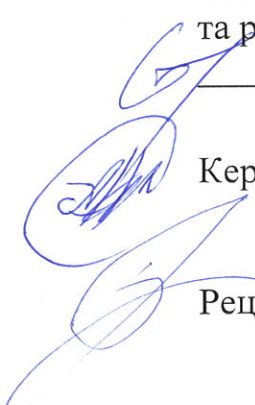


МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
Факультет комп'ютерно-інтегрованих технологій, мехатроніки і робототехніки
Кафедра робототехніки, електроенергетики та автоматизації
ім. проф. Б.Б. Самотокіна

ПОЯСНЮВАЛЬНА ЗАПИСКА
до кваліфікаційної роботи магістра
на тему: «Автоматизована система відстеження рухів рук для взаємодії з
елементами інтерфейсів»

Виконав студент 2 курсу, групи АТ-29м
спеціальності 174 “Автоматизація,
комп'ютерно-інтегровані технології
та робототехніка”

 Статкевич Іван Володимирович
(прізвище та ініціали студента)

Керівник: д., філос. Коавчук А.Р.
(науковий ступінь, вчене звання, посада,
прізвище та ініціали)

Рецензент: к.т.н., доц., Громовий О.А.
(науковий ступінь, вчене звання, посада,
прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ “ЖИТОМИРСЬКА ПОЛІТЕХНІКА”
Факультет комп’ютерно-інтегрованих технологій, мехатроніки і робототехніки
Кафедра робототехніки, електроенергетики та автоматизації
ім. проф. Б.Б. Самотокіна

Спеціальність 174 “Автоматизація, комп’ютерно-інтегровані технології та робототехніка”

ЗАТВЕРДЖУЮ

Завідувач кафедри робототехніки,
електроенергетики та автоматизації
ім. проф. Б.Б. Самотокіна
к.т.н., доц. Громовий О.А.

« 17 » лютого 2025р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра

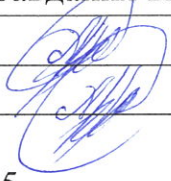
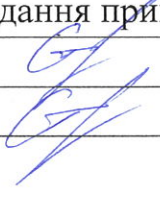
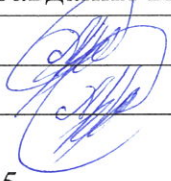
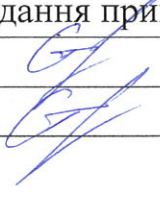
Студент Статкевич Іван Володимирович

1. Тема роботи: Автоматизована система відстеження рухів рук для взаємодії з елементами інтерфейсів
2. Термін здачі закінченої роботи на кафедру «1» грудня 2025 р.
3. Вихідні дані до роботи: матеріали звіту про переддипломну практику, вимоги до автоматизованого відстеження рухів рук для взаємодії з елементами інтерфейсів, відеодані для розпізнавання положення кисті та жестів, методи комп’ютерного бачення, архітектура та принципи роботи MediaPipe Hands, модель координат ключових точок кисті, параметри перетворення координат у координати екрана, методи згладжування координат і фільтрації шумів, вимоги до точності, швидкодії та стійкості роботи системи
4. Зміст пояснювальної записки (перелік питань, які підлягають розробці):
обґрунтування актуальності розробки та визначення сфер застосування системи, огляд і порівняння методів відстеження рук та розпізнавання жестів, аналіз архітектури й принципів роботи MediaPipe Hands, розробка моделі координат кисті та алгоритмів перетворення координат у координати екрана, визначення геометричних і кінематичних характеристик рухів руки, згладжування координат і фільтрація шумів, розробка алгоритмів розпізнавання жестів, розробка структурної, функціональної та алгоритмічної схем системи, побудова UML-

діаграми взаємодії, організація обробки жестів і логування даних, вибір інструментів та технологій, розробка програмних модулів системи, проведення експериментальних досліджень, аналіз та узагальнення результатів тестування.

5. Перелік презентативного матеріалу (з точним визначенням обов'язкових слайдів/креслень): структурна та функціональна схеми системи, схема ключових точок кисті, блок-схема алгоритму розпізнавання жестів, UML-діаграма взаємодії, приклади керування елементами інтерфейсів, результати експериментальних досліджень

6. Консультанти з кваліфікаційної роботи магістра із зазначенням розділів роботи, що їх стосуються

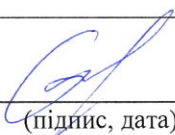
Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1, 2, 3,4	Кравчук А.Р.		
5,6,7	Кравчук А.Р.		

7. Дата видачі завдання 22.09.2025

Календарний план

№ з/п	Назва етапів виконання кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	Підготувати звіт про переддипломну практику	1 тиждень	виконано
2.	Збір та аналіз джерел за темою роботи, огляд існуючих систем	1 тиждень	виконано
3.	Вибір і обґрунтування технічних та програмних засобів	2 тиждів	виконано
4.	Розробка структурної, функціональної та блок-схеми	3 тиждів	виконано
5.	Аналіз отриманих результатів	4 тиждів	виконано
6.	Оформлення пояснювальної записки та презентаційного матеріалу	4 тиждів	виконано

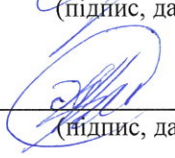
Студент


(підпис, дата)

І.В. Статкевич

(прізвище та ініціали)

Керівник роботи


(підпис, дата)

А.Р. Кравчук

(прізвище та ініціали)

ЗМІСТ

ВСТУП.....	9
1. АКТУАЛЬНІСТЬ РОЗРОБКИ СИСТЕМИ ВІДСТЕЖЕННЯ РУХІВ РУКАМИ ДЛЯ УПАВЛІННЯ ЕЛЕМЕНТАМИ ІНТЕРФЕЙСУ	11
1.1. Історія комп'ютерного бачення та жестових інтерфейсів	11
1.2. Практичне значення та сфери застосування розробленої системи..	12
2. ПОРІВНЯННЯ РІЗНИХ МЕТОДІВ ВІДСТЕЖЕННЯ РУКИ ТА РОЗПІЗНАВАННЯ ЖЕСТІВ	15
2.1. Сегментація за кольором шкіри.....	15
2.2. Контурний аналіз.....	16
2.3. Optical Flow	16
2.4. Фонове віднімання	17
2.5. CNN-архітектури	18
2.6. 3D Hand Pose Estimation	18
2.7. Огляд архітектури, та принципів роботи MediaPipe Hands	19
3. РОЗРОБКА ТА ДОСЛІДЖЕННЯ АЛГОРИТМІВ	24
3.1. Модель координата руки	24
3.2. Перетворення координат в координати екрану.....	24
3.3. Евклідова відстань між landmarkами	25
3.4. Кут між фалангами пальців	25
3.5. Центроїд кисті	26
3.6. Швидкість зміни координати	26
3.7. Згладжування координат та фільтрація шумів	27
3.8. Алгоритми розпізнавання жестів.....	29
4. РОЗРОБКА СХЕМ ТА ДІАГРАМ.....	33
4.1. Структурна схема системи	33
4.2. Функціональна схема системи.....	36
4.3. Алгоритмічна схема	38
4.4. UML-діаграма взаємодії (Sequence Diagram)	40

4.5. Схеми обробки жестів.....	41
4.6. Схеми логування даних.....	42
5. РОЗРОБКА АВТОМАТИЗОВАНОЇ СИСТЕМИ ВІДСТЕЖЕННЯ РУК ТА РОЗПІЗНАВАННЯ ЖЕСТІВ.....	46
5.1. Вибір інструментів та технологій.....	46
5.2. Структура програмного коду	47
5.3. Розробка модулів.....	48
6. ЕКСПЕРЕМЕНТАЛЬНА ЧАСТИНА	58
6.1. Завдання експерименту.....	58
6.2. Умови тестування.....	58
6.3. Опис тестових сценаріїв.....	59
6.4. Узагальнення результатів спостережень	60
ВИСНОВКИ	62
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	63

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

AR - Augmented Reality (Доповнена реальність)

VR - Virtual Reality (Віртуальна реальність)

Landmark - це опорна (ключова) точка, яка описує важливе місце на об'єкті та має координати.

ROI (Region of Interest) - це виділена область зображення, яка описується конкретними піксельними координатами і в якій виконується подальший аналіз.

ВСТУП

Сучасні інформаційні технології активно розвиваються, що змінює підходи до взаємодії людини з комп'ютером. Традиційні засоби керування, такі як клавіатура та миша, поступово доповнюються або замінюються більш природними способами, зокрема безконтактним керуванням за допомогою жестів руки.

Жестове керування є перспективним напрямом, оскільки дозволяє створювати інтуїтивні системи взаємодії, які можуть застосовуватися у віртуальній та доповненій реальності, медицині, промисловості та інших галузях. Такі системи базуються на аналізі відеопотоку зі звичайної камери та не потребують спеціального обладнання. Одним із поширених інструментів для цього є фреймворк Google MediaPipe Hands, що забезпечує визначення ключових точок кисті руки в реальному часі.

Водночас розробка надійної системи жестового керування пов'язана з технічними труднощами, зокрема обробкою шумів, згладжуванням рухів, розпізнаванням жестів і коректним перетворенням координат руки у координати елементів інтерфейсу.

Таким чином, створення системи відстеження рухів руки для взаємодії з користувацьким інтерфейсом є актуальним завданням, що поєднує методи комп'ютерного бачення, машинного навчання та програмної інженерії.

Актуальність роботи

Потреба у більш природних способах керування цифровими системами постійно зростає. Безконтактні інтерфейси набувають широкого поширення, оскільки вони забезпечують зручнішу та інтуїтивнішу взаємодію користувача з пристроями, дозволяють працювати без фізичного контакту, що є особливо важливим у медицині, промисловості та середовищах з підвищеною динамікою руху, а також знижують поріг входу у системи віртуальної та доповненої реальності. Додатковою перевагою таких рішень є можливість використання доступного обладнання, зокрема звичайної вебкамери.

Предмет дослідження

Методи та алгоритми виявлення, відстеження та інтерпретації рухів руки в 2D/3D-просторі для керування елементами інтерфейсу.

Мета роботи

Розробити автоматизовану програмну систему, що забезпечує відстеження рухів руки в реальному часі з використанням MediaPipe Hands та реалізує взаємодію з елементами інтерфейсів різних систем на основі розпізнаних жестів.

Наукова новизна полягає у формалізації набору жестів на основі геометричних і кінематичних параметрів кисті, поєднанні MediaPipe Hands із математичними моделями фільтрації та проєкції координат, а також у створенні масштабованої модульної архітектури системи жестового керування.

1. АКТУАЛЬНІСТЬ РОЗРОБКИ СИСТЕМИ ВІДСТЕЖЕННЯ РУХІВ РУКАМИ ДЛЯ УПРАВЛІННЯ ЕЛЕМЕНТАМИ ІНТЕРФЕЙСУ

1.1. Історія комп'ютерного бачення та жестових інтерфейсів

Комп'ютерне бачення (Computer Vision, CV) сформувалося як окрема наукова галузь у середині ХХ століття з метою навчити комп'ютери аналізувати та інтерпретувати зображення навколишнього середовища. Перші дослідження, що проводилися у 1960-1980-х роках, були зосереджені на базових задачах обробки зображень, таких як виділення контурів, сегментація та фільтрація шумів. У цей період були розроблені класичні алгоритми, зокрема Sobel, Canny та перетворення Гафа, які заклали основу подальшого розвитку комп'ютерного бачення.

У 1990-х роках, із розвитком обчислювальних ресурсів, з'явилися перші інтерактивні системи відстеження рухів руки. Вони переважно базувалися на сегментації за кольором шкіри та аналізі контурів, однак їхня ефективність суттєво залежала від умов освітлення, фону та параметрів камери.

На початку 2010-х років з'явилися перші спеціалізовані пристрої для жестового керування. Зокрема, у 2010 році була представлена система Microsoft Kinect, що використовувала глибинну камеру для розпізнавання рухів усього тіла, а у 2013 році - Leap Motion, високоточний інфрачервоний контролер, орієнтований на відстеження рухів кисті руки. Хоча ці рішення стали важливим кроком у розвитку gesture-control технологій, вони вимагали використання спеціального обладнання, що обмежувало їхню доступність у звичайних користувацьких умовах.

Суттєвий прорив відбувся після 2015 року з розвитком глибинного навчання. Застосування згорткових нейронних мереж дозволило виконувати аналіз відеопотоку в реальному часі на звичайних CPU та GPU. Це сприяло появі компактних і точних моделей для визначення ключових точок руки, які стали менш чутливими до фону, освітлення та кольору шкіри.

У 2019-2020 роках компанія Google представила фреймворк MediaPipe Hands, який став однією з найточніших систем відстеження рук на основі RGB-зображень. Система працює на звичайних камерах ноутбуків і мобільних пристроїв без використання додаткових сенсорів та отримала широке застосування у AR/VR-додатках, інтерфейсах із жестовим керуванням і наукових дослідженнях.

1.2. Практичне значення та сфери застосування розробленої системи

Розроблена в межах даної роботи система відстеження рухів руки та жестового керування має значний потенціал практичного застосування у промисловості, інформаційних технологіях, медицині, освіті та інших галузях. Її ключовою перевагою є можливість безконтактної взаємодії з цифровими інтерфейсами з використанням звичайної RGB-камери без потреби у спеціалізованому обладнанні.

Розглянемо детальніше та приведемо приклади для різних сфер застосування.

1.2.1. Застосування у промисловості

У промислових умовах розроблена система може використовуватися як альтернативний або допоміжний інтерфейс керування технологічними процесами. Зокрема, вона придатна для безконтактної роботи з операторськими панелями, що дозволяє зменшити залежність від традиційних пристроїв введення. Такий підхід є особливо актуальним у виробничих приміщеннях із підвищеними вимогами до чистоти, зокрема у харчовій, фармацевтичній та мікроелектронній промисловості.

1.2.2. Використання в системах автоматизації

Система може бути інтегрована в рішення Smart Home для керування пристроями за допомогою жестів. Вона дозволяє реалізувати керування мультимедійними системами, навігацію по інформаційних панелях, контроль освітлення, проведення презентацій та роботу з цифровими дошками.

Також варто зауважити, що система легко піддається розширенню та адаптації під різні вимоги до середовища де вона може бути застосована.

1.2.3. Застосування у медицині

У медичній сфері розроблений підхід може використовуватися для безконтактного керування медичними інформаційними системами в операційних залах. Це дає змогу лікарям і медичному персоналу взаємодіяти з комп'ютерними системами, не торкаючись клавіатури чи миші, що є особливо важливим під час проведення хірургічних втручань. Такий спосіб керування допомагає дотримуватися вимог стерильності, зменшує ризик забруднення обладнання та підвищує загальний рівень безпеки й зручності роботи в операційній.

1.2.4. AR/VR

Розроблена система може бути використана як базовий компонент для систем доповненої та віртуальної реальності. Жестове керування забезпечує можливість маніпуляції тривимірними об'єктами та створення способів навігації у віртуальних середовищах. Хоча в межах даної роботи реалізовано спрощену 2D/3D-модель, отримані результати можуть бути масштабовані та адаптовані для більш складних AR/VR рішень.

1.2.5. Використання в домашніх умовах

Розроблена система жестового керування може ефективно застосовуватися і в побутових умовах як засіб взаємодії з персональними комп'ютерами та

домашніми цифровими пристроями. Її ключовою перевагою є можливість безконтактного керування без використання традиційних пристроїв введення, що підвищує комфорт і доступність взаємодії для користувачів.

Загальний висновок до розділу 1

Аналіз розвитку комп'ютерного бачення та жестових інтерфейсів підтверджує актуальність теми дослідження. Перехід від класичних методів обробки зображень до сучасних нейромережевих моделей дозволив створювати точні й швидкодіючі системи жестового керування, що працюють на основі звичайних камер без спеціалізованого обладнання.

Сучасні підходи до відстеження рухів руки забезпечують інтуїтивну взаємодію з інтерфейсами в режимі реального часу та відкривають широкі можливості застосування у промисловості, медицині, AR/VR-середовищах і побутових умовах. Таким чином, розробка системи жестового керування є доцільною та має значний потенціал.

2. ПОРІВНЯННЯ РІЗНИХ МЕТОДІВ ВІДСТЕЖЕННЯ РУКИ ТА РОЗПІЗНАВАННЯ ЖЕСТИВ

Розглянемо декілька основних методів відстеження рук та розпізнавання жестів для порівняння та визначення оптимального рішення.

2.1. Сегментація за кольором шкіри

Алгоритми сегментації використовують діапазони значень у просторі HSV або YCrCb. Умови сегментації кольору шкіри в просторі HSV мають вигляд:

$$H \in [0^\circ, 50^\circ], S > 0.23, V > 0.35, \quad (2.1)$$

де

H (hue) - кут у колірному колі, що визначає відтінок. Діапазон **0-50°** охоплює теплу частину спектра (від червоного до жовтого), характерну для тону шкіри.

S (Saturation) - насиченість. Порогове значення **0.23** дозволяє усунути сіруватість та тіні.

V (Value) - яскравість. Умова **$V > 0.35$** відсіює темні області, що не можуть бути шкірою.

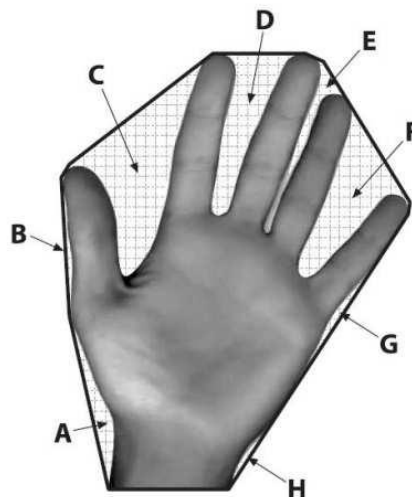
Такі обмеження дозволяють виділити пікселі, найбільш близькі до природних тонів шкіри.

Однак такий метод:

- нестійкий до тіней,
- залежить від локації,
- не працює при слабкому освітленні.

2.2. Контурний аналіз

Рука виділяється як силует, далі знаходиться опукла оболонка.



Виявлення пальців виконується за «дефектами опуклості», але метод ненадійний, особливо при складному фоні.

2.3. Optical Flow

Для аналізу зміни яскравості пікселів у просторі використовується градієнт інтенсивності зображення:

$$\nabla I(x, y) = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right), \quad (2.2)$$

де

$I(x, y, t)$ - інтенсивність пікселя в точці (x, y) у момент часу t .

Частинні похідні $\partial I / \partial x$, $\partial I / \partial y$ визначають зміну інтенсивності по горизонталі й вертикалі.

Вектор $\vec{v}(x, y)$ задає напрямок і швидкість локального переміщення точки між кадрами.

Метод дозволяє визначати рух, але **не визначає форму чи геометрію руки**, тому не підходить для детального жестового аналізу.

Використовується у статичних середовищах:

$$I_{fg} = | I_t - I_{background} | \quad (2.3)$$

Пояснення

I_t - поточний кадр.

$I_{background}$ - модель фону (статичного зображення поза рухом).

I_{fg} - Різниця між ними дає зображення рухомих об'єктів.

Метод добре працює в статичних сценах, але не придатний для складних жестів, бо рука рухається у тривимірному просторі та може частково перекриватися.

2.4. Фонове віднімання

Використовується у статичних середовищах:

$$I_{fg} = | I_t - I_{background} | \quad (2.4)$$

де

I_t - поточний кадр.

$I_{background}$ - модель фону (статичного зображення поза рухом).

I_{fg} - Різниця між ними дає зображення рухомих об'єктів.

Метод добре працює в статичних сценах, але не придатний для складних жестів, бо рука рухається у тривимірному просторі та може частково перекриватися.

Далі буде розглянуто більш новітні підходи що базуються на ML-моделях. У 2010-х роках convolutional neural networks стали основним інструментом для побудови pose estimation.

2.5. CNN-архітектури

Мережі аналізують зображення шляхом згортки:

$$Conv(I) = \sum_{i=1}^k \sum_{j=1}^k w_{ij} I_{ij}, \quad (2.5)$$

де

I_{ij} - значення пікселів у локальному вікні розміром $k \times k$.

w_{ij} - ваги фільтра (ядра згортки).

Результат - зважена сума, що визначає, наскільки локальний фрагмент відповідає певному патерну. Так мережа навчається впізнавати краї, текстури, контури пальців та долоні, що є базою для виявлення landmark'ів.

2.6. 3D Hand Pose Estimation

Тривимірна позиція руки визначається шляхом регресії:

$$(x_i, y_i, z_i) = f(I), \quad (2.6)$$

Функція f - це нейронна модель, яка:

- отримує зображення I ,
- оцінює положення кожної ключової точки руки,
- повертає 3D координати (x , y - у площині кадру, z - відносна глибина).

Таким чином система може визначати рух руки у тривимірному просторі (наближення, віддалення, pinch).

Висновок щодо існуючих методів розпізнавання руки та жестів

Класичні методи відстеження руки, такі як сегментація за кольором шкіри, контурний аналіз, оптичний потік і фонове віднімання, дозволяють виявляти рух або силует руки, проте мають суттєві обмеження. Вони залежать від умов освітлення й фону та не забезпечують точної інформації про положення пальців, що ускладнює розпізнавання складних жестів.

Сучасні підходи на основі згорткових нейронних мереж позбавлені цих недоліків, оскільки автоматично виділяють інформативні ознаки та стабільно визначають ключові точки кисті у просторі. Використання моделей 3D Hand Pose Estimation дозволяє аналізувати рух руки в глибині сцени, що є необхідним для реалізації точного й інтуїтивного жестового керування.

Далі розглянемо практичну реалізацію 3D Hand Pose Estimation, а саме MediaPipe Hands, яку будемо використовувати як основу для реалізації системи.

2.7. Огляд архітектури, та принципів роботи MediaPipe Hands

MediaPipe Hands складається з двох моделей:

1. Palm Detection (BlazePalm)

Виявляє долоню в кадрі.



Рис. 2.1. Виявлення долоні

2. Hand Landmark Model

Визначає 21 landmark руки:

$$L = \{(x_i, y_i, z_i) | i = 1..21\} \quad (2.7)$$

Ця множина містить 21 тривимірну точку, що відповідає суглобам пальців та зап'ястю.

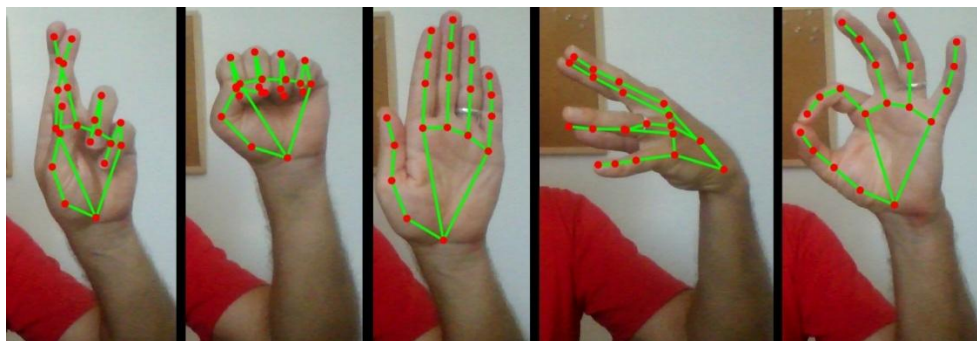


Рис. 2.3. 21 ключова точка кисті.

Схема роботи



Рис. 2.3 - Блок-схема пайплайну MediaPipe Hands.

1. Відеокадр →
2. Palm Detection →
3. ROI (region of interest) →
4. Landmark Model →
5. Нормалізовані координати →
6. Інтерпретація жестів.

Пояснення принципу роботи

MediaPipe повертає нормовані координати:

$$x, y \in [0; 1], \quad (2.8)$$

де

0 - ліва/верхня межа кадру,

1 - права/нижня межа.

Це дозволяє працювати з будь-якою роздільністю камери.

Для перетворення їх у координати екрану:

$$X_{\text{screen}} = x \cdot W_{\text{screen}} \quad (2.9)$$

$$Y_{\text{screen}} = y \cdot H_{\text{screen}},$$

де

$W_{\text{screen}}, H_{\text{screen}}$ - розміри монітора.

Перетворення масштабує нормовану координату до реальних пікселів екрана.

Відстань між landmarkами

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (2.10)$$

Ця евклідова відстань використовується для:

- визначення pinch-жесту (великий палець - вказівний),
- виявлення стискання/розтискання долоні,
- аналізу глибини руху.

Менше d - пальці зближуються (клік), більше - розтискаються.

Швидкість руху кисті:

$$v = \frac{d}{\Delta t}, \quad (2.11)$$

де

d - пройдена відстань між двома кадрами,

Δt - час між кадрами.

Використовується для визначення:

- свайпів,
- різких рухів,
- фільтрації шумів (занадто велика швидкість).

Використання MediaPipe Hands забезпечує ефективну реалізацію 3D Hand Pose Estimation на основі звичайного відеопотоку. Поєднання моделей виявлення долоні та визначення 21 ключової точки кисті дозволяє точно відстежувати положення і рух руки в реальному часі.

Загальний висновок до розділу 2

Порівняння методів відстеження руки показало, що класичні підходи комп'ютерного зору мають суттєві обмеження через залежність від умов освітлення та неможливість точно визначати положення пальців у просторі. Натомість методи на основі згорткових нейронних мереж забезпечують стабільне та точне визначення ключових точок кисті, включно з аналізом руху у глибині сцени.

Застосування підходів 3D Hand Pose Estimation, зокрема фреймворку MediaPipe Hands, дозволяє ефективно відстежувати руку та розпізнавати жести в реальному часі за допомогою звичайної камери, що підтверджує доцільність його використання для систем жестового керування інтерфейсом.

3. РОЗРОБКА ТА ДОСЛІДЖЕННЯ АЛГОРИТМІВ

Метою данного розділу є дослідження алгоритмів для відстеження жестів, згладжування координат та фільтрація шумів для коректної роботи системи.

3.1. Модель координта руки

В основі роботи системи лежить набір із 21 ключової точки (landmarks), які визначають геометричну структуру кисті. Кожна точка задається трійкою координат:

$$L_i = (x_i, y_i, z_i), \quad i = 1..21, \quad (3.1)$$

Кожна ключова точка руки подається як тривимірний вектор координат:

x_i - горизонтальне положення точки на зображенні;

y_i - вертикальне положення;

z_i - відносна глибина точки (чим менше значення, тим ближче точка до камери).

Така модель дозволяє описати геометрію кисті та аналізувати як положення окремих пальців, так і рух руки загалом. Наявність координати глибини дає змогу враховувати рух у тривимірному просторі, що важливо для жестів типу pinch або grab.

3.2. Перетворення координат в координати екрану

Оскільки MediaPipe Hands повертає нормалізовані координати у діапазоні $[0;1]$, для керування курсором необхідно виконати перетворення у піксельні координати екрана:

$$X_{screen} = x_i \cdot W_{screen}, \quad (3.2)$$

$$Y_{screen} = y_i \cdot H_{screen}.$$

Далі потрібно перетворити нормовані координати в координати реального екрана.

Це здійснюється шляхом масштабування на розміри дисплея:

- W_{screen} - ширина монітора;
- H_{screen} - висота монітора.

Таке масштабування забезпечує незалежність системи від роздільної здатності камери та дозволяє коректно працювати на різних пристроях без додаткового калібрування.

3.3. Евклідова відстань між landmarkами

Для аналізу взаємного розташування пальців використовується евклідова відстань між двома ключовими точками:

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}, \quad (3.3)$$

Відстань між двома ключовими точками використовується для:

- визначення жесту pinch (зближення великого та вказівного пальців),
- аналізу ступеня стискання чи розгинання пальців,
- аналізу жестів на основі положення точок.

Використання евклідової відстані дозволяє реалізувати прості та надійні правила класифікації жестів без складних моделей машинного навчання.

3.4. Кут між фалангами пальців

Для більш точного визначення положення пальців використовується обчислення кута між двома векторами, що відповідають фалангам пальця:

$$\vec{a} = L_j - L_i, \vec{b} = L_k - L_j, \quad (3.4)$$

Кут між ними визначається за формулою:

$$\cos (\theta)=\frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \cdot\|\vec{b}\|}, \quad (3.5)$$

Це дозволяє визначати:

- розгинання пальця,
- згинання,
- напрям.

Обчислення кутів є особливо корисним для відрізнєння схожих жестів, де відстані між пальцями можуть бути подібними.

3.5. Центроїд кисті

Центроїд кисті визначається як середнє значення координат усіх landmark-ів:

$$C=\frac{1}{21} \sum_{i=1}^{21} L_i, \quad (3.6)$$

Центроїд - це середнє положення всіх landmark'ів.

Він використовується для:

- визначення загального положення кисті,
- трекінгу руху руки як цілісного об'єкта,
- визначення глобальних жестів (наприклад, swipe всією кистю).

Застосування центроїда дозволяє зменшити вплив дрібних рухів окремих пальців та підвищити стабільність розпізнавання глобальних жестів.

3.6. Швидкість зміни координати

Швидкість руху руки або окремої точки визначається як:

$$v=\frac{d}{\Delta t}, \quad (3.7)$$

Швидкість зміни положення точки використовується для:

- визначення жестів типу swipe,
- відсікання випадкових ривків,
- аналізу плавності руху.

Використання швидкості дозволяє розрізняти навмисні жести та дрібні неконтрольовані рухи.

3.7. Згладжування координат та фільтрація шумів

Відстеження руки супроводжується шумами: дрібні рухи, вібрації, нестабільність моделі. Тому необхідне згладжування сигналу. Розглянемо необхідні алгоритми фільтрації та сгладжування.

3.7.1. Фільтр ковзного середнього

Для координат:

$$x'_t = \frac{1}{n} \sum_{k=0}^{n-1} x_{t-k}, \quad (3.8)$$

де n - розмір вікна.

Фільтр обчислює середнє значення останніх n координат.

Це:

- згладжує “тремтіння” руки,
- робить курсор стабільним,
- прибирає шум відеокадрів.

Згладжує координати шляхом усереднення останніх n значень. Дає високу стабільність, але вводить невелику затримку.

3.7.2. Експоненційне згладжування

$$x'_t = \alpha x_t + (1 - \alpha)x'_{t-1}, \quad (3.9)$$

Це рекурсивний фільтр, де:

$\alpha \in (0,1)$ - параметр згладжування,

нове значення залежить від поточного та попереднього згладженого.

Переваги:

- мінімальна затримка,
- дуже плавний рух,

Рекурсивний фільтр, що забезпечує баланс між плавністю та швидкістю реакції. Найбільш ефективний для керування курсором.

3.7.3. Сигма-фільтрація для відхилень

$$|x_t - x'_{t-1}| < \sigma \Rightarrow x'_t = x'_{t-1}, \quad (3.10)$$

де

x_t - нове виміряне значення координати;

x'_{t-1} - попереднє згладжене значення;

σ - допустиме відхилення (поріг чутливості).

Використовується для відсікання різких викидів, що виникають через помилки детекції. Якщо відхилення перевищує заданий поріг, нове значення ігнорується.

Нижче наведено загальний висновок щодо використання фільтрів у вигляді таблиці.

Таблиця 3.1 - Фільтри

Фільтр	Затримка	Якість	Використання
Ковзне середнє	Середня	Висока	стабільні жести
Експоненційне	Низька	Середня	курсор
Sigma-фільтр	Низька	Середня	пінч, клік

3.8. Алгоритми розпізнавання жестів

Розглянемо алгоритми які будемо використовувати для розпізнавання жестів.

3.8.1. Pinch жест

Pinch - визначається за зменшенням відстані між кінчиками великого та вказівного пальців.

Палець 4 (*thumb_tip*) та 8 (*index_tip*):

$$d_{4,8} < T_{pinch}, \quad (3.11)$$

де T_{pinch} - поріг.

Жест *pinch* відбувається тоді, коли відстань між кінчиком великого пальця (landmark 4 - *thumb_tip*) та кінчиком вказівного пальця (landmark 8 - *index_tip*) стає меншою за порогове значення T_{pinch} .

3.8.2. Scroll жест

Scroll - базується на вертикальній швидкості руху центроїда кисті.

Вертикальна швидкість центроїда:

$$v_y = \frac{y_t - y_{t-1}}{\Delta t}, \quad (3.12)$$

Жест:

$$|v_y| > T_{scroll}, \quad (3.13)$$

Центроїд кисті рухається ввверх/вниз із певною швидкістю:

- якщо вертикальна швидкість $|v_y|$ перевищує поріг T_{scroll} ,
- система інтерпретує це як жест прокручування (*scroll up/down*).

Цей підхід дозволяє розпізнавати вертикальні рухи всієї кисті, а не окремого пальця, що робить жест стабільнішим.

3.8.3. Swipe жест

Swipe - визначається за різким переміщенням руки у горизонтальному або вертикальному напрямку.

Горизонтальна або вертикальна швидкість:

$$v_x > T_{swipe} \text{ або } v_y > T_{swipe}, \quad (3.14)$$

Жест *swipe* визначається за швидким переміщенням кисті:

- по горизонталі (v_x),
- або по вертикалі (v_y).

Якщо швидкість перевищує порогове значення T_{swipe} , система реєструє жест “змахування” - аналог жестів в інтерфейсах смартфонів.

Цей метод працює навіть при невеликій амплітуді руху, оскільки визначальним є саме темп, а не дистанція.

3.8.4. Grab / Release жест

Grab / Release - аналізується за відстанями між пальцями та центроїдом кисті.

Пальці наближаються до центроїда:

$$d_{iC} < T_{grab}, \quad (3.15)$$

Для жесту *grab* (стискання кисті) аналізується відстань між:

- кожним пальцем i ,
- та центроїдом кисті C .

Якщо для більшості пальців:

$$d_{iC} < T_{grab}, \quad (3.16)$$

то пальці вважаються зігнутими всередину - кисть “стиснута” - **grab**.

Якщо значення зворотні (дистанції збільшуються) - **release**.

Такий підхід моделює природну жестикуляцію “захопити / відпустити” без потреби в детальному аналізі кожної фаланги.

Загальні висновки до розділу 3

У цьому розділі було розроблено та розглянуто основні алгоритми, необхідні для роботи системи відстеження руки та розпізнавання жестів у реальному часі.

Перетворення координат руки в координати екрана, а також обчислення відстаней, кутів і швидкості руху дають можливість визначати жести користувача. Для підвищення стабільності роботи системи були застосовані методи згладжування та фільтрації шумів, що дозволяє зменшити вплив випадкових коливань і помилок розпізнавання.

Реалізовані жести базуються на простих і зрозумілих характеристиках руху кисті, завдяки чому система є зручною у використанні.

4. РОЗРОБКА СХЕМ ТА ДІАГРАМ

Метою цього розділу є побудова та опис основних схем системи відстеження рухів руки. Розглядаються структурна, функціональна та алгоритмічна схеми, а також UML Sequence Diagram, схема логування та обробки жестів які допомагають зрозуміти загальну архітектуру системи, взаємодію між її компонентами, обробку даних і послідовність виконання операцій.

4.1. Структурна схема системи

Структурна схема описує архітектуру системи на верхньому рівні, показуючи її основні модулі, функціональні блоки та їхні взаємозв'язки.

Система складається із таких функціональних вузлів:

Відеокамера – відповідає за захоплення відеопотоку в реальному часі з будь-якої RGB-камери. Як правило, використовуються камери з роздільною здатністю 720p або 1080p, які є стандартними для більшості ноутбуків і персональних комп'ютерів, а також доступні як окремі пристрої для інтеграції в інші системи. Данні з камери передаються в модуль препроцесингу (попередньої обробки).

Модуль препроцесингу – відповідає за нормалізацію освітлення, перетворення кольорових просторів з RGB в BGR. В нашій системі роль препроцесингу виконує MediaPipe.

MediaPipe Hands – це головний компонент, що виконує детекцію долоні, виділяє ROI, визначає 21 ключову точку кисті, повертає нормовані координати. Це дозволяє отримати стабільну модель руки в режимі реального часу та використовувати її для розпізнавання жестів і керування інтерфейсом.

Модуль згладжування – приймає координати landmarkів та стабілізує їх, зменшує тремтіння курсора та шуму у даних.

Gesture Detector – визначає який жест виконує користувач: pinch, scroll, swipe, grab/release, алгоритм базується на порогових значеннях.

UI Controller – перетворює жести на події інтерфейсу, такі як, переміщення курсора, клік, скрол, свайп, це компонент який взаємодії з операційною системою та може бути переписаний та адаптований залежно від того в яку систему інтегрується розробка.

Операційна система – система в якій виконуються події, в данному прикладі використовується OS Windows, але розробка може інтегруватись в інші системи, завдяки модульній архітектурі може бути швидко адаптована та запущена на таких пристроях як Raspberry PI, Nvidia Jetson та ін, що дуже корисно при інтеграції в промислові процеси та автоматизовані системи.

Візуалізація – опційна частина яка не впливає на роботу системи, використовується для відображення скелетону руки, FPS, активного жесту. Може бути розширена за потреби, або взагалі вимкнена через конфігураційні файли.

Logger – зберігає данні про роботу систми для подальшого аналізу та пошуку помилок, дебагу. Зберігає FPS, координати, визначені жести, затримки, помилки. Може бути налаштована або вимкнена через конфігураційні файли.

Структурна схема відображає архітектуру системи на верхньому рівні, показуючи її основні модулі, функціональні блоки та як вони пов'язані. Нижче показано структурну схему, також зі схемою можна ознайомитись в **Додатку А**.

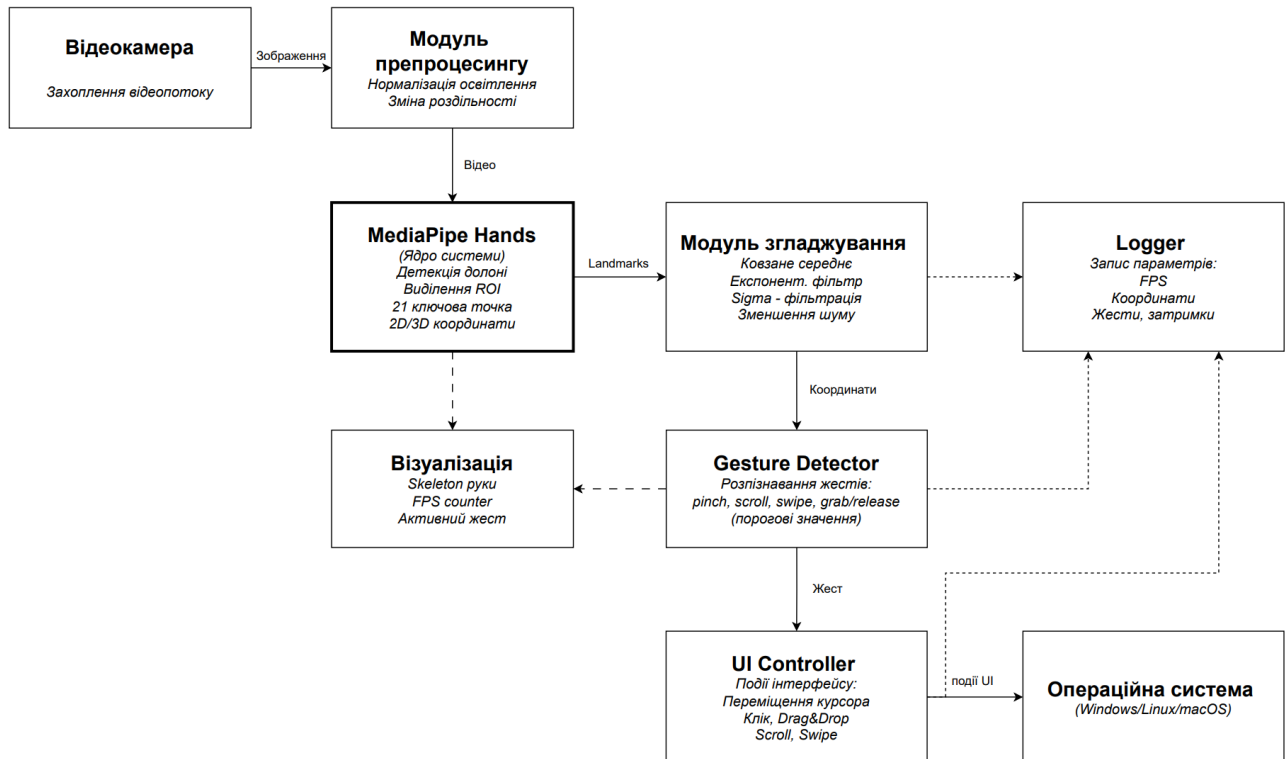


Рис. 4.1. Структурна схема системи жестового керування.

На основі описаних вище модулів побудована структурна схема роботи системи.

MediaPipe Hands є ядром системи яке отримує вхідні данні від камери і виконує основні операції виявлення руки та розпізнавання 21 ключової точки.

Далі данні надходять до модуля Gesture Detector, який розпізнає жести та передає виконання до UI Controller який в свою чергу взаємодії з OS та виконує такі дії як клік, свайп, скрол і тд.

Також система має модуль логування та візуалізації які полегшують аналіз роботи системи та пошук помилок.

Потік даних виглядає так:

1. Камера → MediaPipe: передається зображення.
2. MediaPipe → Smoothing: передаються landmarkи.
3. Smoothing → Gesture Detector: стабілізовані координати аналізуються.
4. Gesture Detector → UI Controller: класифікований жест.

5. UI Controller → ОС: інтерфейсна подія.
6. Будь-який модуль → Logger: робоча інформація для аналізу.

4.2. Функціональна схема системи

Функціональна схема показує, як працює система з точки зору виконуваних функцій і операцій, а не окремих програмних модулів. Вона відображає послідовність обробки даних, основні етапи перетворення інформації та зв'язки між ними - від отримання вхідних даних до формування кінцевого результату.

Функціональна схема складається з таких функціональних блоків:

Отримання кадру – функція *capture_frame()* отримує зображення від камери.

Визначення руки – функція *detect_hand(frame)* отримує фрейм та знаходить долоню, обчислює bounding box, передає данні далі для отримання landmarkів.

Обчислення landmarkів – функція *extract_landmarks(frame)* обчислює та повертає координати 21 точки руки, перевіряє чи рука є в кадрі.

Стабілізація координат – функція *smoth(landmarks)* видаляє шуми, згладжує траєкторію, та передає виконання далі для знаходження жемстів.

Інтерпретація жестів – функція *classify_gesture(smothed_landmarks)* визначає жести: pinch, swipe, scroll, grab та викликає функцію виконання дії.

Виконання дії – функція *execute_action(gesture)* викликає виконання дії в системі куди інтегрована розробка, в нашому прикладі це OS Windows, але в інших випадках функція може бути адаптована залежно від системи.

Логування – опціональна функція *log_data()* яка викликається якщо увімкнено потрібне налаштування в конфігурації, може бути адаптована під потреби користувача або системи в яку розробка інтегрується. Детільніше процес

логування розглядаєть далі. Основна задача логуювання – це збереження даних про роботу системи.

Нижче показана функціональна схема процесу відстеження руки, також зі схемою можна знайомитись в Додатку В.

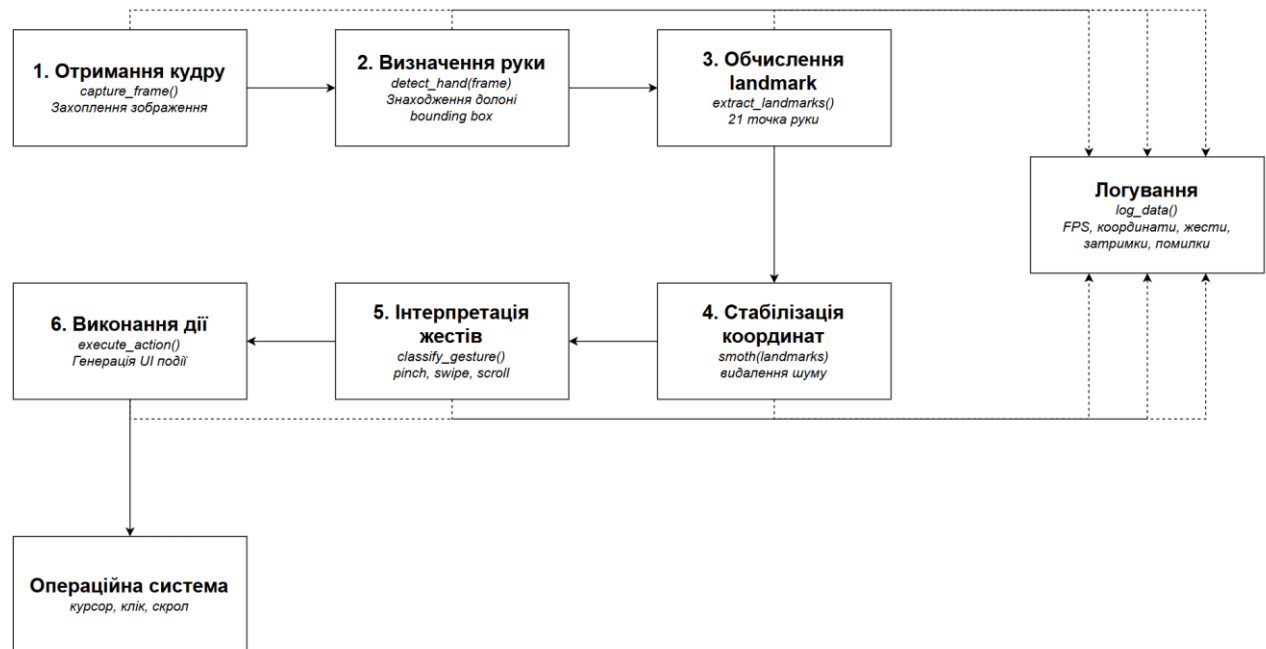


Рис. 4.2. Функціональна схема процесу відстеження руки.

Потік функцій

Отримання кадру → Виявлення → Landmarkи → Згладжування → Жест → Дія → Логуювання.

Функціональна схема показує роботу системи як послідовний процес обробки даних - від отримання відеокадру до виконання відповідної дії в операційній системі.

Виділення окремих етапів дозволяє легко простежити, як інформація перетворюється на керуючу дію, а використання згладжування координат забезпечує більш стабільну роботу та коректне розпізнавання жестів.

Наявність опціонального блоку логуювання спрощує аналіз і налагодження системи, що робить схему зрозумілою, гнучкою та придатною для подальшого розширення.

4.3. Алгоритмічна схема

Алгоритмічна схема показує послідовність виконання операцій у вигляді блок-схеми та відображає логіку роботи алгоритму, включаючи основні обчислювальні кроки, умови переходів і циклічні процеси.

Опис основних етапів алгоритму

1. Старт системи
Ініціалізуються камери, моделі MediaPipe, змінні.
2. Читання нового кадру
3. Чи знайдено руку?
Якщо ні → повернутися до читання кадру.
Якщо так → перейти до обчислення landmarkів.
4. Обчислення landmarkів - MediaPipe повертає 21 точку.
5. Згладжування координат: застосовується фільтр, зменшується “тремтіння”.
6. Класифікація жесту
7. Виклик UI-дії якщо жест pinch → клік, swipe → навігація, scroll → прокрутка, grab → drag&drop.
8. Логування
9. Повернення до циклу

Алгоритмічна схема зображена нижче, також зі схемою можна ознайомитись в Додатку С.

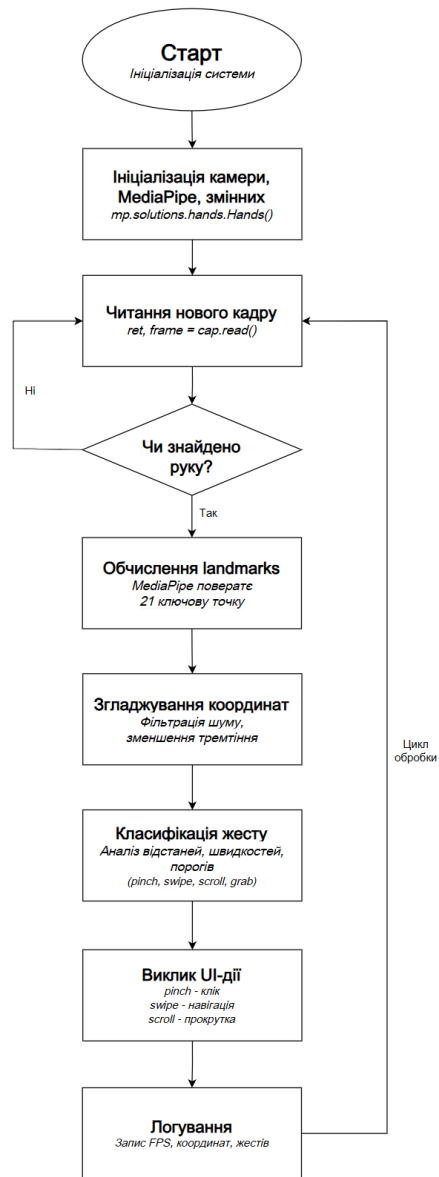


Рис. 4.3. Алгоритмічна схема роботи системи.

Алгоритм складається з дев'яти логічних етапів, компоненти працюють у циклі реального часу.

Вихід з циклу відбувається вручну, додаткові можливості зупинки роботи системи розробляються додатково, залежно від системи куди інтегрується розробка.

4.4. UML-діаграма взаємодії (Sequence Diagram)

Sequence Diagram показує порядок викликів між компонентами системи у часі та відображає обмін повідомленнями під час виконання сценарію. Це дозволяє наочно простежити, як елементи системи взаємодіють між собою в процесі роботи.

Діаграму зображено на риснку нижче, також з діаграмою можна ознайомитись в Додатку D.

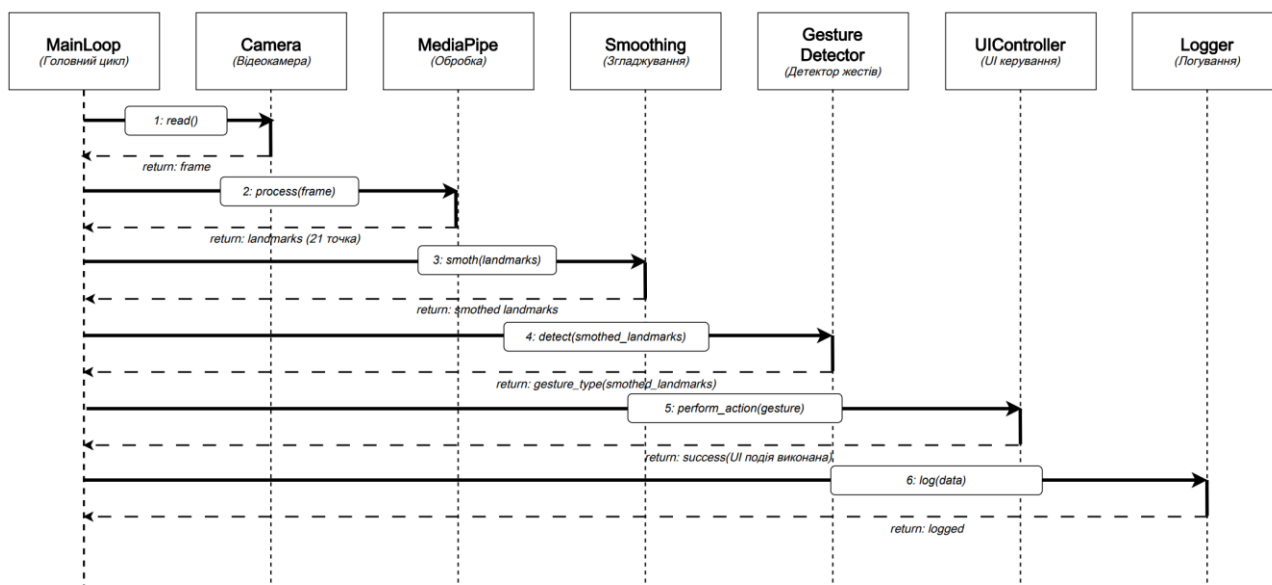


Рис. 4.3. UML-діаграма взаємодії (Sequence Diagram).

Опис взаємодії:

1. MainLoop викликає *Camera.read()*
2. Передає кадр у *MediaPipe.extract_landmarks()*
3. Результат передається в *Smoothing.smooth()*
4. Потім у *GestureDetector.classify_gesture()*
5. Далі — у *UIController.execute_action()*
6. Наприкінці кожної ітерації дані передаються у *Logger.log_data()*

Описана взаємодія компонентів забезпечує послідовну обробку даних у реальному часі - від отримання відеокадру до виконання відповідної дії та

збереження результатів. Такий підхід дозволяє підтримувати стабільну та керовану роботу системи.

4.5. Схема обробки жестів

Ця схема показує логічну структуру аналізу жестів всередині Gesture Detector. Схема показна на рисунку нижче, також зі схемою можна ознайомитись в Додатку Е.

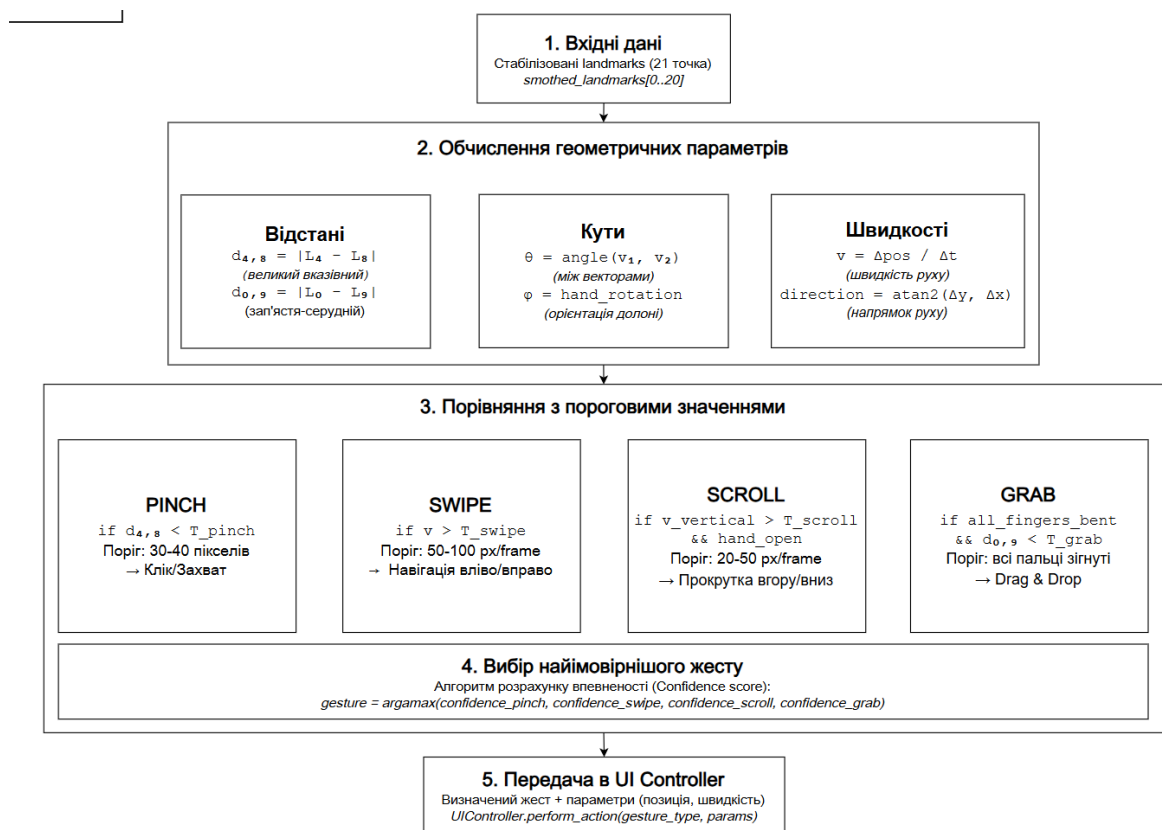


Рис. 4.4. Логічна структура аналізу жестів.

Процес:

- 1. Отримання стабілізованих landmarkів
- 2. Обчислення геометричних параметрів: відстані, кути, швидкості.
- 3. Порівняння з порогами: pinch якщо $d_{4,8} < T_{pinch}$, swipe якщо $v > T_{swipe}$.
- 4. Вибір найімовірнішого жесту
- 5. Передача в UI Controller

4.6. Схема логування даних

Схема відображає структуру даних, що записує Logger. Збирає та записує дані роботи системи в реальному часі.

Формат: *JSON / CSV*

Файл: *gesture_log_YYYYMMDD_HHMMSS.json*

Частота запису: Після закінчення кожної ітерації циклу

6. Таблиця 4.1 Схема логування.

Поле	Значення / Опис	Приклад
timestamp (часова мітка)	Час запису події Формат: <i>ISO 8601</i> Використовується для аналізу послідовності подій	"2025-12-12T14:23:45.123Z" або 1702389825.123
gesture (тип жесту)	Класифікований жест користувача Можливі значення: "pinch", "swipe_left", "swipe_right", "scroll_up", "scroll_down", "grab", "release", "none"	"pinch" "swipe_left" "none"

fps <i>(кадри/секунду)</i>	Частота кадрів на момент запису Використовується для аналізу продуктивності Діапазон: 15-60 FPS	58.3 32.1
landmarks <i>(координати)</i>	Масив координат 21 ключової точки руки Формат: [{x, y, z}, ...] або [x0, y0, z0, x1, y1, z1, ...] Нормалізовані координати (0.0 - 1.0) або абсолютні пікселі Можна зберігати згладжені або сирі дані	[[0.512, 0.634, 0.001],[0.498, 0.521, 0.003],... (21 точка)]
latency <i>(затримка)</i>	Час обробки кадру (мілісекунди) Від захоплення кадру до виконання UI дії Критично для real-time відгуку (ціль: < 50ms)	23.4(ms)
events <i>(UI події)</i>	Список виконаних UI подій Формат: [{action, params}, ...] Типи: "mouse_move", "mouse_click", "scroll", "key_press"	[{"action": "mouse_move","x": 640, "y": 480},{ "action": "click","button": "left"}]

	Включає параметри: координати курсора, силу прокрутки, тощоДозволяє відтворити взаємодію користувача	
--	---	--

Приклад запису в JSON форматі:

```
{
  "timestamp": "2025-12-12T14:23:45.123Z",
  "gesture": "pinch",
  "fps": 58.3,
  "landmarks": [
    [
      0.512, 0.634,
      0.001
    ], ...
  ],
  "latency": 23.4,
  "events": [
    {
      "action": "mouse_click",
      "button": "left",
      "x": 640,
      "y": 480
    }
  ]
}
```

Загальний висновок до розділу 4

У цьому розділі було розглянуто архітектуру та принципи роботи системи відстеження рухів руки на основі побудови структурної, функціональної й алгоритмічної схем, а також UML-діаграми взаємодії. Такий підхід дозволив описати систему як з точки зору її програмної структури, так і з позиції обробки даних у реальному часі.

У сукупності наведені схеми дають цілісне уявлення про роботу системи та будують фундамент для розробки самої системи.

5. РОЗРОБКА АВТОМАТИЗОВАНОЇ СИСТЕМИ ВІДСТЕЖЕННЯ РУК ТА РОЗПІЗНАВАННЯ ЖЕСТІВ

У межах роботи буде розроблена модульна архітектура системи відстеження та інтерпретації жестів руки. Для реалізації системи відстеження жестів був використаний стек технологій, який оптимально поєднує продуктивність, простоту та можливість швидкого прототипування.

5.1. Вибір інструментів та технологій

Основні технології:

1. **Python 3.x** - було обрано як основну мову програмування через простоту використання та наявність великої кількості бібліотек для комп'ютерного бачення.
2. **OpenCV** - використовується для роботи з відеопотоком у реальному часі. Вона відповідає за захоплення кадрів із камери, базову обробку зображень і відображення результатів роботи системи.
3. **MediaPipe Hands** - застосовується для відстеження руки та визначення 21 ключової точки кисті за допомогою звичайної RGB-камери. Перевагою цієї технології є готовий оптимізований алгоритм, який стабільно працює за різних умов освітлення та фону.
4. **PyAutoGUI** - використовується для взаємодії з операційною системою. Вона дозволяє перетворювати розпізнані жести на реальні дії, такі як переміщення курсора, кліки миші, прокрутка або натискання клавіш. Завдяки цьому система може керувати будь-якими програмами без створення окремого користувацького інтерфейсу.

5. **NumPy** - застосовується для виконання математичних обчислень над координатами ключових точок руки. За її допомогою реалізовано обчислення відстаней, кутів, швидкості руху та нормалізацію координат.
6. **JSON** логування - збір даних для аналізу та дебагу.

5.2. Структура програмного коду

Додаток складається з декількох Python-файлів:



Рис 6.1. Файли системи

Кожен модуль має чітку відповідальність, що покращує тестованість та гнучкість системи. Далі розглянемо кожен файл окремо.

5.3. Розробка модулів

5.3.1. Модуль конфігурації (config.py)

Файл *config.py* містить набір основних параметрів, які визначають роботу системи без необхідності змінювати програмний код. До таких параметрів належать:

- параметри камери;
- масштаб обробки зображення;
- пороги детекції та трекінгу;
- коефіцієнти згладжування;
- параметри відображення та логування.

Такий підхід дозволяє швидко адаптувати систему під різні апаратні умови та сценарії використання.

```
CAMERA_WIDTH = 320
CAMERA_HEIGHT = 240
CAMERA_FPS = 60

PROCESSING_SCALE = 0.45
SKIP_FRAMES = 0

MAX_HANDS = 1
DETECTION_CONFIDENCE = 0.5
TRACKING_CONFIDENCE = 0.3
MODEL_COMPLEXITY = 0
REFINE_LANDMARKS = False

CURSOR_SMOOTHING_ALPHA = 0.2
LANDMARK_SMOOTHING_ALPHA = 0.4
You, 33 minutes ago • Refac

SHOW_WINDOW = False
SHOW_LANDMARKS = False
SHOW_CONNECTIONS = False
SHOW_FPS = True
SHOW_GESTURE = True
DISPLAY_SCALE = 1.0

ENABLE_LOGGING = False
LOG_FILE = "gesture_log.jsonl"
```

Рис 6.2. Модуль конфігурації

Для детального ознайомлення відкрийте файл *config.py* в Додатку F.

5.3.2. Профілі продуктивності (performance_profiles.py)

Для експериментів з продуктивністю реалізовано набір профілів роботи:

- *ULTRA_PERFORMANCE*
- *HIGH_PERFORMANCE*
- *BALANCED*
- *HIGH_QUALITY*

Кожен профіль є словником параметрів, який може бути застосований до конфігурації системи через функцію `apply_profile()`.

Це дозволяє:

- порівнювати FPS та точність;
- проводити експерименти для Розділу 6;
- демонструвати адаптивність системи.

```
▼ ULTRA_PERFORMANCE = {  
    'CAMERA_WIDTH': 320,  
    'CAMERA_HEIGHT': 240,  
    'PROCESSING_SCALE': 0.4,  
    'SKIP_FRAMES': 1,  
    'DETECTION_CONFIDENCE': 0.3,  
    'TRACKING_CONFIDENCE': 0.3,  
    'MODEL_COMPLEXITY': 0,  
    'SHOW_LANDMARKS': False,  
    'ENABLE_LOGGING': False,  
    'DISPLAY_SCALE': 0.75,  
}
```

Рис 6.3. *ULTRA_PERFORMANCE* конфігурація

```
HIGH_PERFORMANCE = {  
    'CAMERA_WIDTH': 320,  
    'CAMERA_HEIGHT': 240,  
    'PROCESSING_SCALE': 0.5,  
    'SKIP_FRAMES': 0,  
    'DETECTION_CONFIDENCE': 0.3,  
    'TRACKING_CONFIDENCE': 0.3,  
    'MODEL_COMPLEXITY': 0,  
    'SHOW_LANDMARKS': False,  
    'ENABLE_LOGGING': False,  
    'DISPLAY_SCALE': 1.0,  
}
```

Рис 6.4. HIGH_PERFORMANCE конфігурація

```
BALANCED = {  
    'CAMERA_WIDTH': 640,  
    'CAMERA_HEIGHT': 480,  
    'PROCESSING_SCALE': 0.65,  
    'SKIP_FRAMES': 0,  
    'DETECTION_CONFIDENCE': 0.4,  
    'TRACKING_CONFIDENCE': 0.4,  
    'MODEL_COMPLEXITY': 0,  
    'SHOW_LANDMARKS': False,  
    'ENABLE_LOGGING': False,  
    'DISPLAY_SCALE': 1.0,  
}
```

Рис 6.5. BALANCED конфігурація

```
HIGH_QUALITY = {  
    'CAMERA_WIDTH': 640,  
    'CAMERA_HEIGHT': 480,  
    'PROCESSING_SCALE': 0.8,  
    'SKIP_FRAMES': 0,  
    'DETECTION_CONFIDENCE': 0.6,  
    'TRACKING_CONFIDENCE': 0.6,  
    'MODEL_COMPLEXITY': 1,  
    'SHOW_LANDMARKS': True,  
    'ENABLE_LOGGING': True,  
    'DISPLAY_SCALE': 1.0,  
}
```

Рис 6.6. *HIGH_QUALITY* конфігурація

Для детального ознайомлення відкрийте файл **performance_profiles.py** в Додатку F.

5.3.3. Реалізація модуля відстеження руки (hand_detector.py)

Модуль *hand_detector.py* відповідає за:

- захоплення та обробку відеокадру;
- детекцію руки за допомогою MediaPipe Hands;
- отримання координат 21 ключової точки кисті;
- оптимізоване отримання координат у двох системах.

Детекція руки

Реалізовано метод `find_hands()` який знаходить руку в кадрі. Для кожного кадру виконується:

- Перетворення BGR $\rightarrow$ RGB.
- Передача кадру в MediaPipe Hands.
- Визначення наявності руки.

```

def find_hands(self, frame, draw: bool = True):
    """
    Processes the frame, finds the hand, optionally draws the skeleton.
    :param frame: BGR image (frame from the camera)
    :param draw: whether to draw landmarks on the frame
    :return: modified frame
    """
    img_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    self.results = self.hands.process(img_rgb)

    if self.results.multi_hand_landmarks and draw:
        for hand_lms in self.results.multi_hand_landmarks:
            self.mp_draw.draw_landmarks(
                frame,
                hand_lms,
                self.mp_hands.HAND_CONNECTIONS,
                self.mp_draw.DrawingSpec(color=(0, 255, 0), thickness=2, circle_radius=3),
                self.mp_draw.DrawingSpec(color=(0, 0, 255), thickness=2, circle_radius=2),
            )
    return frame

```

Рис 6.7. Детекція руки

Отримання координат landmarkів

Реалізовано метод `get_landmarks()`, який повертає обидва набори (у пікселях та у нормалізованих координатах) за один виклик. Це зменшує обчислювальні витрати в головному циклі програми.

```

def get_landmarks(
    self, frame
) -> Tuple[List[Tuple[int, float, float, float]], List[Tuple[int, float, float, float]]:
    """
    Returns both pixel and normalized landmarks in one call (optimization).
    :param frame: frame (for converting normalized coordinates into pixels)
    :return: tuple (landmarks_px, landmarks_norm)
    """
    landmarks_px: List[Tuple[int, float, float, float]] = []
    landmarks_norm: List[Tuple[int, float, float, float]] = []
    You, yesterday * Enhance gesture detection and hand tracking per...
    if not self.results or not self.results.multi_hand_landmarks:
        return landmarks_px, landmarks_norm

    img_h, img_w, _ = frame.shape
    hand_lms = self.results.multi_hand_landmarks[0]

    for idx, lm in enumerate[Any](hand_lms.landmark):
        landmarks_norm.append((idx, lm.x, lm.y, lm.z))
        landmarks_px.append((idx, lm.x * img_w, lm.y * img_h, lm.z))

    return landmarks_px, landmarks_norm

```

Рис 6.8. Отримання координат landmarkів

Для детального ознайомлення відкрийте файл **hand_detector.py** в Додатку F.

5.3.4. Згладжування рухів (smoothing.py)

Для забезпечення стабільної взаємодії реалізовано два фільтри згладжування.

Згладжування курсора (CursorSmoother)

Використовується експоненційне згладжування:

- мінімізує тремтіння;
- має низьку затримку;
- оптимізоване за рахунок попередніх обчислень ($1 - \alpha$).

Згладжування виконується у піксельних координатах екрана.

```
class CursorSmoother:
    """
    Simple exponential filter for smoothing the cursor coordinates.
    Optimized for performance.
    """

    def __init__(self, alpha: float = 0.5):
        self.alpha = alpha
        self.one_minus_alpha = 1.0 - alpha
        self.prev_x: Optional[float] = None
        self.prev_y: Optional[float] = None

    def smooth(self, x: float, y: float) -> Tuple[int, int]:
        if self.prev_x is None:
            self.prev_x, self.prev_y = x, y
            return int(x), int(y)

        sx = self.prev_x * self.one_minus_alpha + x * self.alpha
        sy = self.prev_y * self.one_minus_alpha + y * self.alpha

        self.prev_x, self.prev_y = sx, sy
        return int(sx), int(sy)
```

Рис 6.9. Згладжування курсора (CursorSmoother)

Згладжування landmarkів (LandmarkSmoother)

Згладжуються всі 21 landmark:

- кожна координата (x, y, z) обробляється окремо;
- застосовується експоненційний фільтр;
- оптимізований цикл без зайвих операцій.

Цей модуль критично важливий для стабільної детекції жестів.

```
class LandmarkSmoother:
    """
    Smoothing of a complete set of landmarks (list (id, x, y, z)).
    Uses exponential smoothing for each coordinate.
    Optimized for performance.
    """

    def __init__(self, alpha: float = 0.5):
        self.alpha = alpha
        self.one_minus_alpha = 1.0 - alpha
        self.prev_landmarks: Optional[List[Tuple[int, float, float, float]]] = None

    def smooth(
        self, landmarks: List[Tuple[int, float, float, float]]
    ) -> List[Tuple[int, float, float, float]]:
        if self.prev_landmarks is None or len(self.prev_landmarks) != len(landmarks):
            self.prev_landmarks = landmarks
            return landmarks

        smoothed: List[Tuple[int, float, float, float]] = []
        alpha = self.alpha
        one_minus_alpha = self.one_minus_alpha

        for i in range(len(landmarks)):
            idx, x, y, z = landmarks[i]
            _, px, py, pz = self.prev_landmarks[i]

            sx = px * one_minus_alpha + x * alpha
            sy = py * one_minus_alpha + y * alpha
            sz = pz * one_minus_alpha + z * alpha
            smoothed.append((idx, sx, sy, sz))

        self.prev_landmarks = smoothed
        return smoothed
```

Рис 6.10. Згладжування landmarkів (LandmarkSmoother)

Для детального ознайомлення відкрийте файл **smoothing.py** в Додатку F.

5.3.5. Реалізація детекції жестів (gesture_detector.py)

Для аналізу жестів використовуються:

- відстані між пальцями (Pinch);
- середня відстань пальців від центроїда (Grab);
- швидкість центроїда (Swipe, Scroll);
- debounce-час для захисту від хибних спрацьовувань.

Система підтримує:

- *PINCH*
- *GRAB / RELEASE*
- *SCROLL_UP / SCROLL_DOWN*
- *SWIPE_LEFT / SWIPE_RIGHT*

Особливістю реалізації є :

- під час Grab активуються Scroll і Swipe;
- Pinch можливий лише при відсутності Grab;

```
you, yesterday | 1 author (you)
class GestureType:
    NONE = "none"
    MOVE = "move"
    PINCH = "pinch"
    SCROLL_UP = "scroll_up"
    SCROLL_DOWN = "scroll_down"
    SWIPE_LEFT = "swipe_left"
    SWIPE_RIGHT = "swipe_right"
    GRAB = "grab"
    RELEASE = "release"
```

Рис 6.11. Типи жестів

Реалізація `GestureDetector` класу об'ємна тому не була додана до звіту у вигляді скріншоту, для детального ознайомлення відкрийте файл **`smoothing.py`** в Додатку F.

5.3.6. Модуль керування інтерфейсом (`ui_controller.py`)

Модуль `ui_controller.py` відповідає за перетворення жестів у дії операційної системи. Реалізовано:

- переміщення курсора;
- клік;
- прокрутка;

Використання нормалізованих координат дозволяє системі бути незалежною від роздільної здатності екрана.

```
class UIController:
    """
    Class for managing the system interface based on gestures.
    """

    def __init__(self) -> None:
        self.screen_w, self.screen_h = pyautogui.size()

    def map_to_screen(self, x_norm: float, y_norm: float) -> Tuple[int, int]:

    def move_cursor_norm(self, x_norm: float, y_norm: float) -> None: ...

    def click(self) -> None: ...

    def scroll_up(self, amount: int = 80) -> None: ...

    def scroll_down(self, amount: int = 80) -> None: ...

    def swipe_left(self) -> None: ...

    def swipe_right(self) -> None: ...

    def start_drag(self) -> None: ...

    def end_drag(self) -> None: ...

    def perform_gesture(self, gesture: str) -> None: ...
```

Рис 6.12. Методи для взаємодії з інтерфейсом ОС

Для детального ознайомлення відкрийте файл **`ui_controller.py`** в Додатку F.

5.3.7. Головний цикл роботи (**main.py**)

Головний цикл програми реалізує:

- Захоплення кадру з камери.
- Дзеркальне відображення.
- Масштабування кадру для обробки.
- Детекція руки.
- Відновлення координат до оригінального масштабу.
- Згладжування landmarkів.
- Детекція жесту.
- Керування курсором / UI.
- Обчислення FPS.
- (Опціонально) візуалізація.

Реалізація **main.py** об'ємна тому не була додана до звіту у вигляді скріншоту, для детального ознайомлення відкрийте файл **main.py** в Додатку F.

Загальний висновок до розділу 5

У цьому розділі було розроблено та описано автоматизовану систему відстеження рухів руки та розпізнавання жестів із модульною структурою. Обґрунтовано вибір основних технологій і засобів реалізації, а також розглянуто структуру програмного коду та призначення окремих модулів системи.

Запропонована архітектура забезпечує стабільне відстеження руки, коректне розпізнавання жестів і перетворення їх у дії операційної системи в реальному часі. Наявність конфігураційних налаштувань, методів згладжування та механізмів логування робить систему гнучкою, зручною для налаштування та придатною для подальшого розвитку й інтеграції в різні системи

6. ЕКСПЕРЕМЕНТАЛЬНА ЧАСТИНА

Метою експериментального дослідження є перевірка працездатності розробленої системи відстеження рухів руки та жестового керування елементами користувацького інтерфейсу в умовах, наближених до реального використання. У цьому розділі основна увага приділяється не максимальній продуктивності системи, а її стабільності, коректності роботи та зручності взаємодії з користувачем.

Експеримент побудовано таким чином, щоб його можна було легко відтворити без використання спеціалізованого обладнання.

6.1. Завдання експерименту

Для досягнення поставленої мети необхідно виконати такі завдання:

- перевірити стабільність виявлення руки в кадрі;
- оцінити коректність керування курсором;
- протестувати основні жести в реальному сценарії;
- перевірити можливість виконання послідовних дій;

Замість складних лабораторних тестів у роботі використано сценарно-орієнтований підхід, характерний для досліджень у галузі взаємодії людини з комп'ютером. Такий підхід дозволяє оцінити не лише технічну працездатність системи, а й її практичну зручність та корисність для користувача.

6.2. Умови тестування

Апаратні умови

Експеримент проводився з використанням стандартного обладнання:

- персональний ноутбук;
- вбудована вебкамера;
- стандартний монітор;

- відсутність додаткових сенсорів або датчиків.

Програмні умови

Для проведення експерименту використовувались:

- операційна система Windows;
- веббраузер Google Chrome;
- розроблена система жестового керування;

Умови середовища

- освітлення: кімнатне;
- фон: відносно однорідний;
- відстань до камери: 50–80 см;
- кількість користувачів: один.

6.3. Опис тестових сценаріїв

6.3.1. Сценарій 1 – Керування курсором

Мета: перевірка плавності та стабільності руху курсора.

Користувач здійснює рухи рукою в межах поля зору камери, спостерігаючи за реакцією курсора на екрані.

Спостереження:

- курсор слідує за рухом пальця;
- затримка відчувається, але не заважає виконанню дій;
- згладжування зменшує тремтіння.

6.3.2. Сценарій 2 – Клік та вибір елементів

Мета: перевірка жесту Pinch.

Користувач наводить курсор на клікабельні елементи та виконує жест pinch для активації кліку.

Спостереження:

- клік спрацьовує при чітко виконаному жесті;
- потребує короткого звикання;
- можливі одиничні хибні спрацьовування при швидкому русі.

6.3.3. Сценарій 3 – Прокрутка сторінки

Мета: перевірка жестів Scroll Up / Scroll Down.

Користувач прокручує вебсторінки з текстовим контентом.

Спостереження:

- прокрутка виконується плавно;
- найкраще працює при повільних вертикальних рухах;
- зручно для читання контенту.

6.3.4. Сценарій 4 – Навігація між сторінками

Мета: перевірка жестів Swipe Left / Swipe Right.

Користувач переходить між сторінками браузера.

Спостереження:

- жест спрацьовує при чіткому горизонтальному русі;
- потребує достатньої амплітуди руху;
- випадкові спрацьовування трапляються рідко.

Відео з записом експерименту можна переглянути в **Додатку G**.

6.4. Узагальнення результатів спостережень

Таблиця 7.1 – Результати виконання тестових сценаріїв

Сценарій	Працездатність	Коментар
Рух курсора	стабільна	плавний рух
Клік (Pinch)	стабільна	потребує звикання
Scroll	стабільна	зручно для читання
Swipe	задовільна	залежить від швидкості

Відтворюваність експерименту

Запропонований експеримент:

- не потребує спеціального обладнання;
- може бути повторений у будь-яких побутових умовах;
- підходить для демонстрації;
- дозволяє наочно показати принцип роботи системи.

Загальний висновок до розділу 6

Проведене експериментальне дослідження показало, що розроблена система жестового керування є працездатною та придатною для використання в умовах, наближених до реальних. Система стабільно виявляє руку, дозволяє керувати курсором і коректно розпізнає базові жести, такі як клік, прокрутка та свайп, без застосування спеціального обладнання.

Хоча в досліді проводився експеримент з використанням ПК в домашніх умовах, система може інтегруватись в промислові процеси та автоматизовані системи та ряд інших сфер.

ВИСНОВКИ

В межах даної роботи було розроблено та досліджено автоматизовану систему відстеження рухів руки та розпізнавання жестів на основі методів комп'ютерного зору. Актуальність теми зумовлена зростаючою потребою в безконтактних та інтуїтивних інтерфейсах взаємодії людини з комп'ютером, які можуть застосовуватися як у побутових умовах, так і в задачах промислової автоматизації.

У теоретичній частині роботи розглянуто еволюцію підходів до відстеження руки - від класичних методів комп'ютерного зору до сучасних нейромережевих моделей.

У практичній частині розроблено модульну архітектуру системи з використанням фреймворку MediaPipe Hands та бібліотек Python, OpenCV і NumPy. Система реалізує відстеження 21 ключової точки кисті, згладжування координат і розпізнавання базових жестів у реальному часі. Завдяки відсутності потреби у спеціалізованих сенсорах, система може бути розгорнута на різних платформах, таких як Raspberry Pi та NVIDIA Jetson, що робить її придатною для використання в автоматизованих та промислових системах керування.

Експериментальне дослідження підтвердило працездатність розробленої системи без застосування спеціалізованого обладнання. Отримані результати свідчать про можливість використання системи як прототипу жестового інтерфейсу для задач автоматизації, зокрема для керування промисловими інтерфейсами, операторськими панелями та інтерактивними системами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. MediaPipe Hands — Official Google Documentation. URL: https://developers.google.com/mediapipe/solutions/vision/hand_landmarker
2. Zhang Z., “Hand Pose Estimation via Deep Learning”, *IEEE Transactions on PAMI*, 2020.
3. Simon, T., Joo, H., Matthews, I., & Sheikh, Y. “Hand Keypoint Detection in Single Images Using Multiview Bootstrapping”. *CVPR*, 2017.
4. Holz, C., et al. “Leap Motion: 3D Interaction via Natural Hand Motions”. *CHI*, 2013.
5. OpenCV Documentation. URL: <https://docs.opencv.org>
6. PyAutoGUI Documentation. URL: <https://pyautogui.readthedocs.io>
7. Python Official Documentation. URL: <https://docs.python.org>
8. Mao, Y., “A Survey on Gesture Recognition Using Deep Learning”. *ACM Computing Surveys*, 2021.
9. Sridhar S., Muller M., Oulasvirta A., Theobalt C. “Fast and Robust Hand Tracking Using Detection-Guided Optimization”. *CVPR*, 2015.
10. Ren, Z., Yuan, J. “Depth Camera Based Hand Gesture Recognition”. *IEEE ICCV Workshops*, 2011.
11. Shotton, J. et al. “Real-Time Human Pose Recognition in Parts from a Single Depth Image”. *CVPR*, 2011.
12. Colledanchise M., Ögren P. “Behavior Trees in Robotics and AI”. CRC Press, 2018.
13. Goodfellow, I., Bengio, Y., Courville, A. “Deep Learning”. MIT Press, 2016.
14. Canny, J. “A Computational Approach to Edge Detection”. *IEEE PAMI*, 1986.
15. Horn B., Schunck B. “Determining Optical Flow”. *Artificial Intelligence Journal*, 1981.
16. Aggarwal J., Cai Q. “Human Motion Analysis: A Review”. *Computer Vision and Image Understanding*, 1999.
17. Krizhevsky, A., Sutskever, I., Hinton G. “ImageNet Classification with Deep CNNs”. *NIPS*, 2012.

- 18.OpenPose — CMU Perceptual Computing Lab. URL: <https://github.com/CMU-Perceptual-Computing-Lab/openpose>
- 19.BlazePalm Paper — MediaPipe Palm Detection Model. URL: <https://arxiv.org/abs/2006.10230>
- 20.BlazeHand Landmark Model Paper. URL: <https://arxiv.org/abs/2006.10214>
- 21.Microsoft Kinect Hand Tracking SDK Documentation. URL: <https://learn.microsoft.com>
- 22.Zhang, J., et al. “Real-Time Hand Tracking Under Occlusion Using Conditional Regression Forests”. *CVPR*, 2014.
- 23.ISO 9241-960:2023 — Ergonomics of Human-System Interaction.
- 24.HCI Interaction Models. Preece J., Rogers Y., Sharp H. “Interaction Design”, Wiley, 2019.
- 25.Human-Centered AI Interaction — Stanford HAI Publications, 2021.
- 26.Duda, R., Hart, P., Stork, D. “Pattern Classification”, Wiley, 2012.
- 27.Busch A., “Gesture-Based Interfaces in Assistive Technologies”, Springer, 2020.