МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп’ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

Воєвудський Олександр Олександрович

Прикладний програмний інтерфейс відео-керування цифровим двійником / Application Programming Interface of Video Control of a Digital Twin

спеціальність: 122 - Комп'ютерні науки
освітньо-професійна програма - Комп’ютерні науки

Кваліфікаційна робота

Виконав студент групи КН-41
О. О. Воєвудський
_____

Науковий керівник:
к.т.н., доцент І. В. Турченко
_____

Кваліфікаційну роботу
допущено до захисту:
«___» _____ 20___ р.
Завідувач кафедри
_____ Н.В. Дзюбановська

ТЕРНОПІЛЬ – 2026

Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «бакалавр»
спеціальність: 122 – Комп'ютерні науки
освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Н.В. Дзюбановська
«____» _____20__р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Воєвудського Олександра Олександровича
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи

Прикладний програмний інтерфейс відео-керування цифровим двійником / Application Programming Interface of Video Control of a Digital Twin

керівник роботи к.т.н., доцент І.В. Турченко
затверджений наказом по університету від 01 грудня 2025 року № 892.
2. Строк подання студентом закінченої кваліфікаційної роботи 25 травня 2026 р.
3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.
4. Основні питання, які потрібно розробити

- провести огляд та аналіз сучасних технологій людинно-машинної взаємодії, методів комп'ютерного зору та просторових обчислень;
- здійснити постановку задачі проектування;
- спроектувати архітектуру прикладного програмного інтерфейсу;
- розробити алгоритм роботи прикладного програмного інтерфейсу;
- здійснити програмну реалізацію системи керування цифровим двійником мовою Python із накладеним графічним інтерфейсом користувача (HUD);
- провести функціональне тестування розробленого програмного комплексу та оцінити коректність обробки просторових інтенцій оператора.

5. Перелік графічного матеріалу у роботі

- Схема алгоритму роботи прикладного програмного інтерфейсу
- UML-діаграма класів
- UML-діаграма станів

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 01 грудня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Затвердження теми кваліфікаційної роботи, ознайомлення з літературними джерелами та складання плану роботи	до 01.01. 2026 р.	
2	Написання 1 розділу кваліфікаційної роботи	до 01.02. 2026 р.	
3	Написання 2 розділу кваліфікаційної роботи	до 01.04.2026 р.	
4	Написання 3 розділу кваліфікаційної роботи	до 01.05.2026 р.	
5	Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником	до 15.05.2026 р.	
6	Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів	до 25.05.2026 р.	
7	Перевірка кваліфікаційної роботи на оригінальність тексту	до 30.05.2026 р.	
8	Оформлення кваліфікаційної роботи та отримання допуску до захисту	до 10.06.2026 р.	
9	Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії	до 10.06.2026 р.	

Студент _____ О. О. Воєвудський
підпис

Керівник роботи _____ к.т.н., доцент І.В. Турченко
підпис

АНОТАЦІЯ

Кваліфікаційна робота на тему «Прикладний програмний інтерфейс відео керування цифровим двійником» на здобуття освітнього ступеня «бакалавр» зі спеціальності 122 «Комп’ютерні науки» написана обсягом 52 сторінки, містить 6 рисунків, 1 таблицю, 2 додатки та 26 джерел.

Метою роботи є розробка прикладного програмного інтерфейсу відео-керування цифровим двійником.

Об’єктом дослідження є процеси людино-машинної взаємодії в імерсивних цифрових середовищах.

Методи досліджень: системний аналіз, методи комп'ютерного зору та глибокого навчання (MediaPipe BlazePose), математичне моделювання просторових рухів, процедурне програмування (Python) та тестування «чорний ящик».

Результатом роботи є прикладний програмний інтерфейс відео-керування цифровим двійником, реалізований на мові програмування Python. Він дозволяє користувачеві взаємодіяти з імерсивним середовищем за допомогою рухів.

Результати можуть бути використані в ігрових рушіях та інтерактивних симуляціях для впровадження природних методів керування, а також як інклюзивна асистивна технологія для користувачів із порушеннями дрібної моторики.

Ключові слова: ПРИКЛАДНИЙ ПРОГРАМНИЙ ІНТЕРФЕЙС, MEDIAPIPE, ЦИФРОВИЙ ДВІЙНИК, ЛЮДИНО-МАШИННА ВЗАЄМОДІЯ, КОМП'ЮТЕРНИЙ ЗІР, ЕМУЛЯЦІЯ ВВОДУ, ПРОСТОРОВІ ОБЧИСЛЕННЯ.

ANNOTATION

The qualification work on the topic "Application Programming Interface of Video Control of a Digital Twin" for obtaining the "Bachelor" degree in specialty 122 "Computer Science" consists of 52 pages, contains 6 figures, 1 table, 2 appendices, and 26 references.

The purpose of the work is the development of an application programming interface for video control of a digital twin.

The object of research is the processes of human-computer interaction in immersive digital environments.

Research methods: system analysis, computer vision and deep learning methods (MediaPipe BlazePose), mathematical modeling of spatial movements, procedural programming (Python), and black-box testing.

The result of the work is an application programming interface for video control of a digital twin, implemented in the Python programming language. It allows the user to interact with an immersive environment using movements.

The results can be used in game engines and interactive simulations to implement natural control methods, as well as an inclusive assistive technology for users with fine motor impairments.

Keywords: APPLICATION PROGRAMMING INTERFACE, MEDIAPIPE, DIGITAL TWIN, HUMAN-COMPUTER INTERACTION, COMPUTER VISION, INPUT EMULATION, SPATIAL COMPUTING.

ЗМІСТ

ВСТУП

У теперішніх реаліях, у контексті активного розвитку імерсивних технологій та віртуальних середовищ, все більша частина користувачів стикається із новими вимогами до людинно-машинної взаємодії. Традиційні засоби введення інформації, такі як клавіатури, миші та геймпади, створюють штучний бар'єр між природною біомеханікою людини і цифровим простором. Розвиток технологій просторових обчислень та комп'ютерного зору відкриває нові можливості для створення безконтактних систем керування. За таких обставин використання природних інтерфейсів користувача (Natural User Interfaces) перестає бути лише експериментальним напрямом – воно забезпечує доступ до більш глибокого занурення в інтерактивний процес, створюючи нові можливості для симуляцій, гейміфікації та асистивних технологій.

Актуальність теми полягає у тому, що на тлі зростаючої популярності цифрових двійників та інтерактивних аватарів [1, 2, 3], особливо важливим є знаходження ефективних способів їхнього контролю без використання дороговартісного або обмежуючого рухи апаратного забезпечення. На сьогоднішній день переважна частина існуючих рішень вимагає спеціалізованих інфрачервоних датчиків глибини або VR-контролерів, що знижує їх доступність для масового користувача. Це зумовлює потребу в розробці програмних інтерфейсів, здатних працювати зі стандартними RGB-вебкамерами [4]. Поєднання алгоритмів комп'ютерного зору та методів емуляції вводу дозволяє створити комфортне та інклюзивне середовище для керування цифровими об'єктами, забезпечуючи баланс між точністю розпізнавання рухів та мінімальними апаратними вимогами.

Об’єктом дослідження є процеси людинно-машинної взаємодії в імерсивних цифрових середовищах.

Предметом дослідження є інтерактивний програмний інтерфейс для безконтактного відео-керування цифровим двійником на основі методів комп'ютерного зору.

Метою роботи є розробка прикладного програмного інтерфейсу відео-керування цифровим двійником.

Для досягнення поставленої мети необхідно виконати наступні завдання:

– провести огляд та аналіз сучасних технологій людинно-машинної взаємодії, методів комп'ютерного зору та просторових обчислень;

– огляд існуючих аналогів систем безконтактного керування та постановка задачі проектування;

– представити загальну структуру прикладного програмного інтерфейсу

– розробити алгоритм роботи прикладного програмного інтерфейсу;

– здійснити програмну реалізацію системи керування цифровим двійником мовою Python із накладеним графічним інтерфейсом користувача (HUD);

– провести функціональне тестування розробленого програмного комплексу та оцінити коректність обробки просторових інтенцій оператора.

У роботі застосовано такі методи дослідження: аналіз та узагальнення наукової літератури, системний аналіз для проектування архітектури програмного забезпечення, математичне моделювання просторових рухів (евклідова геометрія), методи комп'ютерного зору та глибокого навчання (архітектура BlazePose фреймворку MediaPipe), UML-моделювання (діаграми класів та станів), процедурне програмування (Python).

Практичне значення отриманих результатів: розроблений програмний продукт є готовим проміжним програмним забезпеченням (Middleware), яке може бути використане розробниками інтерактивних додатків для керування віртуальними персонажами без фізичних контролерів. Завдяки налаштовуваним порогам чутливості, система також може слугувати інклюзивною асистивною технологією для людей з порушеннями моторики, поєднуючи ефективність безконтактного керування із комфортом користування.

Результати роботи опубліковано в матеріалах міжнародної науково-практичної конференції студентів і молодих вчених «ICSPM 2026: Інтелектуальні обчислювальні системи та управління проектами», м. Тернопіль, 21–22 травня 2026 р. (додаток Б).

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ПРОЕКТУВАННЯ

1.1 Аналіз предметної області та еволюція технологій людинно-машинної взаємодії

Предметною областю даного кваліфікаційного дослідження є інженерія систем людино-машинної взаємодії (Human-Computer Interaction, HCI) [5] із застосуванням методів комп'ютерного зору для забезпечення безконтактного керування цифровими двійниками. З розвитком обчислювальної техніки парадигма взаємодії людини та комп'ютера пройшла шлях від перфокарт та інтерфейсів командного рядка до графічних інтерфейсів користувача (GUI). Сучасним етапом цієї еволюції є перехід до природних інтерфейсів користувача (Natural User Interfaces, NUI), де взаємодія відбувається на основі інтуїтивних фізичних рухів, жестів та голосових команд, мінімізуючи когнітивне навантаження на оператора.

У контексті сучасних інтерактивних систем, симуляцій та ігрових середовищ (зокрема, двовимірних платформерів) цифровим двійником виступає віртуальний аватар – керована програмна сутність, яка повинна синхронно реагувати на фізичні дії оператора у реальному часі. Традиційна парадигма взаємодії з такими сутностями безальтернативно базується на використанні електромеханічних периферійних пристроїв введення, таких як комп'ютерні клавіатури, миші або геймпади.

Незважаючи на високу швидкість відгуку та точність, електромеханічний підхід має низку суттєвих обмежень. По-перше, він вимагає безперервного тактильного контакту, що жорстко обмежує фізичну мобільність користувача. По-друге, він створює штучний бар’єр між природною біомеханікою людини та цифровим середовищем, змушуючи оператора транслювати складні просторові інтенції через натискання обмеженої кількості кнопок. Крім того, особливо гостро ця проблема постає в умовах промислового виробництва, стерильних медичних приміщень або під час роботи з агресивними середовищами, де фізичний контакт із пристроями керування є ускладненим, небажаним або взагалі неможливим.

Розвиток технологій просторових обчислень (англ. Spatial Computing)

формує нову предметну область [6, 7], що вивчає методи повної або часткової заміни фізичних контролерів на оптичні сенсори [8]. У цій моделі тіло оператора стає безпосереднім пристроєм введення інформації. Фундаментом досліджуваної предметної області є технології машинного зору [9] та глибокого навчання (Deep Learning), зокрема алгоритми оцінки пози тіла людини (Pose Estimation). Завданням цих систем є вилучення структурованої кінематичної інформації (топології скелета) з неструктурованого двовимірного масиву пікселів, який генерується стандартною монокулярною RGB-вебкамерою.

Важливим, і часто недооціненим, аспектом розробки систем безконтактного відеокерування є забезпечення високого рівня інклюзивності та цифрової доступності. Стандартні пристрої введення розроблені з розрахунком на користувачів із повною дрібною моторикою рук. Однак для значної частини людей використання клавіатури чи геймпада є фізично виснажливим або неможливим. До цієї категорії належать особи з ампутаціями верхніх кінцівок, порушеннями координації, церебральним паралічем, а також люди, що страждають на тунельний синдром карпального каналу (RSI) внаслідок тривалої роботи за комп'ютером.

Програмний інтерфейс для відео-керування цифровим двійником пропонує фундаментально новий рівень адаптивності. Оскільки система здатна аналізувати глобальну топологію тіла, керування може здійснюватися не лише пальцями чи кистями, але й нахилами тулуба, рухами плечей або голови. Це відкриває двері для концепції адаптивного геймінгу та безбар'єрної взаємодії (Barrier-free HCI). Крім того, інтеграція таких контролерів в інтерактивні середовища створює передумови для гейміфікації процесів фізичної реабілітації: оператор, виконуючи призначені лікарем терапевтичні рухи, може одночасно керувати віртуальним аватаром, що значно підвищує мотивацію та психологічний комфорт під час відновлення.

Незважаючи на стрімкий розвиток інструментів машинного навчання та наявність оптимізованих моделей для повноростового трекінгу, на сучасному ринку програмного забезпечення спостерігається виражений технологічний розрив. Доступні нейромережеві фреймворки функціонують виключно як пасивні сенсори: вони генерують масив просторових координат, але не здійснюють їх семантичної інтерпретації. Цей неструктурований потік даних неможливо напряму передати до

рушія цифрового двійника.

З іншого боку, існуючі платформи для трансляції рухів зазвичай вимагають прямої, глибокої інтеграції у вихідний код керованого середовища (наприклад, через специфічні плагіни для ігрових рушіїв Unity або Unreal Engine). Такий підхід робить неможливим використання відеокерування для вже скомпільованих програм, до вихідного коду яких немає доступу.

1.2 Огляд і аналіз існуючих аналогів, що реалізують функції предметної області

Проектування та розробка програмного комплексу для безконтактного відео-керування цифровими двійниками вимагає попереднього критичного аналізу існуючих на ринку технологій захоплення руху (англ. Motion Capture) та оцінки пози (англ. Pose Estimation). Для забезпечення об'єктивності дослідження, наявні апаратно-програмні рішення класифіковано за методом отримання просторової інформації та рівнем алгоритмічної абстракції.

Порівняльний аналіз здійснюється за чотирма ключовими критеріями: апаратна залежність (необхідність спеціалізованого обладнання), обчислювальна складність алгоритмів (здатність працювати на центральному процесорі без залучення високопродуктивних графічних прискорювачів), стійкість до зовнішніх перешкод та можливість універсальної системної інтеграції без втручання у вихідний код керованого середовища. За цими критеріями індустріальні рішення поділяються на апаратні системи глибинного зондування, натільні інерціальні сенсори, низькорівневі бібліотеки класичного комп'ютерного зору та високорівневі нейромережеві фреймворки.

Апаратні системи на базі датчиків глибини. Історично найбільш точними інструментами для захоплення рухів тіла без використання маркерів є спеціалізовані апаратні комплекси, що генерують карту глибини простору (Depth Map). Найвідомішими представниками цієї категорії є Microsoft Azure Kinect, Intel RealSense та датчики Apple LiDAR [10]. Робота цих пристроїв базується на двох основних фізичних принципах: проекції структурованого світла (Structured Light)

та вимірюванні часу польоту фотонів (Time-of-Flight, ToF).

У системах структурованого світла інфрачервоний (ІЧ) проектор випромінює на об'єкт відомий патерн точок, а ІЧ-камера зчитує деформацію цієї сітки на нерівностях тіла, що дозволяє тригонометрично обчислити глибину кожного пікселя. Системи ToF працюють за принципом оптичного радара: вони випромінюють модульований інфрачервоний імпульс і за допомогою високошвидкісних сенсорів вимірюють фазовий зсув або час повернення відбитого сигналу. Це дозволяє апаратно відокремити тіло людини від фону та з субміліметровою точністю визначити просторові координати суглобів у тривимірному просторі.

Незважаючи на високу точність, апаратні системи мають фундаментальні недоліки в контексті розробки універсальних інтерфейсів для масового користувача. По-перше, вони створюють жорстку апаратну залежність (Vendor Lock-in), вимагаючи закупівлі дорогого обладнання для кожного робочого місця. По-друге, інфрачервоні датчики є вкрай вразливими до зовнішніх умов: їхня ефективність критично знижується при яскравому сонячному світлі або наявності відбиваючих поверхонь. По-третє, програмні комплекси (SDK) для цих пристроїв зазвичай інтегровані виключно під операційні системи сімейства Windows або вимагають встановлення важких драйверів, що унеможливлює створення кросплатформного та портативного програмного рішення.

Натільні інерціальні системи захоплення руху. Альтернативним апаратним підходом є використання інерціальних вимірювальних модулів (IMU), які кріпляться безпосередньо на тіло оператора (наприклад, спеціалізовані костюми Rokoko або Xsens). Кожен IMU-датчик містить мікроелектромеханічні (MEMS) акселерометри, гіроскопи та магнітометри. Вони вимірюють лінійне прискорення та кутову швидкість окремих сегментів тіла. Дані з датчиків обробляються за допомогою алгоритмів злиття сенсорних даних (англ. Sensor Fusion), таких як розширений фільтр Калмана (EKF), для розрахунку орієнтації суглобів у просторі.

Ці системи забезпечують найнижчу затримку (latency) та не страждають від проблеми візуального перекриття (Occlusion), оскільки не покладаються на камери. Однак для повсякденного керування комп'ютерними програмами чи

інтерактивними двійниками цей підхід є абсолютно непрактичним. Він вимагає тривалого процесу надягання костюма, регулярного апаратного калібрування (через проблему накопичення похибки інтегрування гіроскопів – так званий Drift), а також критично залежить від ємності акумуляторів датчиків.

Низькорівневі бібліотеки класичного комп'ютерного зору У прагненні відмовитися від спеціалізованого обладнання розробники звертаються до методів обробки монокулярного (2D) відеопотоку зі звичайних RGB-вебкамер. Фундаментальним інструментом у цій сфері є відкрита бібліотека комп'ютерного зору OpenCV [11, 12]. Спроби реалізувати трекінг тіла виключно класичними методами базуються на алгоритмах віднімання фону (англ. Background Subtraction) та аналізу оптичного потоку (англ. Optical Flow). Метод Лукаса-Канаде, який часто застосовується для відстеження рухів, аналізує зміщення інтенсивності пікселів між сусідніми кадрами.

Застосування виключно класичних алгоритмів для детекції біомеханіки людини є алгоритмічно нестабільним. Вони вимагають ідеального контрасту між користувачем і фоном, жорстко залежать від стабільності освітлення і не здатні самостійно ідентифікувати специфічні анатомічні вузли (наприклад, відрізнити лікоть від коліна без наявності кольорових маркерів на одязі). Використання таких методів для повноростового трекінгу призводить до високої частоти хибних спрацьовувань та робить неможливим побудову надійного кінематичного скелета (рисунок 1.1).

Високорівневі та низькорівневі нейромережеві фреймворки Вирішення проблеми оптичного трекінгу стало можливим завдяки впровадженню глибоких нейронних мереж (англ. Deep Learning). На ринку існують потужні дослідницькі фреймворки, такі як OpenPose [13], що використовують архітектуру згорткових мереж з полями афінності частин (Part Affinity Fields, PAFs) для одночасного багатокористувацького трекінгу 2D-скелетів з винятковою точністю. Однак архітектура OpenPose є занадто обчислювально важкою: для досягнення частоти обробки у 30 кадрів за секунду вона вимагає наявності високопродуктивних графічних процесорів (GPU) з підтримкою технології CUDA. На стандартних центральних процесорах (CPU) швидкість інференсу падає до 2-5 кадрів за секунду,

що робить цей інструмент непридатним для систем керування в реальному часі на типових користувацьких комп'ютерах.

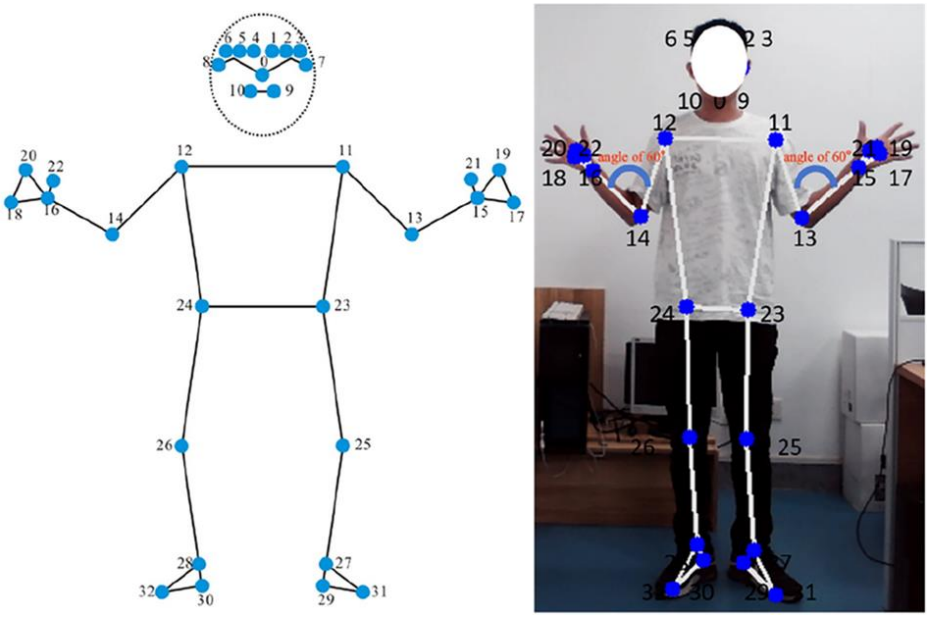

Рисунок 1.1 – Візуалізація топології скелета нейромережевою архітектурою

Оптимальним компромісом між швидкодією та точністю є використання фреймворку Google MediaPipe Pose. Його архітектура (BlazePose) оптимізована для виконання мобільних та десктопних інференсів безпосередньо на центральному процесорі. Завдяки використанню двоступеневого конвеєра (модуль детекції області інтересу та модуль точної регресії координат 33 точок), MediaPipe забезпечує стабільний трекінг з мінімальними апаратними вимогами. Саме тому він є найкращим сенсорним модулем для інтеграції у розроблювану систему.

Проблема системної інтеграції та формування технологічного розриву Незважаючи на довершеність моделей MediaPipe, з архітектурної точки зору вони вирішують лише половину задачі. Будь-яка нейромережа генерації скелета працює як пасивний сенсор – вона видає масив сирих векторних координат, але не здійснює їх семантичної інтерпретації. На сучасному ринку програмного забезпечення виявлено суттєвий технологічний розрив: відсутні готові, легковагові програмні рішення (Middleware), які б перетворювали кінематичні дані від камери у стандартизовані команди операційної системи.

Існуючі платформи для трансляції рухів (наприклад, специфічні плагіни для

ігрових рушіїв Unity або Unreal Engine) вимагають прямої інтеграції у вихідний код цифрового двійника. Вони передають дані за допомогою складних мережевих протоколів (OSC або WebSockets), що змушує розробників керованого середовища писати додатковий код для прийому та обробки цих координат. Такий підхід унеможливлює використання відеокерування для вже скомпільованих програм або ігор, до вихідного коду яких немає доступу.

Для об'єктивного підсумку результатів дослідження, порівняльну характеристику існуючих підходів наведено у таблиці 1.1 [14].

Таблиця 1.1 – Порівняльна характеристика технологій для відео-керування

Технологія / Рішення	Апаратна база	Алгоритмічний підхід	Мережева інтеграція	Основний недолік
Сенсори глибини (Microsoft Kinect)	Інфрачервоні камери, ToF-датчики	Апаратна генерація карти глибини простору	Власні закриті SDK, прив'язка до ОС Windows	Висока вартість обладнання, чутливість до освітлення
Інерціальні костюми (Rokoko, Xsens)	Натільні IMU-сенсори (акселерометри)	Злиття сенсорних даних (Sensor Fusion, EKF)	Плагіни для професійних 3D-редакторів	Потреба у носінні костюма, проблема накопичення похибки
OpenPose / Класичне CV	Стандартна RGB веб-камера	Вузькоспеціалізовані важкі нейромережі (PAFs)	Відсутня (видає JSON масиви координат)	Вимагає потужних дискретних GPU для роботи в реальному часі
Пропоноване рішення (Кастомний API)	Стандартна RGB веб-камера	MediaPipe Pose, евклідова математика	Емуляція ОС вводу (Віртуальна клавіатура/миша)	Потребує етапу проектування математичних порогів та гістерезису

1.3 Постановка задачі проектування та обґрунтування вибору засобів реалізації

На основі проведеного аналізу предметної області та існуючих технологічних рішень було виявлено суттєвий недолік сучасних систем відео-керування – відсутність універсального, апаратно-незалежного проміжного програмного забезпечення, яке б працювало на рівні операційної системи. Більшість існуючих рішень або вимагають дорогого обладнання (сенсори глибини), або потребують втручання у вихідний код цільової програми.

Зважаючи на вищевикладене, метою кваліфікаційної роботи є проектування структури та практична реалізація прикладного програмного інтерфейсу для безконтактного відео-керування цифровим двійником. Розроблюваний інтерфейс повинен функціонувати як інтелектуальний жест-контролер, що перетворює динамічні рухи тіла оператора, зафіксовані звичайною вебкамерою, у стандартизовані команди системного вводу.

При розробці важливо звернути увагу на наступні дії:

1) формалізація алгоритмів розпізнавання: розробити логіку інтерпретації кінематичних даних, що базується на відносних векторах, для забезпечення стабільної роботи системи незалежно від відстані оператора до камери;

2) проектування механізмів стабілізації: впровадити програмні методи фільтрації шумів та логіку гістерезису для запобігання хибним спрацьовуванням та багаторазовому перемиканню станів;

3) розробка модуля емуляції вводу: реалізувати низькорівневу трансляцію жестів у системні переривання клавіатури та миші, що забезпечить прозорість роботи API для будь-якого зовнішнього програмного забезпечення;

4) створення інтерфейсу візуального контролю: розробити модуль HUD (Head-Up Display) для реального часу візуалізації скелета та активних станів системи.

Для успішної реалізації поставлених завдань було обрано мову програмування Python [15]. Вибір зумовлений її статусом як стандарту де-факто у сфері комп'ютерного зору та штучного інтелекту. Python володіє багатим

екосистемою бібліотек, які дозволяють зосередитися на розробці алгоритмів, не відволікаючись на низькорівневе керування пам'яттю, що критично важливо для швидкої ітеративної розробки архітектури API.

Як базовий інструмент для обробки відеопотоку обрано бібліотеку OpenCV (Open Source Computer Vision Library). Вона забезпечує високу продуктивність під час захоплення кадрів, їх апаратного віддзеркалення та перетворення колірних просторів. OpenCV є кросплатформним рішенням, що відповідає вимозі універсальності розроблюваного API.

Для вирішення завдання оцінки пози тіла обрано фреймворк Google MediaPipe [16], а саме модель BlazePose. На відміну від аналогів (наприклад, OpenPose), MediaPipe оптимізований для виконання інференсу на центральному процесорі (CPU) в режимі реального часу. Модель здатна ідентифікувати 33 ключові точки тіла з високою точністю, що є достатнім для побудови складної кінематичної моделі керування. Використання MediaPipe дозволяє зробити систему доступною для користувачів із середньостатистичними потужностями ПК, не вимагаючи наявності дорогих дискретних відеокарт.

Для забезпечення взаємодії з операційною системою та емуляції апаратних подій обрано бібліотеку PyDirectInput [17]. Це рішення дозволяє генерувати синтетичні натискання клавіш, які сприймаються операційною системою як реальні події вводу. Це є ключовим фактором для створення Middleware-системи, оскільки дозволяє керувати цифровим двійником (наприклад, у 2D-платформері) без необхідності модифікації коду самої гри.

Окрім технічних засобів, необхідно визначити функціональні вимоги, яким має відповідати розроблений API:

1) ергономічність: система повинна автоматично враховувати дзеркальність рухів оператора;

2) адаптивність: логіка розпізнавання жестів має спиратися на антропометричні пропорції тіла, що робить систему придатною для людей різного зросту та статури;

3) надійність (Robustness): алгоритм повинен коректно обробляти ситуації часткового перекриття точок скелета, використовуючи показники

достовірності (confidence scores) від нейромережі;

4) інклюзивність: архітектура системи повинна дозволяти налаштування порогів чутливості, забезпечуючи можливість керування для осіб з обмеженою рухливістю.

Таким чином, інтеграція обраних технологій дозволить створити цілісне програмне рішення, яке заповнює існуючий технологічний розрив і забезпечує новий рівень взаємодії людини з цифровими двійниками.

2 СТРУКТУРА, АЛГОРИТМІЧНЕ ТА АПАРАТНЕ ЗАБЕЗПЕЧЕННЯ ПРИКЛАДНОГО ПРОГРАМНОГО ІНТЕРФЕЙСУ

2.1 Загальна структура проектованого прикладного програмного інтерфейсу

Проектований прикладний програмний інтерфейс відео-керування цифровим двійником побудований за принципом слабкої зв'язності компонентів (Loose Coupling). Це забезпечує гнучкість налаштування системи, високу швидкість інференсу нейромережі та стабільність роботи на рівні операційної системи. Архітектура системи базується на патерні конвеєрної обробки даних (Data Pipeline), де кожен наступний етап строго залежить від результатів роботи попереднього модуля, але функціонує незалежно в загальному циклі виконання.

Основним інженерним завданням архітектури є перетворення неструктурованого аналогового відеосигналу у високодетерміновані дискретні команди системного вводу без накопичення часової затримки. Загальна структурна схема системи (конвеєр) включає п’ять основних функціональних блоків:

– блок захоплення та попередньої обробки відеопотоку (Input & Preprocessing Module): цей модуль взаємодіє безпосередньо з апаратною частиною (вебкамерою) за допомогою методів бібліотеки OpenCV. Основними завданнями блоку є ініціалізація відеопотоку, виділення буфера пам'яті для кадрів та зчитування матриць пікселів у режимі реального часу. Критично важливим елементом препроцесингу на цьому етапі є горизонтальне віддзеркалення (flip) кадру. Це необхідно для створення інтуїтивного ефекту «дзеркала» для оператора: рух правої руки людини має синхронно відображатися як рух правої частини віртуального скелета на екрані. Також у цьому модулі відбувається конвертація колірного простору з формату BGR (стандарт OpenCV) у формат RGB (робочий стандарт нейромережевих тензорів MediaPipe);

– модуль нейромережевого інференсу (Pose Detection Engine): ядром цього блоку виступає фреймворк комп'ютерного зору MediaPipe Pose. Модуль отримує підготовлений RGB-кадр і виконує просторову регресію [18] для ідентифікації ключових точок тіла (landmarks). На виході підсистема генерує структурований масив із 33 вузлів (рисунок 2.1), кожен з яких містить нормалізовані координати

простору (x, y, z) та розрахований коефіцієнт достовірності (visibility score). Якщо рівень впевненості моделі падає нижче встановленого порогу VISIBILITY_THRESHOLD, координати відкидаються як зашумлені. Це працює як первинний фільтр безпеки, що запобігає хаотичній та непередбачуваній поведінці цифрового двійника при поганому освітленні або перекритті силуету оператора сторонніми об'єктами;

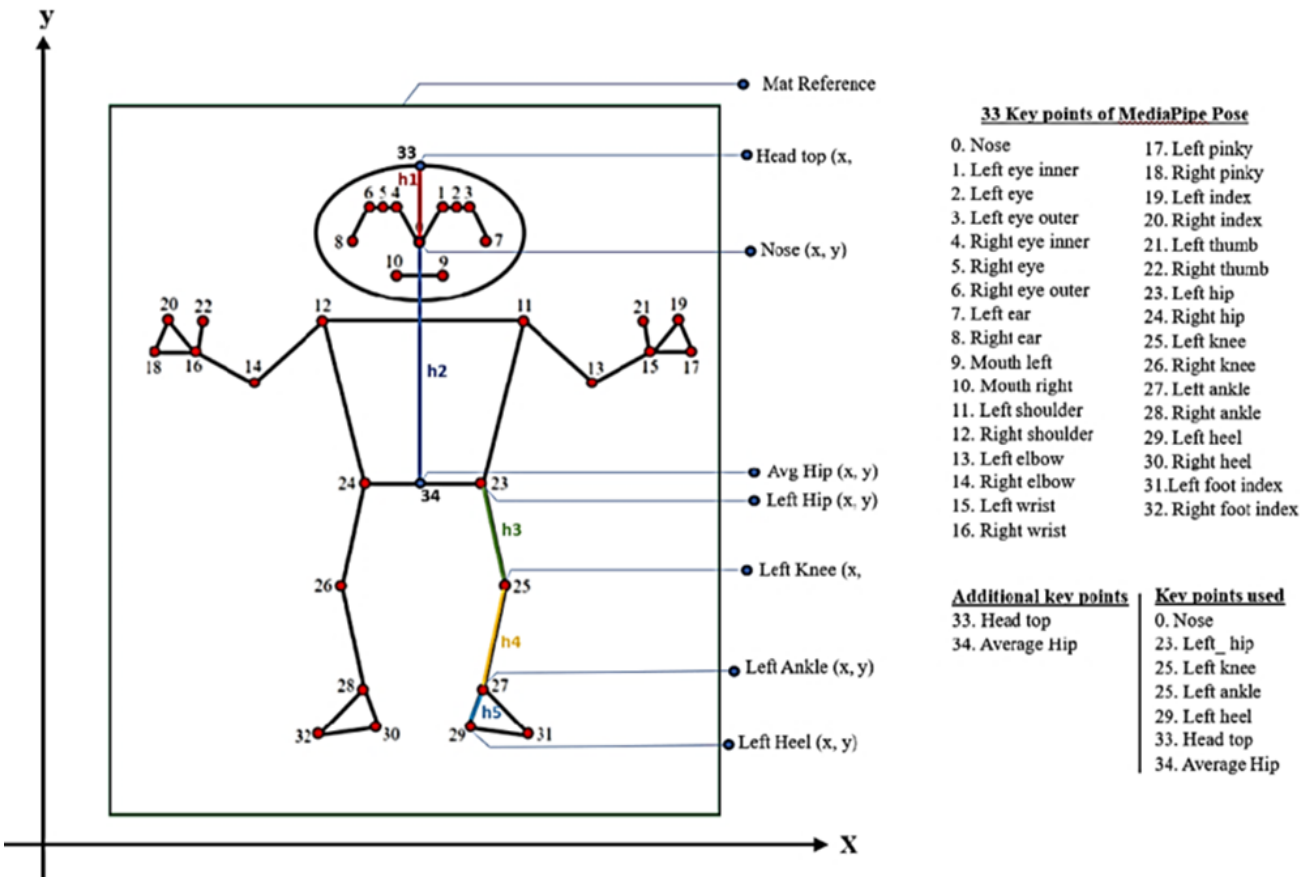

Рисунок 2.1 – Топологія 33 ключових точок кінематичної моделі тіла (MediaPipe Pose)

– модуль семантичної інтерпретації жестів (Decision Logic Module): інтелектуальне ядро системи, де відбувається математичне перетворення сирих координат у логічні стани (інтенції). Програмна логіка цього модуля оперує відносними відстанями, векторами та центроїдами. Основні розрахункові операції включають аналіз нахилів (обчислення зміщення центроїдів плечового поясу та тазу для керування вектором руху), детекцію статичних поз (порівняння висоти кистей щодо рівня обличчя для активації стрибка) та управління динамічними

атаками (аналіз швидкості зміни глибини по осі Z). Важливою властивістю цього модуля є масштабна інваріантність (Scale Invariance): оскільки всі розрахунки базуються на відносних пропорціях тіла, а не на абсолютних пікселях, система залишається стабільною незалежно від фізичної відстані оператора до камери;

– модуль стабілізації та емуляції системного вводу (Input Emulation & Hysteresis Module): цей блок відповідає за трансляцію логічних станів у переривання операційної системи. Його головна інженерна особливість – програмна реалізація логіки гістерезису. Модуль керує словником поточних станів (State Dictionary). Коли логічний модуль фіксує жест, емулятор перевіряє поточний стан відповідної клавіші. Якщо вона деактивована – генерується подія затискання keyDown. Подія відпускання keyUp генерується лише тоді, коли координати оператора повертаються в нейтральну «мертву зону» (Deadzone). Такий підхід повністю усуває ефект усуває ефект багаторазового перемикання логічних станів (keyboard chattering), який неминуче виникає через природний мікротремор людських м'язів та шум матриці камери;

– модуль візуального зворотного зв'язку та HUD (Visualization Module): система телеметрії, призначена для калібрування та інформування користувача. Модуль накладає згенеровану графову маску кінематичної моделі поверх оригінального відеопотоку та рендерить текстову інформацію про стан системи: списки активних тригерів, статуси готовності динамічних рухів (механіка Reload) та відстані між вузлами.

Запропонована структура конвеєра гарантує високу модульність: будь-який із зазначених блоків (наприклад, метод детекції скелета або бібліотека емуляції апаратного вводу) може бути модифікований чи замінений без необхідності рефакторингу всієї бізнес-логіки додатка. Це позиціонує розробку як універсальне Middleware-рішення, придатне для інтеграції в широкий спектр імерсивних середовищ. Зокрема, така архітектурна ізоляція компонентів дозволяє безперешкодно масштабувати систему шляхом впровадження нових математичних моделей для розпізнавання складніших багатовекторних жестів. Крім того, чітке розмежування процесів захоплення, обчислення та трансляції сигналів забезпечує повну апаратну незалежність розробки, суттєво подовжуючи життєвий цикл

створеного програмного продукту.

2.2 Алгоритм роботи прикладного програмного інтерфейсу

Алгоритм функціонування розробленого прикладного програмного інтерфейсу базується на безперервній циклічній обробці відеоданих у режимі реального часу. В основі алгоритму лежить строгий ітеративний процес, який проектувався з урахуванням жорстких обмежень щодо часу виконання (Latency). Загальний цикл обробки одного кадру можна розділити на декілька послідовних етапів:

1) ініціалізації та препроцесингу;
2) вилучення та фільтрації біометричних ознак;
3) математичної логіки та детекції інтенцій;
4) стабілізації та емуляції вводу;
5) візуалізації (HUD).

Етап ініціалізації та препроцесингу описано нижче.

Функціонування прикладного програмного інтерфейсу розпочинається з активації відеозахвату через інтерфейс cv2.VideoCapture. Алгоритм працює в нескінченному циклі (while loop), поки відеопотік залишається відкритим [19] або не отримано сигнал переривання.

На кожній ітерації циклу з буфера камери зчитується матриця пікселів поточного кадру. Отриманий кадр негайно піддається апаратному віддзеркаленню по горизонтальній осі (cv2.flip). Ця матрична операція є критичною для правильної просторової орієнтації оператора: без дзеркального відображення рух лівої руки людини сприймався б як рух правої частини скелета на екрані, що порушує принципи ергономіки та робить інтуїтивне керування неможливим.

Наступним кроком є застосування афінного перетворення колірної схеми з BGR на RGB, оскільки внутрішні тензори моделі BlazePose оптимізовані та навчені виключно на RGB-даних.

На етапі вилучення та фільтрації біометричних ознак підготовлений кадр передається в об'єкт Pose фреймворку MediaPipe. Алгоритм виконує просторову

регресію та вилучає 33 ключові точки тіла. Для кожної точки виконується обов'язкова перевірка на достовірність за параметром visibility. У розробленому API встановлено жорсткий поріг VISIBILITY_THRESHOLD (0.8), що відсікає всі координати, в яких нейромережа впевнена менше ніж на 80%. Цей механізм діє як високоефективний програмний фільтр низьких частот (Low-pass filter) [20], що дозволяє уникнути хаотичних рухів віртуального аватара у випадках, коли частини тіла оператора тимчасово перекриваються іншими об'єктами або виходять за межі кута огляду об'єктива.

Наступним є етап математичної логіки та детекції інтенцій. Після отримання валідних координат алгоритм переходить до розрахунку інтенцій оператора. Основним математичним апаратом для аналізу кінематики в межах даної системи є обчислення відстаней між визначеними ключовими вузлами. Оскільки відеопотік проектовано на площину екрана, а фреймворк повертає нормалізовані координати у діапазоні [0, 1], для визначення взаємного розташування точок застосовується класична формула евклідової відстані [D]у двовимірному просторі [21]:

$$D = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}, \quad (2.1)$$

де D - евклідова відстань;

x1, y1 - координати першої ключової точки;

x2, y2 - координати другої ключової точки.

Використання нормалізованих координат робить цю математичну модель масштабно інваріантною: результати розрахунків не залежать від роздільної здатності кадру. Загальна логіка детекції жестів, що використовує цей та інші кінематичні апарати, розділена на статичну та динамічну:

– взаємодія та тригери станів (Відновлення ресурсу): алгоритм безперервно вимірює просторове відхилення кінцівок. Зіставлення розрахованого значення евклідової відстані D між зап'ястками та обличчям із заздалегідь заданим порогом MANA_THRESHOLD дозволяє системі фіксувати специфічний жест «пиття мани»

(емуляція кліку правою кнопкою миші);

– навігаційна логіка (Стрій): алгоритм реалізує метод керування через нахил тулуба. Для цього розраховується відносне зміщення (tilt) між центром плечового поясу та центром тазової області. Враховуючи попереднє горизонтальне віддзеркалення відеопотоку та напрямок осі X фреймворку MediaPipe, значення розраховується як різниця середніх арифметичних координат X відповідних точок:

$$tilt = \frac{x_{left\ shoulder} + x_{right\ shoulder}}{2} - \frac{x_{left\ hip} + x_{right\ hip}}{2} \quad (2.2)$$

де tilt – обчислене значення нахилу тулуба оператора по горизонтальній осі;

$x_{left\ shoulder}, x_{right\ shoulder}$ – координати лівого та правого плеча по осі X відповідно;

$x_{left\ hip}, x_{right\ hip}$ – координати лівого та правого стегна (тазу) по осі X відповідно.

Якщо розраховане значення tilt перевищує поріг STEER_THRESHOLD, алгоритм генерує подію руху вправо (клавіша «D»), а відхилення у від'ємний бік викликає рух вліво (клавіша «A»);

– вертикальна навігація (Стрибок): стрибок («Space») детектується через порівняння вертикальних координат Y зап’ястків та носа. Якщо обидві руки підняті вище рівня обличчя, алгоритм реєструє інтенцію до стрибка;

– динаміка атак (Z-аналіз та Reloading): найскладнішою частиною алгоритму є детекція ударів. Вона базується на аналізі швидкості зміни синтетичної координати Z (псевдо-глибини, яку фреймворк розраховує відносно центру мас тіла). Для кожної руки в пам'яті зберігається попереднє значення глибини prev_z. Якщо поточне значення Z стає меншим за попереднє на величину PUNCH_FORWARD_SPEED, реєструється динамічна атака (випад). Важливою інженерною особливістю є впровадження механізму перезарядки рухів (l_reloaded, r_reloaded). Атака не може бути повторена, поки рух кисті оператора назад не перевищить поріг швидкості, визначений константою RELOAD_BACK_SPEED.

Це реалізує ергономічну модель «удар – повернення», що достовірно імітує механіку фізичного бою та запобігає випадковим подвійним спрацьовуванням (Double-triggering).

Фаза стабілізації та емуляції вводу. Завершальним логічним етапом ітераційного циклу є трансляція обчислених інтенцій оператора в апаратні переривання операційної системи. Оскільки системні виклики (зокрема, методи pydirectinput.keyDown()) є ресурсомісткими операціями введення-виведення, їх надмірне використання може призвести до синхронного блокування основного обчислювального потоку процесора.

З метою оптимізації продуктивності та усунення ефекту високочастотного хибного спрацьовування (keyboard chattering), алгоритм застосовує структуру даних у вигляді словника станів (key_states). Ініціалізація події натискання виконується одноразово в момент перетину біометричною ознакою заданого математичного порогу. Протягом усього часу утримання оператором відповідної пози стан тригера фіксується на логічному рівні, що виключає циклічне дублювання апаратних сигналів до ядра операційної системи.

Подія деактивації (keyUp) генерується виключно після припинення виконання жесту та повернення просторових координат у межі нейтральної зони нечутливості («мертвої зони»).

Етап візуалізації (HUD) – паралельно з емуляцією вводу, алгоритм оновлює графічне вікно зворотного зв'язку. На екран за допомогою оптимізованих методів відмальовування виводяться графова сітка скелета, список активних системних подій (наприклад, «ACTIVE: A SPACE LMB») та технічні метрики (відстані, готовність до атаки), що дозволяє оператору відкалібрувати свою дистанцію до камери.

Завершення кожної ітерації супроводжується викликом cv2.waitKey(1), який не лише оновлює графічне вікно, але й віддає процесорний час операційній системі, запобігаючи «зависанню» основного потоку.

Схема алгоритму прикладного програмного інтерфейсу наведена на рисунку 2.2.

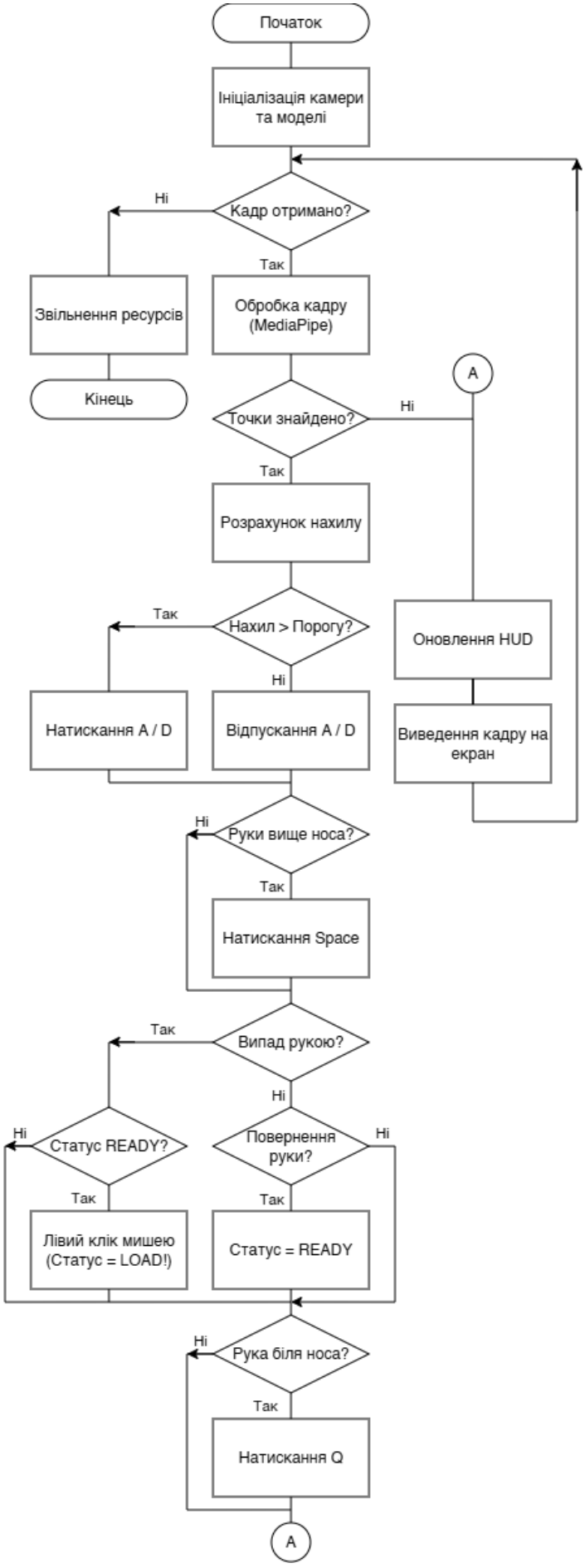

Рисунок 2.2 – Схема алгоритму роботи прикладного програмного інтерфейсу

Обчислювальна складність запропонованого ітеративного алгоритму становить O(N) для N=33 ключових точок. Завдяки лінійному характеру математичних операцій та застосуванню хеш-таблиць для збереження станів, конвеєр має детермінований час виконання. Це гарантує стабільну роботу розробленого проміжного програмного забезпечення у фоновому режимі без створення обчислювального навантаження на центральний процесор паралельно з роботою цифрового двійника.

2.3 Апаратне і програмне забезпечення

Для забезпечення стабільної та безперервної роботи розробленого прикладного програмного інтерфейсу (API), а також мінімізації затримок передачі сигналів (input lag) між рухами оператора та реакцією цифрового двійника, архітектура системи висуває чіткі вимоги до апаратного та програмного середовища.

До апаратного забезпечення комп'ютерної системи оператора висуваються такі мінімальні вимоги:

– процесор (CPU): багатоядерний процесор архітектури x86_64 із базовою тактовою частотою не менше 2.0 ГГц (наприклад, Intel Core i3 8-го покоління або аналогічний). Завдяки оптимізованій архітектурі BlazePose, яка виконує обчислення тензорів високоефективними блоками, наявність дискретного графічного прискорювача (GPU) не є обов'язковою;

– оперативна пам'ять (RAM): щонайменше 4 ГБ вільної пам'яті. Оскільки інференс (передбачення) нейромережі відбувається на центральному процесорі, система чутлива до пропускної здатності пам'яті для швидкого переміщення матриць зображень між кешем процесора та оперативною пам'яттю (рекомендується використання двоканального режиму);

– пристрій захоплення зображення: стандартна RGB-вебкамера з роздільною здатністю від 640x480 пікселів та стабільною частотою кадрів не менше 30 FPS. Окрім роздільної здатності, важливою вимогою є кут огляду об'єктива (Field of View, FOV) не менше 60 градусів, що дозволяє захопити верхню частину тулуба та

розмах рук оператора на робочій відстані 1.5–2 метри від монітора. Ефективність детекції також залежить від рівня освітлення: недостатнє світло спричиняє розмиття в русі (motion blur) на матриці камери, що знижує точність позиціонування ключових точок. Використання спеціалізованих інфрачервоних датчиків або камер глибини не вимагається, що суттєво підвищує доступність системи.

Програмне середовище розробленого комплексу базується на мові програмування Python версії 3.9 (або вище). Вибір Python зумовлений його роллю як високорівневого оркестратора, що підтримує оптимізовані C++ біндінги (bindings) для ресурсомістких бібліотек. Розгортання потребує встановлення наступних пакетів через менеджер залежностей: opencv-python, mediapipe та pydirectinput. Бібліотека OpenCV виконує не лише функцію асинхронного захоплення кадрів з буфера камери, але й забезпечує конвертацію колірних просторів та рендеринг графічного інтерфейсу (HUD).

Ключовою інженерною особливістю розробленого API є механізм його інтеграції з цільовим імерсивним середовищем (сучасним ігровим рушієм або симулятором), під яке воно було спроектоване [22]. На відміну від традиційних методів інтеграції, які використовують мережеві сокети, модифікацію вихідного коду програми або ін'єкції в пам'ять процесу (DLL-injection), дана система функціонує як незалежне проміжне програмне забезпечення (Middleware).

Процес підключення API до середовища реалізується через механізм абсолютної апаратної абстракції:

1) програма працює у фоновому режимі, зчитуючи біометричні координати оператора та перетворюючи їх на логічні стани.

2) традиційні засоби автоматизації інтерфейсу (наприклад, стандартні функції Windows SendMessage) ігноруються більшістю сучасних тривимірних рушіїв, оскільки останні зчитують стан периферії безпосередньо через апаратні порти. Тому модуль pydirectinput звертається до низькорівневих функцій операційної системи (зокрема, до бібліотеки winuser.h [23] та функції SendInput [24]).

3) ОС Windows формує структури даних типу INPUT з прапорцем

KEYEVENTF_SCANCODE і генерує апаратні скан-коди переривань (DirectX Input), які повністю ідентичні сигналам від фізичної USB-клавіатури або миші.

4) цільовий ігровий рушій (наприклад, Unity або Unreal Engine), у якому функціонує цифровий двійник, перехоплює ці сигнали штатною підсистемою обробки вводу на рівні ядра (Ring 0).

Такий підхід забезпечує повну прозорість розробленого API для цільового додатка. Графічний рушій не розпізнає програмної емуляції, сприймаючи вхідні дані як дії реального користувача за фізичною клавіатурою. Це не лише гарантує 100% сумісність із будь-якими програмами та захист від систем Anti-Cheat, але й дозволяє гнучко переналаштовувати API під різні цифрові середовища шляхом простої зміни словника розпізнаваних клавіш (наприклад, перепризначення тригера нахилу з клавіші «A» на іншу системну команду) без втручання в архітектуру самої гри. Операційна сумісність даного рішення обмежується сімейством ОС Windows (Windows 10, Windows 11), оскільки механізм емуляції тісно пов'язаний зі специфікою драйверів вводу/виводу ядра Windows NT.

Комплексне використання цих програмних та апаратних рішень дозволяє досягти загальної наскрізної затримки системи (End-to-End Latency) в межах 30–50 мілісекунд. З них близько 10–15 мс витрачається на інференс моделі MediaPipe, а решта часу йде на препроцесинг, генерацію переривань та відмальовування кадру. Такий показник є цілком прийнятним для керування цифровим двійником, забезпечуючи комфортну синхронізацію між фізичними рухами оператора та віртуальними діями.

3 РЕАЛІЗАЦІЯ ПРИКЛАДНОГО ПРОГРАМНОГО ІНТЕРФЕЙСУ ВІДЕО-КЕРУВАННЯ ЦИФРОВИМ ДВІЙНИКОМ

3.1 Реалізація програмного забезпечення

Практична реалізація розроблюваного прикладного програмного інтерфейсу відео-керування орієнтована на створення стійкого, кросплатформного Middleware-модуля. Основним інженерним завданням на етапі реалізації є побудова архітектури, здатної безперервно транслювати аналогові біометричні дані у низькорівневі системні події без накопичення обчислювальної затримки.

Під час практичної реалізації прикладного програмного інтерфейсу відео-керування цифровим двійником інформаційне забезпечення було структуровано для мінімізації затримки обробки даних. Воно визначає архітектуру вхідних та вихідних потоків, логічну організацію обробки біометричних ознак та параметричну модель налаштування системи.

Вхідна інформація системи обробляється в межах головного циклу програми та поділяється на два основні потоки: – первинний відеопотік: неструктурований масив кадрів, отриманий у режимі реального часу з RGB-вебкамери. Кожен кадр захоплюється за допомогою методів бібліотеки OpenCV, має колірну модель BGR та фіксовану роздільну здатність для забезпечення стабільного фреймрейту під час роботи нейромережі; – параметричні константи: набір числових порогів (Thresholds), заданих в архітектурі скрипта, що визначають чутливість системи до рухів оператора. До ключових параметрів належать STEER_THRESHOLD (мінімальний поріг нахилу тулуба), PUNCH_FORWARD_SPEED (швидкість динамічного випаду руки) та VISIBILITY_THRESHOLD (рівень довіри до детекції точок моделлю MediaPipe).

Вихідна інформація розробленої системи також має дворівневу структуру, розділяючи системні команди та візуальний зворотний зв'язок: – системні події (OS Events): дискретні сигнали керування, що генеруються модулем pydirectinput та транслюються через низькорівневі API Windows у віртуальні апаратні скан-коди клавіатури (наприклад, натискання клавіш «A», «D», «Space») та події кліків кнопок миші; – візуальна телеметрія (HUD): графічна інформація, що накладається

поверх оригінального відеопотоку за допомогою функцій cv2.putText. Вона включає відображення координат активних точок скелета, текстові індикатори поточного стану системи (наприклад, «READY» або «LOAD!» для відкату рук), а також логування активних тригерів дій оператора.

Архітектурна побудова системи базується на концепції модульного конвеєра обробки даних (Data Pipeline). Програмний комплекс функціонує в межах єдиного життєвого циклу, де інформаційні зв'язки між елементами мають суворо послідовний характер.

Взаємодія компонентів розпочинається з модуля захоплення відео, який ініціалізує оптичний сенсор і передає сирий матричний масив пікселів до підсистеми препроцесингу. Тут кадр дзеркально відображається та трансформується у потрібний колірний простір. Очищений та підготовлений тензор даних надходить до нейромережевого конвеєра MediaPipe Pose, який виконує просторову регресію та повертає структурований масив об'єктів ключових точок тіла.

Отримані координати аналізуються ядром математичної логіки, яке обчислює евклідові відстані, вектори та центроїди, зіставляючи їх із конфігураційними константами чутливості.

Якщо жест розпізнано, інформаційний сигнал передається до підсистеми стабілізації та емуляції системного вводу, яка безпосередньо взаємодіє з підсистемами переривань операційної системи. Паралельно з цим, результати обробки та графова маска скелета передаються до модуля візуального інтерфейсу (HUD) для відображення зворотного зв'язку на екрані користувача.

Оскільки розробка ведеться на мові Python, модель даних представлена як ієрархія класів та словників станів.

Діаграма класів (Class Diagram) системи включає:

— Landmark Model: Об'єкт, що містить 33 точки зі властивостями x (float), y (float), z (float) та visibility (float);

— Key State Manager: Словник (Dictionary), де ключем є назва клавіші (string), а значенням – її поточний логічний стан (boolean);

— Gesture Profiler: структура констант, що зберігає математичні пороги

для детекції;

Попри те, що програмний комплекс реалізовано за допомогою функціонально-орієнтованого підходу мови Python для мінімізації накладних витрат пам'яті, його архітектурна бізнес-логіка піддається чіткій об'єктній декомпозиції. Проектовану діаграму класів можна представити через взаємодію кількох ключових логічних сутностей (рисунок 3.1).

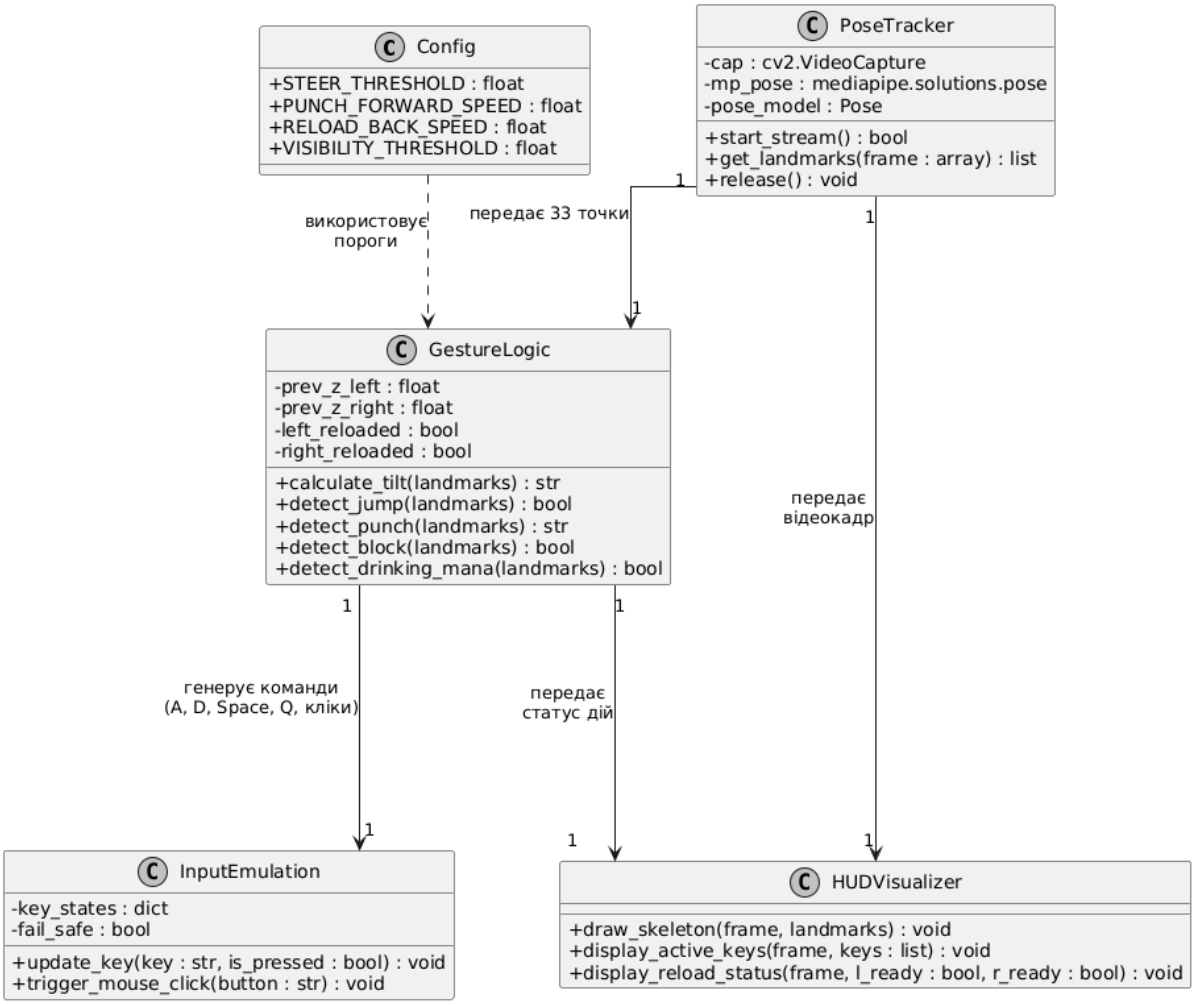

Рисунок 3.1 – UML-діаграма класів

Центральним керуючим компонентом системи виступає клас PoseTracker, який відповідає за ініціалізацію життєвого циклу програми, управління процесом відеозахоплення та виклик методів нейромережевого конвеєра MediaPipe. Програмна логіка розпізнавання рухів інкапсульована у спеціалізованому модулі GestureInterpreter. Цей компонент реалізує алгоритми кінематичних обчислень,

зокрема визначення вектора нахилу тулуба (tilt) та аналіз градієнта зміни просторової глибини за віссю Z для реєстрації динамічних випадів (ударів).

Клас InputEmulationEngine відповідає за збереження поточних станів клавіш та реалізацію алгоритму гістерезису. Він керує словником станів і викликає низькорівневі методи затискання та відпускання кнопок. Візуальне представлення даних покладено на клас HUDVisualizer, який інкапсулює логіку малювання графічних елементів і виведення текстових метрик на кадр за допомогою засобів комп'ютерного зору.

Елементи користувацького інтерфейсу та ядро системи безперервно змінюють свої стани залежно від дій оператора в кадрі. Логіку функціонування системи та HUD можна описати за допомогою детермінованого автомата станів (рисунок 3.2).

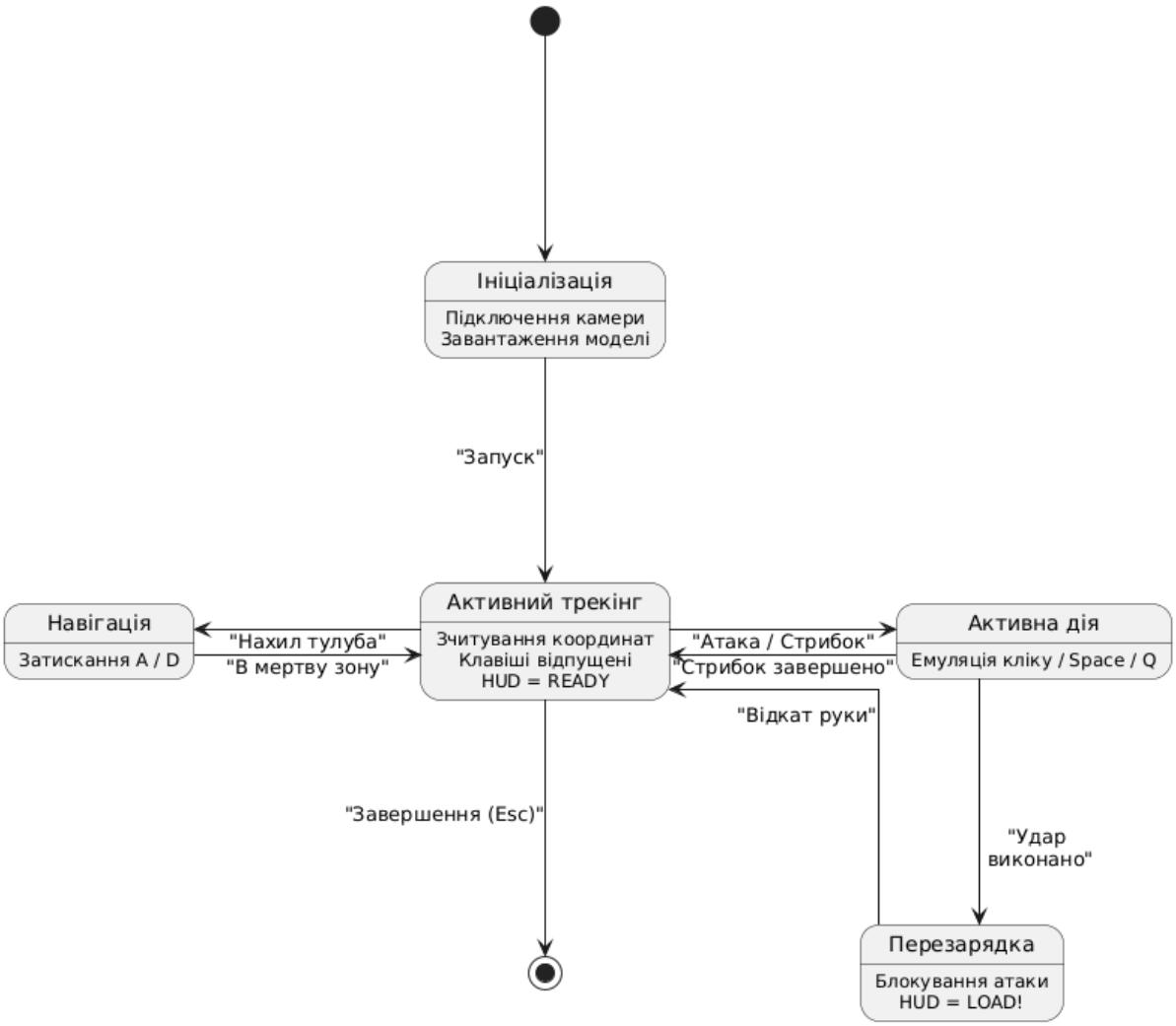

Рисунок 3.2 – UML-діаграма станів

Основним станом системи є стан очікування або ініціалізації, під час якого програма перевіряє доступність вебкамери та завантажує нейромережеву модель у пам'ять. Після успішного старту система переходить у стан активного трекінгу (Active Tracking), де на екрані відображається відеопотік із накладеною сіткою скелета, а всі індикатори клавіш мають статус False.

При виконанні оператором певного руху (наприклад, нахилу корпусу або випаду руки), показники виходять за межі встановлених порогів чутливості, і відповідний елемент інтерфейсу переходить у стан тригера (Gesture Triggered). У цей момент на HUD текстовий маркер змінює колір або статус на ACTIVE, а емулятор фіксує затискання клавіші.

Для динамічних жестів (таких як удари руками) передбачено специфічний стан перезарядки (Action Cooldown / Reload). Після фіксації швидкого руху вперед за віссю Z, система блокує повторне викликання команди, поки рух кисті оператора назад не перевищить поріг швидкості, визначений константою RELOAD_BACK_SPEED. Після повернення кінцівки в нейтральне положення, інтерфейс HUD знову відображає статус READY, повертаючи автомат у стан базового моніторингу.

Вибір технологічного стека продиктований вимогами до швидкодії, кросплатформності та здатності працювати на апаратному забезпеченні без дискретних графічних прискорювачів.

Мова програмування Python обрана як провідна платформа для розробки інтелектуальних систем завдяки наявності високоефективних біндінгів до низькорівневих C++ бібліотек комп'ютерного зору. Застосування бібліотеки OpenCV дозволяє реалізувати оптимізоване захоплення та препроцесинг кадрів із мінімальним використанням ресурсів центрального процесора.

Інтеграція фреймворку Google MediaPipe (модель BlazePose) є ключовим інженерним рішенням проекту. На відміну від альтернативних важких нейромереж (наприклад, OpenPose), архітектура BlazePose використовує комбінований підхід з детектора обличчя/тулуба та регресійної моделі точок, що дозволяє досягати стабільних 30+ кадрів за секунду (FPS) безпосередньо при інференсі на CPU.

Особливе значення має використання бібліотеки PyDirectInput замість стандартних засобів емуляції вводу ОС. Більшість сучасних віртуальних середовищ та ігрових рушіїв використовують низькорівневий API DirectInput (або XInput) для опитування периферійних пристроїв, повністю ігноруючи стандартні синтетичні повідомлення віконного менеджера Windows. PyDirectInput генерує реальні скан-коди апаратних переривань на рівні драйвера ОС, що гарантує абсолютну прозорість розроблюваного API для будь-якого зовнішнього цифрового двійника чи комп'ютерної гри.

Архітектурний опис структури програмного коду описаний нижче.

Повний вихідний код, що реалізує описану бізнес-логіку системи, винесено у відповідні додатки кваліфікаційної роботи, проте його структурну організацію доцільно розглянути в межах даного розділу. Програма має чіткий лінійний поділ на три функціональні блоки:

1) конфігураційний блок: розташований на початку програми, де задаються глобальні математичні константи (швидкості випадів, координатні пороги нахилів, ліміти часу затискання ATTACK_HOLD_SEC та фільтри видимості). Це дозволяє гнучко адаптувати API під архітектуру конкретного ігрового середовища без зміни логіки самого алгоритму;

2) допоміжна підсистема станів (update_key): функція, яка приймає назву клавіші та логічний прапорець її необхідного стану. Вона виконує роль прошарку стабілізації: перевіряє поточне значення у словнику key_states, викликає методи pydirectinput.keyDown або keyUp лише в моменти реальної зміни стану, усуваючи дублювання сигналів;

3) головний операційний цикл життєвого циклу: реалізує логіку "while True", де відбувається безперервний ітераційний процес обробки кадрів, математичний аналіз кінематичних зв'язків між точками, трекінг часових міток для розрахунку час перезарядки та динамічне оновлення інтерфейсу HUD. Для підвищення надійності системи параметр безпеки pydirectinput.FAILSAFE примусово переведено у стан False, що запобігає аварійному завершенню скрипту при випадковому зміщенні курсору миші до кутів екрана операційної системи.

Програмний код представлений в додатку А.

3.2 Інтерфейс користувача

Користувацький інтерфейс у розроблюваному прикладному програмному інтерфейсі відео-керування проектувався з урахуванням концепції імерсивної взаємодії. Оскільки система функціонує як проміжне програмне забезпечення (Middleware), що працює у фоновому режимі або паралельно з керованим цифровим двійником, класичний графічний інтерфейс (GUI) із традиційними елементами керування (кнопками, формами, випадними меню) є концептуально недоцільним. Замість нього реалізовано систему візуального зворотного зв'язку реального часу – накладений дисплей (Head-Up Display, HUD), який рендериться безпосередньо поверх вхідного відеопотоку.

Згідно з базовими принципами проектування людино-машинних інтерфейсів (HCI), візуальна підсистема розроблена з урахуванням наступних вимог та адаптацій:

– допомога у виконанні завдань (інтуїтивність та легкість освоєння): головне завдання інтерфейсу на етапі калібрування – дати користувачеві чітке розуміння того, як саме алгоритм комп'ютерного зору інтерпретує його рухи. Для цього на відеопотік у режимі реального часу накладається графова топологічна модель кінематичного скелета (за допомогою утиліт малювання mp.solutions.drawing_utils). Оператор візуально контролює зв'язки між ключовими анатомічними вузлами, що дозволяє йому інтуїтивно коригувати своє положення в кадрі, освітлення чи дистанцію до об'єктива, не вивчаючи складних технічних інструкцій;

– усунення просторового дискомфорту та ергономіка взаємодії: щоб користувач не відчував когнітивного дисонансу під час роботи з програмою, інтерфейс в обов'язковому порядку рендерить дзеркально відбитий відеопотік. Це створює природний ефект «цифрового дзеркала», де фізичний рух правої руки оператора синхронно відповідає руху правої руки на екрані монітора, що є критично важливим для збереження орієнтації у просторі;

– інформативність та запобігання помилковим діям (Error Prevention): для забезпечення користувача актуальною інформацією щодо поточного стану

математичної моделі, на HUD виводиться структурована текстова телеметрія (за допомогою функції cv2.putText), яка розділена на два логічні блоки:

а) індикатор активних дій (ACTIVE): динамічний рядок, що логує всі поточні затиснуті клавіші та системні тригери (наприклад, «ACTIVE: A SPACE LMB» або специфічний маркер «DRINKING»). Це дозволяє оператору миттєво верифікувати, чи правильно система розпізнала його позу, і чи не відбулося хибного спрацьовування;

б) індикатори станів перезарядки (READY / LOAD!): ключовий елемент обробки хибних дій та зворотного зв'язку для динамічних жестів. Оскільки система вимагає повернення руки після удару (механізм гістерезису та часу перезарядки), користувач може не розуміти, чому його наступний швидкий випад не реєструється. Кольорові текстові індикатори (зелений «READY» для базового стану та червоний «LOAD!» для стану перезарядки) для лівої (L) та правої (R) руки чітко сигналізують оператору, що системі потрібен зворотний рух кінцівки для фіксації наступної атаки. Також виводиться розрахована поточна відстань від зап'ястка до обличчя (Dist Face), що допомагає відкалібрувати амплітуду рухів під свій зріст;

– гнучкість та можливість відключення для досвідчених користувачів: враховуючи, що API призначене для роботи як фоновий драйвер вводу, рендеринг відеопотоку та накладання HUD створюють додаткове навантаження на центральний процесор. Згідно з принципами адаптивного дизайну, архітектура програми дозволяє дослідникам та досвідченим користувачам повністю відключити візуалізацію (модуль cv2.imshow), перевівши систему з режиму налагодження (Debug) у режим продуктової експлуатації (Production). У цьому стані працює виключно математичне ядро обробки та генерація системних переривань, що максимізує швидкодію комп'ютера під час гри.

Загальний вигляд розробленого користувацького інтерфейсу в режимі активного трекінгу, що демонструє роботу графової сітки та текстової телеметрії, наведено на рисунку 3.3.

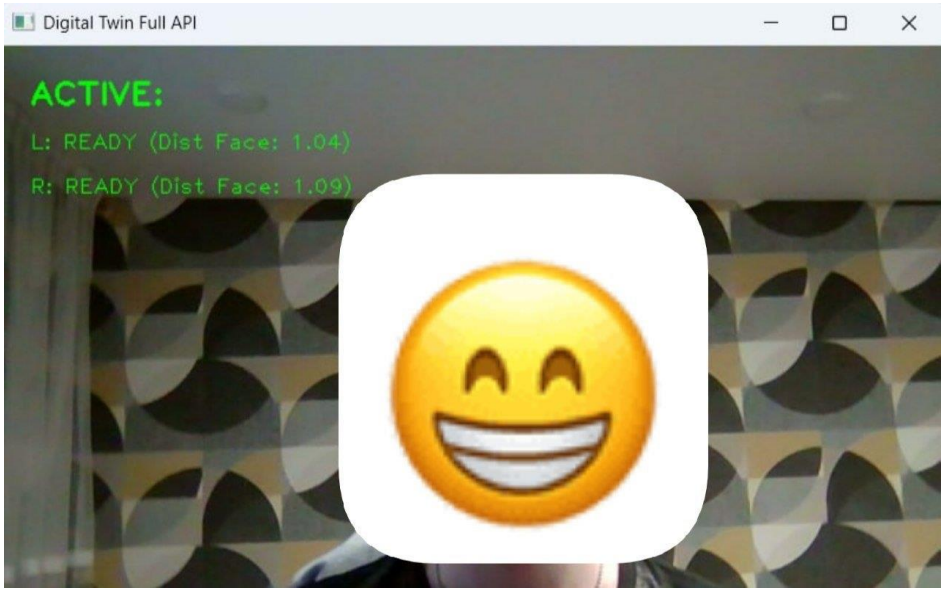

Рисунок 3.3 – Візуальний інтерфейс користувача (HUD) у режимі активного трекінгу

Таким чином, розроблений інтерфейс виконує роль технологічного вікна (Dashboard) у внутрішні процеси нейромережі, забезпечуючи швидку адаптацію нових користувачів та надаючи вичерпний інструментарій для діагностики і калібрування системи перед сеансом безконтактного керування.

3.3 Тестування розробленої програми

Тестування розробленого прикладного програмного інтерфейсу відео-керування проводилося з метою перевірки коректності розпізнавання біометричних жестів, стабільності роботи конвеєра даних та надійності емуляції системних переривань. Оскільки система базується на обробці неструктурованого відеопотоку в реальному часі, основним методом валідації було обрано функціональне тестування за принципом «чорного ящика» [25] з фокусом на граничні умови (Edge Cases) та стрес-тестування поведінки алгоритму при нестандартних рухах оператора.

У процесі ітеративної розробки та проведення тестових сесій було виявлено та усунуто низку критичних проблем, які впливали на керованість цифрового двійника. Результати тестування ключових сценаріїв наведено нижче.

Тест-кейс 1: обробка мікрофлуктуацій на межі порогу чутливості (англ. Keyboard Chattering):

— опис сценарію: оператор виконує навігаційний жест (нахил тулуба) і фіксує тіло точно на межі координати активації (STEER_THRESHOLD = 0.05);

— виявлена проблема: природний м'язовий тремор та шум оптичного сенсора призводили до того, що значення tilt постійно коливалося навколо порогового значення. Це викликало багаторазове відправлення подій keyDown та keyUp зі швидкістю понад 20 разів на секунду. Цифровий двійник у керованому середовищі починав хаотично «смикатися» на місці;

— результат доопрацювання: Запроваджено словник станів (key_states). Повторне тестування підтвердило, що система тепер надсилає сигнал натискання лише один раз і чекає гарантованого повернення користувача в «мертву зону» для відпускання клавіші. Аномалію повністю усунуто.

Тест-кейс 2: калібрування динаміки ударів (Z-Axis Depth Testing):

— опис сценарію: виконання серії швидких випадів рукою вперед для емуляції натискання кнопок миші (атаки);

— виявлена проблема: початковий алгоритм фіксував зміну глибини, але реєстрував один фізичний рух як серію з 3-4 ігрових ударів. Крім того, зворотний рух руки (повернення у вихідну позицію) іноді хибно розпізнавався нейромережею як нова атака;

— результат доопрацювання: впроваджено логіку часу перезарядки (RELOAD_BACK_SPEED = -0.05). Тестування оновленого механізму показало, що атака успішно фіксується лише при різкому подоланні порогу PUNCH_FORWARD_SPEED, а наступна стає можливою виключно після того, як індикатор HUD для відповідної руки змінить статус із LOAD! на READY.

Тест-кейс 3: втрата видимості анатомічних вузлів (англ. Occlusion Handling):

— опис сценарію: рука оператора різко виходить за межі кута огляду вебкамери або перекривається іншим об'єктом;

— виявлена проблема: фреймворк MediaPipe намагався екстраполювати (вгадати) положення втраченої точки. Це призводило до різкого стрибка координат і, як наслідок, до мимовільної активації стрибка або атаки.

— результат доопрацювання: активовано жорстку фільтрацію за параметром довіри (VISIBILITY_THRESHOLD = 0.8). Після цього тестування показало, що система коректно ігнорує «невидимі» точки, блокуючи виконання жестів до моменту повернення кінцівки в кадр.

Тест-кейс 4: системні конфлікти бібліотеки емуляції (Failsafe Crash):

— опис сценарію: під час активного керування цифровим двійником оператор випадково виконує рух мишею, що зсуває системний курсор у кут екрана ОС;

— виявлена проблема: бібліотека pydirectinput має вбудований механізм безпеки: якщо курсор досягає кута екрана, генерується виняток FailSafeException, що призводило до миттєвого аварійного завершення (Crash) всього програмного коду прикладного програмного інтерфейсу;

— результат доопрацювання: параметр pydirectinput.FAILSAFE було примусово переведено у стан False. Стрес-тестування з хаотичними рухами миші підтвердило стабільність процесу – аварійні зупинки API припинилися.

Проведені процедури функціонального тестування підтвердили працездатність розробленого прикладного програмного інтерфейсу. Програмний комплекс успішно ініціалізує відеопотік, стабільно розпізнає ключові точки з частотою, достатньою для керування в реальному часі, та коректно транслює їх у системні команди. Впроваджені механізми стабілізації (гістерезис, пороги видимості, час перезарядки) довели свою ефективність, усунувши проблеми хибних спрацьовувань та забезпечивши високий рівень ергономіки взаємодії користувача з цифровим двійником.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було:

1. Проведено огляд та аналіз сучасних технологій людино-машинної взаємодії, методів комп'ютерного зору та просторових обчислень. На основі цього обґрунтовано доцільність створення системи керування з використанням стандартної RGB-вебкамери замість дорогого обладнання.

2. Здійснено постановку задачі проектування. Визначено ключові вимоги до швидкодії та обрано технологічний стек для створення проміжного програмного забезпечення (Middleware).

3. Спроектовано архітектуру прикладного програмного інтерфейсу. Побудовано конвеєр обробки даних та розроблено логіку стабілізації сигналів на базі гістерезису для усунення хибних спрацьовувань.

4. Розроблено алгоритм роботи прикладного програмного інтерфейсу. Використання евклідової геометрії дозволило створити стійку математичну модель, яка розпізнає жести незалежно від відстані користувача до об'єктива.

5. Здійснено програмну реалізацію системи керування цифровим двійником мовою Python із накладеним графічним інтерфейсом користувача (HUD). Система генерує апаратні скан-коди на рівні драйвера ОС, що гарантує сумісність із будь-якими ігровими рушіями без модифікації їхнього коду.

6. Проведено функціональне тестування розробленого програмного комплексу та оцінено коректність обробки просторових інтенцій оператора. Результати підтвердили стабільність обробки даних у реальному часі та високу точність розпізнавання рухів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Grieves M., Vickers J. Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems. Transdisciplinary Perspectives on Complex Systems. Springer, Cham, 2017. P. 85-113.
2. El Saddik A. Digital Twins: The Convergence of Multimedia Technologies. IEEE MultiMedia. 2018. Vol. 25, no. 2. P. 87–92.
3. Barricelli B. R., Casiraghi E., Fogli D. A Survey on Digital Twin: Definitions, Characteristics, Applications, and Design Implications. IEEE Access. 2019. Vol. 7. P. 167653–167671.
4. Wachs J. P., Kölsch M., Stern H., Edan Y. Vision-based hand-gesture applications. Communications of the ACM. 2011. Vol. 54, no. 2. P. 60–71.
5. Dix A., Finlay J., Abowd G. D., Beale R. Human-Computer Interaction. 3rd ed. Prentice Hall, 2003. 834 p.
6. Shekhar S., Vold P. Spatial computing. Cambridge, MA: The MIT Press, 2020. 256 p.
7. Kumari S., Tiwari S., Tyagi A. K. Spatial Computing: Opportunities and Challenges for the Next Decade. International Conference on Hybrid Intelligent Systems. Cham: Springer Nature Switzerland, 2023. P. 143-155.
8. Goel A., Goel A. K., Kumar A. The role of artificial neural network and machine learning in utilizing spatial information. Spatial Information Research. 2022. Vol. 31, No. 3. P. 275-284.
9. Szeliski R. Computer Vision: Algorithms and Applications. 2nd ed. Cham : Springer, 2022. 926 p.
10. Zhang Z. Microsoft Kinect Sensor and Its Effect. IEEE MultiMedia. 2012. Vol. 19, No. 2. P. 4-10.
11. Офіційна документація бібліотеки комп'ютерного зору OpenCV. URL: https://docs.opencv.org/.
12. Bradski G., Kaehler A. Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library. Sebastopol : O'Reilly Media, 2017. 1018 p.

13. Cao Z., Hidalgo G., Simon T., Wei S. E., Sheikh Y. OpenPose: Realtime Multi-Person 2D Pose Estimation. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2021. Vol. 43, No. 1. P. 172-186.

14. Воєвудський О.О., Турченко І.В. Безконтактний інтерфейс відео-керування цифровим двійником в імерсивному цифровому середовищі. ICSPM 2026: Інтелектуальні обчислювальні системи та управління проектами: збірник тез доповідей міжнародної науково-практичної конференції студентів і молодих вчених (Тернопіль, 21-22 травня 2026 р). Тернопіль: ЗУНУ, 2026. С.23-26.

15. Офіційна документація мови програмування Python 3. URL: https://docs.python.org/3/.

16. Документація Google MediaPipe: Pose Landmark Detection Guide. URL: https://developers.google.com/mediapipe/solutions/vision/pose_landmarker.

17. Офіційна документація бібліотеки PyDirectInput. URL: https://pypi.org/project/PyDirectInput/.

18. Lugaresi C. et al. MediaPipe: A Framework for Perceiving and Processing Reality. CVPR Workshop on Computer Vision for Augmented and Virtual Reality. 2019. URL: https://xr.cornell.edu/.

19. Кормен Т., Лейзерсон Ч., Рівест Р., Стайн К. Вступ до алгоритмів. 3-тє вид. Київ: К.І.С., 2019. 1288 с.

20. Smith S. W. The Scientist and Engineer's Guide to Digital Signal Processing. San Diego : California Technical Publishing, 1997. 640 p.

21. Black P. E. Euclidean distance. NIST Dictionary of Algorithms and Data Structures. URL: https://www.nist.gov/dads/HTML/euclidndstnc.html

22. Qi Q., Tao F. Digital Twin and Big Data Towards Smart Manufacturing and Industry 4.0: 360 Degree Comparison. IEEE Access. 2018. Vol. 6. P. 3520–3533.

23. Python Software Foundation. ctypes – A foreign function library for Python. Python 3.10 documentation. URL: https://docs.python.org/3/library/ctypes.html.

24. Microsoft Learn. SendInput function (winuser.h) – Win32 API. Microsoft Windows Console. URL: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-sendinput.

25. ДСТУ ISO/IEC 25010:2016. Системна та програмна інженерія. Вимоги до якості систем і програмних засобів та її оцінювання (ISO/IEC 25010:2011, IDT). Вид. офіц. Київ: ДП «УкрНДНЦ», 2017. 33 с.

26. Комар М.П., Саченко А.О., Васильків Н.М., Гладій Г.М., Коваль В.С., Ліп'яніна-Гончаренко Х.В. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки» за першим (бакалаврським) рівнем вищої освіти. Тернопіль: ЗУНУ, 2024. 52 с.

Додаток А

Код програми

```python
import cv2
import mediapipe as mp
import pydirectinput
import time
import math

pydirectinput.FAILSAFE = False


PUNCH_FORWARD_SPEED = 0.11
RELOAD_BACK_SPEED = -0.05
SHOULDER_ALIGN_THRESHOLD = 0.20
VISIBILITY_THRESHOLD = 0.8
MANA_THRESHOLD = 0.1

ATTACK_HOLD_SEC = 0.05
ATTACK_COOLDOWN_SEC = 0.15
STEER_THRESHOLD = 0.05
CROSS_THRESHOLD = 0.15


key_states = {'space': False, 'a': False, 'd': False, 'q': False}
mouse_l_pressed = False
mouse_r_pressed = False

last_attack_time = 0
mouse_release_time = 0

l_reloaded = False
r_reloaded = False
prev_l_z = None
prev_r_z = None

def update_key(key_name, should_press):
    if should_press and not key_states[key_name]:
        pydirectinput.keyDown(key_name)
        key_states[key_name] = True
        print(f"--- KEY: {key_name.upper()} Pressed ---")
    elif not should_press and key_states[key_name]:
        pydirectinput.keyUp(key_name)
        key_states[key_name] = False
        print(f"--- KEY: {key_name.upper()} Released ---")

mp_pose = mp.solutions.pose
mp_drawing = mp.solutions.drawing_utils
cap = cv2.VideoCapture(0)

print("--- СИСТЕМА ДИНАМІЧНОГО БОЮ + МАНА ЗАПУЩЕНА ---")

with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5) as pose:
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret: break
        current_time = time.time()

        frame = cv2.flip(frame, 1)
        image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        image.flags.writeable = False
        results = pose.process(image)
        image.flags.writeable = True
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        if results.pose_landmarks:
            lm = results.pose_landmarks.landmark
            nose, l_sh, r_sh = lm[0], lm[11], lm[12]
            l_wrist, r_wrist = lm[15], lm[16]
            l_hip, r_hip = lm[23], lm[24]

            # 1. Розрахунок Z та Швидкість (Атака)
            curr_l_z = l_sh.z - l_wrist.z
            curr_r_z = r_sh.z - r_wrist.z
```

```
            l_speed = (curr_l_z - prev_l_z) if prev_l_z is not None else 0
            r_speed = (curr_r_z - prev_r_z) if prev_r_z is not None else 0

            # 2. ПЕРЕЗАРЯДКА (Reset)
            if l_speed < RELOAD_BACK_SPEED and l_wrist.visibility > VISIBILITY_THRESHOLD:
                if not l_reloaded: l_reloaded = True
            if r_speed < RELOAD_BACK_SPEED and r_wrist.visibility > VISIBILITY_THRESHOLD:
                if not r_reloaded: r_reloaded = True

            # 3. ВИРІВНЮВАННЯ ТА ЛОГІКА УДАРУ
            l_is_aligned = abs(l_wrist.y - l_sh.y) < SHOULDER_ALIGN_THRESHOLD
            r_is_aligned = abs(r_wrist.y - r_sh.y) < SHOULDER_ALIGN_THRESHOLD
            time_since_last = current_time - last_attack_time

            l_strike = l_speed > PUNCH_FORWARD_SPEED and l_reloaded and l_is_aligned and l_wrist.visibility > VISIBILITY_THRESHOLD
            r_strike = r_speed > PUNCH_FORWARD_SPEED and r_reloaded and r_is_aligned and r_wrist.visibility > VISIBILITY_THRESHOLD

            if (l_strike or r_strike) and time_since_last > ATTACK_COOLDOWN_SEC:
                pydirectinput.mouseDown(button='left')
                mouse_l_pressed = True
                last_attack_time = current_time
                mouse_release_time = current_time + ATTACK_HOLD_SEC
                if l_strike: l_reloaded = False
                else: r_reloaded = False
                print(f"!!! PUNCH !!!")

            prev_l_z, prev_r_z = curr_l_z, curr_r_z

            if mouse_l_pressed and current_time >= mouse_release_time:
                pydirectinput.mouseUp(button='left')
                mouse_l_pressed = False


            # --- ВІДНОВЛЕННЯ МАНИ (Q) ---
            # Рахуємо відстань від носа до кистей (Pythagoras)
            dist_l = math.hypot(l_wrist.x - nose.x, l_wrist.y - nose.y)
            dist_r = math.hypot(r_wrist.x - nose.x, r_wrist.y - nose.y)

            is_drinking = dist_l < MANA_THRESHOLD or dist_r < MANA_THRESHOLD
            update_key('q', is_drinking)

            # --- СТАНДАРТНІ РУХИ ---
            is_block = abs(l_wrist.x - r_sh.x) < CROSS_THRESHOLD and abs(r_wrist.x - l_sh.x) < CROSS_THRESHOLD
            if is_block and not mouse_r_pressed:
                pydirectinput.mouseDown(button='right'); mouse_r_pressed = True
            elif not is_block and mouse_r_pressed:
                pydirectinput.mouseUp(button='right'); mouse_r_pressed = False

            update_key('space', l_wrist.y < nose.y and r_wrist.y < nose.y)
            tilt = ((l_sh.x + r_sh.x) / 2) - ((l_hip.x + r_hip.x) / 2)
            update_key('a', tilt < -STEER_THRESHOLD)
            update_key('d', tilt > STEER_THRESHOLD)

            # ---HUD ---
            active = [k.upper() for k, v in key_states.items() if v]
            if mouse_l_pressed: active.append("LMB")
            if mouse_r_pressed: active.append("RMB")
            if is_drinking: active.append("DRINKING") # Позначка в HUD

            cv2.putText(image, f"ACTIVE: {' '.join(active)}", (20, 40), 1, 1.5, (0, 255, 0), 2)

            # Статус рук
            l_st = "READY" if l_reloaded else "LOAD!"
            r_st = "READY" if r_reloaded else "LOAD!"

            cv2.putText(image, f"L: {l_st} (Dist Face: {dist_l:.2f})", (20, 70), 1, 1, (0, 255, 0) if l_reloaded else (0, 0, 255), 1)
            cv2.putText(image, f"R: {r_st} (Dist Face: {dist_r:.2f})", (20, 100), 1, 1, (0, 255, 0) if r_reloaded else (0, 0, 255), 1)

            mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_pose.POSE_CONNECTIONS)

        cv2.imshow('Digital Twin Full API', image)
        if cv2.waitKey(10) & 0xFF == ord('q'): break

for k in key_states:
    if key_states[k]: pydirectinput.keyUp(k)
if mouse_l_pressed: pydirectinput.mouseUp(button='left')
if mouse_r_pressed: pydirectinput.mouseUp(button='right')
cap.release()
cv2.destroyAllWindows()
```

Додаток Б

Копія опублікованих результатів

Міністерство освіти і науки України
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

Матеріали

І Міжнародної науково-практичної конференції студентів і молодих вчених

«ICSPM 2026: Intelligent Computing Systems and Project Management /

Інтелектуальні обчислювальні системи та управління проєктами»

21–22 травня 2026 р.

м. Тернопіль - 2026

ЗМІСТ

УДК 004.5:004.93

БЕЗКОНТАКТНИЙ ІНТЕРФЕЙС ВІДЕО-КЕРУВАННЯ ЦИФРОВИМ ДВІЙНИКОМ В ІМЕРСИВНОМУ ЦИФРОВОМУ СЕРЕДОВИЩІ

Воєвудський Олександр Олександрович,
здобувач першого (бакалаврського) рівня вищої освіти,
sashaspaidr13@gmail.com

Турченко Ірина Василівна,
к.т.н., доцент, доцент кафедри інформаційно-обчислювальних систем і управління,
itu@wunu.edu.ua
ORCID: 0000-0002-9441-6669
Західноукраїнський національний університет

Керування цифровими двійниками вимагає застосування ефективних методів людино-машинної взаємодії (Human-Computer Interaction, HCI) [1]. Традиційні апаратні пристрої введення інформації, такі як клавіатури, комп'ютерні миші чи геймпади, обмежують фізичну мобільність оператора та демонструють низьку ефективність при маніпулюванні тривимірними об'єктами в режимі реального часу. Цей підхід вимагає безперервного тактильного контакту і створює штучний бар’єр між природною біомеханікою людини та цифровим середовищем. Особливо гостро ця проблема постає в умовах створення імерсивних ігрових середовищ та тренажерів, де фізичний рух користувача є ключовим елементом взаємодії

Розвиток технологій просторових обчислень (Spatial Computing) формує нову предметну область, що вивчає методи повної заміни фізичних контролерів на оптичні сенсори [2]. Зазначений підхід передбачає, що тіло оператора стає безпосереднім пристроєм введення інформації. Незважаючи на стрімке оновлення інструментів машинного навчання, на ринку спостерігається технологічний розрив: існуючі моделі [3-4] генерують неструктурований масив просторових координат, який неможливо напряму передати до ігрового рушія. Виникає об'єктивна необхідність у розробці спеціалізованого проміжного програмного забезпечення (Middleware API) безконтактного (жестового) програмного контролера, здатного виконувати фільтрацію шумів та перетворення рухів у стандартизовані апаратні команди.

Проектування програмного інтерфейсу для безконтактного відеокерування вимагає аналізу існуючих технологій захоплення руху (Motion Capture). Індустріальні рішення можна класифікувати за методом отримання просторової інформації:

1) апаратні системи глибинного зондування: Microsoft Azure Kinect, Intel RealSense використовують інфрачервоні камери та ToF-датчики для апаратної

генерації карти глибини простору. Їхнім недоліком є висока вартість обладнання, чутливість до сонячного освітлення та жорстка прив'язка до специфічних ОС;

2) інерціальні системи (IMU): натільні костюми на зразок Rokoko використовують акселерометри та гіроскопи, застосовуючи алгоритми злиття сенсорних даних (Sensor Fusion). Вони забезпечують низьку затримку, але вимагають тривалого процесу надягання костюма та страждають від проблеми накопичення похибки інтегрування [5];

3) класичний комп'ютерний зір (OpenCV): спроби реалізувати трекінг алгоритмами субтракції фону або оптичного потоку (метод Лукаса-Канаде) є вкрай нестабільними та вимагають ідеального контрасту;

4) високорівневі нейромережі (OpenPose): використовують архітектуру з полів афінності частин (PAFs). Забезпечують виняткову точність, проте вимагають наявності високопродуктивних дискретних GPU для роботи в реальному часі [6].

У таблиці 1 наведено порівняння технологій для відео-керування.

Таблиця 1 – Порівняльна характеристика технологій для відео-керування

Технологія / Рішення	Апаратна база	Алгоритмічний підхід	Основний недолік
Сенсори глибини (Microsoft Kinect)	Інфрачервоні камери, ToF-датчики	Апаратна генерація карти глибини	Висока вартість, чутливість до освітлення
Інерціальні костюми (Rokoko, Xsens)	Натільні IMU-сенсори	Злиття сенсорних даних (Sensor Fusion)	Потреба у носінні костюма, похибка гіроскопів
OpenPose	Стандартна RGB веб-камера	Важкі нейромережі (PAFs)	Вимагає потужних дискретних GPU
Пропоноване рішення (Кастомний API)	Стандартна RGB веб-камера	MediaPipe Pose, евклідова математика	Потребує етапу проєктування математичних порогів

Оптимальним компромісом для створення Middleware API є фреймворк Google MediaPipe Pose (архітектура BlazePose). Він забезпечує стабільний повноростовий трекінг 33 ключових точок тіла (Landmarks) безпосередньо на центральному процесорі (CPU) користувацького комп'ютера [7].

Розроблений програмний інтерфейс відеокерування базується на концепції конвеєрної обробки даних (Data Pipeline Architecture) у єдиному синхронному циклі. Обрана схема компонування складає п'ять взаємопов'язаних модулів:

1) підсистема захоплення відеопотоку: використовує бібліотеку OpenCV для ініціалізації веб-камери. Кожен кадр підлягає обов'язковому апаратному віддзеркаленню (для інтуїтивності управління) та конвертації колірного простору з BGR в RGB;

2) нейромережевий екстрактор: модуль на базі MediaPipe Pose працює в режимі безперервного трекінгу, повертаючи структурований тензор з 33 просторовими векторами тіла;

3) ядро математичної логіки: найбільш наукоємний компонент, який відмовився від абсолютних екранних координат. Логіка побудована на аналізі відносних векторних зміщень та обчисленні евклідових відстаней між центроїдами анатомічних вузлів;

4) підсистема емуляції системного вводу: використовує бібліотеку pydirectinput для генерації низькорівневих апаратних переривань. Це гарантує, що ігровий рушій (DirectX/OpenGL) розпізнає сигнали від віртуальної клавіатури/миші без затримок [8];

5) модуль візуального зворотного зв'язку (HUD): Накладає на відеокадр графову структуру розпізнаного кінематичного скелета та виводить телеметричні показники і статуси активних клавіш (ACTIVE).

Основною проблемою предметної області є архітектурна невідповідність між безперервною природою людських рухів та дискретним характером системного введення даних. Унаслідок природного м'язового тремору безпосереднє перетворення координат у події натискання клавіш призводить до виникнення високочастотних коливань (так званого «дребізгу»). Для усунення цього ефекту в ядро системи інтегровано механізми мертвої зони (deadzone) та гістерезису.

Математично розпізнавання жестів базується на розрахунку евклідової відстані між ключовими точками у просторі [9]:

$$D = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}, \quad (1)$$

де D – евклідова відстань;

x1, y1 – координати першої ключової точки;

x2, y2 – координати другої ключової точки.

Ця логіка інтегрована зі словником станів (State Dictionary), що функціонує як скінченний автомат. Сигнал на затискання клавіші генерується рівно один раз при перетині порогу, а відпускання – лише після повернення біометрії у нейтральну зону.

Для керування цифровим двійником у формі ігрового аватару було розроблено наступний словник семантичних трансляцій кінематики:

1) переміщення (Клавіші A / D): алгоритм розраховує центроїд осі плечей та центроїд таза. Різниця цих центроїдів формує вектор нахилу тулуба вліво або вправо, що ініціює рух;

2) стрибок (Space): реєструється, коли координати обох кистей стають візуально вищими за Y-координату носа (точка 0 за MediaPipe);

3) атака (Ліва кнопка миші): динамічний бойовий рух (різкий випад рукою вперед). Для запобігання "фантомним" атакам впроваджено систему фізичної «перезарядки» (cooldown) – наступний удар неможливий без повернення кисті до грудей;

4) блок (Права кнопка миші): захисна стійка детектується через перехрещення рук (X-координата лівої кисті перетинає вертикальну вісь правого плеча);

5) відновлення мани (Клавіша Q): імерсивний жест піднесення кисті безпосередньо до рота або носа гравця.

Розроблений програмний інтрфейс успішно закриває виявлений технологічний розрив, виконуючи функцію універсального Middleware API. Синтез методів оптичного трекінгу (MediaPipe), векторної математики гістерезису та низькорівневої емуляції вводу (pydirectinput) дозволив створити безконтактний програмний контролер інтефейсу людинно-машинної взаємодії. Інтерфейс відеокерування функціонує як «прозорий» віртуальний пристрій, забезпечуючи стабільну трансляцію фізичних рухів у команди керування цифровим двійником в реальному часі без необхідності залучення додаткового дороговартісного обладнання.

Список використаних джерел

1. Dix A., Finlay J., Abowd G. D., Beale R. Human-Computer Interaction. 3rd ed. Prentice Hall, 2003. 834 p.

2. Shekhar S., Vold P. Spatial computing. Cambridge, MA : The MIT Press, 2020. 256 p. ISBN 978-0262538046.

3. 4Kumari S., Tiwari S., Tyagi A. K. Spatial Computing: Opportunities and Challenges for the Next Decade. International Conference on Hybrid Intelligent Systems. Cham : Springer Nature Switzerland, 2023. P. 143-155

4. Goel A., Goel A. K., Kumar A. The role of artificial neural network and machine learning in utilizing spatial information. Spatial Information Research. 2022. Vol. 31, no. 3. P. 275.

5. Grieves M., Vickers J. Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems. Transdisciplinary Perspectives on Complex Systems. Springer, 2017. P. 85-113.

6. Cao Z., Hidalgo G., Simon T., Wei S. E., Sheikh Y. OpenPose: Realtime Multi-Person 2D Pose Estimation. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2021. Vol. 43, No. 1. P. 172-186

7. Lugaresi C. et al. MediaPipe: A Framework for Perceiving and Processing Reality. CVPR Workshop on Computer Vision for Augmented and Virtual Reality. 2019. URL: https://xr.cornell.edu/s/NewTitle_May1_MediaPipe_CVPR_CV4ARVR_Workshop_2019.pdf

8. PyDirectInput: Python library for controlling mouse and keyboard. URL: https://pypi.org/project/PyDirectInput/

9. Black P. E. Euclidean distance. NIST Dictionary of Algorithms and Data Structures. URL: https://www.nist.gov/dads/HTML/euclidndstnc.html