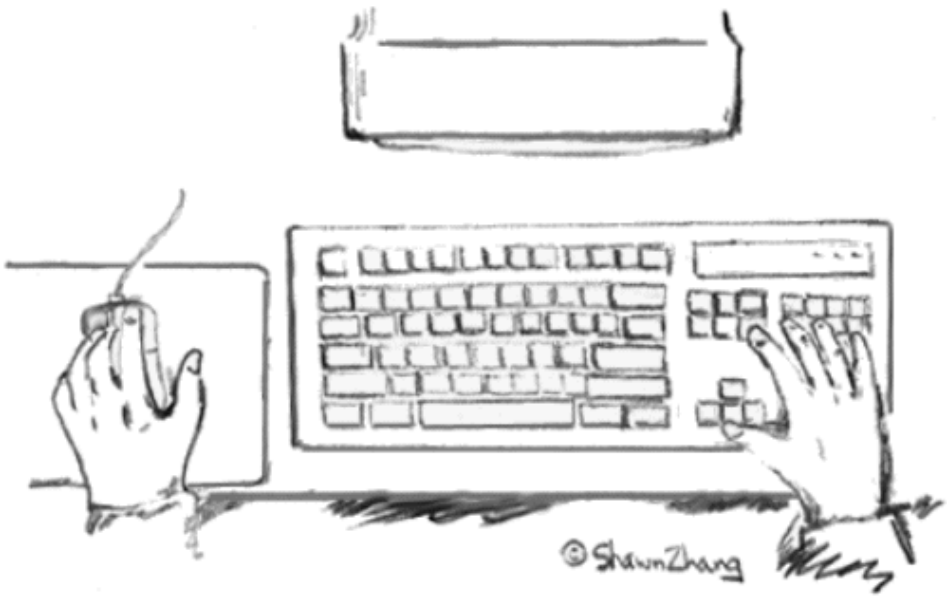


Рис. 1.4 Приклад асиметричного використання рук відповідно до моделі Гіара

Модель доводить, що завдання, які передбачають використання обох рук, можуть бути ефективнішими, якщо проєктувати інтерфейси з урахуванням природного розподілу ролей між руками. Це особливо актуально в розробці інтерфейсів для пристроїв з мультисенсорним управлінням, жестовим введенням або для віртуальної реальності.

1.4 Огляд сучасних HCI-систем та способів взаємодії з комп'ютером

З розвитком технологій з'явилися нові, інноваційні способи комунікації з комп'ютером. На практиці через конкретні апаратно-програмні комплекси, які дозволяють користувачу природно та інтуїтивно взаємодіяти з цифровим середовищем.

Leap Motion Controller 2 – це компактний інфрачервоний сенсор, що відстежує рухи рук і пальців з високою точністю у тривимірному просторі. Пристрій не потребує носимих елементів та ідеально підходить для взаємодії з віртуальними об'єктами в середовищах VR та AR. Завдяки точному оптичному відстеженню пальців, Leap Motion широко використовується в медицині, освітніх платформах та в інтерактивних іграх (рис. 1.5).

Рис. 1.5 Пристрій Leap Motion Controller 2 [4]

Intel RealSense – це серія камер, які поєднують кольорову та глибинну зйомку. RealSense здатен розпізнавати об'єкти, глибину сцени, обличчя, а також жести. Камера RealSense SR300, наприклад, може відстежувати рухи рук, голови, вирази обличчя. Вона має більший діапазон дії, ніж Leap Motion, але нижчу точність в деталізації пальців. Використовується в безпечній автентифікації, робототехніці, UX-дизайні, а також в комп’ютерному зорі (рис. 1.6).

Рис. 1.6 Пристрій Intel RealSense SR300[5]

Azure Kinect DK – це багатофункціональний сенсорний пристрій, розроблений компанією Microsoft як спадкоємець класичного Kinect. Він поєднує в собі камеру глибини, кольорову камеру високої роздільної здатності, масив мікрофонів і інерційний модуль. Azure Kinect призначений не для геймінгу, а насамперед для розробників, які створюють системи комп’ютерного зору, штучного інтелекту та інтерактивні додатки. Завдяки високоточному відстеженню положення тіла, Azure Kinect широко застосовується у медицині, промисловості, робототехніці та проєктах доповненої реальності. Пристрій підтримує

відстеження до 32 суглобів людського тіла, а також має розширену зону дії та високу якість даних глибини (рис. 1.7).

Рис. 1.7 Пристрій Azure Kinect DK для операційної системи Windows [6]

У таблиці 1.1 наведено їх основні характеристики :

Таблиця 1.1

Порівняння параметрів пристроїв відстеження жестів

Параметр	Leap Motion Controller 2	Intel RealSense	Azure Kinect DK
Діапазон дії	25 – 60 см	20 – 150 см	25 – 540 см
Точність	0.01 мм	0.1 – 1 мм	0.5 – 2 мм
Відстеження пальців	Так	Обмежене	Ні

Загалом, кожен з пристроїв – Leap Motion, Intel RealSense та Microsoft Kinect – має свої переваги та обмеження, і вибір залежить від конкретних вимог проєкту. Таким чином, у сучасних HCI-системах важливо обирати інструмент з урахуванням точності, відстані роботи, типу взаємодії (жести, голос, рухи тіла) та галузі використання.

Висновок до розділу 1

У першому розділі було проведено комплексний аналіз основ людино-машинної взаємодії , її класифікації, моделей та реальних прикладних сфер використання. Розглянуто еволюцію інтерфейсів від текстових до мультимодальних, що поєднують голос, жести та зір. Здійснено аналіз таких моделей, як GOMS, закон Фіттса та модель Гіара, які дають змогу якісно проєктувати ефективні інтерфейси взаємодії.

Особливу увагу приділено сучасним апаратним засобам для взаємодії через жести: Leap Motion, Intel RealSense та Azure Kinect. Проведене порівняння цих технологій за ключовими параметрами дозволило визначити їхні можливості та обмеження.

Узагальнення наданої теоретичної бази створило надійний фундамент для подальшої розробки системи, що використовує жести як основний канал взаємодії з комп’ютером.

2 АНАЛІЗ ТЕХНОЛОГІЙ ТА РОЗРОБКА МОДЕЛІ ДЛЯ РОЗПІЗНАВАННЯ ЖЕСТІВ

2.1 Вибір технологій для реалізації системи

Система взаємодії з комп’ютером за допомогою жестів має на меті забезпечити зручне, інтуїтивне керування без використання фізичних пристроїв введення, таких як миша чи клавіатура. Основна мета полягає у створенні альтернативного способу керування комп’ютером, а також для побудови сучасних HCI-систем , які використовують природні жести як інтерфейс.

Розроблена система має виконувати дві ключові функції:

- Керування мишею – забезпечення контролю за положенням курсора, а також реалізація основних дій, таких як клацання (лівим або правим кнопками), прокручування, масштабування, перетягування об’єктів тощо;
- Введення тексту – шляхом розпізнавання статичних жестів, що відповідають літерам української жестової абетки, з подальшим виводом цих символів на екран (рис. 2.1).

Ці функції висувають різні вимоги до технічної реалізації. Для керування мишею важливо досягти високої точності та стабільності у відстеженні рухів руки в реальному часі. Відстеження повинно бути достатньо швидким, щоб забезпечити плавність руху курсора та швидке реагування на жести, що відповідають кліку або іншим діям. У цьому випадку важливими є такі характеристики системи, як низька затримка, точне визначення просторових координат пальців і стійкість до зовнішніх факторів (освітлення, фон, положення тіла тощо).

З іншого боку, для введення тексту система повинна бути здатна точно розпізнавати статичні пози руки, які відповідають літерам. Тут акцент робиться не на динаміці, а на класифікації форми руки. Модель повинна мати здатність до узагальнення, аби впізнавати літери за різних умов зйомки (кут, освітлення, незначні відхилення у жестах). У зв’язку з цим постає потреба в попередньо підготовленому датасеті з мітками та в навчанні згорткової нейронної мережі.

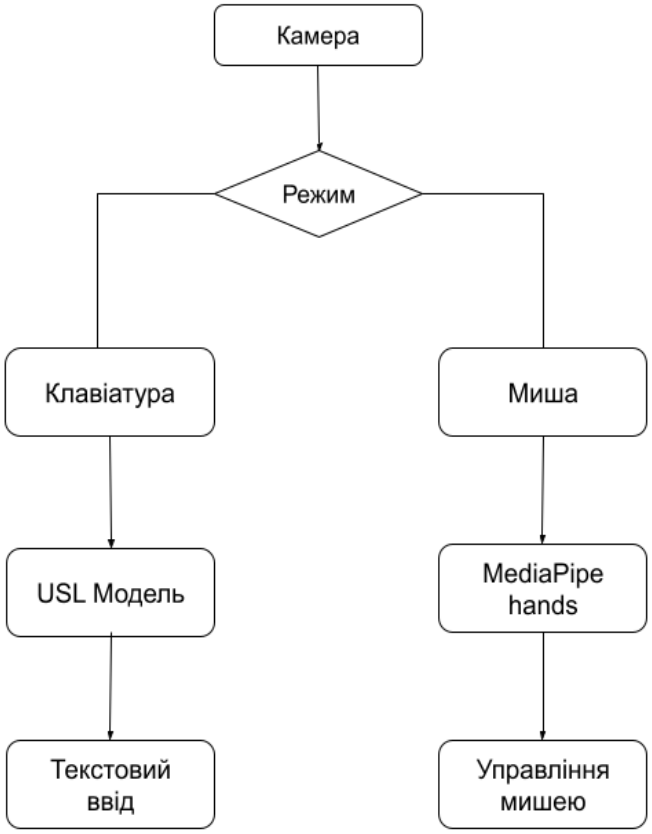


Рис. 2.1 Структурна схема системи керування комп’ютером за допомогою жестів

2.2 Огляд популярних бібліотек та фреймворків для розпізнавання жестів рук

Для реалізації модуля керування мишею необхідно мати точне та стабільне відстеження положення руки в реальному часі. У цьому підрозділі буде розглянуто найпопулярніші бібліотеки та фреймворки, які забезпечують детекцію та трекінг руки на зображенні або відеопотоці.

Серед таких рішень особливу увагу заслуговують фреймворки, які вже довели свою ефективність у задачах комп’ютерного зору та взаємодії з користувачем за допомогою жестів. Нижче буде наведено огляд найпоширеніших з них.

2.2.1 MediaPipe Hands

MediaPipe Hands – це крос-платформна бібліотека комп'ютерного зору від Google, призначена для високоточного виявлення і відстеження рук у режимі реального часу. Вона здатна визначати 21 ключову точку на кожній руці, включно із суглобами пальців і долонею (рис. 2.2).

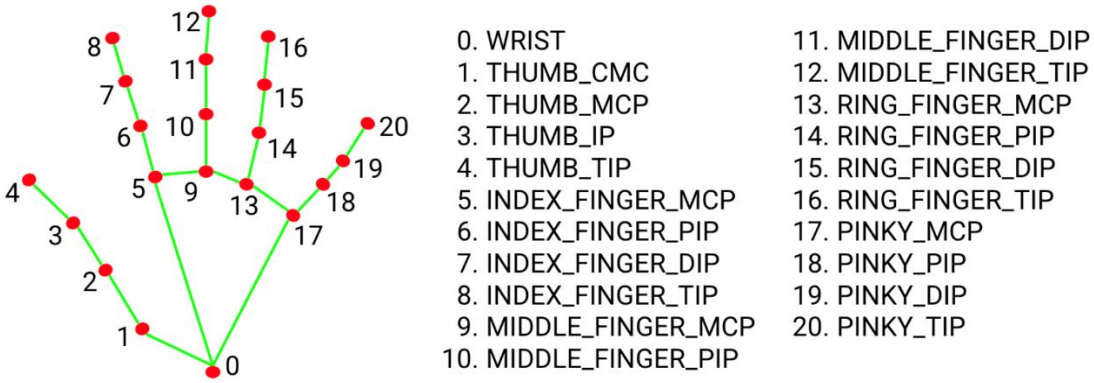


Рис. 2.2 Схема розташування 21 контрольної точки на руці [7].

Переваги:

- Робота в реальному часі на пристроях середньої потужності;
- Вбудована класифікація орієнтації руки;
- Простота інтеграції з Python;
- Висока точність в складних умовах;

Недоліки:

- Обмежений контроль над архітектурою;
- Працює лише з руками (без трекінгу всього тіла);

2.2.2 OpenPose

OpenPose – це система оцінки поз, розроблена дослідниками з Університету Карнегі-Меллона , яка може виявляти і відстежувати людське тіло в реальному часі і точно визначати його позу в тривимірному просторі [8]. Вона добре відома як перша система оцінки поз кількох людей у реальному часі, яка точно виявляє ключові точки людського тіла, рук, обличчя та ніг на окремих зображеннях.

Переваги:

- Висока точність при трекінгу людей;
- Підтримка повного скелету;
- Надійність у складних ситуаціях;

Недоліки:

- Високі апаратні вимоги;
- Складна інтеграція;
- Повільна робота центрального процесора;

2.2.3 YOLO Keypoint Detection

YOLO Keypoint Detection – це модифікація популярного алгоритму виявлення об'єктів YOLO, яка додатково визначає ключові точки на виявлених об'єктах. Цей підхід поєднує в собі переваги швидкого виявлення об'єктів з можливістю точної локалізації характерних точок, як-от суглоби людини, кути автомобіля або інші значущі частини об'єктів.

Переваги:

- Висока швидкість;
- Гнучкість у навчанні під конкретну задачу;
- Можливість суміщення з іншими задачами;

Недоліки:

- Відсутність готових моделей для трекінгу рук;
- Необхідність створення власного датасету;
- Складність реалізації;

2.3 Вибір бібліотеки для розпізнавання рук

Для вибору моделі трекінгу руки було проведено експериментальне порівняння двох популярних підходів: MediaPipe Hands та YOLO. В якості еталонних даних використовувався FreiHAND, який містить координати контрольних точок руки. Порівняння проводилося за наступними метриками:

- Success rate – частка кадрів, на яких модель успішно виявила руку (рис. 2.3).
- Average MSE – середньоквадратичне відхилення між прогнозованими та реальними координатами (рис. 2.4).
- Median MSE – медіанне відхилення, менш чутливе до викидів (рис. 2.5).
- Average processing time – середній час обробки одного кадру (рис. 2.6).

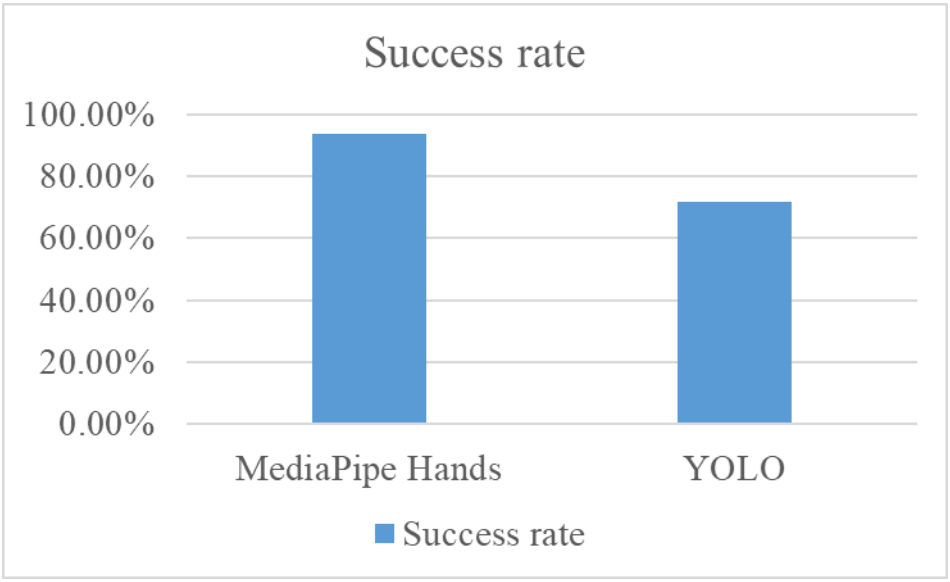


Рис. 2.3 Порівняння точності моделей MediaPipe Hands та YOLO за метрикою за коефіцієнтом успішності.

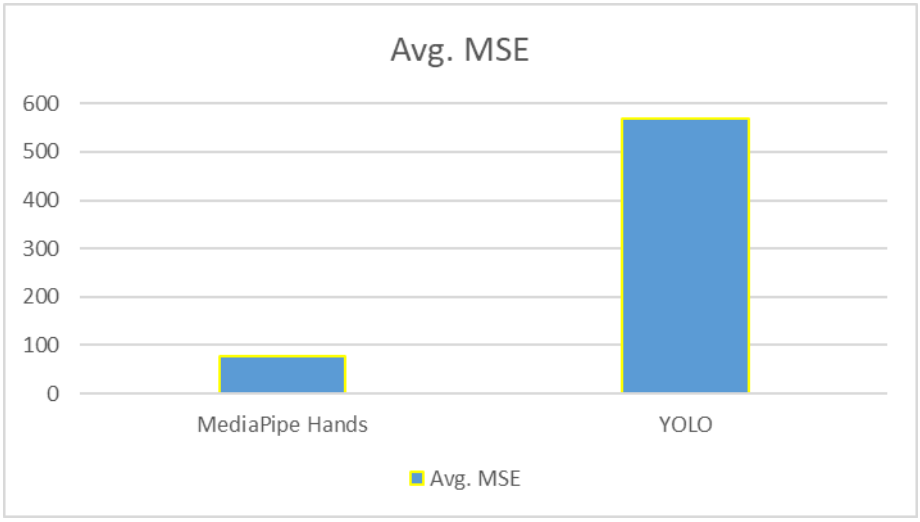


Рис. 2.4 Порівняння середнього значення MSE для моделей MediaPipe Hands та YOLO.

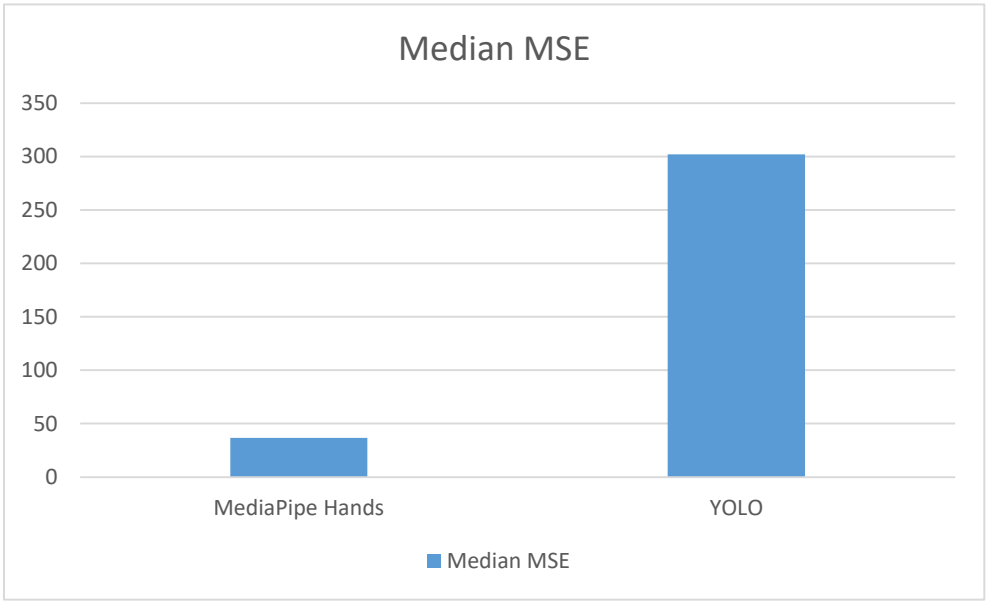


Рис. 2.5 Порівняння медіанного значень MSE для моделей MediaPipe Hands та YOLO.

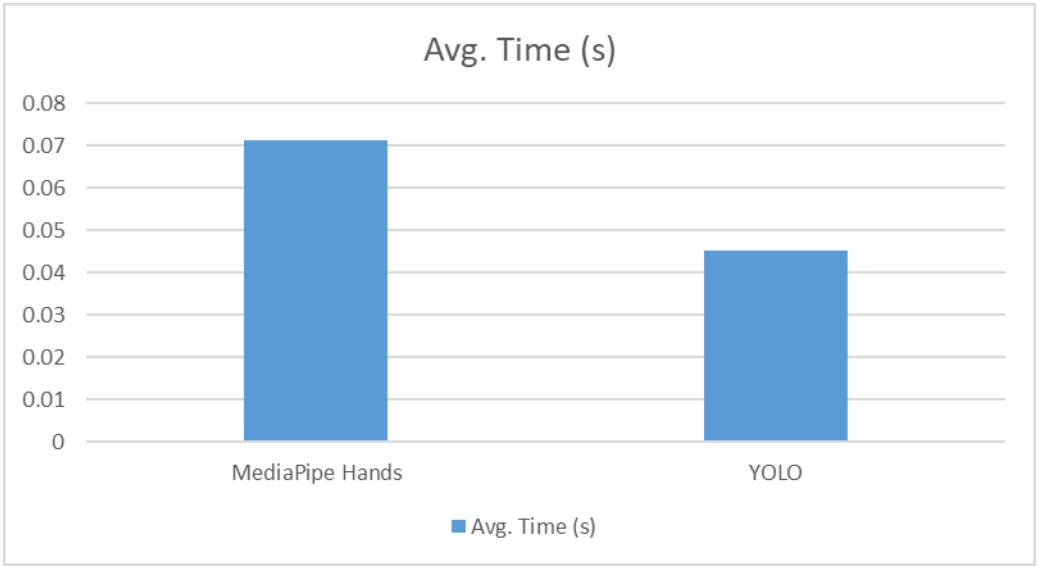


Рис. 2.6 Порівняння точності моделей MediaPipe Hands та YOLO за метрикою Avg. Time.

MediaPipe Hands продемонструвала значно кращу точність локалізації ключових точок при вищій частоті успішного виведення руки Хоча YOLO має перевагу за швидкістю обробки, його точність є недостатньою для задач, що вимагають точність трекінгу, як керування мишею. Таким чином, MediaPipe Hands було обрано як основну модель для реалізації системи.

OpenPose, незважаючи на свою популярність та високу точність у загальних задачах позової оцінки, не був включений до експериментального порівняння

через кілька важливих причин. По-перше, він відзначається складністю інтеграції та налаштування в середовищі розробки, що ускладнює його використання в невеликих проєктах. По-друге, OpenPose має високі вимоги до апаратного забезпечення, що робить його запуск проблематичним на менш потужних пристроях. Крім того, ця бібліотека орієнтована на повне тіло, тому обробляє надлишкову інформацію, що не є релевантним для задач, сфокусованих лише на кисті руки.

2.4 Формування моделі для класифікації жестів

На відміну від задачі керування мишею, яка вимагає безперервного відстеження положення руки, задача введення тексту базується на розпізнаванні статичних жестів, які відповідатимуть літерам жестової абетки. Для цього необхідно розробити модель, здатну класифікувати зображення руки в певному положенні як одну з фіксованих категорій.

2.4.1 Підготовка та обробка даних

Перед початком навчання необхідно обрати та підготувати якісний набір даних. Для успішного навчання моделі потрібна велика кількість різноманітних прикладів, які допомагають моделі узагальнювати інформацію та підвищують її точність і стабільність. Особливо це важливо для розпізнавання жестів, оскільки жести можуть виконуватися різними користувачами за різних умов освітлення, ракурсів та фону.

Незважаючи на велику кількість відкритих датасетів для жестової мови, більшість з них адаптовані під американську або міжнародну жестову мову, що не відповідає особливостям українського алфавіту. Тому було прийнято рішення створити власний датасет, орієнтований на українську жестову мову.

Було визначено 37 класів, які відповідають літерам українського алфавіту від “А” до “Я” та додаткові три класи “Space” (пробіл), “Nothing” (відсутність жесту), “Delete” (видалення символу) (рис. 2.7). Для кожного з класів передбачено приблизно однакову кількість з метою забезпечити збалансованість вибірки.

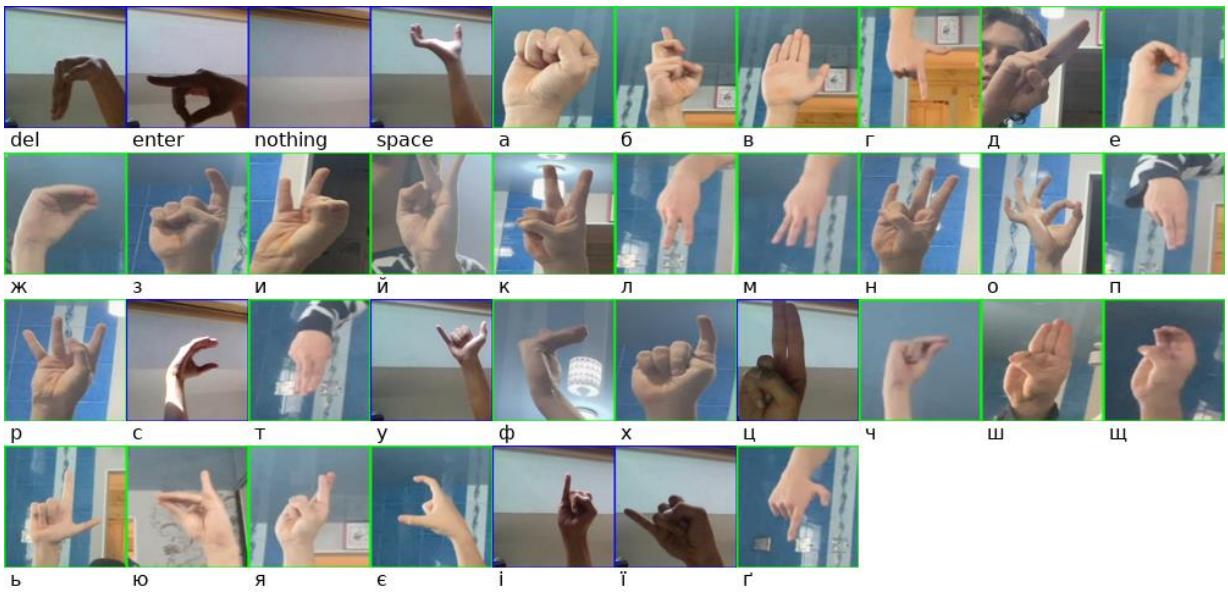


Рис. 2.7 Приклад зображення для кожного класу датасету

Для збору даних використовувалась бібліотека MediaPipe Hands, яка дозволяє виявляти руку на зображенні в реальному часі. Джерелом відео слугувала веб-камера.

Під час збору даних була демонстрація кожного жесту перед камерою. На кожному кадрі система автоматично визначала координати руки, будувала навколо неї прямокутник, вирізала цю область із кадру, масштабувала її до розміру 224x224 пікселів та зберігала як окреме зображення. Такий підхід дозволив отримати велику кількість прикладів без потреби у ручній розмітці або зовнішніх анотаціях.

Усі зображення були збережені у відповідні директорії відповідно до назви класу, що значно спростило їх подальше опрацювання та використання під час етапу навчання моделі. Така організація структури даних забезпечила зручність у доступі до кожної категорії зображень. Для кожного класу було зібрано приблизно 3000 прикладів, що дало змогу сформувати збалансований та досить репрезентативний набір даних. Це, у свою чергу, позитивно вплинуло на якість і стабільність процесу навчання моделі, підвищивши її здатність до точного розпізнавання жестів.

Оскільки всі приклади були зроблені в домашніх умовах, прийнято рішення застосувати аугментацію для збільшення загального обсягу даних та підвищення загальної якості навчання. Серед застосованих методів були: зміна яскравості та

контрасту, обертання, масштабування, зсув зображення, Gaussian Blur, а також додавання випадкового шуму. Завдяки цим трансформаціям вдалося змоделювати різні умови освітлення та положення руки, що підвищило стійкість моделі до варіативності вхідних даних і сприяло кращому узагальненню під час реального використання (рис. 2.8).

```
transform = A.Compose([
    A.RandomBrightnessContrast(brightness_limit=0.2, contrast_limit=0.2, p=0.5),
    A.Rotate(limit=7, p=0.5),
    A.ShiftScaleRotate(shift_limit=0.03, scale_limit=0.05, rotate_limit=5, p=0.7),
    A.GaussianBlur(blur_limit=3, p=0.3),
    A.Resize(224, 224)
])
```

Рис. 2.8 Приклад конфігурації аугментації, що використовувалась для обробки зображень

Після створення та аугментації датасету, наступним кроком стала його підготовка до навчання моделі. На цьому етапі датасет було поділено на тренувальну та тестову вибірки у співвідношенні 80/20. Для цього використано функцію random_split, яка дозволяє випадковим чином розділити набір зображень на дві частини (рис. 2.9).

```
full_dataset = datasets.ImageFolder(root=data_dir, transform=train_transform)

train_size = int(0.8 * len(full_dataset))
test_size = len(full_dataset) - train_size
train_dataset, test_dataset = random_split(full_dataset,
                                           [train_size, test_size])

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True,
                          num_workers=4, pin_memory=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False,
                         num_workers=4, pin_memory=True)
```

Рис. 2.9 Фрагмент коду розподілу зображень на тренувальну та тестову вибірки

Кожна з вибірок була завантажена за допомогою DataLoader з відповідними параметрами, такими як розмір пакету (batch_size=64), кількість потоків для обробки (num_workers=4) та використання закріплення в пам’яті (pin_memory=True). Важливо, що тренувальна вибірка завантажувалась із перемішуванням (shuffle=True), щоб уникнути можливого упередження під час

навчання, а тестова – без перемішування (shuffle=False), що забезпечує стабільність результатів під час оцінювання точності моделі.

Загальний обсяг датасету становить понад 327 тисяч зображень, з яких 262 тисячі використано для навчання, та 65 тисяч – для тестування. Така структура дозволяє ефективно використовувати датасет у процесі тренування нейронної мережі та забезпечує можливість його розширення або модифікації в майбутньому без зміни логіки обробки.

2.4.2 Архітектура згорткової нейронної мережі

Для розпізнавання жестів було розроблено згорткову нейронну мережу (рис. 2.10). Мережа складається з чотирьох блоків, за якими слідують три повнозв'язні шари. Кінцевий шар відповідає за класифікацію одного з 37 класів жестів. CNN є одним з найефективніших типів нейронних мереж у задачах комп’ютерного зору, оскільки здатна автоматично витягувати релевантні просторові ознаки з вхідного зображення.

```
class CNNModel(nn.Module):
    def __init__(self):
        super(CNNModel, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 75, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(75),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )

        self.layer2 = nn.Sequential(
            nn.Conv2d(75, 50, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(50),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )

        self.layer3 = nn.Sequential(
            nn.Conv2d(50, 25, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(25),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )

        self.layer4 = nn.Sequential(
            nn.Conv2d(25, 10, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(10),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.flatten = nn.Flatten()
        self.fc1 = nn.Sequential(
            nn.Linear(14 * 14 * 10, 512),
            nn.ReLU(),
            nn.Dropout(0.3)
        )

        self.fc2 = nn.Sequential(
            nn.Linear(512, 128),
            nn.ReLU(),
            nn.Dropout(0.3)
        )

        self.fc3 = nn.Linear(128, 37)  # 37 класів

    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.fc2(x)
        x = self.fc3(x)
```

Рис. 2.10 Архітектура згорткової нейромережі для розпізнавання жестів

Розроблена модель складається з чотирьох послідовних згорткових блоків, кожен з яких виконує обробку зображення через згортку, нормалізацію, активацію та субдискретизацію (рис. 2.11). Завдяки цьому поступово зменшується

розмір зображення і збільшується глибина ознак, що дозволяє моделі ефективно вчитись.

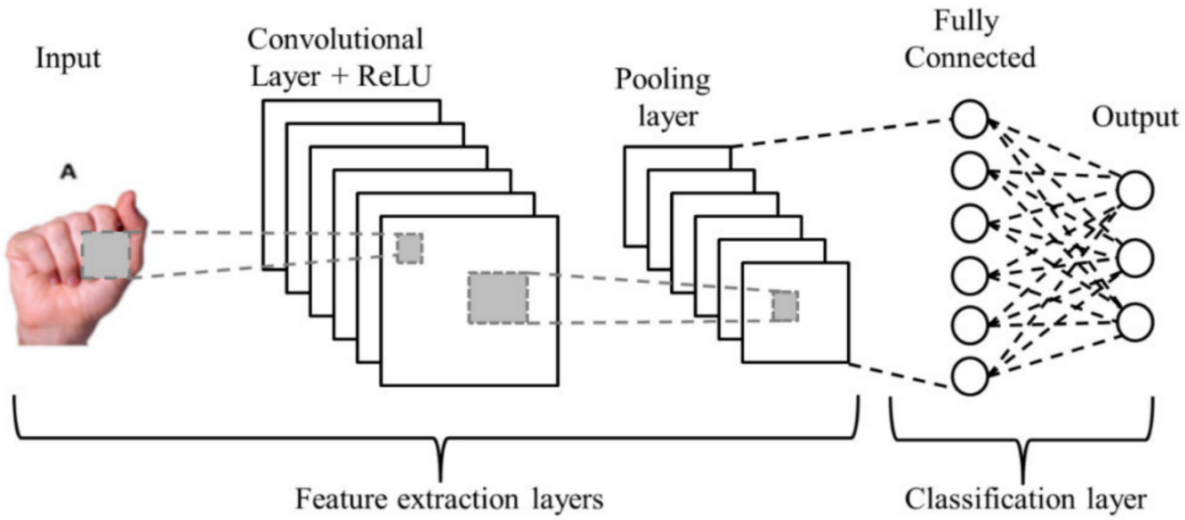


Рис. 2.11 Схема згорткової нейронної мережі для класифікації зображень [9]

Основні компоненти моделі:

- Згорткові шари (Convolutional layers) – головний елемент CNN, який виконує фільтрацію просторових ознак. Кожен фільтр виконує операцію згортки.
- Шари нормалізації (Batch Normalization) – після кожної згортки йде нормалізація для стабілізації та прискорення процесу навчання. Математично визначається наступним чином (2.1):

$$\hat{x} = \frac{x-\mu}{\sqrt{\sigma^2+\epsilon}} * ٦ + \beta \quad (2.1)$$

де :

  - μ – середнє та дисперсія по батчу,
  - σ^2 – дисперсія (variance), обчислена по міні-батчу.
  - $٦, \beta$ - навчальні параметри,
  - ϵ – мала константа для чисельної стійкості.
- Функції активації ReLU – нелінійні функії, що дозволяють моделі вчити складні залежності. Вона є стандартною у глибокому навчанні завдяки простоті та ефективності. Математично має такий вигляд (2.2):

$$\mathrm{ReLU}(x) = \max(0,x) \quad (2.2)$$

де :

- x – вхідний сигнал для функції,
- max() – функція, яка повертає більше значення з двох.

Вона дозволяє уникнути складних обчислень і проблеми зникнення градієнта. Вигляд функції ReLU: від'ємні значення обмежені нулем, а додатні – зростають лінійно (рис. 2.12).

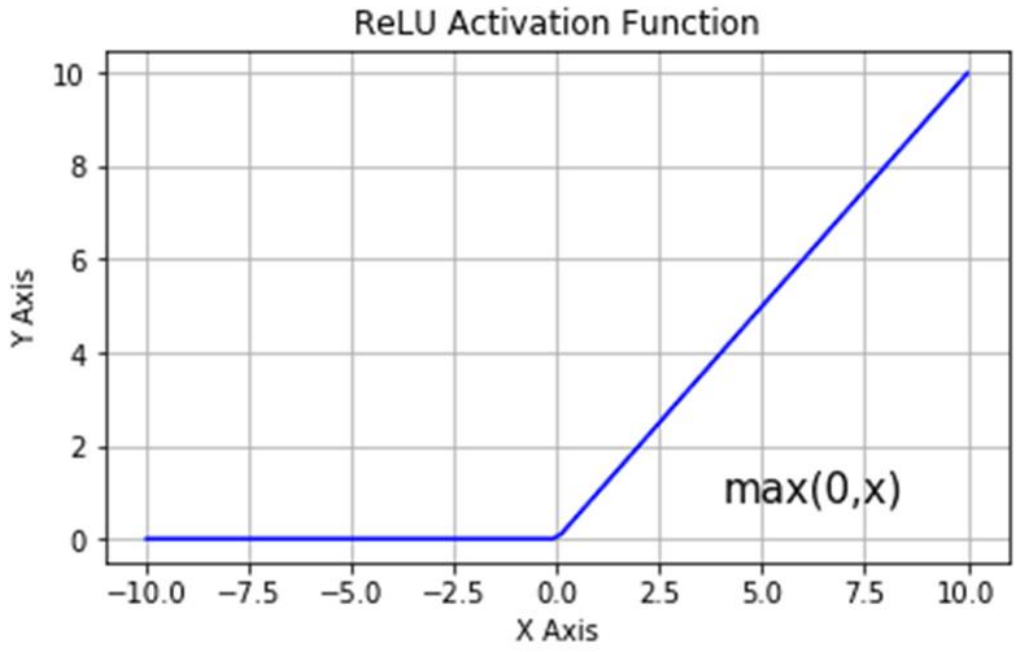


Рис. 2.12 Приклад графіку функції ReLU [10]

- Dropout – використовується як метод регуляризації для запобігання перенавчанню моделі. Під час навчання випадковим чином "вимикається" певна частина нейронів, що змушує модель не залежати від конкретних шляхів у мережі, а вчитись підсумовувати інформацію.
- Операція Max Pooling – зменшує розмірність зображення та виділяє найважливіші ознаки.
- Flatten-шар – перетворює тривимірну карту ознак у вектор для подальшої обробки повнозв’язними шарами.
- Повнозв’язний шар (Fully Connected) – складаються з двох прихованих шарів з ReLU та Dropout для запобігання перенавчанню. Завершальний шар має 37 нейронів – за кількістю класів жестів.

Завдяки цьому модель стала здатною класифікувати саме ті жести, які будуть використовуватися в подальшому.

2.4.3 Навчання та аналіз результатів

Перед початком навчання моделі було обрано відповідні інструменти оптимізації (рис. 2.13). В якості оптимізатора використовувався AdamW – модифікований алгоритм адаптивної оптимізації, що поєднує метод моментів з регуляризацією через вагове загасання. Його оновлення ваг проводиться за формулою (2.3):

$$\theta_{t+1} = \theta_t - \eta * (\frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}} + \lambda * \theta_t) \quad (2.3)$$

де :

- $\hat{m}_t$ – скориговані перші та другі моменти градієнта;
- η – швидкість навчання (learning rate) ;
- λ – коефіцієнт регуляції (weight decay) ;
- $\hat{m}_t$ – скориговане (bias-corrected) середнє значення градієнтів першого порядку, яке розраховується як експоненційне згладжування градієнтів з поправкою на зміщення;
- $\hat{v}_t$ – скориговане середнє значення квадратів градієнтів (другий момент) з поправкою на зміщення;
- ϵ – мале додатне число, яке запобігає діленню на нуль;
- λ – коефіцієнт вагової регуляції (weight decay), який додає регуляризаційний штраф до оновлення, щоб уникнути перенавчання та зменшити складність моделі;
- $\lambda * \theta_t$ – термін L2-регуляризації, який притягує значення параметрів ближче до нуля;

Для динамічного зменшення швидкості навчання протягом епох застосовувався планувальник CosineAnnealingLR, який зменшує значення lr за косинусною кривою, сприяючи стабільній оптимізації.

Функцією втрат була обрана крос-ентропія (CrossEntropyLoss) – стандартне рішення для багато класової класифікації, яке дозволяє ефективно порівнювати

розподіл передбачень з правильними мітками класів. Формально вона виглядає як (2.4):

$$\mathcal{L}_{CE} = -\sum_{i=1}^{C} y_i * \log(p_i) \quad (2.4)$$

де :

- C – кількість класів;
- y_i – правильна мітка;
- p_i – ймовірність, передбачень моделлю для класу i;
- $\log(p_i)$ – логарифм передбаченої ймовірності.

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = CNNModel().to(device)
optimizer = optim.AdamW(model.parameters(), lr=LR, weight_decay=1e-4)
scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=NUM_EPOCHS)
criterion = nn.CrossEntropyLoss()
```

Рис. 2.13 Ініціалізація моделі, оптимізатора, функції втрат та планувальника навчальної ставки у PyTorch

Після підготовки архітектури моделі та формування датасету було здійснено повноцінне навчання мережі. Процес навчання складався з двох основних етапів: тренування та валідації, які виконувались у циклі протягом 10 епох.

Під час тренувальної фази модель переводилась у режим train(), де обчислювалась функція втрат (loss), здійснювалась зворотна передача помилки (backpropagation), та оновлення ваг за допомогою оптимізатора. Для запобігання нестабільності навчання використовувалось обмеження градієнта (gradient clipping) (рис. 2.14).

```python
def train_epoch(model, loader, criterion, optimizer, scheduler=None):
    model.train()
    total_loss = 0.0

    for inputs, labels in tqdm(loader, desc="Training"):
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()

        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()

        total_loss += loss.item() * inputs.size(0)

    if scheduler:
        scheduler.step()

    return total_loss / len(loader.dataset)
```

Рис. 2.14 Функція навчання моделі на одній епосі

Фаза валідації виконувалась без оновлення ваг – у режимі eval(), що дозволяло об’єктивно оцінити здатність моделі узагальнювати нові, невідомі дані. На цьому етапі обчислювались втрата та точність класифікації (accuracy), які фіксувалися після кожної епохи(рис. 2.15).

```python
def validate(model, loader, criterion):
    model.eval()
    total_loss = 0.0
    correct = 0

    with torch.no_grad():
        for inputs, labels in tqdm(loader, desc="Validation"):
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)

            loss = criterion(outputs, labels)
            total_loss += loss.item() * inputs.size(0)

            _, preds = torch.max(outputs, 1)
            correct += (preds == labels).sum().item()

    accuracy = correct / len(loader.dataset) * 100
    return total_loss / len(loader.dataset), accuracy
```

Рис. 2.15 Функція перевірки моделі (валідації)

Для оцінки ефективності використовувалися такі метрики, як втрата (loss) на тренувальній та валідаційній вибірках, а також точність (accuracy) на валідаційному наборі даних. З графіку видно, що втрата на тренувальній вибірці швидко зменшується, що свідчить про успішне навчання моделі. Втрата на валідаційній вибірці також знижується і залишається стабільною, що вказує на відсутність перенавчання (overfitting) на даному етапі (рис. 2.16).

Рис. 2.16 Зміна функції втрат на тренувальній і валідаційній вибірках під час навчання моделі

Також оцінювалася валідаційна точність після кожної епохи (рис. 2.17). Високі показники точності свідчать про хорошу здатність моделі узагальнювати нові дані та правильно класифікувати жести, що не входили до тренувального набору.

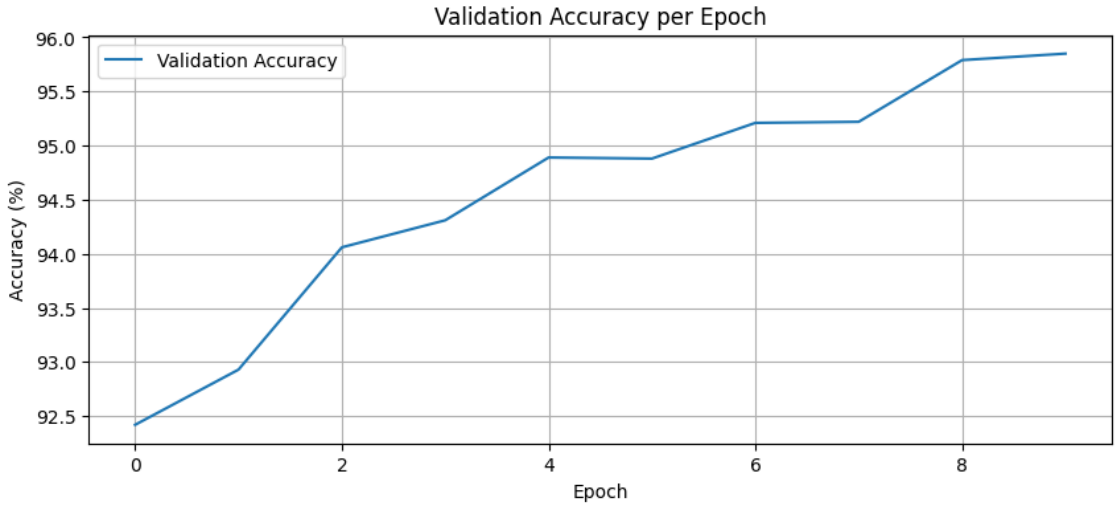


Рис. 2.17 Зміна точності під час перевірки на валідаційній вибірці

Таким чином, результати навчання демонструють ефективність побудованої моделі та коректність вибору гіперпараметрів. Отримані метрики свідчать про здатність моделі до узагальнення.

Висновок до розділу 2

У другому розділі було розглянуто ключові етапи проєктування системи розпізнавання жестів, які охоплюють вибір технологій, створення датасету,

побудову моделі та її навчання. Було порівняно бібліотеки MediaPipe Hands, YOLO та OpenPose, і на основі практичного експерименту обрано MediaPipe Hands як найефективніше рішення для трекінгу рук у реальному часі завдяки високій точності, швидкості та простоті інтеграції.

Особливу увагу приділено створенню власного датасету для української жестової мови, що дало змогу адаптувати систему до національного контексту. Проведено аугментацію даних для підвищення стійкості моделі до варіацій умов (освітлення, пози рук тощо).

Було розроблено згорткову нейронну мережу для класифікації жестів. Модель продемонструвала високу точність розпізнавання під час навчання та валідації, що підтверджує ефективність архітектури та підготовлених даних.

Таким чином, у цьому розділі було створено технічну та алгоритмічну основу, що забезпечує надійну і точну роботу системи розпізнавання жестів – одну з ключових складових розробки.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Архітектура програмного забезпечення

Система розроблена на мові програмування Python, що забезпечує зручну інтеграцію інструментів комп'ютерного зору, нейронних мереж та управління пристроями. Основу рішення складають такі бібліотеки:

- MediaPipe – для детекції руки та визначення ключових точок.
- PyTorch – для побудови та навчання згорткової нейронної мережі, яка класифікує жести.
- OpenCV – для захоплення та попередньої обробки відео з камери.
- PyAutoGUI – для управління мишею та введення тексту з клавіатури.
- Tkinter – для виводу повідомлень поверх усіх інших вікон.

Для розробки системи використовувався PyCharm – це інтегроване середовище розробки (IDE) для мови програмування Python, розроблене компанією JetBrains. Воно забезпечує потужні інструменти для написання, налагодження та тестування програмного коду. У процесі розробки він використовувалося як основне середовище для зручного редагування коду, налагодження логіки роботи програми та управління проєктною структурою. Вона поділена на кілька логічних модулів які підвищують читабельність та зручність підтримки коду(рис. 3.1).

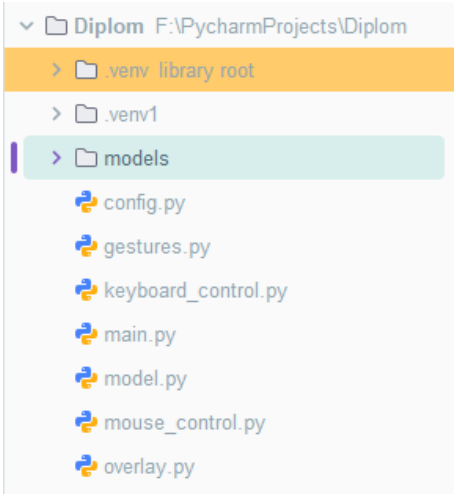


Рис. 3.1 Структура проєкту: основні модулі системи розпізнавання жестів у середовищі PyCharm

Опис структури проєкту:

- models/model.pth – файл із збереженою натренованою нейронною мережею, реалізованою у середовищі PyTorch.

- config.py – файл конфігурації, який містить параметри налаштувань (шляхи до файлів, параметри обробки жестів, налаштування керування клавіатурою та мишею).

- main.py – головний скрипт, що ініціалізує всі необхідні компоненти, завантажує модель та запускає основний цикл роботи.

- model.py – модуль, який відповідає за завантаження нейронної мережі з файлу, підготовку вхідних даних та отримання передбачень від моделі.

- gestures.py – обробка відеопотоку, виявлення руки, попередня обробка зображення та передача вхідних даних до моделі для класифікації жестів.

- keyboard_control.py – реалізація управління клавіатурою: емулювання натискань клавіш залежно від розпізнаного жесту.

- mouse_control.py – реалізація управління мишею: рух курсору, кліки, скролінг тощо.

- overlay.py – модуль для накладання інформації на екран, наприклад, відображення розпізнаного жесту або статусу системи.

Функціонування системи ґрунтується на обробці відеопотоку з камери в реальному часі, після чого залежно від обраного режиму активується одна з двох гілок логіки: керування мишею або введення тексту (рис. 3.2). У режимі керування мишею система за допомогою MediaPipe Hands визначає положення руки та обчислює координати для керування курсором. У режимі введення тексту рука ізольовується в окрему область, яка подається на вхід нейронної моделі для класифікації жесту. Після підтвердження результату відповідний символ передається у вікно введення.

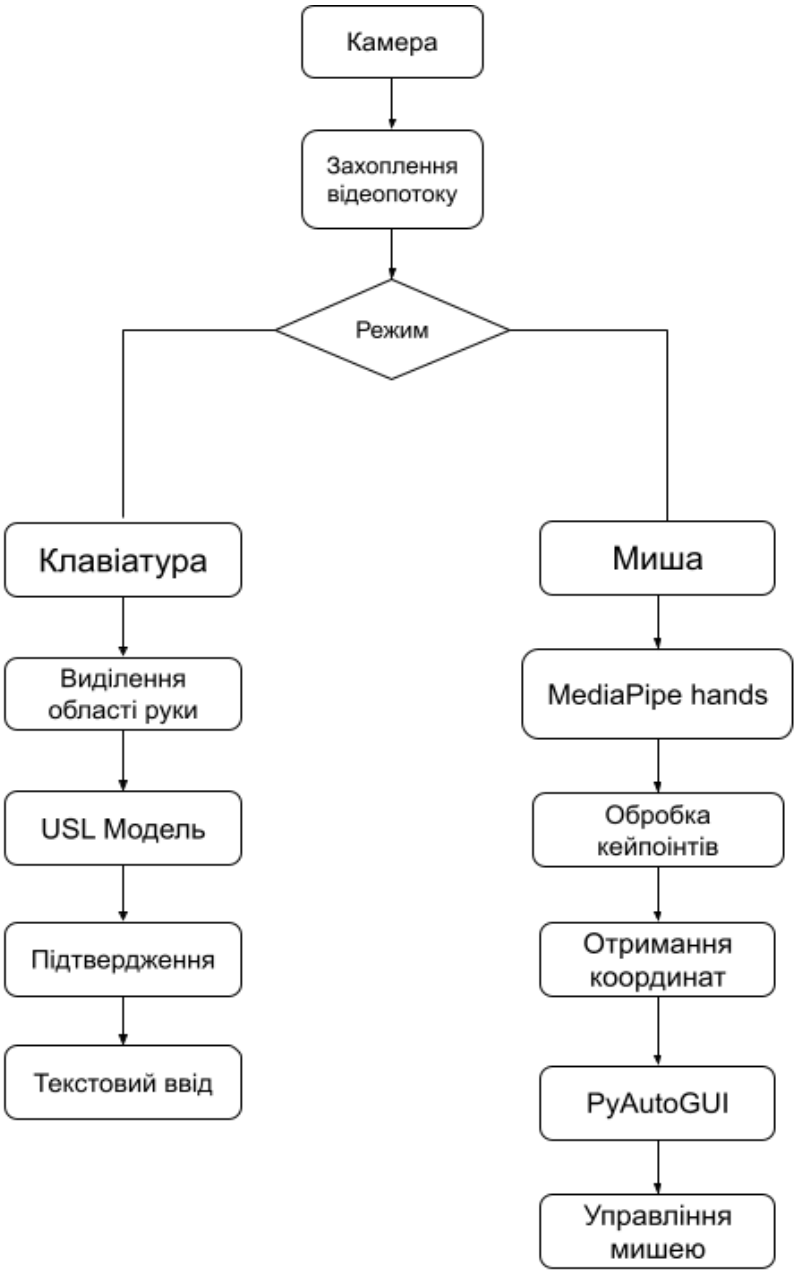


Рис. 3.2 Блок-схема роботи системи в режимах "Миша" та "Клавіатура"

3.2 Реалізація режиму керування мишею

Режим керування мишею реалізовано у модулі mouse_control.py за допомогою бібліотеки PyAutoGUI, що дозволяє програмно здійснювати переміщення курсора, кліки та прокрутку сторінок. Основна функція – handle_mouse_control, яка відповідає за обробку положення руки, розпізнавання жестів та керування мишею.

Для розпізнавання жестів, використовується 21 ключова точка на руці в режимі реального часу (рис. 3.3). Кожна точка має координати де x та y

нормалізовані відносно розмірів зображення. Особливу увагу приділено таким точкам, як кінчик вказівного пальця, середнього пальця, а також основі долоні. На основі їхнього положення та взаємного розташування визначаються жести. Алгоритм роботи реалізовано як послідовність етапів, кожен з яких відповідає за окремий тип взаємодії.

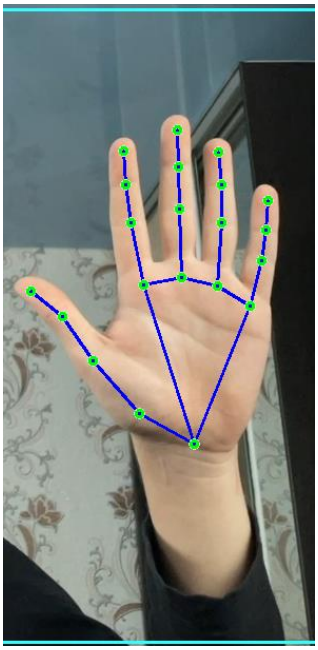


Рис. 3.3 Виявлення ключових точок у робочій області

Для визначення активного жесту коли рука знаходиться у положенні активного керування, розраховується положення кінчика вказівного пальця у пікселях відносно роздільної здатності екрану (рис. 3.4). Далі координати перетворюються до нормалізованих величин у межах прямокутника керування (x1, y1, x2, y2), а потім – до координат екрану з урахуванням згладжування. Координати курсора передаються у фоновий потік, який викликає функцію pyautogui.moveTo. Це дозволяє забезпечити плавність та уникнути зависання головного потоку.

```
if is_active_gesture(landmarks):
    x_pixel = int(landmarks.landmark[8].x * width)
    y_pixel = int(landmarks.landmark[8].y * height)

    clipped_x = max(x1, min(x_pixel, x2))
    clipped_y = max(y1, min(y_pixel, y2))

    norm_x = (clipped_x - x1) / rect_width
    norm_y = (clipped_y - y1) / rect_height

    x_monitor = int(norm_x * config.monitor_width)
    y_monitor = int(norm_y * config.monitor_height)

    if config.prev_x is None:
        config.prev_x, config.prev_y = x_monitor, y_monitor
    else:
        x_monitor = int(config.SMOOTHING_FACTOR * x_monitor + (1 - config.SMOOTHING_FACTOR) * config.prev_x)
        y_monitor = int(config.SMOOTHING_FACTOR * y_monitor + (1 - config.SMOOTHING_FACTOR) * config.prev_y)
        config.prev_x, config.prev_y = x_monitor, y_monitor

    x_monitor = max(config.SAFE_MARGIN, min(config.monitor_width - config.SAFE_MARGIN, x_monitor))
    y_monitor = max(config.SAFE_MARGIN, min(config.monitor_height - config.SAFE_MARGIN, y_monitor))
    overlay.show_message(f"Управління мишею", duration=1.0)
    move_mouse_thread(x_monitor, y_monitor)
else:
    config.prev_x, config.prev_y = None, None
```

Рис. 3.4 Розрахунок координат пікселя в просторі екрана на основі кінчика вказівної точки

Для реалізації алгоритму прокрутки використовується жест з піднятими вказівним та середнім пальцями (рис. 3.5), система зчитує y-координату середнього пальця у нормалізованому вигляді. Щоб визначити напрям та інтенсивність руху, її порівнюють із координатою з попереднього кадру. Таким чином, обчислюється різниця положення пальця за вертикаллю – дельта зміщення.

```
def is_two_fingers_up(landmarks):  1 usage
    return (landmarks.landmark[8].y < landmarks.landmark[6].y and
            landmarks.landmark[12].y < landmarks.landmark[10].y and
            landmarks.landmark[16].y > landmarks.landmark[14].y and
            landmarks.landmark[20].y > landmarks.landmark[18].y)
```

Рис. 3.5 Перевірка жесту з піднятими двома пальцями

Якщо ця зміна перевищує заданий поріг, система інтерпретує рух як жест прокрутки. На основі дельти масштабується величина скролу, і команда передається у фоновий потік для виконання прокрутки через pyautogui.scroll (рис. 3.6).

```
if is_two_fingers_up(landmarks):
    current_y = landmarks.landmark[12].y
    overlay.show_message(f"Скролл", duration=1.0)
    if config.prev_two_finger_y is not None:
        delta_raw = current_y - config.prev_two_finger_y
        if abs(delta_raw) > config.SCROLL_THRESHOLD:
            delta = int(delta_raw * 100 * config.SCROLL_SENSITIVITY)
            scroll_thread(delta)
    config.prev_two_finger_y = current_y
else:
    config.prev_two_finger_y = None
```

Рис. 3.6 Реалізація вертикального скролінгу за допомогою жесту з двома піднятими пальцями

Натискань миші спрацює коли виявляє жест, що відповідає натисканню, система переходить до перевірки стабільності пози. Це необхідно, щоб уникнути випадкових або хибних спрацювань при короткочасному збігу координат. Для цього запроваджено лічильник, який збільшується на кожному кадрі, якщо клік-жест залишається активним. Коли кількість послідовних кадрів із жестом перевищує поріг, виконується відповідна команда для імітації натискання миші. Система підтримує два типи кліків:

- лівий клік – виконується за допомогою одного жесту (наприклад, стиснута долоня), за допомогою команди pyautogui.click();
- правий клік – викликається іншим спеціально визначеним жестом , з використанням pyautogui.click(button='right') (рис. 3.7).

```
if is_click(landmarks):
    if config.click_counter >= config.CLICK_STABILITY_FRAMES:
        pg.click()
        overlay.show_message(f"Клік лівою кнопкою миші", duration=1.0)
        config.click_counter = 0
    else:
        config.click_counter += 1
else:
    config.click_counter = max(0, config.click_counter - 1)
```

Рис. 3.7 Реалізація стабільного кліку лівою кнопкою миші за допомогою жесту

Для перемикання режиму керування передбачено спеціальний жест – махання відкритою долонею. Його виявлення виконується функцією is_waving_hand (рис. 3.8), яка аналізує послідовність положень руки за останні кадри. Якщо рух руки має характерну “махання руки” траєкторію в горизонтальній площині та долоня відкрита, жест виконується.

```python
def is_waving_hand(current_landmarks, history, frame_width):  2 usages
    if len(history) < 5:
        return False

    if config.WAVE_OPEN_PALM_REQUIRED and not is_open_palm(current_landmarks):
        return False

    wrist_positions = [lm.landmark[0].x * frame_width for lm in history[-10:]]
    min_x = min(wrist_positions)
    max_x = max(wrist_positions)
    amplitude = max_x - min_x

    if amplitude < frame_width * 0.15:
        return False

    direction_changes = 0
    for i in range(1, len(wrist_positions) - 1):
        prev = wrist_positions[i - 1]
        curr = wrist_positions[i]
        next_pos = wrist_positions[i + 1]

        if (curr > prev and curr > next_pos) or (curr < prev and curr < next_pos):
            direction_changes += 1

    return direction_changes >= 1
```

Рис. 3.8 Функція is_waving_hand яка визначає, чи виконує користувач жест "махання рукою".

Щоб уникнути багаторазового спрацювання, реалізовано обмеження по часу, тобто мінімальний інтервал між жестами перемикання. Якщо всі умови дотримано, викликається функція set_regime, яка змінює активний режим з "mouse" на "keyboard" і повідомляє про це через накладку інтерфейсу(рис. 3.9).

```
if landmarks:
    if not config.WAVE_OPEN_PALM_REQUIRED or is_open_palm(landmarks):
        config.mouse_wave_history.append(landmarks)

    config.mouse_wave_history = config.mouse_wave_history[-20:]

current_time = time.time()
if (len(config.mouse_wave_history) >= 5 and
        current_time - config.last_mouse_wave_time > config.WAVE_COOLDOWN and
        is_waving_hand(landmarks, config.mouse_wave_history, width)):
    set_regime( new_regime: "keyboard", overlay)
    config.last_mouse_wave_time = current_time
    config.mouse_wave_history = []

return frame
```

Рис. 3.9 Обробка історії рухів руки і розпізнавання жесту махання з переходом у режим керування клавіатурою

Уся логіка працює з урахуванням фільтрації шумів, використовуючи історію положення руки та межі безпечної області для уникнення виходу курсора за межі екрану.

3.3 Реалізація режиму введення тексту

Режим введення тексту реалізовано у модулі keyboard_control. Для симуляції натискання клавіш використовується бібліотека PyAutoGUI, розпізнавання жестів виконується нейронною мережею. Основна функція – handle_keyboard_control, яка обробляє зображення руки, виділяє область з кистю, виконує класифікацію жесту і керує введенням тексту.

За координатами ключових точок визначається прямокутник, який охоплює всю кисть з додатковим запасом пікселів. З цієї області формується окреме зображення для подальшої класифікації (рис. 3.10). Такий підхід дозволяє сконцентрувати увагу моделі лише на жестовій частині зображення, виключаючи зайві фонові елементи. Це підвищує точність та швидкість роботи системи.

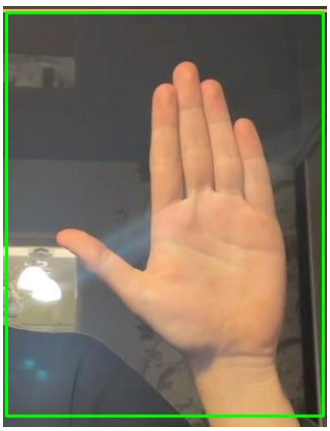


Рис. 3.10 Виділення області кисті руки на основі координат ключових точок для подальшої класифікації

Виділена область з рукою конвертується у формат, який підходить для подачі на вхід нейронної мережі. Спочатку зображення переводиться з кольорової моделі Blue-Green-Red (синій-зелений-червоний) у модель Red-Green-Blue (червоний-зелений-синій)**.** Потім воно конвертується у формат об’єкта PIL.Image, після чого застосовуються послідовні трансформації: зміна розміру, нормалізація пікселів і перетворення у тензор PyTorch (рис. 3.11). В результаті формується тензор фіксованого розміру та стандартного формату, який може ефективно оброблятися моделлю.

```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

Рис. 3.11 Послідовність трансформацій для попередньої обробки зображень: масштабування, перетворення у тензор і нормалізація

Підготовлене зображення передається на обчислювальний пристрій – графічний процесор . За допомогою операції torch.argmax вибирається клас з найбільшим значенням ймовірності, який і вважається передбаченим жестом (рис. 3.12). Цей результат далі використовується для управління введенням тексту.

```
with torch.no_grad():
    output = model(img)
    pred = torch.argmax(output, dim: 1).item()
    predicted_char = classes[pred]
```

Рис. 3.12 Процес передбачення класу

Для уникнення випадкових або помилкових вводів у систему введення тексту зроблено механізм подвійної перевірки – підтвердження символів. Якщо один і той же символ розпізнається стабільно протягом 1.5 секунди, система переходить у режим підтвердження(рис. 3.13).

```
if not config.confirm_mode:
    if predicted_char == config.last_char:
        if config.char_start_time and (now - config.char_start_time >= 1.5):
            config.confirm_mode = True
            config.confirm_start_time = now
            config.current_char = predicted_char
            overlay.show_message(f"Розпізнано: {config.current_char} – Підтвердити? 👍 / 🤘", duration=1.0)
    else:
        config.last_char = predicted_char
        config.char_start_time = now
```

Рис. 3.13 Перевірка стабільності розпізнаного символу та активація режиму підтвердження введення

У цьому режимі користувач має можливість підтвердити або відхилити розпізнаний символ за допомогою певних жестів: жест відкритої долоні слугує сигналом підтвердження, а жест із піднятим вказівним та мізинцем а – скасуванням.

Якщо підтвердження або відхилення символу не відбувається протягом 3 секунд, система автоматично скасовує введення поточного символу і повертається до режиму розпізнавання(рис. 3.14).

```
if now - config.confirm_start_time > 3:
    overlay.show_message("Час вичерпано – жест скасовано.", duration=1.0)
    config.confirm_mode = False
    config.last_char = None
    config.char_start_time = None
    config.gesture_confirm_start_time = None
```

Рис. 3.14 Скасування введення символу у випадку перевищення часу очікування підтвердження

Після підтвердження символ або команда виконуються через PyAutoGUI. Зазвичай це натискання клавіш (Backspace, Enter, Space) або вставлення тексту через буфер обміну з подальшим симулюванням команди Ctrl+V для вставки (рис. 3.15). Такий підхід застосовується через те, що бібліотека не підтримує безпосереднє введення тексту українською мовою через функцію typewrite або аналогічні методи. Використання буфера обміну дозволяє вводити будь-які символи незалежно від мови, оскільки операційна система вставляє текст безпосередньо в активне поле введення, обходячи обмеження PyAutoGUI.

```
def execute_command(command):  1 usage
    if command == 'del':
        pg.press('backspace')
    elif command == 'enter':
        pg.press('enter')
    elif command == 'nothing':
        pass
    elif command == 'space':
        pg.press('space')
    else:
        pyperclip.copy(command.strip())
        time.sleep(0.05)
        pg.hotkey( *args: 'ctrl', 'v')
```

Рис. 3.15 Виконання команди введення: обробка службових команд або вставка тексту через буфер обміну

Аналогічно до режиму керування мишею, розпізнається жест “махання рукою” для повернення до режиму миші.

Для зручності користувача у системі реалізовано візуальне накладання тексту та підказок безпосередньо на екран поверх усіх активних вікон. Це забезпечує постійний візуальний зворотний зв’язок із системою, не відволікаючи користувача від основної діяльності.

В області екрана відображаються :

- Поточний розпізнаний символ;
- Сповіщення про зміну режимів;
- Підтвердження або зміна режиму;

Основні властивості вікна:

- overrideredirect(True) – вимикає рамку вікна;
- attributes("-topmost", True) – розміщення поверх усіх інших вікон;
- show_message – відображення пріоритетних повідомлень, які не перекриваються символами (рис. 3.16).

```
def show_message(self, message, duration=2.0):  10 usages (10 dynamic)
    self.current_text = message
    self.label.config(text=message)
    self.root.deiconify()
    self.message_until = time.time() + duration
    self.update()
```

Рис. 3.16 Відображення текстового повідомлення в графічному інтерфейсі з автоматичним приховуванням через заданий інтервал часу

3.4 Тестування системи в реальних умовах

Після завершення розробки систему було протестовано з використанням різних сценаріїв взаємодії користувача. Основна мета тестування – перевірити стабільність, точність розпізнавання жестів, зручність керування та адаптивність до змін середовища.

Під час тестування з різним освітленням система проходила перевірку в різних умовах освітлення для оцінки її надійності та стабільності. Було проведено тестування при нормальному денному освітленні. Та при слабкому освітленні, щоб дослідити вплив тіней, шумів зображення та нестачі світла на точність розпізнавання жестів. Для оцінки точності розпізнавання в режимах миші та введення тексту при різному освітленні були проведені окремі тести.

Результати тестування у режимі керування мишею наведено в таблиці 3.1**,** а результати для введення тексту **–** у таблиці 3.2**.**

Таблиця 3.1

Вплив відстані на точність у режимі керування мишею

Освітлення	Точність позиціонування (%)
Добре	80%
Слабке	65%

Таблиця 3.2

Вплив відстані на точність у режимі введення тексту

Освітлення	Точність розпізнавання (%)	Помилки
Добре	84%	6
Слабке	68%	12

Результати тестування показали, що система працює стабільно при денному, коректно розпізнаючи жести й забезпечуючи належну взаємодію. У складніших умовах – при слабкому або нестабільному світлі – точність розпізнавання дещо знижується.

Окрім умов освітлення, тестування також охоплювало перевірку ефективності системи на різних відстанях між користувачем і камерою. Було проведено тестування при малій відстані, та при середній відстані, яка відповідає типовому сценарію використання за столом.

Результати подано у таблиці 3.3 для керування мишею та таблиці 3.4 для режиму введення тексту.

Таблиця 3.3

Вплив відстані на точність у режимі керування мишею

Відстань до камери	Точність позиціонування
30 см	67%
60 см	85%

Таблиця 3.4

Вплив відстані на точність у режимі введення тексту

Відстань до камери	Точність розпізнавання	Помилки
30 см	70%	12
60 см	80%	7

Результати показали, що найвища точність досягається при середній відстані. Водночас механізми підтвердження та стабілізації дозволяють зберігати функціональність навіть у менш сприятливих умовах.

Для перевірки працездатності режиму введення було проведено тестування розпізнавання слова “мама”, яке складається з повторюваних жестів однієї й тієї самої літери. Потрібно послідовно показувати жест для літери “М” та “А” , утримуючи його стабільно протягом 1,5 секунд для активації режиму підтвердження. Після цього виконувалася команда підтвердження жесту, і літера додавалася до текстового рядка. Потім аналогічну процедуру повторювали для наступних літер. Повне слово формувалося поступово: “М”, “А”, “М”, “А”, з урахуванням затримки на підтвердження та необхідності стабільного показу кожного жесту. У результаті система успішно розпізнала й сформувала слово (рис. 3.17 – 3.20).

Рис. 3.17 Розпізнавання жесту для літери “М”

Рис. 3.18 Підтвердження жесту для літери “М”

Рис. 3.19 Розпізнавання жесту для літери “А”

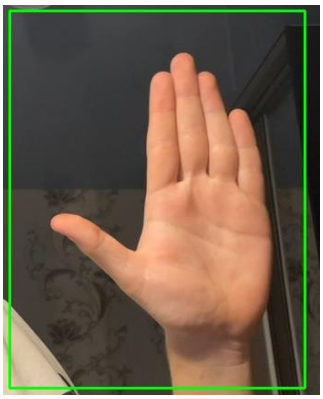


Рис. 3.20 Підтвердження жесту для літери “А”

Висновок до розділу 3

У третьому розділі розглянуто практичну реалізацію розробленої системи, що включає керування мишею та введення тексту за допомогою жестів руки. Було реалізовано повноцінний програмний продукт, побудований з використанням мов Python та бібліотек MediaPipe Hands, PyTorch, OpenCV, PyAutoGUI та інших. Було розділено архітектуру системи на логічні модулі, що значно спростило підтримку та масштабування проєкту.

У режимі керування мишею реалізовано відстеження положення руки, інтерпретацію жестів для натискань лівою кнопкою миші, прокрутки та перемикання режимів. У режимі введення тексту – інтеграція з нейромережею для класифікації символів з української жестової абетки, підтвердження символів та підтримка української мови через буфер обміну. Особливу увагу приділено плавності інтерфейсу, зниженню помилок при швидких жестах і зворотному зв’язку з користувачем.

Результати тестування показали високу точність виконання дій, стабільну реакцію системи в режимі реального часу і зручність у використанні. Було виявлено деякі чинники, що впливають на ефективність (наприклад, освітлення), що дозволяє визначити напрямки подальших покращень.

Узагальнюючи, можна сказати, що у цьому розділі реалізовано практичну реалізацію системи, що відповідає вимогам, поставленим на початку роботи, і підтверджує доцільність використання жестового управління в сучасних HCI-системах.

ВИСНОВКИ

Основною метою мого дослідження було створення інтерфейсу, який дозволяє керувати комп’ютером без фізичного контакту, використовуючи лише жести рук, що розширює можливості традиційної взаємодії з комп’ютером.

Було проведено аналіз сучасних підходів до людино-машинної взаємодії та способів реалізації жестового керування. Визначено основні типи HCI, їх моделі, а також переваги та обмеження апаратних і програмних засобів, що існують на ринку.

Здійснено аналіз популярних бібліотек комп’ютерного зору для розпізнавання жестів рук, таких як MediaPipe Hands, OpenPose та YOLO. На основі порівняння точності та швидкодії обрано MediaPipe Hands як найоптимальніший варіант.

Розроблено власний датасет української жестової абетки, що включає 37 класів, із подальшою аугментацією для покращення узагальнення моделі. На основі цих даних проведено навчання згорткової нейронної мережі.

Реалізовано програмне забезпечення з двома режимами роботи: керування мишею та введення тексту жестами. Забезпечено підтримку введення українських символів.

Було проведено тестування в реальних умовах і продемонструвала високу точність, низьку затримку та стабільну роботу. Результати підтверджують ефективність підходу для покращення HCI.

Реалізована система продемонструвала стабільну роботу в режимі реального часу, без помітних затримок, що свідчить про її придатність для практичного застосування.

ПЕРЕЛІК ПОСИЛАНЬ

1. Ganatra A., Panchal B. Y., Singhal T., Rawal H., Singh A. Introduction to Explainable AI // *Explainable AI in Health Informatics*. – July 2024. – С. 1–31.
2. Langote M., Saratkar S., Kumar P., Goyal D. Human–computer interaction in healthcare: Comprehensive review // *AIMS Bioengineering*. – 2024. – Т. 11, № 3. – С. 343–390.
3. Kumar, V. N. What Does Fitts's Law Mean for Web Apps? [Електронний ресурс] / Vignesh Nandha Kumar. – 2 жовтня 2015. – Режим доступу: https://inside.recruiterbox.com/what-does-fittss-law-mean-for-web-apps-628203098c5e (дата звернення: 20.04.2025).
4. Ultraleap – Products [Електронний ресурс]. – Режим доступу: https://www.ultraleap.com/products/ (дата звернення: 24.04.2025).
5. Kinect DK / Microsoft Azure [Електронний ресурс]. – Режим доступу: https://azure.microsoft.com/ru-ru/products/kinect-dk (дата звернення: 24.04.2025).
6. Intel RealSense [Електронний ресурс]. – Режим доступу: https://www.intelrealsense.com/ (дата звернення: 24.04.2025).
7. MediaPipe Hand Landmarker / Google AI [Електронний ресурс]. – Режим доступу: https://ai.google.dev/edge/mediapipe/solutions/vision/hand_landmarker (дата звернення: 30.04.2025).
8. What is OpenPose? / RoboFlow Blog [Електронний ресурс]. – Режим доступу: https://blog.roboflow.com/what-is-openpose/ (дата звернення: 01.05.2025).
9. Fregoso J., Gonzalez C. I., Martinez G. E. A Survey on Human–Robot Collaboration: Models and Tools for Cognitive Human–Robot Interaction [Електронний ресурс] // 2021. – Vol. 10, №3. – Стаття 139. – Режим доступу: https://www.mdpi.com/2075-1680/10/3/139 (дата звернення: 09.05.2025).
10. What is ReLU and Sigmoid Activation Function? / NoMiDL [Електронний ресурс]. – Режим доступу: https://www.nomidl.com/deep-learning/what-is-relu-and-sigmoid-activation-function/ (дата звернення: 12.05.2025).

11. Dovetail Editorial Team. How to use Fitts' Law in UX design [Електронний ресурс] / Dovetail Editorial Team. – 27 квітня 2023. – Режим доступу: https://dovetail.com/ux/what-is-fitts-law/ (дата звернення: 15.05.2025).

12. Introduction to Human Computer Interface (HCI) / GeeksforGeeks [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/introduction-to-human-computer-interface-hci/ (дата звернення: 24.04.2025).

13. Vishnu, A.M. Dropout in Convolutional Neural Networks (CNN) [Електронний ресурс] // Medium. – Режим доступу: https://medium.com/@vishnuam/dropout-in-convolutional-neural-networks-cnn-422a4a17da41 (дата звернення: 14.05.2025).

14. Convolutional Neural Networks for Beginners / [Електронний ресурс]. – Режим доступу: https://serokell.io/blog/introduction-to-convolutional-neural-networks (дата звернення: 16.05.2025).

15. Rosebrock, A. Convolutional Neural Networks (CNNs) and Layer Types [Електронний ресурс] / Adrian Rosebrock // *PyImageSearch*. – 14 травня 2021. – Режим доступу: https://pyimagesearch.com/2021/05/14/convolutional-neural-networks-cnns-and-layer-types/ (дата звернення: 15.05.2025).

16. Pykes, K. AdamW Optimizer in PyTorch [Електронний ресурс] / Kurtis Pykes // *DataCamp*. – 21 жовтня 2024 – Режим доступу: https://www.datacamp.com/tutorial/adamw-optimizer-in-pytorch (дата звернення: 03.05.2025).

17. Van Otten, N. Cosine Annealing in Machine Learning Simplified: Understand How It Works [Електронний ресурс] / Neri Van Otten // *Spot Intelligence*. – 29 квітня 2024. – Режим доступу: https://spotintelligence.com/2024/04/29/cosine-annealing-in-machine-learning/ (дата звернення: 05.05.2025).

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

1

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ШТУЧНОГО ІНТЕЛЕКТУ

КВАЛІФІЦКАЦІЙНА РОБОТА на тему: «Система керування комп'ютером з використанням нейромережевої моделі розпізнавання жестів»

Виконав: здобувач вищої освіти
гр. ШД-42 Ярослав ЯРОШЕНКО
Керівник: кандидат технічних наук, доцент
Олександр ЗВЕНІГОРОДСЬКИЙ

2

Мета роботи – збільшення ефективності HCI при наборі текстів і взаємодії з графічними інтерфейсами

Об'єкт дослідження – людино-машинна взаємодія та її практичне застосування.

Предмет дослідження – алгоритми розпізнавання жестів.

Основні завдання кваліфікаційної роботи:

1. Провести аналіз існуючих систем для розпізнавання жестів
2. Розробити структуру програмної системи.
3. Вибрати програмні засоби для розробки системи.
4. Реалізувати програмну систему керування за допомогою жестів.
5. Провести тестування працездатності та точності системи в реальних умовах.

3

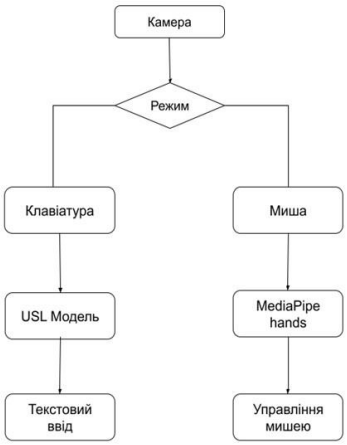
Існуючі системи

Leap Motion Controller 2

Intel RealSense Depth Camera

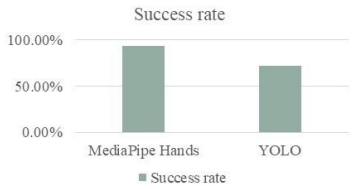
Azure Kinect DK

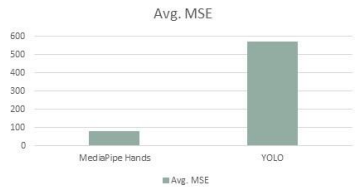
Структурна схема роботи системи

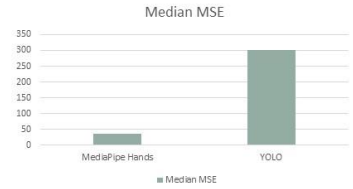


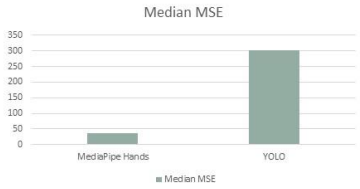
Робота в одному з двох режимів: введення тексту або керування мишею.
Залежно від вибраного режиму використовується або модель USL для розпізнавання жестової абетки, або MediaPipe Hands для взаємодії з курсором.

Порівняння моделей для розпізнавання рук





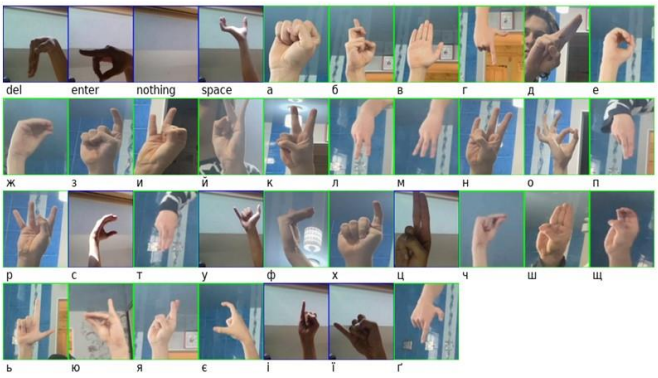




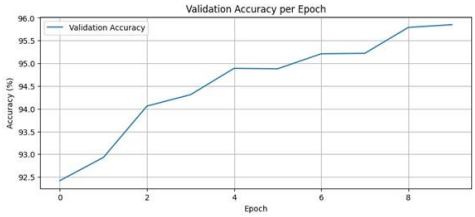
Використані технології

Python 3.10 — основна мова програмування.
MediaPipe Hands — визначення ключових точок руки в реальному часі.
OpenCV — попередня обробка зображень, захоплення відео з камери.
PyTorch — побудова та навчання згорткової нейронної мережі для розпізнавання жестів.
PyAutoGUI — емуляція руху миші, кліків, прокрутки.
Tkinter — створення GUI для перемикання режимів.

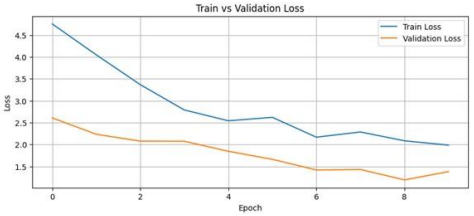
Приклади зображень з датасету



Графіки навчання моделі

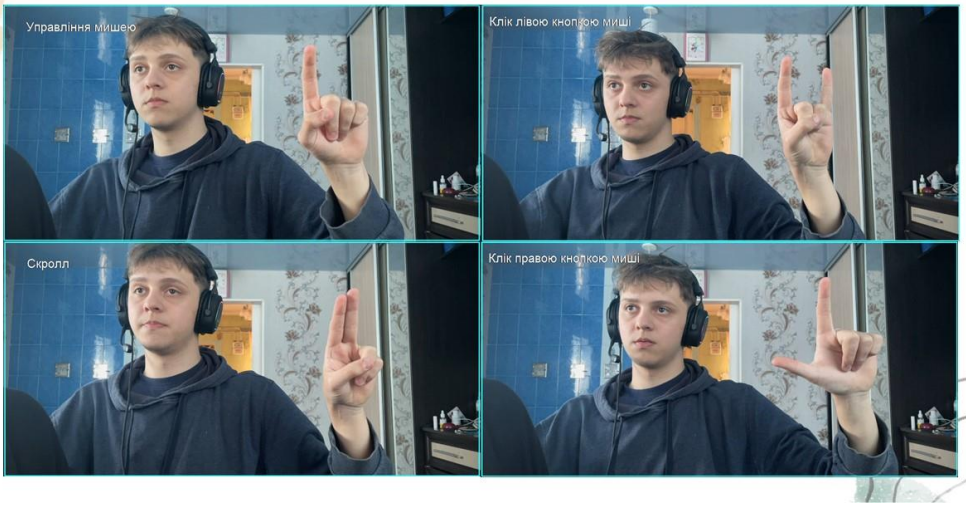


Графік точності показує поступове зростання на валідаційній вибірці.

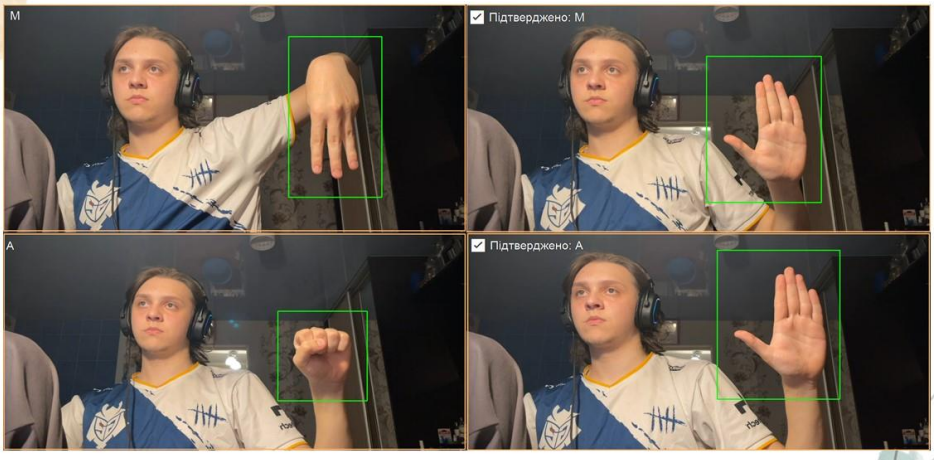


Графік втрат демонструє зменшення функції втрат із кожною епохою.

Демонстрація проекту



Демонстрація проекту



Висновки

- Розроблено функціональну систему керування комп’ютером за допомогою жестів.
- Проведено аналіз сучасних HCI-рішень.
- Вибрані програмні засоби для розробки системи.
- Створено датасет української жестової абетки.
- Навчено згорткову нейронну мережу для класифікації жестів