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

КОВТУНЕНКО Андрій Олександрович

Програмний модуль виявлення та відслідковування облич із використанням дрона Ryze Tello/
Software Module for Face Detection and Tracking Using Ryze Tello Drone

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

Кваліфікаційна робота

Виконав студент групи КН-42
А.О. Ковтуненко
_____

Науковий керівник
к.т.н., доцент П.Є. Биковий
_____

Кваліфікаційну роботу допущено до захисту
«____»_____20___р.
Завідувач кафедри
_____ Н.В. Дзюбановська

Тернопіль-2026 р.

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
КОВТУНЕНКУ Андрію Олександровичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: Програмний модуль виявлення та відслідковування облич із використанням дрона Ryze Tello/ Software Module for Face Detection and Tracking Using Ryze Tello Drone

керівник роботи к.т.н., доцент П.Є. Биковий
затверджений наказом по університету від 01 грудня 2025 року № 894.
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 аналіз предметної області та існуючих систем автономного відеосупроводу з використанням дронів, виявити їхні переваги та недоліки;
- обґрунтувати вибір моделі комп'ютерного зору та бібліотек для розробки;
- спроектувати та розробити архітектуру рограмного модуля, включаючи модуль асинхронної обробки відеопотоку, модуль біометричної ідентифікації цілі, та модуль керування дроном на базі пропорційно-диференціального регулятора;
- реалізувати програму у вигляді десктопного застосунку з графічним інтерфейсом, вікном відеопотоку та інтеграцією з контролером дрона DJI Tello;
- провести тестування розробленого модуля, оцінити стабільність утримання цілі та точність біометричної ідентифікації;

5. Перелік графічного матеріалу у роботі:
- дерево функцій бізнес-процесу автономного відеосупроводу;
- функціональна структура системи;
- UML-діаграма класів бізнес-логіки програмної системи;
- UML-діаграма станів графічного інтерфейсу користувача.

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	Огляд літературних джерел відповідно до предметної області 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 15.03.2026 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 _____ А.О. Ковтуненко
підпис

Керівник роботи _____ к.т.н., доцент П.Є. Биковий
підпис

АНОТАЦІЯ

Кваліфікаційна робота на тему «Програмний модуль виявлення та відслідковування облич із використанням дрона Ryze Tello» для здобуття ступеня бакалавра за спеціальністю 122 «Комп'ютерні науки» освітньо-професійної програми «Комп'ютерні науки» виконана на 74 сторінках, містить 14 рисунків, 5 додатків та 33 джерел використаної літератури.

Метою роботи є розробка програмного модуля для автономного розпізнавання та відстежування облич із використанням дрона Ryze Tello з метою забезпечення автоматизованого динамічного відеозапису рухомого доповідача в режимі реального часу.

У результаті виконання роботи розроблено програмний модуль, який забезпечує автоматичне виявлення, ідентифікацію та відстежування облич із використанням дрона Ryze Tello. Реалізовано механізм формування біометричного підпису на основі ключових точок обличчя, алгоритм повторного захоплення цілі у разі тимчасової втрати відстежування, а також систему керування польотом дрона з використанням PID-регулятора. Крім того, розроблено графічний інтерфейс користувача для моніторингу відеопотоку та керування процесом відстежування.

Результати роботи можуть бути використані для автоматизованого відеозапису лекцій, презентацій, тренінгів, конференцій, навчальних заходів та інших подій, що потребують автономного супроводу людини без залучення оператора камери.

Ключові слова: КОМП'ЮТЕРНИЙ ЗІР, РОЗПІЗНАВАННЯ ОБЛИЧ, ВІДСТЕЖУВАННЯ ОБ'ЄКТІВ, БЕЗПІЛОТНИЙ ЛІТАЛЬНИЙ АПАРАТ, RYZE TELLO, MEDIAPIPE FACE MESH, PID-РЕГУЛЯТОР, PYTHON.

ANNOTATION

Qualification work on the topic «Software Module for Face Detection and Tracking Using Ryze Tello Drone» for the Bachelor's degree in specialty 122 «Computer Science» under the educational and professional program «Computer Science» is written on 74 pages and contains 14 figures, 5 appendices and 33 references.

The aim of the work is to develop a software module for autonomous face recognition and tracking using the Ryze Tello drone in order to provide automated dynamic video recording of a moving speaker in real time.

As a result of the work, a software module was developed that provides automatic face detection, identification, and tracking using a Ryze Tello drone. A biometric signature generation mechanism based on facial landmarks, a target reacquisition algorithm for temporary tracking loss, and a drone flight control system using a PID controller were implemented. In addition, a graphical user interface was developed for video stream monitoring and tracking control.

The results of the work can be applied to automated video recording of lectures, presentations, training sessions, conferences, educational events, and other activities requiring autonomous tracking of a person without the involvement of a camera operator.

Keywords: COMPUTER VISION, FACE RECOGNITION, OBJECT TRACKING, UNMANNED AERIAL VEHICLE, RYZE TELLO, MEDIAPIPE FACE MESH, PID CONTROLLER, PYTHON.

ЗМІСТ

ВСТУП

У сучасних умовах проведення тренінгів, корпоративних презентацій та відеозйомки доповідачів дедалі більшого поширення набуває гібридний формат роботи, коли частина аудиторії присутня безпосередньо в залі, а інша переглядає трансляцію або запис онлайн [1]. Якісна відеозйомка доповідача, який активно рухається під час виступу, стала не просто технічним завданням, а важливим чинником ефективності сприйняття інформації та залучення аудиторії [2]. Динамічна зйомка дозволяє зберегти природність виступу, підвищує зацікавленість онлайн-глядачів і забезпечує можливість подальшого використання відеоматеріалів для архівування та повторного перегляду [3, 4].

На сьогодні зйомка рухомого доповідача здійснюється переважно трьома способами: за допомогою оператора, стаціонарних PTZ-камер з функціями автоматичного відстеження або мобільних пристроїв. Ручна операторська зйомка забезпечує достатню гнучкість, однак потребує постійної участі людини та може супроводжуватися помилками, спричиненими втомою або людським фактором [5]. Стаціонарні PTZ-камери з функціями штучного інтелекту здатні автоматично слідкувати за об'єктом, проте мають обмеження щодо мобільності та часто втрачають ціль при швидких переміщеннях або зміні умов освітлення [6]. Використання мобільних пристроїв забезпечує лише статичну зйомку і не дозволяє автоматично супроводжувати рухомий об'єкт. У результаті якість відеоматеріалів може суттєво знижуватися, що негативно впливає на сприйняття контенту.

Актуальність роботи обумовлена потребою створення доступних, мобільних та ефективних засобів автоматизованої динамічної відеозйомки рухомих об'єктів без залучення додаткового персоналу. Використання сучасних алгоритмів комп'ютерного зору та безпілотних літальних апаратів дозволяє реалізувати автономне розпізнавання та відстеження людини в режимі реального часу, що відкриває нові можливості для проведення презентацій, лекцій, тренінгів та інших публічних заходів.

Запропонованим рішенням є програмне забезпечення для дрона Ryze Tello, яке реалізує автономне розпізнавання обличчя та відстеження рухомої цілі в реальному часі. На відміну від стандартних алгоритмів трекінгу, що базуються лише на виявленні об’єкта, запропоноване рішення використовує адаптивну векторну біометрію. Система будує індивідуальний профіль особи на основі нейромережевої моделі Face Mesh бібліотеки MediaPipe, яка дозволяє отримувати координати 468 ключових точок обличчя в реальному часі [7]. Використання співвідношень між ключовими точками обличчя дозволяє ідентифікувати конкретну людину та ігнорувати інших осіб у кадрі навіть у випадках тимчасової втрати візуального контакту. Для стабілізації положення дрона застосовується цифровий PID-регулятор по трьох осях – повороту, висоти та дистанції до цілі залежно від її положення в кадрі [8].

Метою даної роботи є розробка програмного модуля автономного розпізнавання та відстеження обличчя людини за допомогою дрона Ryze Tello для забезпечення автоматизованої динамічної відеозйомки рухомого доповідача в режимі реального часу.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– провести аналіз предметної області та існуючих систем автономного відеосупроводу з використанням дронів;

– обґрунтувати вибір моделі комп’ютерного зору та програмних бібліотек для розробки;

– спроектувати та розробити архітектуру програмної системи, включаючи модуль асинхронної обробки відеопотоку, модуль біометричної ідентифікації цілі та модуль керування дроном;

– реалізувати програму у вигляді десктопного застосунку з графічним інтерфейсом користувача та інтеграцією з дроном Ryze Tello;

– провести тестування системи та оцінити точність біометричної ідентифікації й стабільність утримання цілі в кадрі.

Об’єктом дослідження є процес автоматизованого відеосупроводу людини за допомогою безпілотного літального апарата.

Предметом дослідження є методи комп’ютерного зору, біометричної ідентифікації та автоматичного відстеження облич із використанням безпілотного літального апарата Ryze Tello.

Методи дослідження:

– аналіз наукової літератури та існуючих програмних рішень у сфері комп’ютерного зору та автономного керування безпілотними літальними апаратами;

– використання методів цифрової обробки зображень та нейромережевих алгоритмів розпізнавання облич;

– застосування технології MediaPipe Face Mesh для екстракції біометричних ознак обличчя;

– використання методів автоматичного керування та PID-регулювання для стабілізації руху дрона;

– тестування програмного забезпечення та аналіз отриманих результатів.

Практичне значення роботи полягає у створенні програмного модуля, який забезпечує автоматичне розпізнавання та відстеження обличчя людини за допомогою дрона Ryze Tello без необхідності постійного ручного керування. Розроблене рішення може бути використане для відеозйомки лекцій, презентацій, тренінгів, конференцій, освітніх заходів та інших подій, де необхідний автоматичний супровід доповідача або іншого рухомого об'єкта.

Під час виконання та оформлення кваліфікаційної роботи враховано вимоги методичних рекомендацій щодо підготовки бакалаврських робіт за спеціальністю 122 «Комп’ютерні науки» [9].

Результати роботи були апробовані на І Міжнародній науково-практичній конференції студентів та молодих учених «Intelligent Computing Systems and Project Management (ICSPM 2026)», м. Тернопіль, Україна, та опубліковані у збірнику матеріалів конференції [10]. Текст публікації наведено у Додатку Д.

1 АНАЛІЗ АВТОНОМНОГО ВІДЕОСУПРОВОДУ ЗА ДОПОМОГОЮ ДРОНА

1.1 Опис предметної області

Предметною областю є процес автоматизованого відеосупроводу динамічних об'єктів за допомогою безпілотних літальних апаратів. Блок-схема дерева функцій зображена на рисунку 1.1.

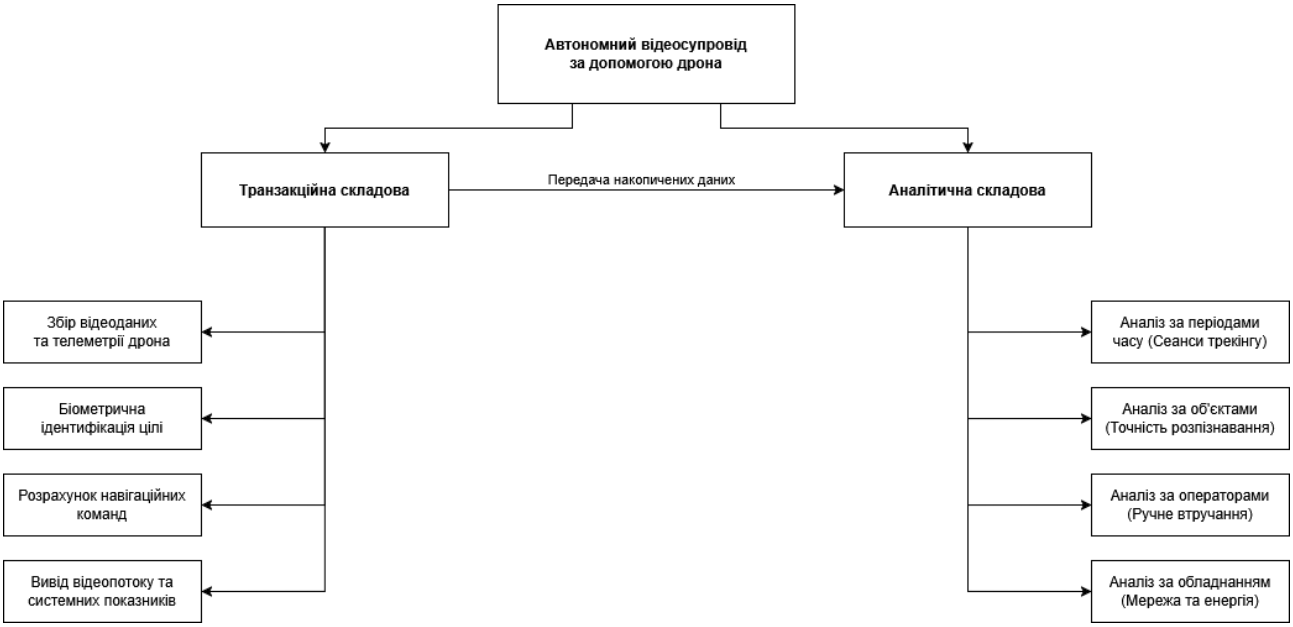

Рисунок 1.1 – Дерево функцій бізнес-процесу

Глобальний бізнес-процес «Автономний відеосупровід об'єктів за допомогою дрона» структурно розділено на дві фундаментальні складові: транзакційну та аналітичну.

Транзакційна складова забезпечує безперервний збір, накопичення та первинну обробку кількісних даних про поточний стан об’єкта управління (дрона) та зовнішнього середовища. Транзакційні функції бізнес-процесу:

– Збір відеоданих та телеметричної інформації: отримання мережевих UDP-пакетів з відеокамери, декодування кадрів та паралельне зчитування показників заряду батареї.

– Біометрична ідентифікація: обробка кадрів нейромережевою моделлю та розрахунок ступеня схожості з еталоном у реальному часі.

– Розрахунок навігаційних команд: розрахунок векторів лінійних та кутових швидкостей дрона на основі відхилення центру обличчя від центру кадру та перетворення їх у стандартизовані RC-команди для дрона.

– Вивід результатів: виведення відеопотоку та системних показників дрона у графічний інтерфейс для оператора.

Уся отримана в процесі польоту транзакційна інформація – тривалість польоту, кількість втрат цілі, рівні заряду, спрацьовування ручного перехоплення – може бути використана для подальшого аналізу.

Аналітична складова відповідає за агрегацію та глибокий аналіз кількісних показників, накопичених транзакційною складовою. Аналітична підсистема забезпечує дослідження даних у наступних розрізах і вимірах:

– За періодами часу: Аналіз ефективності відстеження облич в розрізі робочих сесій. Досліджується загальна тривалість автономного польоту, середній час безперервного утримання цілі в кадрі до її втрати, а також динаміка розряду акумулятора залежно від інтенсивності руху у різні періоди роботи системи.

– За об'єктами супроводу: Аналіз якості біометричного розпізнавання. Оцінюється середнє значення біометричної похибки для різних цілей, частота хибних спрацювань алгоритму на сторонні об’єкти та стабільність розпізнавання при різних ракурсах конкретної особи.

– За операторами системи: Аналіз взаємодії оператора з системою. Збирається статистика щодо кількості ручних втручань у процес автономного польоту, частоти використання функції екстреної посадки та середнього часу, який потрібен оператору для калібрування системи на новій цілі.

– За обладнанням та локаціями: Оцінка апаратної надійності. Аналізується рівень втрати мережевих UDP-пакетів залежно від умов закритого приміщення, теплові навантаження на процесор наземної станції під час відстеження та стабільність роботи дрона у різних зонах офісу чи аудиторії.

1.2 Огляд і аналіз існуючих рішень

Для оцінки поточного стану розвитку розпізнавання та відстеження облич в дронах було проведено аналіз існуючих проектів, які спеціалізуються у цій сфері.

Проект користувача BobMcDear реалізований на Python з використанням бібліотеки OpenCV та попередньо навченої моделі SSD на базі Caffe. Для роботи з дроном відбувається його підключення до комп'ютера через Wi-Fi, Після зльоту дрон в реальному часі обробляє відеопотік та розпізнає обличчя за допомогою DNN-модуля. Система вважає обличчям об'єкт, який перетнув поріг впевненості 0,75 і обличчя з найбільшим рівнем впевненості автоматично центрується в кадрі за допомогою простого PID-подібного регулятора. Оскільки обличчя для відстеження обирається програмою автоматично, вибрати конкретне вручну можливості немає. Через це також програма не здатна обробляти більше одного обличчя, вона також не містить механізмів відновлення трекінгу при втраті обличчя та не має функції електронної стабілізації зображення [11].

Інший проект, який реалізував користувач juanmapf97, використовує іншу модель розпізнавання, а саме каскад Хаара. Це швидка та невибаглива модель, яка може запускатися на слабких системах без проблем з продуктивністю, якщо важчі моделі система не витримує, проте вона поступається в точності попередньо навченим нейронним мережам. Дрон отримує відеопотік, виявляє обличчя та коригує положення за координатами XYZ. Програма підтримує виявлення кількох облич одночасно, проте для відстеження програма обирає одне автоматично, без можливості ручного вибору, як і в попередньому проекті. Для відстеження вибирається найближче до дрона обличчя. Через використання відносно простої моделі програма чутлива до змін освітлення, поворотів голови та швидких рухів, а також не здатна відрізняти обличчя, через що дрон зможе з легкістю зплутати одне обличчя з іншим [12].

Проект Tello Sandbox від користувача youngsoul – ще одна реалізація стеження за обличчям за допомогою Python-скрипту та OpenCV. Дрон автономно центрує об’єкт і намагається підтримувати задану відстань, використовуючи багатопроцесорну обробку відео. Для розпізнавання облич тут також використовується каскади Хаара [13]. Приклад роботи алгоритму відстеження зображений на рисунку 1.2.

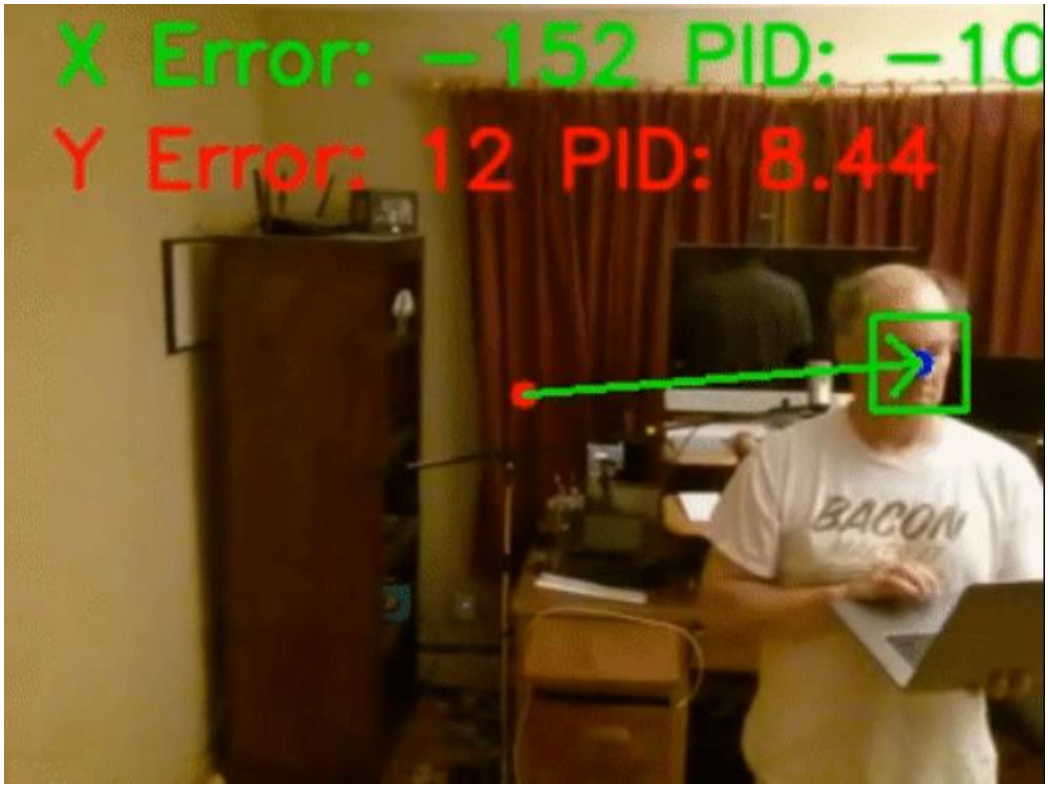

Рисунок 1.2 – Візуалізація центрування обличчя

Наукова робота Boonsongsrikul та Eamsaard пропонує систему Real-Time Human Motion Tracking by Tello EDU Drone з використанням моделі MediaPipe Holistic. Ця модель значно просунутіша за каскади Хаара чи SSD, оскільки вона не лише бачить де людина в кадрі, а й розуміє, як вона рухається. MediaPipe будує точки у ключових місцях тіла та обличчя людини, завдяки чому дрон не втрачає людину навіть якщо повернути голову в бік чи якщо освітлення не ідеальне. Схема побудови точок зображена на рисунку 1.3.

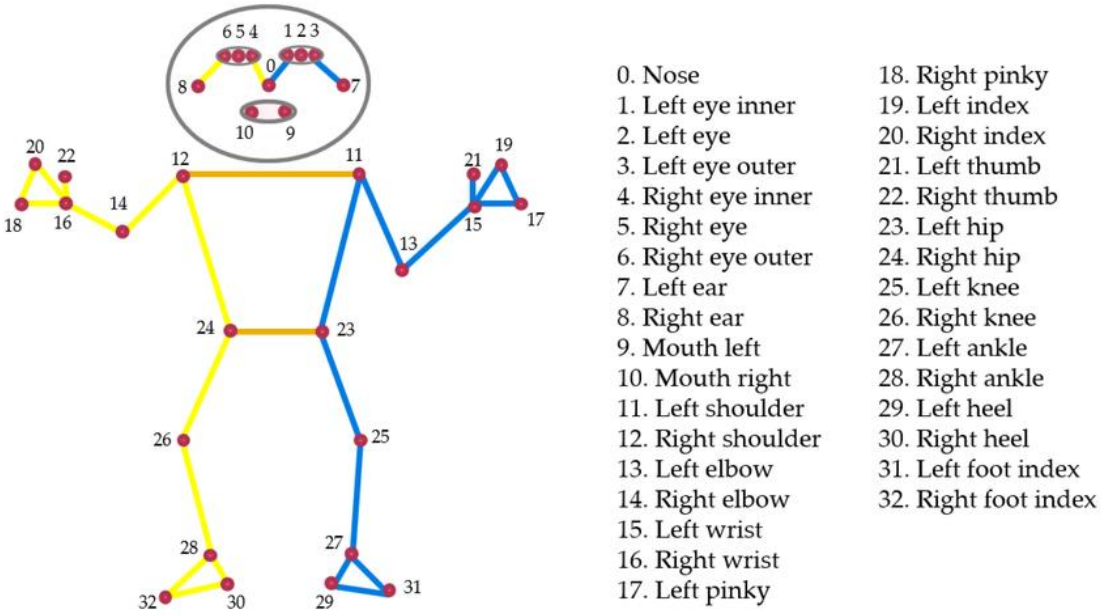

Рисунок 1.3 – Розташування точок на тілі та обличчі

Дрон стежить за людиною на дистанції 2-10 м, повертаючись або переміщаючись вслід за людиною, і під час дослідження демонстрував високу точність (96,67–100 %). Однак програма все ще не містить жодного алгоритму для відрізняння людей, тому дрон все ще може плутатись [14].

Рішення carlo98 поєднує розпізнавання облич на базі згорткових нейронних мереж з автономним трекінгом. На відміну від попередніх проектів, це рішення також пропонує алгоритм уникнення зіткнень. Він базується на попередньо навченій нейромережі, яка класифікує кадри на два типи – “шлях вільний” та “є перешкода”. Рішення приймається на основі того, чи перетнув кадр встановлений поріг схожості. Перед використанням нейромережу потрібно навчити вручну, керуючи дроном та позначаючи кадри, як free (шлях вільний) або blocked (є перешкода). Розпізнавання облич тут теж здійснюється за допомогою нейронної мережі. Створюється SVM-класифікатор, який навчається на підготовлених фотографіях, розрізняючи людей на них. Відеопотік з дрона аналізується на наявність облич у кадрі, і коли обличчя знайдене – будується вектор ознак. Вони передаються у класифікатор, який порівнює їх з ознаками облич з підготовлених фотографій, і якщо є збіг – дрон починає стежити за цим обличчям, центруючи його в кадрі. Хоча це рішення для розпізнавання облич є

дуже точним та здатне ідентифікувати конкретну людину, воно обмежене лише тими людьми, на фотографіях яких навчався класифікатор. Також постає питання в доцільності використання цього методу при відстеженні в реальному часі, оскільки побудова вектора ознак обличчя та робота класифікатора з ними може спричинити значні затримки та надмірне навантаження на систему [15].

Реалізація користувача Akbonline подібно до багатьох схожих проектів використовує OpenCV та каскади Хаара для розпізнавання та відстеження облич. Програма дає можливість як ручного керування дроном, так і автономний режим, який позиціонує виявлене обличчя в центрі кадру автоматично слідує за ним. Схема роботи програми зображена на рисунку 1.4.

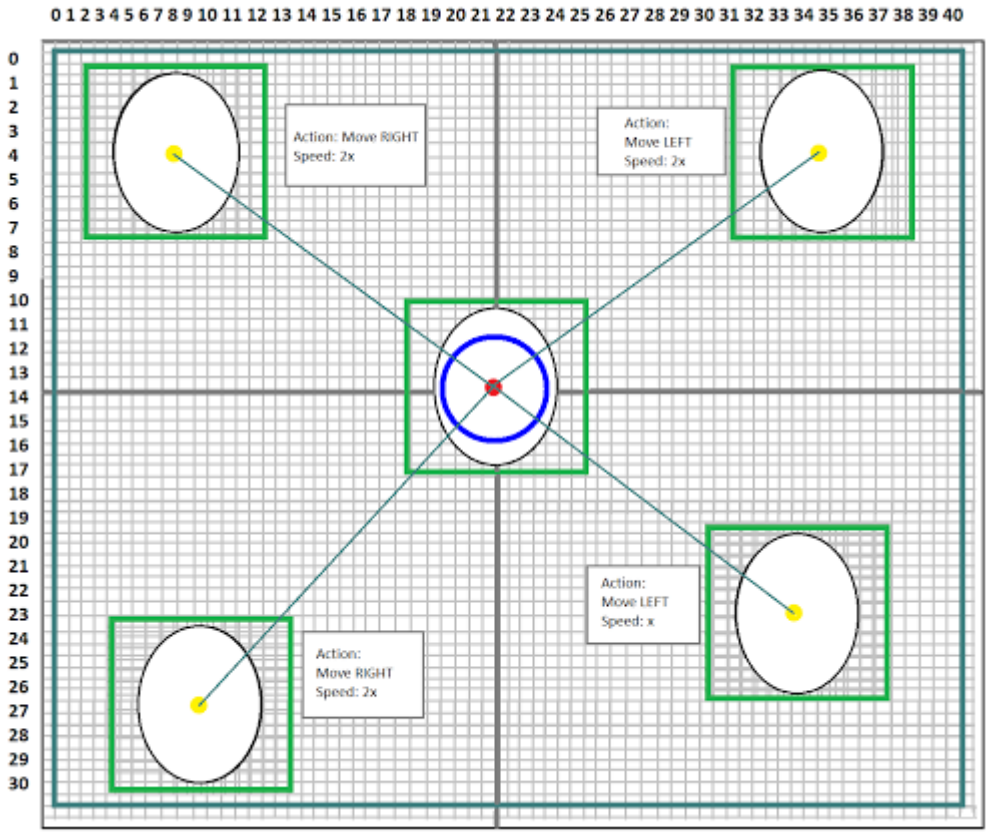

Рисунок 1.4 – Схема роботи програми

Для керування дроном запускається місцевий Flask-сервер та відкривається веб-інтерфейс. Там можна змінити тип керування (ручний/автоматичний) та режими польоту, серед яких доступні прискорений

режим, патрулювання за заданим маршрутом та виконання переворотів в повітрі [16].

Проект користувача Jacob-Pitsenberger використовує MediaPipe Face Mesh у поєднанні з OpenCV. Система забезпечує базове розпізнавання та підрахунок кількості виявлених облич, проте не передбачає відстеження облич чи векторної ідентифікації [17].

Рішення користувача shijq23 є класичною реалізацією розпізнавання та відстеження обличчя з використанням PID-контролера. Дрон зависає на заданій висоті та розпізнає найближче обличчя за допомогою каскадів Хаара, після чого центрує його в кадрі. Програма також записує відео з камери дрона та зберігає його на диску комп'ютера, до якого під'єднаний дрон. При натисканні будь-якої клавіші клавіатури скрипт зупиняється та дрон приземляється. Як і інші подібні прості рішення, ця програма не забезпечує ідентифікацію конкретної особи [18]. Скріншот роботи програми зображений на рисунку 1.5.

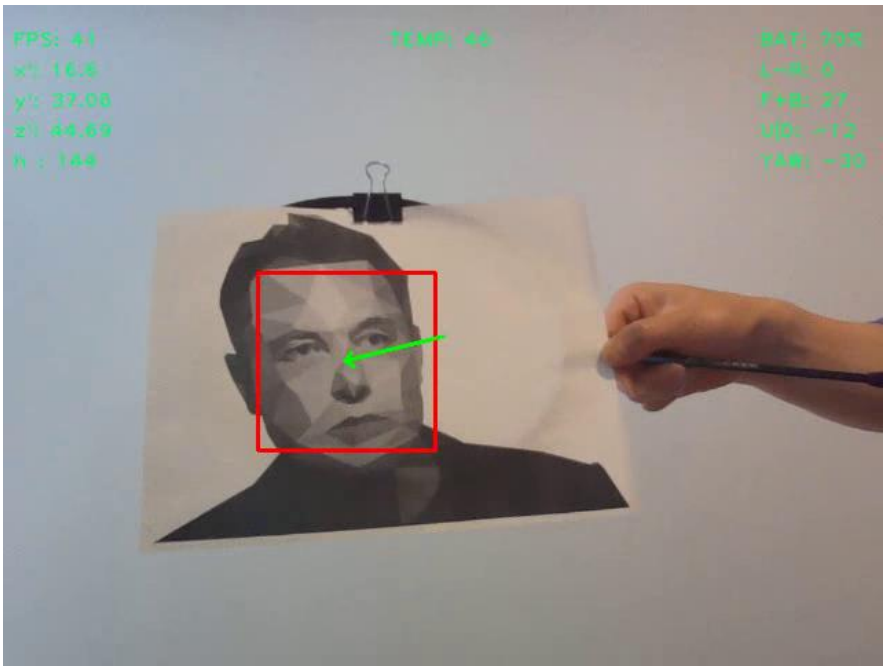

Рисунок 1.5 – Скріншот інтерфейсу програми

Більш просунутим є рішення користувача Matthewjsiv, яке переходить від простої детекції обличчя до трекінгу ключових точок тіла за допомогою PoseNet (TensorFlow). Система відстежує ніс для керування поворотами дрона і його

вертикальним положенням, а відстань між плечима та стегнами – для оцінки руху вперед/назад. Команди формуються за допомогою базового PID-регулятора, що дозволяє автоматично підтримувати людину в центрі кадру. Також є можливість перемкнутися на ручне керування і рухати дрон за допомогою клавіатури. Хоча при тестування програма продемонструвала прийнятну роботу, PID-контролер потребує доопрацювання, а відсутність розпізнавання напрямку повороту спікера призводить до неадекватної реакції дрона. Також через відсутність навіть базового алгоритму ідентифікації дрон ризикує плутати людей, які в його полі зору, якщо в кадрі людей більше ніж одна [19]. Демонстрація роботи програми зображена на рисунку 1.6.

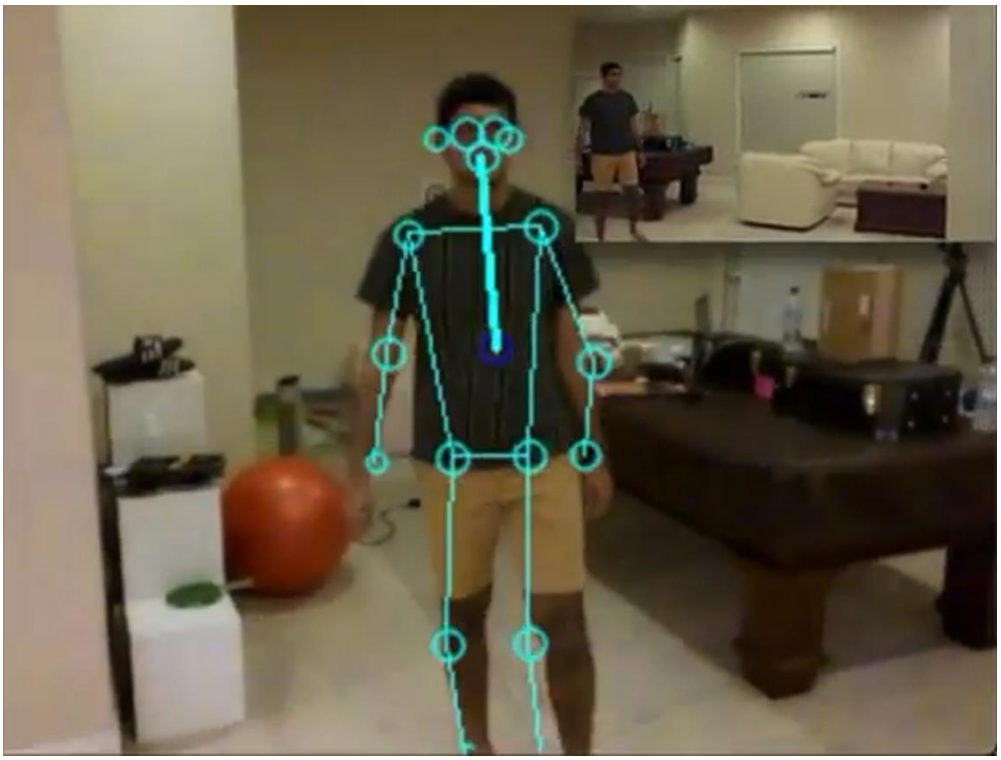

Рисунок 1.6 – Демонстрація роботи програми

Рішення користувача mzarnecki пропонує унікальний підхід до розпізнавання облич – окрім класичних каскадів Хаара проект підтримує розпізнавання облич за допомогою моделі GPT-4.1-mini. При використанні каскадів Хаара, після того як дрон підніметься в повітря він починає обертатися навколо власної осі (на ~36° за ітерацію) допоки не знайде обличчя. Після цього

дрон трохи піднімається вверх, відлітає назад та самостійно приземляється. У випадку використання моделі GPT логіка дій дрона відрізняється. Модель отримує опис кадру від дрона та на основі цього генерує текстові команди, які перетворюються на дії дрона (піднятися, опуститися, обернутися, тощо). Варто зазначити, що модель не аналізує кадри піксель за пікселем, а отримує числові дані, такі як розмір та координати прямокутника, який був намальований навколо обличчя, яке було розпізнане за допомогою каскадів Хаара. Надійність роботи моделі GPT та передбачуваність її дій викликають питання, окрім цього іншого методу відстеження облич в програмі не передбачено [20].

Аналіз існуючих рішень для розпізнавання та відстеження облич за допомогою дрона Ryze Tello показав, що більшість із них використовують прості та застарілі рішення. Каскади Хаара, який відноситься до таких, використовується у 55% відстотках випадків, що підтверджує необхідність подальших досліджень у сфері розпізнавання та відстеження облич та впровадження сучасніших та ефективніших рішень.

1.3 Постановка задачі

Метою даної роботи є розробка програмного модуля автономного розпізнавання та відстеження обличчя людини за допомогою дрона Ryze Tello для забезпечення автоматизованої динамічної відеозйомки рухомого доповідача в режимі реального часу.

Для досягнення поставленої мети були сформовані такі вимоги:

– автономна робота після зльоту без необхідності постійного ручного керування;

– розпізнавання обличчя в реальному часі за допомогою камери дрона;

– безперервне відстеження руху цілі;

– автоматична підтримка висоти польоту;

– низька затримка відеопотоку;

– простий та зрозумілий інтерфейс.

2 АЛГОРИТМІЧНЕ ТА ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ МОДУЛЯ

2.1 Загальна структура програмного модуля

Структура розроблюваної системи складається з кількох основних функціональних модулів, які працюють та взаємодіють між собою в реальному часі. Схема, яка відображає функціональну структуру системи, зображена на рисунку 2.1.

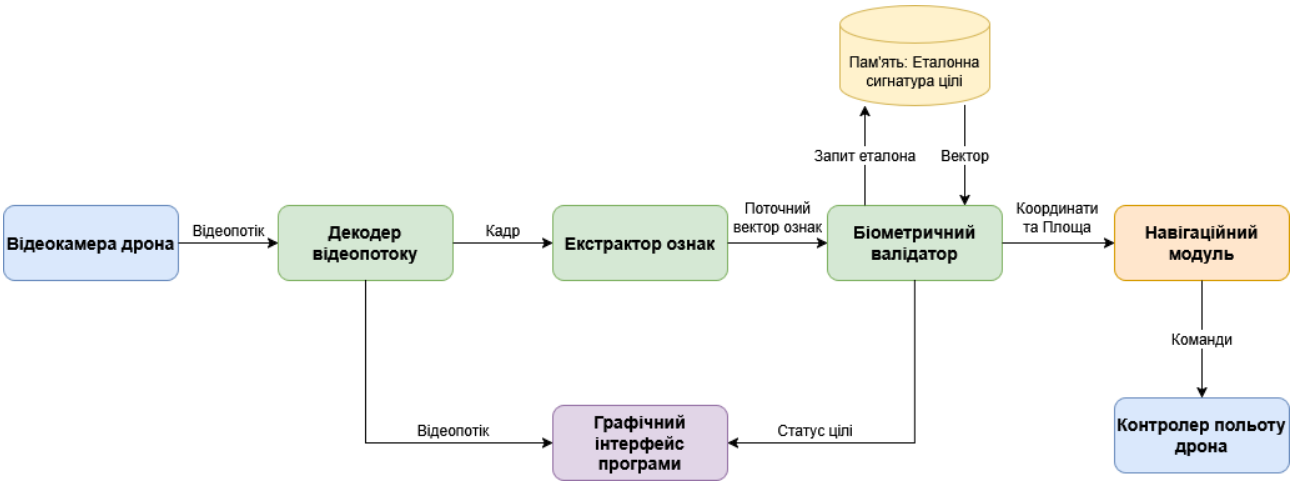

Рисунок 2.1 – Функціональна структура

Процес ініціюється відеокамерою дрона, яка виступає головним джерелом інформації про зовнішнє середовище. Далі згенерований нею відеопотік через мережу Wi-Fi передається до комп’ютера, де його приймає декодер відеопотоку. Цей модуль розпаковує мережеві пакети, конвертує простір кольорів та нормалізує розмір кадру (до роздільної здатності 720p). На виході він формує чисті матриці RGB-пікселів, які паралельно відправляються до екстрактора ознак та виводяться в графічний інтерфейс системи.

Модуль екстрактора ознак обробляє вхідний RGB-кадр, визначаючи чи містить він обличчя. Знайденим обличчям присвоюється набір точок, кожна з яких відображає певну частину обличчя та для кожної з яких обчислюються координати. З цієї інформації формується масив даних, який далі передається до біометричного валідатора. Цей модуль порівнює отримані дані з еталонними, що

зберігаються в пам'яті. Якщо ступінь схожості з еталоном вищий за пороговий, розраховуються координати центру обличчя та його площа в пікселях, які передаються далі в навігаційний модуль.

Навігаційний модуль використовує ці дані для обчислення просторової помилки цілі, тобто ступеня її відхилення від центру кадру. Далі, використовуючи пропорційно-диференціальний закон регулювання, модуль надсилає дрону команди – здійснити поворот, змінити висоту, відстань до цілі, тощо, щоб обличчя завжди залишалося в центрі кадру. Наявність диференціальної складової забезпечує плавні повороти та гальмування дрона, без різких ривків та інерційних перельотів, коли дрон, надто різко повертаючись за обличчям, не може помістити його в центр кадру.

2.2 Алгоритмічне забезпечення

Робота алгоритму розпочинається з етапу глобальної ініціалізації. На цій стадії програма встановлює мережеве з'єднання з дроном через мережу Wi-Fi. Після успішного підключення система надсилає команду на активацію бортової камери та починає приймати відеопотік з неї. Для того, щоб затримки в передачі мережевих пакетів не блокували роботу основного алгоритму, процес зчитування відеопотоку виноситься в окремий фоновий потік виконання. Це дозволяє головному циклу програми завжди миттєво отримувати останній доступний кадр із буфера пам'яті, уникаючи гальмування роботи системи при нестабільному сигналі Wi-Fi.

Після встановлення з’єднання програма починає роботу з перевірки рівня заряду батареї дрона. Якщо він нижчий ніж 10%, програма не дасть дрону злетіти, щоб, коли заряд повністю закінчиться, він не впав у польоті та не пошкодився та не травмував людей. Перевірка рівня заряду батареї відбувається протягом всього часу роботи алгоритму, тому якщо заряд опуститься до критичного рівня під час польоту дрона та відстеження обличчя, програма негайно перерве виконання будь-яких поточних завдань, заблокує можливість

ручного керування та автоматично приземлить дрон. Якщо ж показники в нормі, алгоритм продовжує штатну роботу і переходить до етапу збору візуальних даних.

Після перевірки стану акумулятора програма переходить до обробки відеопотоку. Система витягує з буфера найсвіжіший кадр та проганяє його через процес базової підготовки: воно масштабується до оптимальної роздільної здатності, яка забезпечує баланс між чіткістю деталей обличчя та швидкістю математичних обчислень, а колірна гама конвертується у формат, придатний для аналізу. Після цього підготовлений кадр передається до інтелектуального ядра системи – модуля детекції. Використовуючи нейромережеву модель комп'ютерного зору бібліотеки MediaPipe, алгоритм перевіряє чи присутні на зображенні обличчя. Якщо обличчя присутнє, програма малює рамку навколо нього. Щоб почати відстеження оператору потрібно обрати цільове обличчя, натиснувши на відповідну рамку в інтерфейсі програми. Після цього програма переходить до сканування обличчя.

В основі алгоритму розпізнавання лежить припущення, що співвідношення між певними точками обличчя є індивідуальними для кожної людини та залишаються відносно стабільними незалежно від рухів обличчя та ракурсу зйомки [21]. Для реалізації цієї логіки використовується раніше згадуваний метод екстракції ключових точок обличчя, де кожне обличчя в кадрі представляється у вигляді набору точок [22]:

$$P = p_1, p_2, \dots, p_{468} \tag{2.1}$$

де p – точка обличчя.

Кожна точка має свої координати [22]:

$$p_i = (x_i, y_i, z_i) \tag{2.2}$$

Однією з головних проблем при розпізнаванні об’єктів за допомогою рухомого дрона є постійна зміна відстані між камерою та об’єктом. При наближенні дрона відстань між ключовими точками в пікселях зростає, а при віддаленні – зменшується. Для усунення цієї залежності алгоритм передбачає обов’язкову нормалізацію.

Для цього обирається базовий вектор нормалізації L_{base} , який характеризує загальний масштаб обличчя в конкретному кадрі. Найбільш стабільною рисою для цього є ширина (відстань між скроневими кістками) та висота обличчя.

Формула для обчислення ширини обличчя [23]:

$$W = \sqrt{(x_{right} - x_{left})^2 + (y_{right} - y_{left})^2}\,, \tag{2.3}$$

де x_{right}, x_{left} – позиції точок на горизонтальній осі кадру;

y_{right}, y_{left} – позиції точок на вертикальній осі кадру.

Висота обличчя облислюється за аналогічною формулою [23]:

$$H = \sqrt{(x_{bottom} - x_{top})^2 + (y_{bottom} - y_{top})^2}\,, \tag{2.4}$$

де $x_{\text{bottom}}, x_{top}$ – позиції точок на горизонтальній осі кадру;

y_{bottom}, y_{top} – позиції точок на вертикальній осі кадру.

Алгоритм передбачає, що кожна виміряна відстань D_i між двома довільними точками обличчя p_a та p_b приводиться до нормалізованого виду R_i шляхом ділення на відповідний компонент L_{base} . Якщо вимірюється горизонтальний параметр (наприклад, відстань між очима), нормалізація відбувається відносно ширини W [24]:

$$R_{horizontal} = \frac{\sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}}{W}\,. \tag{2.5}$$

Якщо вимірюється вертикальний параметр (наприклад, відстань від носа до підборіддя), нормалізація відбувається відносно висоти H [24]:

$$R_{vertical} = \frac{\sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}}{H}. \quad (2.6)$$

Такий підхід дозволяє отримати набір безрозмірних коефіцієнтів, які будуть ідентичними як на відстані одного метра, так і на відстані п’яти метрів від обличчя.

Алгоритм оперує п’ятьма ключовими антропометричними параметрами, які разом утворюють вектор ознак обличчя. Вибір саме цих параметрів обґрунтований їхньою низькою залежністю від емоційного стану та високою анатомічною стабільністю:

R1 (Відстань між очима): Відношення відстані між внутрішніми кутами очей до загальної ширини обличчя. Це найбільш стабільний параметр, оскільки положення очей у черепі є фіксованим.

R2 (Ширина рота): Відношення відстані між кутами губ до ширини обличчя. Хоча цей параметр може змінюватися під час розмови, він є важливим.

R3 (Вертикаль «Очі – Ніс»): Відношення вертикальної відстані від лінії очей до кінчика носа до висоти обличчя.

R4 (Вертикаль «Ніс – Рот»): Відношення відстані від кінчика носа до центральної лінії рота до висоти обличчя.

R5 (Вертикаль «Рот – Підборіддя»): Відношення відстані від лінії рота до нижньої точки підборіддя до висоти обличчя.

Таким чином, для кожного обличчя в кадрі формується вектор $V = [R_1, R_2, R_3, R_4, R_5]$. Процес розпізнавання полягає у порівнянні поточного вектора $V_{current}$, який обчислюється в реальному часі, коли дрон бачить обличчя, із еталонним вектором V_{target}, облисленим під час сканування обличчя.

Для визначення ступеня схожості двох облич було відкинуто бінарну логіку ("схожий/несхожий") на користь імовірнісної математичної моделі. Програма розраховує загальний показник відхилення за метрикою Мангеттена, з впровадженням вагових коефіцієнтів w_i для кожного параметра. Необхідність вагових коефіцієнтів обумовлена тим, що пропорції обличчя є по різному надійними для ідентифікації, адже одні риси можуть змінюватись за рухом міміки, тоді як інші залишаться однаковими завжди. Наприклад, відстань між очима стала завжди, а ширина рота постійно змінюється під час розмови.

Формула розрахунку відхилення [25]:

$$S = \sum_{i=1}^{5} w_i \cdot |R_{target,i} - R_{current,i}| . \quad (2.7)$$

Для мінімізації випадкових похибок під час першого захоплення обличчя, алгоритм передбачає “калібрування”. Замість того, щоб зчитувати пропорції обличчя з одного кадру, програма збирає дані протягом N кадрів, поки людина дивиться в камеру. 30 кадрів буде достатньо для уникнення випадкових похибок, а також не надто тривалим, оскільки 30 кадрів це лише 1 секунда часу.

Фінальний еталонний вектор V_{target} розраховується як середнє арифметичне значення за всіма 30 кадрами [26]:

$$V_{target,i} = \frac{1}{N} \sum_{j=1}^{N} V_{j,i} . \quad (2.8)$$

Такий підхід дозволить відфільтрувати мікроколивання точок через шум матриці камери чи випадкові зміни освітлення.

Після того, як цільове обличчя проскановане та його сигнатура збережена в пам'яті програми, починає працювати система відстеження облич.

Головною метою системи відстеження є утримання обличчя в центрі кадру, тобто мінімізація різниці між поточним станом і бажаним, яким і є обличчя в центрі кадру. Цю різницю назвемо помилкою та будемо її

обчислювати для кожної з осей руху дрона: поворот, висота та відстань до об’єкта.

Помилка повороту показує відхилення обличчя від центральної вертикальної осі кадру та обчислюється за такою формулою [27]:

$$e_{yaw} = cx - \frac{W}{2}, \tag{2.9}$$

де cx – центр обличчя по горизонтальній осі;

W – ширина кадру.

Помилка висоти показує відхилення обличчя від центральної горизонтальної осі кадру. Координата Y в зображеннях зазвичай рахується зверху вниз, тому формула така [27]:

$$e_{alt} = \frac{H}{2} - cy\ , \tag{2.10}$$

де H – висота кадру;

cy – центр обличчя по вертикальній осі.

Відстань до об’єкта визначається через зміну площі обличчя A, тому встановимо еталонний діапазон площі $[A_{\min}, A_{\max}]$. Помилка відстані виникає тоді, коли поточне значення A виходить за межі цього діапазону.

Якщо площа обличчя менша за еталон ($A < A_{\min}$), значить воно занадто далеко і дрон отримує команду приблизитись. В такому випадку помилка відстані обчислюється за такою формулою [27]:

$$e_{dist} = A_{\min} - A\ . \tag{2.11}$$

Якщо площа більша за еталон ($A > A_{\max}$) – обличчя занадто близько, дрон отримує команду віддалитись. Помилка обчислюється за наступною формулою [27]:

$$e_{dist} = A_{\max} - A\,. \tag{2.12}$$

Для того, щоб рухи дрона були плавними, що необхідно для відеозйомки, концепція відкидає звичайне керування («рухатись/стояти») на користь пропорційно-інтегрально-диференціального регулятора. Це дозволяє дрону адаптувати швидкість руху залежно від того, наскільки швидко й різко рухається доповідач.

Загальна формула керуючого сигналу U(t) для кожної осі виглядає так [28]:

$$U(t) = K_p \cdot e(t) + K_i \int_0^t e\,(\tau)d\tau + K_d \frac{de(t)}{dt}\,, \tag{2.13}$$

де K_p, K_i, K_d – коефіцієнти;

$e(t)$ – помилка;

$\int_0^t e\,(\tau)d\tau$ – сума всіх помилок від початку відстеження до поточного моменту;

$\frac{de(t)}{dt}$ – швидкість зміни помилки.

Формула складається з трьох частин: пропорційної, інтегральної та диференціальної.

Пропорційна частина $(K_p \cdot e(t))$ визначає наскільки швидко дрон буде рухатися. Якщо коефіцієнт K_p завеликий дрон буде рухатися занадто різко, якщо занизький – занадто повільно. Множачи його на помилку $e(t)$, ми доб'ємся того, що чим дальше обличчя від центру кадру, тим різкіше дрон буде його вирівнювати в центр, і навпаки – чим ближче обличчя до центру, тим плавніше буде центрування. Це дозволить дрону встигати за обличчям доповідача, а також забезпечить плавну зйомку без надмірно різких рухів.

Інтегральна частина відповідає за накопичення помилки з часом, щоб, якщо вона зберігається занадто довго, компенсувати її. Ця частина потрібна для компенсації відхилення під дією зовнішніх сил, таких як вітер. Він може здувати

дрон і через це обличчя не буде в центрі. Тут і втручається інтегральна частина системи. Проте в приміщенні немає вітру, тому K_i встановимо 0.

Диференціальна частина відповідає за гальмування дрона, щоб при центруванні обличчя він не перетинав центр через інерцію та не рухався, як маятник.

Коефіцієнти налаштовуються вручну. Вони визначають наскільки сильно кожна частина формули впливатиме на рух дрона.

Під час зйомки в реальному часі обличчя може на мить перекриватися якимись предметами чи іншими людьми, або людина може просто повернутися вбік і обличчя не буде нормально видно. Щоб подібні ситуації не викликали проблем, програма передбачає систему накопичення чи втрату впевненості. Алгоритм не дозволяє змінювати стан всієї системи (“обличчя відстежується/обличчя не відстежується”) на основі лише одного кадру. Замість цього в програмі передбачений лічильник помилок, який рахує кількість кадрів в яких поріг схожості обличчя з еталоном був перетнутий. Якщо кількість кадрів перевищує встановлену, то дрон переходить у стан пошуку, припиняє рух та очікує на появу обличчя в кадрі. Дрон порівнює ступінь схожості кожного обличчя, яке потрапляє в кадр, з еталоном і якщо, ступінь нижчий, ніж поріг, дрон проводить верифікацію цього обличчя. Для успішної верифікації ступінь схожості має бути нижчим, ніж поріг протягом встановленої кількості кадрів. Лише після цього дрон продовжить відстеження. Використання такої системи дозволить запобігти захопленню стороннього обличчя, коли у кадрі їх багато, наприклад, при виступі доповідача перед аудиторією.

Кінцевим етапом є трансляція результатів роботи. Сформований пакет команд керування відправляється на контролер літального апарата для фізичного виконання. Паралельно з цим алгоритм формує вивід відеопотоку для оператора. На вихідний кадр накладається додаткова інформація: рамки навколо знайдених облич, біометрична сітка, поточний статус системи, показники схожості та рівень заряду акумулятора. Це інформація виводиться в графічний інтерфейс програми, забезпечуючи оператору повний контроль над ситуацією.

2.3 Інформаційне забезпечення

Інформаційне забезпечення програмного модуля являє собою сукупність даних, інформаційних потоків, структур зберігання та механізмів їх обробки, які забезпечують функціонування програмного модуля автономного виявлення та відстеження обличчя за допомогою дрона Ryze Tello. Основним завданням інформаційного забезпечення є організація безперервного обміну даними між компонентами програмного модуля та забезпечення своєчасного прийняття рішень щодо керування безпілотним літальним апаратом.

Основним джерелом інформації виступає відеопотік, що передається з камери дрона Ryze Tello через бездротовий канал зв'язку Wi-Fi. Кожен кадр відеопотоку надходить до програмного модуля у цифровому форматі та обробляється в режимі реального часу. Отримані зображення містять інформацію про навколишнє середовище, положення людей у кадрі та інші візуальні характеристики сцени. Частота надходження кадрів повинна забезпечувати достатню плавність відображення та швидкість реакції програмного модуля на зміни положення цілі.

Після отримання чергового кадру здійснюється його попередня обробка, яка включає перетворення кольорового простору, масштабування та підготовку даних до подальшого аналізу. Підготовлене зображення передається до нейромережевої моделі MediaPipe Face Mesh, яка виконує локалізацію обличчя та визначає координати 468 характерних точок. Кожна точка описується набором координат, що дозволяють визначити просторову структуру обличчя користувача.

На основі отриманих координат формується біометричний опис обличчя. Для цього розраховуються нормалізовані відстані між окремими групами контрольних точок, які характеризують геометричні особливості людини. Сукупність таких параметрів утворює біометричний вектор ознак, який використовується для подальшої ідентифікації цільового користувача. Під час первинного захоплення цілі формується еталонна сигнатура обличчя, яка

зберігається в оперативній пам’яті та використовується протягом усього сеансу роботи програми.

Важливим компонентом інформаційного забезпечення є модуль порівняння біометричних ознак. Для оцінювання подібності між поточним обличчям та еталонною сигнатурою використовується метрика Manhattan Distance. У процесі аналізу кожного кадру система визначає ступінь схожості між наявними обличчями та раніше зафіксованою ціллю. Це дозволяє не лише утримувати ціль у кадрі, але й повторно ідентифікувати її після тимчасового зникнення або перекриття іншими об'єктами.

Окрім біометричних даних, система використовує інформацію про просторове положення цілі відносно центру кадру. Для кожного виявленого обличчя визначаються координати його геометричного центру, ширина та висота області виявлення, а також площа прямокутника, що охоплює обличчя. Отримані параметри використовуються для обчислення помилок позиціонування, які надалі передаються до підсистеми керування дроном.

Інформаційний обмін між модулем комп’ютерного зору та модулем керування здійснюється через внутрішні програмні структури даних. Підсистема керування отримує інформацію про зміщення цілі відносно центру кадру та визначає необхідні коригувальні дії. На основі розрахованих значень PID-регулятор формує команди керування, які передаються безпосередньо на дрон через бібліотеку DJITelloPy. Таким чином забезпечується автоматичне коригування положення безпілотного літального апарата та утримання обличчя користувача в центрі зображення.

Додатково програмний модуль обробляє телеметричні дані дрона. До таких даних належать рівень заряду акумулятора, статус підключення, інформація про виконання команд та поточний стан польоту. Отримані відомості використовуються для контролю працездатності системи та відображаються в графічному інтерфейсі користувача.

Значна частина інформації формується та використовується виключно в оперативній пам’яті комп’ютера без збереження до зовнішніх сховищ даних.

Такий підхід дозволяє мінімізувати затримки під час обробки відеопотоку та забезпечити роботу системи в режимі реального часу. До тимчасових даних належать поточні відеокадри, координати ключових точок, біометричні вектори ознак, результати порівняння сигнатур та службова інформація про стан відстеження.

На рисунку 2.2 наведено схему інформаційних потоків програмного модуля виявлення та відстеження облич із використанням дрона Ryze Tello. Основним джерелом даних виступає відеокамера дрона, яка формує безперервний відеопотік та передає окремі кадри до підсистеми комп'ютерного зору.

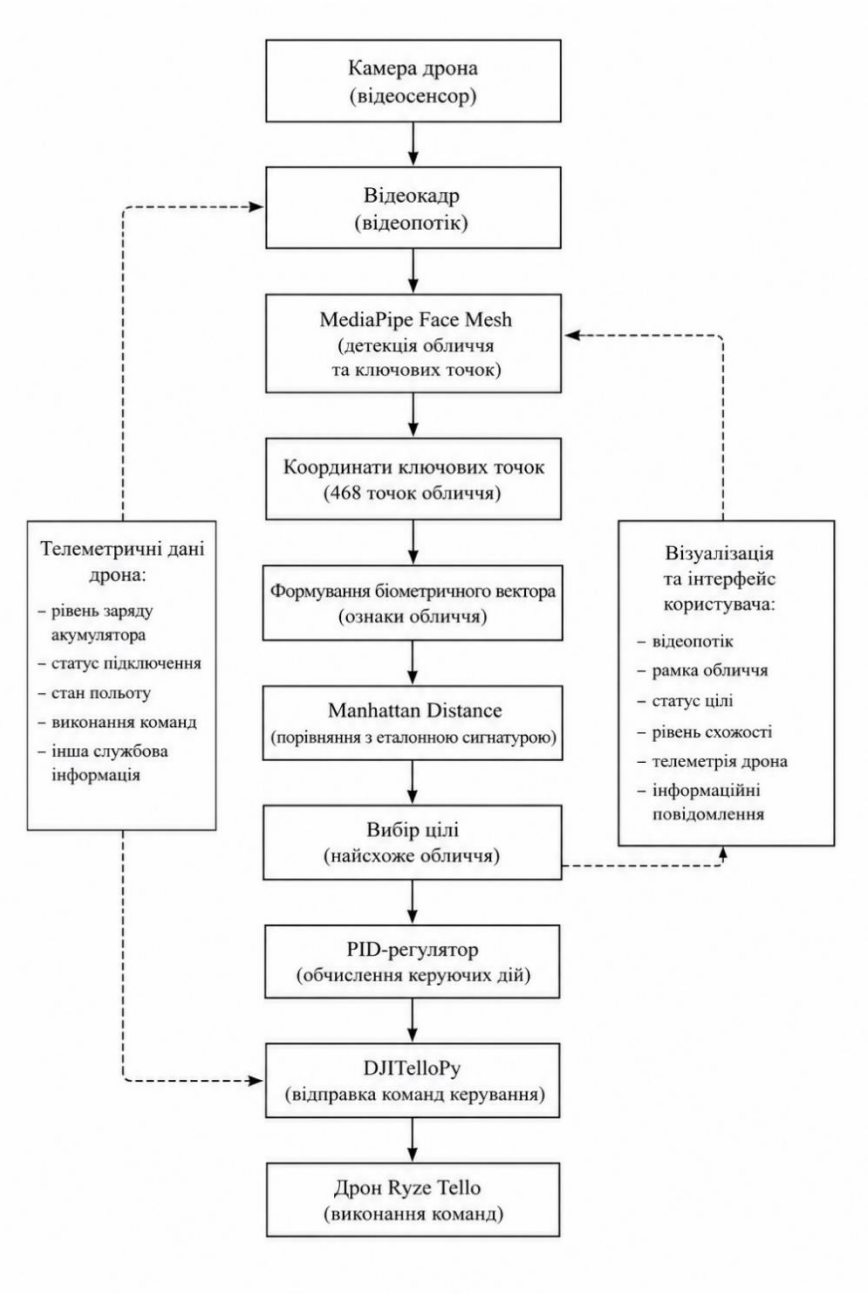

Рисунок 2.2 – Схема інформаційних потоків програмного модуля

Схема, наведена на рисунку відображає послідовність перетворення інформації від моменту отримання відеоданих до формування керуючих команд для безпілотного літального апарата. Отримані кадри обробляються за допомогою бібліотеки MediaPipe Face Mesh, яка виконує детекцію обличчя та визначає координати 468 ключових точок. На основі цих координат формується біометричний вектор ознак, що використовується для ідентифікації користувача. Порівняння поточного набору ознак з еталонною сигнатурою здійснюється за допомогою метрики Manhattan Distance, що дозволяє визначити найбільш схоже обличчя та обрати його як ціль для подальшого супроводу.

Після вибору цілі обчислюються параметри відхилення її положення відносно центру кадру, які передаються до PID-регулятора. Регулятор формує керуючі сигнали для корекції положення дрона, після чого відповідні команди через бібліотеку DJITelloPy надсилаються безпосередньо до безпілотного літального апарата Ryze Tello.

Паралельно із процесом обробки відеопотоку система отримує телеметричні дані дрона, зокрема інформацію про рівень заряду акумулятора, статус підключення, виконання команд та поточний стан польоту. Вся службова інформація разом із відеопотоком, рамкою обличчя, рівнем схожості та повідомленнями про стан системи відображається у графічному інтерфейсі користувача.

Таким чином інформаційне забезпечення програмного модуля формує єдиний цикл збору, обробки, аналізу та передачі даних між усіма компонентами системи. Узгоджена робота інформаційних потоків забезпечує стабільне функціонування алгоритмів комп’ютерного зору, коректну ідентифікацію користувача та ефективне керування дроном Ryze Tello в режимі реального часу.

3 ПРОГРАМНО-ТЕХНОЛОГІЧНЕ ЗАБЕЗПЕЧЕННЯ МОДУЛЯ

3.1 Реалізація програмного забезпечення

Структура програмного модуля базується на багаторівневій архітектурі, що дозволяє відокремити інтерфейс користувача від обчислювальної логіки та низькорівневої взаємодії з апаратною частиною. Система організована у вигляді взаємопов’язаних модулів, кожен з яких відповідає за конкретне завдання. Система складається з елементів описаних нижче.

Модуль графічного інтерфейсу (gui.py): Верхній рівень ієрархії. Його призначенням є ініціалізація візуального середовища за допомогою бібліотеки CustomTkinter, обробка вводу від оператора та відображення результатів роботи системи у реальному часі. Цей модуль керує станами програми та виводить інформацію для оператора.

Модуль обчислювальної логіки та трекінгу (tracker.py): Ядро системи. Він розпізнає обличчя за допомогою бібліотеки комп'ютерного зору MediaPipe, розраховує біометричні сигнатури та команди руху для дрона. Тобто буквально займається трансформацією сирих координат точок обличчя у вектори швидкостей для дрона.

Модуль обробки відеопотоку (videograbber.py): Реалізує рівень доступу до даних. Він працює у окремому потоці виконання, забезпечуючи безперебійне зчитування UDP-пакетів та їх декодування за допомогою OpenCV.

Модуль конфігурації (config.py): Містить глобальні константи, мережеві адреси, коефіцієнти PID та пороги чутливості. Це дозволяє при потребі змінювати поведінку системи в одному місці, без необхідності шукати необхідні змінні в коді.

Зовнішні бібліотеки (SDK): Модуль djitellopy дозволяє взаємодіяти з апаратним рівнем контролера польоту DJI Tello.

Взаємодія між елементами системи відбувається за ієрархічним принципом. Модуль gui.py ініціює створення класів VideoGrabber та FaceTracker, встановлюючи між ними інформаційний канал. Дані передаються у вигляді

матриць зображень від модуля захоплення до модуля інтерфейсу, який передає їх модулю трекінгу для аналізу.

Зворотний зв’язок реалізований через передачу об'єктів стану. tracker.py повертає до gui.py структуровані дані про координати цілі та розраховані швидкості, після чого останній через бібліотеку djitellopy відправляє команди дрону. Всі модулі звертаються до config.py для отримання параметрів функціонування, що забезпечує цілісність налаштувань у всій системі [26, 27].

3.1.1 Діаграма класів

Головним інтеграційним вузлом програмної системи виступає клас MainWindow. Його призначенням є координація роботи всіх інших підсистем та управління глобальним життєвим циклом програми. У межах бізнес-логіки цей клас виконує роль системної пам'яті: він зберігає поточні прапорці станів (активне стеження, очікування, калібрування), а також такі дані, як еталонна біометрична сигнатура користувача, просторові координати захопленого обличчя та масив натиснутих клавіш для можливості ручного керування. Головний метод цього класу запускає безперервний цикл оновлення, який щокадру опитує сенсори дрона, передає дані до аналітичних модулів та фінальні рішення на виконавчі механізми.

Інтелектуальним ядром системи, що відповідає за розпізнавання облич та розрахунок команд керування дроном, є клас FaceTracker. Призначення цього модуля – виконання всіх складних математичних алгоритмів та операцій комп'ютерного зору. Клас містить об'єкт нейромережевого детектора та зберігає критичні налаштування для обчислення команд керування дроном: коефіцієнти просторових помилок (по осях повороту, нахилу та висоти), лічильники втрати цілі та пороги біометричної чутливості. Методи цього класу приймають оброблене зображення, проводять пошук облич, розраховують нормалізовані вектори ознак, здійснюють імовірнісне порівняння виявлених облич з еталоном та генерують готові значення для пропорційно-диференціального регулятора.

Забезпеченням системи безперервним потоком вхідної візуальної інформації займається клас VideoGrabber. Цей модуль було спроектовано з урахуванням вимог багатопотоковості. Його головне призначення – відеокремлення мережевих операцій зчитування UDP-пакетів від основного обчислювального процесу. Клас створює окремий фоновий потік виконання, який безперервно приймає, декодує та буферизує відеокадри від камери дрона. Така архітектура дає можливість процесам математичного аналізу завжди працювати з найактуальнішим кадром, а можливі мережеві затримки не призведуть до гальмування чи блокування всієї програми. UML-діаграма класів зображена на рисунку 3.1.

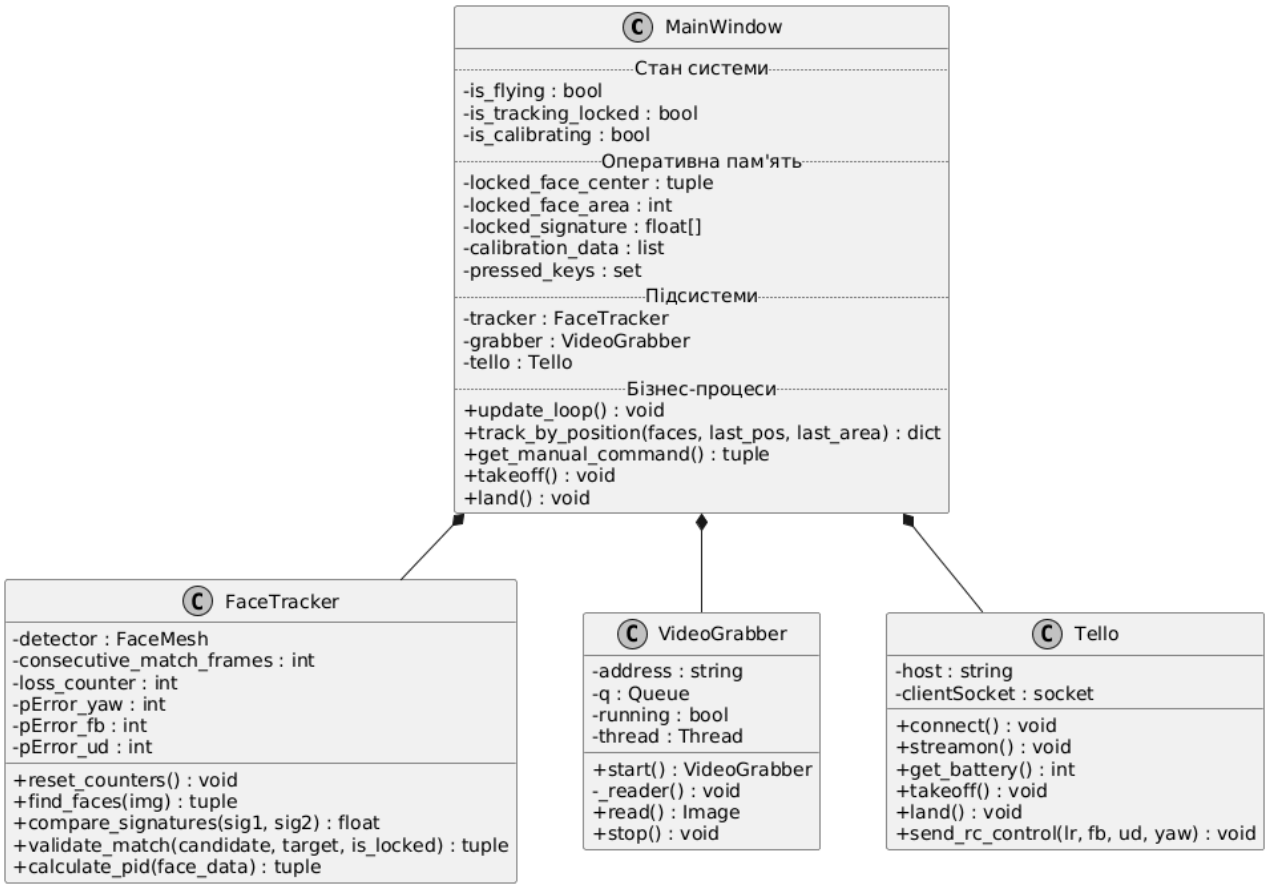

Рисунок 3.1 – UML-діаграма класів

Взаємодія з апаратною частиною безпілотника реалізована через клас Tello. Призначенням цього модуля є приховування низькорівневих специфікацій мережевого обміну та протоколів дрона. Клас відповідає за відкриття

клієнтських сокетів, ініціалізацію відеопотоку та отримання телеметричних даних (наприклад, заряду акумулятора). Його найважливіша функція полягає у трансляції розрахованих математичним ядром швидкостей у стандартизовані команди керування по чотирьох осях та відправці їх безпосередньо на контролер польоту дрона.

Між класом MainWindow та трьома іншими модулями встановлено зв'язок типу «композиція». Це означає, що головна програма повністю володіє цими об'єктами і керує їхнім життєвим циклом. Такий структурний підхід робить архітектуру максимально гнучкою: у випадку необхідності переходу на іншу модель літального апарата або використання іншої нейромережі для детекції облич, достатньо буде переписати лише один відповідний клас, залишаючи всю іншу логіку автономного супроводу абсолютно незмінною.

3.1.2 Діаграма станів

Життєвий цикл графічного інтерфейсу розпочинається з відкриття програми на комп’ютері оператора. Відразу після запуску система переходить у початковий стан «Ініціалізація системи». У цьому стані графічна оболонка вже відмальована операційною системою та відображається на екрані комп’ютера, проте будь-які інтерактивні взаємодії з користувачем поки недоступні. Головним завданням системи на цьому етапі є підготовка комунікаційного середовища підсистеми комп’ютерного зору. Програма у фоновому режимі очікує на встановлення UDP-з'єднання з мережевим контролером дрона та надходження першого декодованого та обробленого кадру з відеопотоку. Блокування елементів керування на цьому етапі було впроваджено, щоб запобігти надсиланню команд польоту буквально в пустоту, оскільки дрон поки не готовий їх приймати, що призвело б до переповнення мережевих буферів або аварії на етапі зльоту. Щойно внутрішній диспетчер потоків отримує сигнал про успішне з’єднання з дроном та фіксує наявність стабільного, неперервного відеосигналу, система генерує автоматичний внутрішній тригер, який переводить інтерфейс у перший інтерактивний стан – «Очікування».

Перебування системи у стані «Очікування» означає повну готовність дрона та алгоритмів до виконання цільових завдань супроводу. Оператор у графічному інтерфейсі програми отримує доступ до чистого відеопотоку з камери дрона в режимі реального часу, на який накладається лише діагностична графіка. Текстовий статусний рядок під областю відеопотоку інформує користувача повідомленням про те, що система готова до роботи. Незважаючи на пасивність цього стану, внутрішні алгоритми комп'ютерного зору вже працюють, аналізуючи кожен вхідний кадр з камери дрона. Усі обличчя, які нейромережевий детектор бачить на відеопотоці з камери, виділяються зеленими графічними маркерами, такими як рамки навколо виявлених облич або сітки з точок, які накладаються на головні точки на обличчі. Графічна оболонка може перебувати в цьому стані необмежено довго, не виконуючи жодних рухів апаратом, очікуючи на команду від оператора, яка ініціює наступний логічний етап роботи системи.

Перехід від режиму пасивного спостереження до активних дій відбувається виключно за командою оператора. Тригером для зміни стану є натискання лівої кнопки миші в межах області одного з розпізнаних облич на області з відеопотоком. Ця дія переводить систему у стан «Калібрування» – алгоритм обчислює сигнатуру обраного обличчя, як було описано в попередньому розділі. Коли система перебуває в цьому стані, інтерфейс припиняє обмальовувати всі обличчя в кадрі, окрім обраного, виводячи натомість інформацію про стан калібрування. Текстовий індикатор статусу змінюється на повідомлення про процес збору даних, яке супроводжується лічильником оброблених кадрів. Оскільки процес збору кадрів займає певний час, виділення цієї операції в окремий стан інтерфейсу дозволяє оператору розуміти, що програма робить в даний час, тобто виконує складне багатоетапне моделювання обличчя. У цьому ж стані активується перша лінія ручного скасування: якщо оператор усвідомив, що обрав хибну особу через випадковий клік, він має змогу натиснути праву кнопку миші, щоб примусово перервати процес, що негайно поверне систему назад у стан очікування, повністю

очистивши частково заповнені масиви оперативної пам'яті. UML-діаграма станів зображена на рисунку 3.2.

У випадку, якщо процес збору біометричних даних завершується успішно та без переривань, алгоритм автоматично формує внутрішню подію створення еталона, яка є тригером для переходу графічного інтерфейсу в його головний робочий стан – «Активне стеження». На цьому етапі рамка або сітка захопленої цілі змінюється із зеленого кольору на червоний, чітко відокремлюючи обране обличчя від будь-яких інших, що можуть випадково перетинати поле зору об’єктива. Найважливішою функціональною особливістю цього стану є виведення поверх відеокадру параметра метричного відхилення – математичного показника просторової різниці між поточним вектором ознак та еталонним, просканованим та обчисленим раніше.

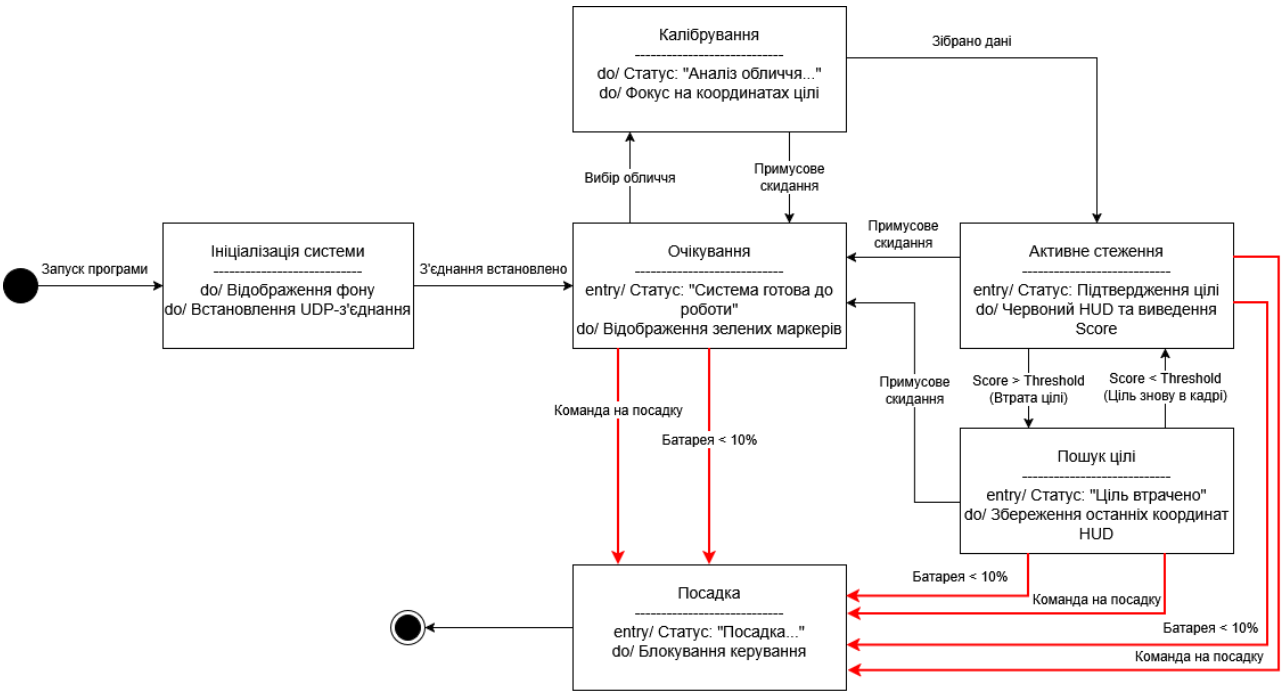

Рисунок 3.2 – UML-діаграма станів

Таким чином оператор здатний в режимі реального часу візуально контролювати рівень впевненості алгоритму: що нижче значення похибки відображається на екрані, то точніше та надійніше система утримує ціль.

Одночасно інтерфейс підтверджує факт початку безперервної трансляції розрахованих навігаційних команд до дрона, сигналізуючи оператору про те, що дрон наразі перебуває під повним автономним керуванням пропорційно-диференціального регулятора. Аналогічно до попереднього режиму, архітектура зберігає за оператором право втрутитись у керування: натискання правої кнопки миші в будь-яку мить польоту призведе до примусового розриву прив'язки і скидання інтерфейсу в базовий режим очікування з одночасною зупинкою дрона.

3.1.3 Обґрунтування вибору програмних засобів реалізації

У якості мови програмування було обрано Python, що зумовлено його популярністю у сфері розробки систем штучного інтелекту та аналізу даних. Основна перевага Python полягає у наявності величезної кількості оптимізованих бібліотек, які дозволяють виконувати складні математичні операції над відеопотоком у реальному часі, попри інтерпретовану природу мови. Високорівневий синтаксис Python забезпечує швидку ітерацію розробки, що дозволило оперативно впроваджувати та тестувати зміни в алгоритмі біометричної валідації. Також Python дозволяє легко реалізувати багатопотоковість, яка є критично важливою для даного дослідження: один потік відповідає за отримання та декодування відео, другий – за логіку розпізнавання обличчя, а третій – за безпосередню комунікацію з дроном. Така архітектурна гнучкість у поєднанні з підтримкою більшості сучасних фреймворків комп’ютерного зору робить Python чудовим вибором для створення надійного програмного забезпечення, здатного працювати в умовах високого навантаження на центральний процесор при обробці зображень високої роздільної здатності.

Для ідентифікації та трекінгу обличчя було обрано передову модель MediaPipe Face Mesh та бібліотеку cvzone. MediaPipe від компанії Google базується на машинному навчанні та дозволяє реконструювати 468 ключових точок обличчя в тривимірному просторі. На відміну від застарілих методів, таких як каскади Хаара, Face Mesh використовує одну нейронну мережу для всього процесу, що забезпечує високу швидкість роботи навіть на мобільних

процесорах без використання дискретних відеокарт. Це дозволило створювати детальні моделі облич, де кожна точка має свої координати, що стало базою для розрахунку вектора біометричних ознак [7]. Бібліотека cvzone, яка є високорівневою надбудовою для MediaPipe, дозволила значно скоротити обсяг коду та спростити розробку, автоматизуючи процеси ініціалізації детектора та маніпуляцій із координатами [29]. Використання цих двох інструментів дозволило системі не просто знаходити обличчя, а розрізняти риси, що критично важливо для методу нормалізації відстаней, який дозволяє зберігати точність ідентифікації незалежно від того, наскільки далеко дрон знаходиться від цілі чи наскільки швидко він рухається [22].

Обробка відеопотоку та організація візуального фреймворку реалізовані за допомогою OpenCV, яка є найбільш потужною та розповсюдженою бібліотекою для задач комп'ютерного зору. OpenCV забезпечує повний цикл роботи з зображеннями, від захоплення Raw-потоку з UDP-порту до виконання складних перетворень, таких як колірна конвертація, фільтрація шумів та накладання графічних інтерфейсів безпосередньо на відеоряд. У цьому дослідженні OpenCV виконує роль головного графічного двигуна, який відповідає за підготовку кадрів до аналізу нейромережею та фінальну візуалізацію результатів для оператора. Завдяки оптимізованим алгоритмам, що лежать в основі Python-обгортки OpenCV, процес обробки кожного кадру займає лічені мілісекунди, що дозволяє підтримувати частоту кадрів на рівні, достатньому для коректної роботи PID-регулятора [30].

Комунікація з дроном здійснюється через бібліотеку djitellopy, яка є найстабільнішим програмним інтерфейсом для взаємодії з дронами Tello на мові Python. Ця бібліотека дозволяє взаємодіяти з дроном через протокол UDP, автоматизуючи процеси відправки команд керування польотом та отримання телеметричних даних. Використання djitellopy дозволило зосередитися на розробці інтелектуальних алгоритмів, не відволікаючись на низькорівневе управління сокетами чи обробку втрачених пакетів даних. Бібліотека підтримує передачу відео у форматі H.264, надаючи зручний інтерфейс для доступу до

кожного кадру через OpenCV. Також djitellopy може динамічно зчитувати стан батареї, температуру процесора та висоту польоту, що дозволило реалізувати систему безпеки, яка автоматично ініціює посадку при критичному зниженні заряду акумулятора [31].

Математична обробка даних та побудова користувацького інтерфейсу базуються на використанні NumPy та Tkinter. Всі операції над векторами координат обличчя виконуються через масиви NumPy. Це дає значно вищу швидкість обчислень порівняно із стандартними циклами Python [32]. Для створення графічної оболонки було обрано CustomTkinter, який, будучи частиною Tkinter, стандартної бібліотеки Python, забезпечує максимальну сумісність та стабільність. CustomTkinter дозволив створити мінімалістичний та приємний на вигляд інтерфейс, в якому користувачу буде легко розібратися [33].

3.2 Інтерфейс користувача

Графічний інтерфейс користувача виступає головним комунікаційним мостом між оператором та автономною обчислювальною підсистемою комплексу (рисунок 3.3).

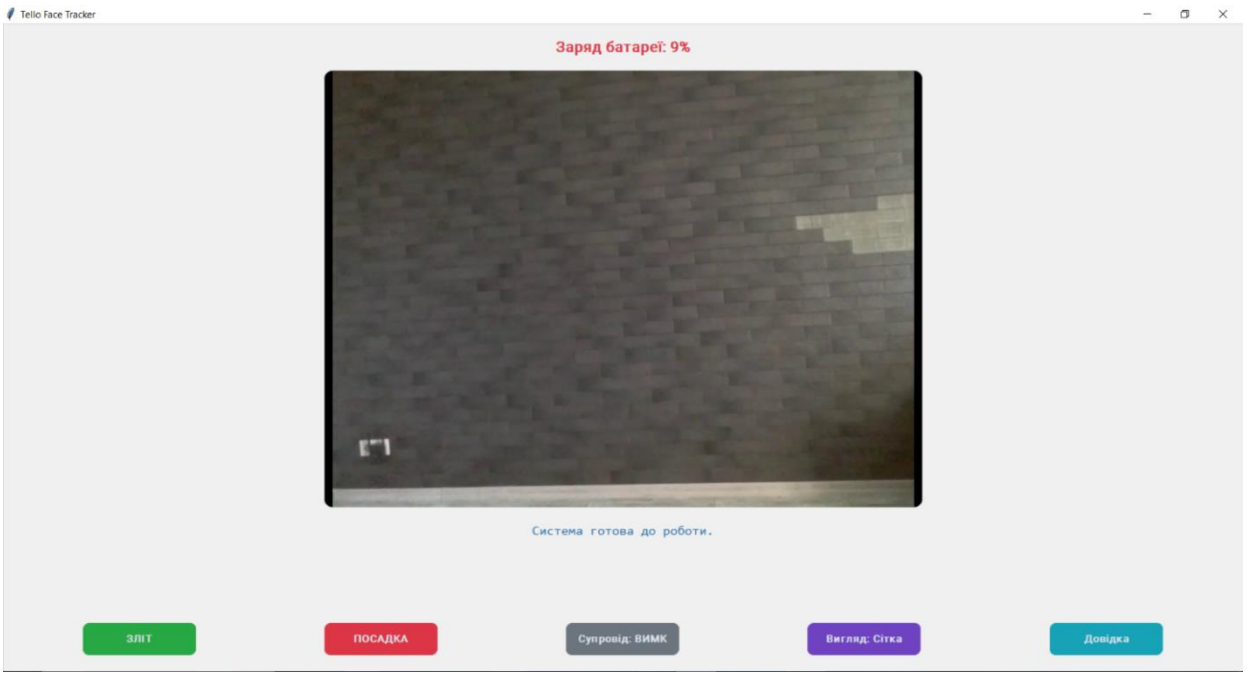

Рисунок 3.3 – Інтерфейс програми

Для реалізації візуальної оболонки було обрано бібліотеку CustomTkinter, яка є надбудовою над стандартним модулем tkinter мови Python. Вибір цієї бібліотеки дозволив відмовитися від застарілих методів відображення на користь сучасного дизайну з апаратним прискоренням [26, 27].

Головне вікно програми поділене на кілька логічних зон. З самого верху знаходиться індикатор рівня заряду батареї дрона. Для того, щоб було зручно переглядати інформацію цей індикатор динамічно змінює колір залежно від рівня заряду. Програма у фоновому режимі опитує контролер польоту кожні п’ять секунд. Якщо заряд батареї перевищує п'ятдесят відсотків, текст відображається зеленим кольором; при падінні рівня заряду нижче двадцяти відсотків колір автоматично змінюється на жовтий, а при досягненні критичної позначки – на червоний. Це дозволяє оператору зручно контролювати рівень заряду батареї дрона.

Центральну і найбільшу частину екрана займає область відеомонітора. Оскільки матриці комп'ютерного зору формату OpenCV не підтримуються графічними оболонками напряму, програма використовує проміжну конвертацію масивів пікселів в об'єкти CTkImage з частотою тридцять кадрів за секунду. Безпосередньо на відеоряд накладається векторна графіка, яка візуалізує роботу прихованих математичних алгоритмів. У базовому режимі всі виявлені системою обличчя обмальовуються зеленими рамками або деталізованою сіткою ключових точок, залежно від вибору оператора. Окрім візуалізації, поверхня відеомонітора є головним інтерактивним елементом, з яким можна взаємодіяти через мишку. Натискання лівої кнопки миші в межах цієї області запускає передачу координат кліку до математичного ядра, який за допомогою них розуміє, яке обличчя було вибране, якщо у кадрі їх декілька. Натискання правої кнопки припиняє стеження та переводить систему в стан очікування.

Після того, як було вибрано обличчя для відстеження, сітка або рамка навколо нього стає червоною, що наочно підтверджує факт фокусування дрона на ньому. Над обмежувальною рамкою з'являється текстовий маркер «LOCKED»

та числовий індикатор Score, який з точністю до тисячних відображає поточну похибку біометричної валідації. Завдяки цій індикації оператор може візуально оцінювати, наскільки алгоритму «складно» утримувати ціль при поточному освітленні або ракурсі. Для забезпечення максимальної видимості цих даних на тлі різнокольорових офісних стін або вікон, програмний код використовує жирні шрифти з високою контрастністю.

Інтерфейс програми при відстеженні обличчя зображений на рисунку 3.4.

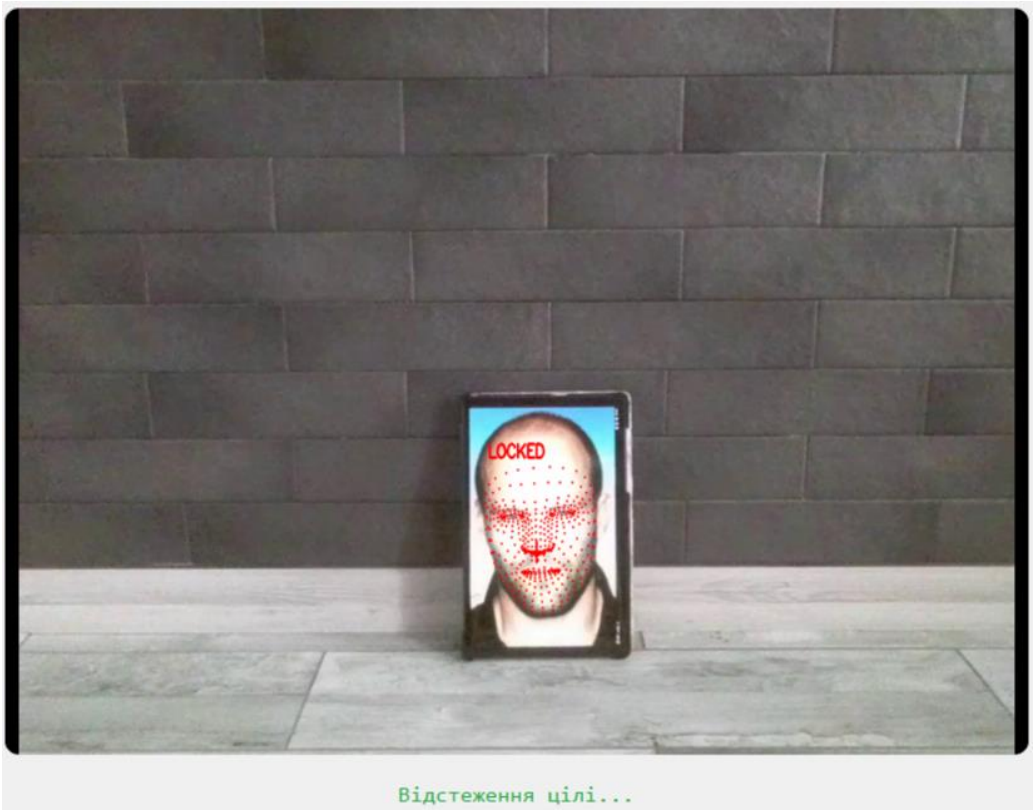

Рисунок 3.4 – Відстеження обличчя

Під областю відеомонітора розташована зона системних статусів, яка містить динамічний рядок стану, який описує поточний режим роботи системи. Як і у випадку з індикатором рівня заряду батареї дрона, цей рядок також динамічно змінює колір: синій колір символізує процес ініціалізації або збору калібрувальних даних, який також відображає лічильник кадрів, зелений підтверджує успішний зліт або впевнене стеження, а червоний сигналізує про втрату цілі чи екстрену посадку.

Нижній ярус графічного інтерфейсу займають кнопки для керування та налаштувань. Зліва розташовані дві головні команди життєвого циклу польоту: зелена кнопка «Зліт» та червона кнопка «Посадка». Рознесення цих функцій на окремі кнопки керування мінімізує ризик випадкової помилки оператора, яка могла б призвести до падіння дрона. Правіше розміщені кнопки-перемикачі, які дозволяють керувати поведінкою алгоритмів. Кнопка активації супроводу дозволяє вмикати або вимикати розрахунок похибки наближення, а кнопка зміни вигляду дозволяє оператору перемикатися між відображенням біометричної сітки та прямокутною рамкою. Замикає панель кнопка «Довідка» , яка викликає спливаюче вікно з додатковою інформацією про ручне керування дроном.

Важливою, хоча й візуально прихованою складовою інтерфейсу є система перехоплення подій клавіатури. Графічна оболонка безперервно відстежує натискання клавіш керування рухом (WASD для горизонтальної площини, Shift/Ctrl для висоти та Q/E для обертання). При натисненні на одну із них, відбувається негайна натиснення конвертація у внутрішній вектор ручного керування, який домішується до автоматичних команд PID-регулятора. Особливий пріоритет відведено клавіші пробілу, натискання якої відправляє команду на екстрену посадку незалежно від поточного стану відеопотоку чи алгоритмів розпізнавання. Завдяки реалізації багатопотоковості для мережевих завдань та використанню оптимізованого циклу оновлення window.after, графічний інтерфейс залишається здатним негайно реагувати на команди оператора, клавіатурного вводу та натискань кнопок не перериваючи процес розрахунку алгоритмів розпізнавання та відстеження.

3.3 Тестування розробленого модуля

Для перевірки працездатності модуля було проведено кілька тестів. Тестування здійснювалося в закритому приміщенні зі штучним та денним світлом.

Спершу було протестовано точність виявлення облич з використанням нейромережевої моделі комп'ютерного зору MediaPipe Face Mesh. Процедура тестування передбачала розташування людини перед камерою дрона на різних дистанціях від 1 до 4 метрів, а також при різних рівнях освітленості, від денного в обідній час, до штучного світла від люстри в темну пору доби. Результати тестування показали високу якість модуля детекції. Алгоритм успішно розпізнавав обличчя незалежно від типу освітлення. Проте, при віддаленні цілі на відстань більше, ніж приблизно 3 метри, дрон починав втрачати її, якщо ціль відверталася або перекривала більшу частину обличчя, в той час як при ближчій відстані система успішно розпізнавала обличчя повторно. Це може бути пов'язано із нездатністю Tello видавати якісь відео вищу, ніж, посередню за сучасними мірками, HD (1280x720 пікселів).

Далі було протестовано наскільки добре дрон відстежував зафіксоване обличчя та утримував його в центрі кадру. Для цього було обрано обличчя, яке дрон просканував та відкалібрувався, після чого ціль починала рухатися по кімнаті з різною швидкістю. Результати показали, що при русі цілі дрон надійно утримував обличчя в межах центральної зони екрана. Завдяки правильно підібраному диференціальному коефіцієнту, система успішно гасила інерцію: коли людина різко зупинялася, дрон також плавно гальмував без ефекту маятника, при якому дрон гойдався би з боку в бік, щоразу повертаючись занадто різко в неможливості відцентрувати обличчя. Увімкнення функції контролю дистанції довело здатність дрона автоматично відлітати назад при наближенні людини та наближатися при її віддаленні.

Також було перевірено те, чи плутає дрон зафіксоване обличчя з іншим в кадрі. Після фіксації дрона на відстеженні конкретного обличчя в кадрі з'являлося інше. Спочатку вони просто були на фоні, а потім переміщалися перед зафіксованим обличчям. Результати підтвердили високу ефективність розробленого алгоритму порівняння векторів ознак облич. Алгоритм розраховував показник похибки для всіх виявлених облич. Під час візуального перекриття, коли стороння людина проходила безпосередньо перед доповідачем,

система фіксувала різкий стрибок похибки і просто ігнорувала стороннє обличчя, фіксуючись на оригінальній цілі як тільки вона знову з'являлася в полі зору. Завдяки порівнянню кожного обличчя з еталоном, система майже в 100% випадків могла відрізнити потрібне обличчя від решти. Система помилилася лише один раз. Поведінка програми при перекриванні зафіксованого обличчя іншим зображена на рисунках 3.5 та 3.6.

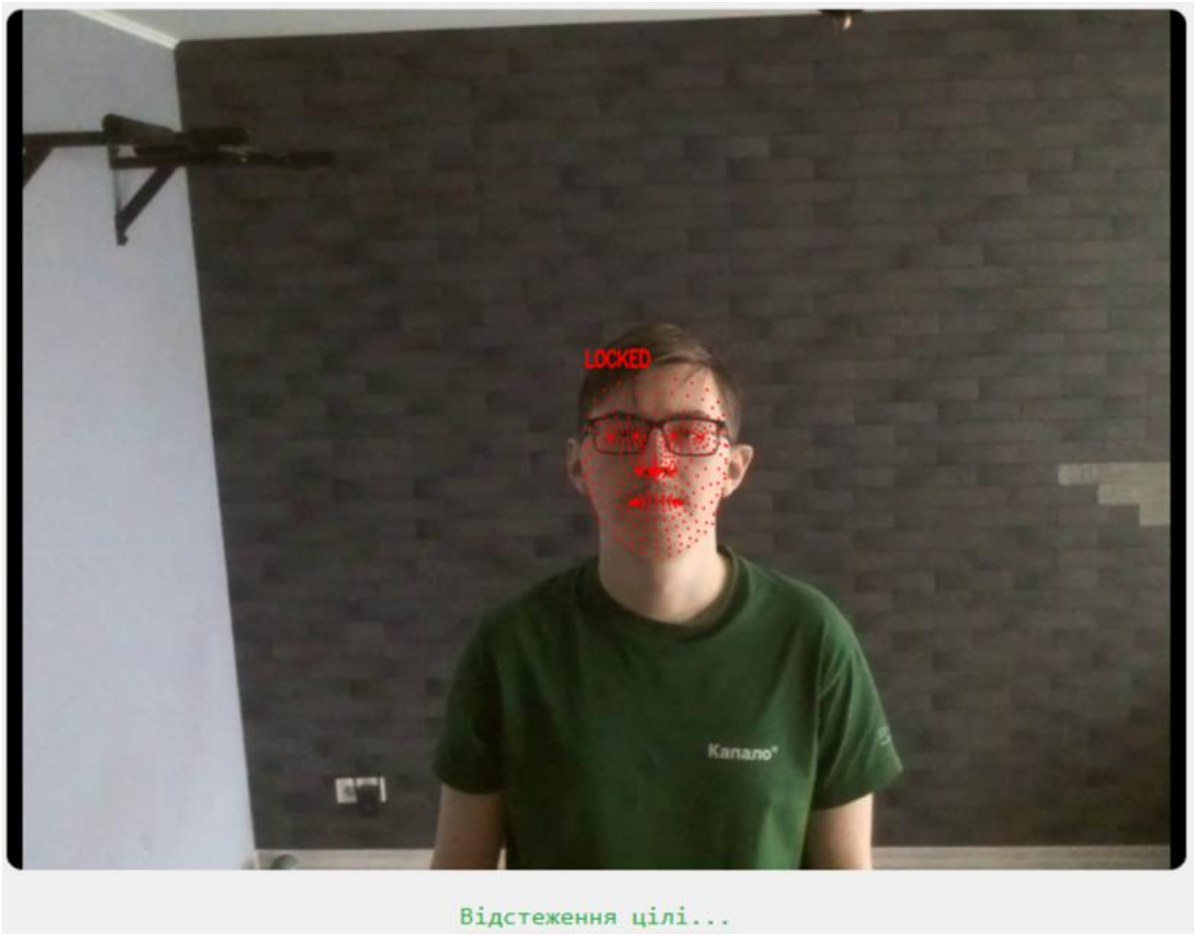

Рисунок 3.5 – Відстеження обличчя

В кінці було протестовано чи втрачала система ціль при її різкому русі та як поводилася після втрати. Ціль різко відходила за межі кута огляду камери і поверталася спиною до дрона. Як і очікувалося, при зникненні обличчя з поля зору камери дрона система негайно зупиняла дрон, обнуляючи всі вектори швидкостей та надсилаючи контролеру польоту команду на зависання. Дрон залишався нерухомим у точці останнього візуального контакту. При поверненні цілі в кадр, система повторно порівнювала її обличчя з еталоном та, упевнившись

що це те саме обличчя, автоматично продовжувала стеження. Без жодного додаткового втручання оператора чи повторного калібрування.

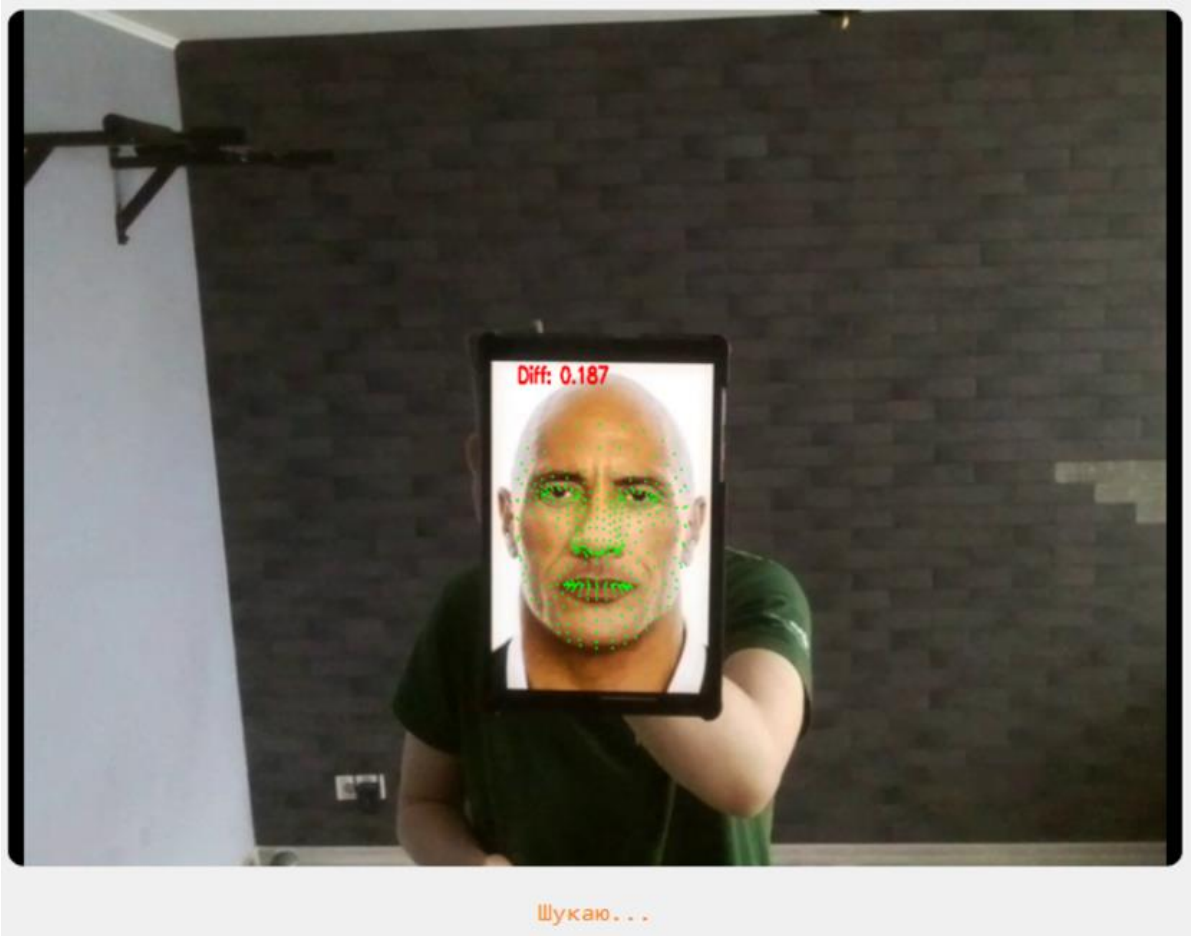

Рисунок 3.6 – Перекривання зафіксованого обличчя іншим

Результати тестування дозволяють стверджувати, що розроблене програмне забезпечення повністю виконує покладені на нього функції. Модуль біометричної ідентифікації працює стабільно та здатен точно відрізняти ціль у натовпі. Автоматичне керування дроном на базі PID-регулятора забезпечує плавний супровід без різких ривків, з планом дій для дрона у випадку непередбачуваної втрати візуального контакту з ціллю.

ВИСНОВКИ

1. Розроблено програмний модуль для дрона Ryze Tello, який реалізує автономне відстеження рухомого доповідача. Розробка включає алгоритм виявлення та розпізнавання обличчя та відстеження руху людини в реальному часі. Впровадження такої системи дозволить уникнути людського фактора помилок та мінімізує витрати на обладнання та персонал.

2. Система працює в повністю автономному режимі, оператор має лише вибрати об’єкт відстеження. Після зльоту дрон за допомогою вбудованої камери здійснює постійний аналіз відеопотоку, виявляє обличчя та обчислює відхилення відносно центру кадру. На основі цих даних PID-регулятор формує команди для руху дрона, забезпечуючи плавне стеження за об’єктом. Дрон автоматично коригує висоту та відстань до цілі для збереження оптимального масштабу та композиції. У разі втрати об’єкта система переходить у режим пошуку, а після виявлення відновлює супровід.

3. У розробці застосовано сучасні бібліотеки комп’ютерного зору – MediaPipe Face Mesh та OpenCV для обробки зображень у реальному часі. Керування дроном здійснюється через бібліотеку djitellopy. Алгоритм відстеження реалізований з використанням PID-регулятора. Система здатна працювати на звичайному ноутбуці.

4. Розробка має низку переваг порівняно з існуючими рішеннями. По-перше, вона повністю автономна і не потребує постійного втручання оператора. По-друге, завдяки компактності Ryze Tello забезпечує високу мобільність у закритих приміщеннях. По-третє, вартість рішення рази нижча за комерційні PTZ-системи чи професійні дрони. По-четверте, система легко розгортається та не вимагає спеціальної інфраструктури.

5. Розроблена система відкриває широкі перспективи для подальшого вдосконалення. Можлива інтеграція з хмарними сервісами для автоматичного завантаження та обробки відео, додавання розпізнавання кількох доповідачів з автоматичним перемиканням між ними, а також реалізація функції голосового

керування. Перспективним напрямком є розробка мобільного додатка, який дозволить запускати систему одним дотиком. Довгостроково система може бути масштабована для більших приміщень та інтегрована з іншими дронами для багатоточкової зйомки. Загалом, проведене дослідження заклало надійну основу для створення комерційно привабливого продукту, який суттєво підвищить ефективність гібридних корпоративних заходів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The Hybrid Learning Model: Adopt It for Corporate Training? URL: https://elmlearning.com/blog/hybrid-learning-model-for-corporate-training/ (дата звернення: 18.03.2026).
2. Autonomous Camera Tracking System Using Image Processing for Dynamic Educational Content Creation. URL: https://www.researchgate.net/publication/391174387_Autonomous_Camera_Tracking_System_Using_Image_Processing_for_Dynamic_Educational_Content_Creation (дата звернення: 24.03.2026).
3. Livestreaming vs Pre-Recorded: How Social Viewing Strategies Impact Consumers' Viewing Experiences and Behavioral Intentions. URL: https://www.researchgate.net/publication/327813328_Livestreaming_vs_pre-recorded_How_social_viewing_strategies_impact_consumers%27_viewing_experiences_and_behavioral_intentions (дата звернення: 06.04.2026).
4. The Art of Drone Videography in Corporate Videos. URL: https://www.vinc.ca/post/the-art-of-drone-videography-in-corporate-videos (дата звернення: 29.03.2026).
5. What Are the Challenges of Hand-Held Shooting with a Professional Camera. URL: https://mackmediagroup.com/what-are-the-challenges-of-hand-held-shooting-with-a-professional-camera/ (дата звернення: 11.04.2026).
6. A New Guide to PTZ Camera: Why It's All Changed. URL: https://www.mylumens.com/en/Blog_detail/162/A-New-Guide-to-PTZ-Camera-Why-Its-All-Changed (дата звернення: 27.03.2026).
7. Lugaresi, C., et al. (2019). MediaPipe: A Framework for Building Perception Pipelines. Google AI Research.
8. Ang, K. H., Chong, G., & Li, Y. (2005). PID control system analysis, design, and technology. IEEE Transactions on Control Systems Technology.
9. 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.

10. Ковтуненко О. О., Биковий П. Є. Програмний модуль виявлення та відслідковування облич із використанням дрона Ryze Tello // Intelligent Computing Systems and Project Management (ICSPM 2026) : матеріали I Міжнародної науково-практичної конференції студентів та молодих учених. Тернопіль : Західноукраїнський національний університет, 2026. С. 27–29.

11. tello-face-tracker. URL: https://github.com/BobMcDear/tello-face-tracker (дата звернення: 14.03.2026).

12. Tello-Face-Recognition. URL: https://github.com/juanmapf97/Tello-Face-Recognition (дата звернення: 03.04.2026).

13. tello-sandbox. URL: https://github.com/youngsoul/tello-sandbox (дата звернення: 20.03.2026).

14. Real-Time Human Motion Tracking by Tello EDU Drone. URL: https://www.researchgate.net/publication/367093080_Real-Time_Human_Motion_Tracking_by_Tello_EDU_Drone (дата звернення: 08.04.2026).

15. tello-ai-features. URL: https://github.com/carlo98/tello-ai-features (дата звернення: 15.03.2026).

16. Autonomous-Drone-based-on-Facial-Recognition-and-Tracking. URL: https://github.com/Akbonline/Autonomous-Drone-based-on-Facial-Recognition-and-Tracking (дата звернення: 30.03.2026).

17. Face Detection over Webcam or Tello Drone Video Stream (Python + OpenCV). URL: https://github.com/Jacob-Pitsenberger/Face-Detection-over-Webcam-or-Tello-Drone-Video-Stream-Python-OpenCV (дата звернення: 04.04.2026).

18. alpha_drone. URL: https://github.com/shijq23/alpha_drone (дата звернення: 22.03.2026).

19. Person-Tracking-Tello-Drone. URL: https://github.com/Matthewjsiv/Person-Tracking-Tello-Drone (дата звернення: 09.04.2026).

20. flying-drone-object-detection. URL: https://github.com/mzarnecki/flying-drone-object-detection (дата звернення: 01.04.2026).

21. Facial Anthropometric Measurements and Photographs — An Interdisciplinary Study. URL: https://www.researchgate.net/publication/347155670_Facial_Anthropometric_Measurements_and_Photographs_-_An_Interdisciplinary_Study (дата звернення: 26.03.2026).

22. Real-Time Facial Surface Geometry from Monocular Video on Mobile GPUs. URL: https://arxiv.org/abs/1907.06724 (дата звернення: 17.03.2026).

23. Euclidean distance. URL: https://www.britannica.com/science/Euclidean-distance (дата звернення: 12.04.2026).

24. OpenSfM Geometry Models. URL: https://opensfm.readthedocs.io/en/latest/geometry.html (дата звернення: 31.03.2026).

25. Generating Job Recommendations Based on User Personality and Gallup Tests. URL: https://www.mdpi.com/1999-4893/18/5/275 (дата звернення: 07.04.2026).

26. Mean. URL: https://www.britannica.com/science/mean (дата звернення: 19.03.2026).

27. A Tutorial on Visual Servo Control / S. Hutchinson, G. Hager, P. Corke. — IEEE Transactions on Robotics and Automation, 1996. — 33 p.

28. Proportional Integral Derivative Controller in Control System. URL: https://www.geeksforgeeks.org/electronics-engineering/proportional-integral-derivative-controller-in-control-system (дата звернення: 02.04.2026).

29. CVZone. URL: https://github.com/cvzone/cvzone (дата звернення: 05.04.2026).

30. Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library / G. Bradski, A. Kaehler. – Sebastopol : O'Reilly Media, 2018. – 1016 p.

31. DJITelloPy. URL: https://github.com/damiafuentes/DJITelloPy (дата звернення: 28.03.2026).

32. The NumPy array: a structure for efficient numerical computation. URL: https://arxiv.org/abs/1102.1523 (дата звернення: 13.04.2026).

33. Custom Tkinter: Build Modern, Stylish Python GUIs Easily. URL: https://www.customproc.com/custom-tkinter-guide/ (дата звернення: 25.03.2026).

Додаток А
Лістинг коду класу MainWindow

```
import customtkinter as ctk
import tkinter.messagebox as messagebox
from PIL import Image, ImageTk
import cv2
import time
import threading
import math
import numpy as np
from djitellopy import Tello

import config
from videograbber import VideoGrabber
from tracker import FaceTracker, BIOMETRIC_THRESHOLD

class MainWindow:
    def __init__(self, root):
        self.window = root
        self.window.title("Tello Face Tracker")
        self.window.geometry("950x800")

        self.tracker = FaceTracker()
        self.grabber = None
        self.tello = Tello(host=config.DRONE_IP)

        # Стан
        self.is_distance_active = False
        self.is_flying = False
        self.show_mesh = True

        # Пам'ять
        self.locked_face_center = None
        self.locked_face_area = 0
        self.locked_signature = None

        self.is_tracking_locked = False

        # Калібрування
        self.is_calibrating = False
        self.calibration_data = []
        self.calibration_frames_target = 20

        self.last_battery_check = 0
        self.pressed_keys = set()

        self.setup_ui()
        self.setup_input()
        self.connect_drone()
        self.update_loop()
```

```
        def setup_ui(self):
            # Головний шрифт для статусів
            font_main   =   ctk.CTkFont(family="Roboto",   size=18,
weight="bold")
            font_status = ctk.CTkFont(family="Consolas", size=16)

            # Рівень заряду батареї
            self.battery_label = ctk.CTkLabel(
                self.window,
                text="Заряд батареї: --%",
                font=font_main,
                text_color="#a8a8a8"
            )
            self.battery_label.pack(pady=(15, 5))

            # Відео з камери дрона
            self.video_label = ctk.CTkLabel(self.window, text="",
fg_color="black", corner_radius=10)
            self.video_label.pack(pady=10)
            self.video_label.focus_set()

            self.video_label.bind("<Button-1>",
self.on_mouse_click)
            self.video_label.bind("<Button-3>",
self.cancel_tracking)

            # Статус системи
            self.status_label = ctk.CTkLabel(
                self.window,
                text="Система готова до роботи",
                font=font_status,
                text_color="#1f6aa5"
            )
            self.status_label.pack(pady=5)

            # Панель кнопок
            controls_frame       =       ctk.CTkFrame(self.window,
fg_color="transparent")
            controls_frame.pack(side="bottom",  fill="x",  pady=20,
padx=20)

            controls_frame.grid_columnconfigure((0,  1,  2,  3,  4),
weight=1)

            btn_opts = {"height": 40, "corner_radius": 8, "font":
ctk.CTkFont(weight="bold")}

            self.btn_takeoff = ctk.CTkButton(
                controls_frame,  text="ЗЛІТ",  fg_color="#28a745",
hover_color="#218838",
                command=self.takeoff, **btn_opts
            )
            self.btn_takeoff.grid(row=0, column=0, padx=10)
```

```
            self.btn_land = ctk.CTkButton(
                controls_frame,                    text="ПОСАДКА",
fg_color="#dc3545", hover_color="#c82333",
                command=self.land, **btn_opts
            )
            self.btn_land.grid(row=0, column=1, padx=10)

            self.btn_dist = ctk.CTkButton(
                controls_frame,       text="Супровід:       ВИМК",
fg_color="#6c757d", hover_color="#5a6268",
                command=self.toggle_dist, **btn_opts
            )
            self.btn_dist.grid(row=0, column=2, padx=10)

            self.btn_view = ctk.CTkButton(
                controls_frame,       text="Вигляд:       Сітка",
fg_color="#6f42c1", hover_color="#59339d",
                command=self.toggle_view, **btn_opts
            )
            self.btn_view.grid(row=0, column=3, padx=10)

            self.btn_info = ctk.CTkButton(
                controls_frame,                    text="Довідка",
fg_color="#17a2b8", hover_color="#138496",
                command=self.show_help, **btn_opts
            )
            self.btn_info.grid(row=0, column=4, padx=10)

        # Вибір обличчя кліком на ліву кнопку миші
        def on_mouse_click(self, event):
            gui_x, gui_y = event.x, event.y
            scale_x = config.FRAME_WIDTH / 720
            scale_y = config.FRAME_HEIGHT / 540
            click_x = int(gui_x * scale_x)
            click_y = int(gui_y * scale_y)

            self.locked_face_center = (click_x, click_y)
            self.locked_face_area = 0
            self.locked_signature = None

            self.is_tracking_locked = True
            self.is_calibrating = True
            self.calibration_data = []
            self.tracker.reset_counters()

            self.status_label.configure(text="Сканування
обличчя...", text_color="#1f6aa5")

        # Скидання відстеження кліком на праву кнопку миші
        def cancel_tracking(self, event=None):
            self.is_tracking_locked = False
            self.is_calibrating = False
```

```
            self.locked_face_center = None
            self.locked_signature = None
            self.tracker.reset_counters()
            self.status_label.configure(text="Відстеження
зупинено. Чекаю нову ціль.", text_color="gray")

        def track_by_position(self, faces, last_pos, last_area):
            if not faces or last_pos is None: return None
            best_face = None
            min_dist = float('inf')
            tx, ty = last_pos
            for face in faces:
                fx, fy = face["center"]
                f_area = face["area"]
                dist = math.hypot(fx - tx, fy - ty)

                is_area_ok = True
                if last_area > 0:
                    ratio = f_area / last_area
                    if ratio < 0.6 or ratio > 1.4: is_area_ok =
False

                if dist < min_dist and is_area_ok and dist < 200:
                    min_dist = dist
                    best_face = face
            return best_face

        def     draw_face_on_image(self,     img,     face_data,
is_locked=False, draw_mesh=True, score=None):
            cx, cy = face_data["center"]
            x1, y1, x2, y2 = face_data["bbox"]

            color = (0, 0, 255) if is_locked else (0, 255, 0)

            if draw_mesh:
                for point in face_data["id"]:
                    cv2.circle(img, point, 1, color, cv2.FILLED)
            else:
                pad = 10
                cv2.rectangle(img,   (x1-pad,   y1-pad),   (x2+pad,
y2+pad), color, 2)

            label = "LOCKED" if is_locked else ""
            cv2.putText(img,       label,       (x1,       y1-10),
cv2.FONT_HERSHEY_SIMPLEX, 0.6, color, 2)

            if score is not None:
                score_text = f"Diff: {score:.3f}"
                score_color   =   (0,   255,   0)   if   score   <
BIOMETRIC_THRESHOLD else (0, 0, 255)
                cv2.putText(img,    score_text,    (x1,    y1-35),
cv2.FONT_HERSHEY_SIMPLEX, 0.6, score_color, 2)
```

```
            if is_locked:
                cv2.line(img, (cx-10, cy), (cx+10, cy), (0, 0,
255), 2)
                cv2.line(img, (cx, cy-10), (cx, cy+10), (0, 0,
255), 2)

            return img

        # Основний цикл
        def update_loop(self):
            if self.grabber is not None:
                frame = self.grabber.read()

                if time.time() > 5:
                    try:
                        bat = self.tello.get_battery()
                        b_color = "#28a745" if bat > 50 else
("#ffc107" if bat > 20 else "#dc3545")
                        self.battery_label.configure(text=f"Заряд
батареї: {bat}%", text_color=b_color)
                        self.last_battery_check = time.time()
                    except: pass

                if frame is not None:
                    frame_resized       =       cv2.resize(frame,
(config.FRAME_WIDTH, config.FRAME_HEIGHT))
                    frame_processed,           faces           =
self.tracker.find_faces(frame_resized)

                    target_face = None

                    if self.is_tracking_locked:

                        if self.is_calibrating:
                            if  self.locked_face_center  is  not
None:
                                target_face                     =
self.track_by_position(faces,             self.locked_face_center,
self.locked_face_area)

                                if target_face:

self.calibration_data.append(target_face["signature"])
                                    count = len(self.calibration_data)

self.status_label.configure(text=f"Збір                    даних...
{count}/{self.calibration_frames_target}", text_color="#1f6aa5")

                                    self.locked_face_center         =
target_face["center"]
                                    self.locked_face_area           =
target_face["area"]
```

```
                                if count >=
self.calibration_frames_target:
                                    data_np =
np.array(self.calibration_data)
                                    self.locked_signature =
np.mean(data_np, axis=0).tolist()
                                    self.is_calibrating = False
                                    self.tracker.reset_counters()

self.status_label.configure(text="Обличчя просканировано. Початок
відстеження.", text_color="#28a745")
                                else:

self.status_label.configure(text="Не виявлено обличчя для
калібрування", text_color="#dc3545")

                            else:
                                if self.locked_face_center is not
None:
                                    target_face =
self.track_by_position(faces, self.locked_face_center,
self.locked_face_area)

                                    if target_face:
                                        self.locked_face_center =
target_face["center"]
                                        self.locked_face_area =
target_face["area"]

self.status_label.configure(text="Відстеження цілі...",
text_color="#28a745")
                                    else:
                                        self.locked_face_center = None
                                        self.tracker.reset_counters()

self.status_label.configure(text="Ціль втрачено. Пошук...",
text_color="#fd7e14")
                                else:
                                    best_candidate = None
                                    best_score = 999.0

                                    for face in faces:
                                        score =
self.tracker.compare_signatures(face["signature"],
self.locked_signature)
                                        if score < best_score:
                                            best_score = score
                                            best_candidate = face

                                    cand_sig =
best_candidate["signature"] if best_candidate else None
```

```
                                    is_verified_target,     msg     =
self.tracker.validate_match(
                                        cand_sig,
                                        self.locked_signature,
                                        is_already_locked=False
                                    )

                                    status_color   =   "#28a745"   if
is_verified_target else "#fd7e14"

self.status_label.configure(text=msg, text_color=status_color)

                                    if     is_verified_target     and
best_candidate:
                                        target_face = best_candidate
                                        self.locked_face_center      =
target_face["center"]
                                        self.locked_face_area        =
target_face["area"]

                        pid_yaw, pid_fb, pid_ud = 0, 0, 0
                        if faces:
                            for face in faces:
                                face_score = None
                                if  self.locked_signature  is  not  None
and self.locked_face_center is None:
                                     face_score                      =
self.tracker.compare_signatures(face["signature"],
self.locked_signature)

                                is_target = (face == target_face)

                                frame_processed                      =
self.draw_face_on_image(
                                    frame_processed,  face,  is_target,
self.show_mesh, score=face_score
                                )

                                if is_target:
                                    pid_yaw,    pid_fb,    pid_ud    =
self.tracker.calculate_pid(face)

                        man_lr,    man_fb,    man_ud,    man_yaw    =
self.get_manual_command()
                        final_lr,  final_fb,  final_ud,  final_yaw  =
man_lr, man_fb, man_ud, man_yaw

                        if self.is_tracking_locked and target_face and
not self.is_calibrating:
                            if final_yaw == 0: final_yaw = pid_yaw
                            if final_ud == 0: final_ud = pid_ud
                            if self.is_distance_active and final_fb ==
0: final_fb = pid_fb
```

```
                    if self.is_flying:
                        self.tello.send_rc_control(final_lr,
final_fb, final_ud, final_yaw)

                    img_rgb    =    cv2.cvtColor(frame_processed,
cv2.COLOR_BGR2RGB)
                    img_pil = Image.fromarray(img_rgb)
                    img_tk   =   ctk.CTkImage(light_image=img_pil,
dark_image=img_pil, size=(720, 540))

                    self.video_label.configure(image=img_tk)
                    self.video_label.image = img_tk

            self.window.after(10, self.update_loop)

        # Підключення до дрона
        def connect_drone(self):
            def _connect():
                try:
                    self.tello.connect()
                    self.tello.streamoff()
                    self.tello.streamon()
                    bat = self.tello.get_battery()
                    self.battery_label.configure(text=f"Заряд
батареї: {bat}%")
                    time.sleep(1)
                    self.grabber                                =
VideoGrabber(config.UDP_VIDEO_ADDRESS).start()
                except Exception as e:
                    self.status_label.configure(text=f"Помилка
підключення: {e}", text_color="#dc3545")
            threading.Thread(target=_connect, daemon=True).start()

        # Ручне керування
        def setup_input(self):
            self.window.bind("<KeyPress>", self.key_down)
            self.window.bind("<KeyRelease>", self.key_up)

        def key_down(self, event):
            self.pressed_keys.add(event.keysym.lower())
            if event.keysym.lower() == 'space': self.land()

        def key_up(self, event):
            k = event.keysym.lower()
            if k in self.pressed_keys: self.pressed_keys.remove(k)

        def get_manual_command(self):
            lr, fb, ud, yv = 0, 0, 0, 0
            speed = config.MANUAL_SPEED
            pk = self.pressed_keys
            if 'w' in pk: fb = speed
            if 's' in pk: fb = -speed
```

```
            if 'a' in pk: lr = -speed
            if 'd' in pk: lr = speed
            if 'q' in pk: yv = -speed
            if 'e' in pk: yv = speed
            if 'shift_l' in pk: ud = speed
            if 'control_l' in pk: ud = -speed
            return lr, fb, ud, yv

        # Зліт
        def takeoff(self):
            try:
                self.tello.takeoff()
                self.tello.send_rc_control(0,0,25,0)
                self.is_flying = True
                self.status_label.configure(text="Зліт  виконано",
text_color="#28a745")
            except: pass

        # Посадка
        def land(self):
            self.cancel_tracking()
            try:
                self.tello.land()
                self.is_flying = False
                self.status_label.configure(text="Посадка...",
text_color="#dc3545")
            except: pass

        # Супровід
        def toggle_dist(self):
            self.is_distance_active = not self.is_distance_active
            if self.is_distance_active:
                self.btn_dist.configure(text="Супровід:    УВІМК",
fg_color="#fd7e14", hover_color="#e36209")
            else:
                self.btn_dist.configure(text="Супровід:     ВИМК",
fg_color="#6c757d", hover_color="#5a6268")

        # Сітка/рамка
        def toggle_view(self):
            self.show_mesh = not self.show_mesh
            if self.show_mesh:
                self.btn_view.configure(text="Вигляд:      Сітка",
fg_color="#6f42c1", hover_color="#59339d")
            else:
                self.btn_view.configure(text="Вигляд:      Рамка",
fg_color="#007bff", hover_color="#0056b3")

        # Довідка
        def show_help(self):
            messagebox.showinfo("Інфо", "ЛКМ по обличчі: Захопити
ціль\nПКМ:     Скинути     ціль\nПробіл:     Посадка\nWASD:     Рух
```

```
вперед/назад/вліво/вправо\nQ/E:   Поворот   проти/за   годинниковою
стрілкою\nShift/Ctrl: Вгору/вниз")
            self.video_label.focus_set()

        # Закриття програми
        def close(self):
            if self.grabber: self.grabber.stop()
            self.window.destroy()
```

Додаток Б
Лістинг коду класу FaceTracker

```
import cvzone
from cvzone.FaceMeshModule import FaceMeshDetector
import cv2
import numpy as np
import math
import config

# Налаштування біометрії
BIOMETRIC_THRESHOLD = 0.1   # Поріг упевненості
CONFIDENCE_FRAMES = 30      # Кількість кадрів для калібрування

class FaceTracker:
    def __init__(self):
        self.detector      =      FaceMeshDetector(maxFaces=5,
minDetectionCon=0.1, minTrackCon=0.5)

        self.pid = config.PID_COEFFICIENTS
        self.pError = 0
        self.w = config.FRAME_WIDTH
        self.h = config.FRAME_HEIGHT
        self.fbRange = config.FACE_AREA_RANGE

        # Лічильники
        self.consecutive_match_frames = 0

    def reset_counters(self):
        self.consecutive_match_frames = 0

    # Пошук облич
    def find_faces(self, img):
        # draw=False -> Трекер не малює, він тільки шукає
        img,    faces    =    self.detector.findFaceMesh(img,
draw=False)
        processed_faces = []

        if faces:
            for face in faces:
                left = face[234]
                right = face[454]
                top = face[10]
                bottom = face[152]
                cx, cy = face[1]

                w_face = right[0] - left[0]
                h_face = bottom[1] - top[1]
                area = w_face * h_face

                signature                                    =
self.calculate_extended_signature(face)
```

```
                    face_data = {
                        "id": face,
                        "center": (cx, cy),
                        "bbox":   (left[0],   top[1],   right[0],
bottom[1]),
                        "area": area,
                        "signature": signature
                    }
                    processed_faces.append(face_data)

            return img, processed_faces

        # Обчислення профіля обличчя
        def calculate_extended_signature(self, face):
            face_width   =   math.hypot(face[234][0]-face[454][0],
face[234][1]-face[454][1])
            face_height   =   math.hypot(face[10][0]-face[152][0],
face[10][1]-face[152][1])

            if  face_width  ==  0  or  face_height  ==  0:  return
[0,0,0,0,0]

            # Вектори ознак обличчя
            # R1
            dist_eyes_inner       =       math.hypot(face[133][0]-
face[362][0], face[133][1]-face[362][1])
            r1 = dist_eyes_inner / face_width
            # R2
            dist_mouth   =   math.hypot(face[61][0]-face[291][0],
face[61][1]-face[291][1])
            r2 = dist_mouth / face_width
            # R3
            eye_y = (face[133][1] + face[362][1]) / 2
            nose_y = face[1][1]
            r3 = abs(eye_y - nose_y) / face_height
            # R4
            mouth_y = (face[61][1] + face[291][1]) / 2
            r4 = abs(nose_y - mouth_y) / face_height
            # R5
            chin_y = face[152][1]
            r5 = abs(mouth_y - chin_y) / face_height

            return [r1, r2, r3, r4, r5]

        # Порівняння виявленого обличчя з еталоном
        def compare_signatures(self, sig1, sig2):
            if not sig1 or not sig2: return 999.0

            diff_1 = abs(sig1[0] - sig2[0]) * 2.0
            diff_2 = abs(sig1[1] - sig2[1]) * 0.5
            diff_3 = abs(sig1[2] - sig2[2]) * 1.5
            diff_4 = abs(sig1[3] - sig2[3]) * 1.0
```

```
            diff_5 = abs(sig1[4] - sig2[4]) * 1.0

            return diff_1 + diff_2 + diff_3 + diff_4 + diff_5

        # Відстеження
        def validate_match(self, current_sig, target_sig,
is_already_locked):
            if is_already_locked:
                self.reset_counters()
                return True, "Відстеження цілі..."

            score = self.compare_signatures(current_sig,
target_sig)

            if score < BIOMETRIC_THRESHOLD:
                self.consecutive_match_frames += 1
                msg = f"Перевірка...
({self.consecutive_match_frames}/{CONFIDENCE_FRAMES})"
                if self.consecutive_match_frames >=
CONFIDENCE_FRAMES:
                    self.reset_counters()
                    return True, "Ціль підтверджено"
            else:
                if self.consecutive_match_frames > 0:
                    self.consecutive_match_frames -= 1
                msg = "Пошук..."

            return False, msg

        # Обчислення команд керування дроном
        def calculate_pid(self, face_data):
            if face_data is None: return 0, 0, 0
            area = face_data["area"]
            cx, cy = face_data["center"]
            fb, yaw, ud = 0, 0, 0

            error_yaw = cx - self.w // 2
            yaw = self.pid[0] * error_yaw + self.pid[1] *
(error_yaw - self.pError)
            yaw = int(np.clip(yaw, -100, 100))
            if -45 < error_yaw < 45: yaw = 0; error_yaw = 0
            self.pError = error_yaw

            if area != 0:
                if area < self.fbRange[0]: fb = 25
                elif area > self.fbRange[1]: fb = -25

            error_ud = (self.h // 2) - cy
            ud = int(error_ud * 0.25)
            ud = int(np.clip(ud, -50, 50))
            if -45 < error_ud < 45: ud = 0

            return yaw, fb, ud
```

Додаток В

Лістинг коду класу VideoGrabber

```
import cv2
import threading
import time

class VideoGrabber:
    def __init__(self, addr):
        self.cap = cv2.VideoCapture(addr)
        self.cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)
        self.grabbed, self.frame = self.cap.read()
        self.stopped = False
        self.lock = threading.Lock()

    # Створення окремого потоку
    def start(self):
        threading.Thread(target=self.update,          args=(),
daemon=True).start()
        return self

    def update(self):
        while not self.stopped:
            if not self.cap.isOpened():
                time.sleep(0.1)
                continue

            grabbed, frame = self.cap.read()

            if grabbed:
                with self.lock:
                    self.grabbed = grabbed
                    self.frame = frame
            else:
                time.sleep(0.01)

    def read(self):
        with self.lock:
            return self.frame if self.grabbed else None

    def stop(self):
        self.stopped = True
        self.cap.release()
```

Додаток Г

Налаштування системи

```
# Налаштування мережі
DRONE_IP = '192.168.10.1'
UDP_VIDEO_ADDRESS = 'udp://192.168.10.1:11111'

# Налаштування зображення
FRAME_WIDTH = 1280
FRAME_HEIGHT = 720

# Налаштування PID контролера [Kp, Ki, Kd]
PID_COEFFICIENTS = [0.15, 0.15, 0]

# Діапазон площі обличчя [min, max]
FACE_AREA_RANGE = [25000, 35000]

# Швидкість ручного керування
MANUAL_SPEED = 50
```

Додаток Д

Копія публікації

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

УДК 004.8:005.8](063)

Присвячено 60-річчю Західноукраїнського національного університету

ICSPM 2026: Інтелектуальні обчислювальні системи та управління проєктами : збірник тез доповідей І Міжнародної науково-практичної конференції студентів і молодих вчених (Тернопіль, 21–22 травня 2026 року). – Тернопіль : ЗУНУ, 2026. – 190 с.

До збірника увійшли тези доповідей учасників І Міжнародної науково-практичної конференції студентів і молодих вчених «ICSPM 2026: Intelligent Computing Systems and Project Management / Інтелектуальні обчислювальні системи та управління проєктами», що відбулася на базі кафедри інформаційно-обчислювальних систем і управління факультету комп'ютерних інформаційних технологій Західноукраїнського національного університету.

Матеріали відображають результати досліджень за такими напрямами:

1. Інтелектуальні обчислення та програмні системи.
2. Проєктно-орієнтоване управління та процеси прийняття рішень.
3. Штучний інтелект, аналітика даних і кібернетика.
4. Прикладні технології та інформаційно-комунікаційні системи.
5. Сучасні інформаційні технології в економіці, праві та освіті.

Рекомендовано до друку на засіданні кафедри інформаційно-обчислювальних систем і управління Західноукраїнського національного університету (протокол №12 від 26.05.2026 р.).

За зміст наукових праць, точність наведених цитувань, перекладу та достовірність фактологічних матеріалів відповідальність несуть автори публікацій. У збірнику збережено авторську стилістику, пунктуацію та орфографію.

ЗМІСТ

УДК 004.932.2:629.7

ПРОГРАМНИЙ МОДУЛЬ ВИЯВЛЕННЯ ТА ВІДСЛІДКОВУВАННЯ ОБЛИЧ ІЗ ВИКОРИСТАННЯМ ДРОНА RYZE TELLO

Ковтуненко Андрій Олександрович, студент,
Kovtunenko.andrii2005@gmail.com

Науковий керівник: Биковий Павло Євгенович,
к.т.н., доцент, доцент кафедри інформаційно-обчислювальних систем і управління,
Західноукраїнський національний університет,
pb@wunu.edu.ua
ORCID: 0000-0002-5705-5702

У сучасних умовах галузь безпілотних літальних апаратів переживає період надзвичайно стрімкого технологічного прогресу та масового впровадження у найрізноманітніші сфери діяльності. Дрони еволюціонували від простих дистанційно керованих пристроїв до складних автономних систем, здатних виконувати високоточну відеозйомку, моніторинг об'єктів та інтелектуальний супровід цілей без постійної участі людини.

Одним із найбільш актуальних завдань розвитку сучасної робототехніки є створення надійних систем автономного відеосупроводу, які здатні працювати у складних динамічних середовищах. Значна частина існуючих рішень досі використовує застарілі алгоритми, такі як каскади Хаара [1]. Вони мають низьку точність, чутливі до змін освітлення та, найголовніше, не здатні ідентифікувати конкретну особу серед інших, що призводить до частої втрати цілі в багатолюдних середовищах. Це зумовлює необхідність розробки нових програмних модулів на базі сучасних нейромережевих моделей та адаптивної біометрії.

Запропоноване рішення впроваджує метод адаптивної векторної біометрії, що базується на нейромережевій моделі Face Mesh бібліотеки MediaPipe [2]. Ця передова модель здатна визначати координати 468 ключових точок обличчя, надаючи достатньо даних для роботи алгоритму ідентифікації. Мовою програмування було обрано Python, що дозволяє легко інтегрувати оптимізовані бібліотеки комп'ютерного зору та взаємодії з дроном.

Взаємодія з апаратною частиною дрона відбувається через бібліотеку djitellopy, яка відповідає за обробку телеметрії та відправку команд дрону [3]. Для обробки відеопотоку використано OpenCV. Інтерфейс користувача розроблений на базі CustomTkinter. Він забезпечує виведення відеопотоку з камери дрона, елементи керування дроном, та відображення телеметричних даних, таких як рівень заряду акумулятора та поточна похибка валідації цілі у реальному часі.

Функціональні можливості програми включають повністю автономний політ після вибору цілі. Оператор у графічному інтерфейсі обирає об'єкт для відстеження одним кліком миші, після чого система формує індивідуальний біометричний профіль обличчя, отримуючи координати ключових точок обличчя в реальному часі.

Програма безперервно контролює стан дрона. Вона автоматично зчитує рівень заряду акумулятора та ініціює примусову безпечну посадку при падінні заряду нижче критичної позначки.

Також передбачена система інтелектуальної пам'яті: якщо ціль на мить зникла з поля зору (наприклад, її перекрив інший об’єкт), дрон зависає в точці останнього контакту і автоматично відновлює стеження, як тільки ідентифікує потрібне обличчя порівнявши його з еталонним профілем.

При потребі оператор у будь-який момент може перехопити керування за допомогою клавіатури або натиснути клавішу «пробіл» для екстреної посадки.

Процес ідентифікації облич полягає у порівнянні поточного профілю із еталонним, який обчислюється під час стартового калібрування системи при виборі цілі (середнє арифметичне обчислених векторів за 30 кадрами). Під час відстеження система обчислює векторні коефіцієнти для кожного обличчя, яке потрапляє в кадр, та порівнює їх із коефіцієнтами, отриманими під час калібрування системи.

Якщо похибка, тобто різниця між коефіцієнтами поточного обличчя і еталонними, перевищує встановлений поріг – система вважає їх різними обличчями і продовжує процес порівняння, поки похибка не опуститься нижче порогу. Похибка розраховується за метрикою Мангеттена з використанням вагових коефіцієнтів [4]. Чим більш стабільною та незалежною від рухів є ознака – тим більший ваговий коефіцієнт вона має.

Окрім модуля розпізнавання в системі присутній також модуль відстеження. Принцип його роботи полягає в утриманні обличчя в центрі кадру. Відхилення центру обличчя від центру кадру - це просторова похибка. На основі неї обчислюються параметри, які далі перетворюються в команди керування дроном. Вони вказують дрону в якому напрямку рухатися, щоб обличчя завжди було в центрі кадру.

За плавність рухів відповідає пропорційно-диференціальний регулятор, який регулює швидкість руху дрона залежно від величини просторової похибки, а також забезпечує плавне гальмування дрона при наближенні центра обличчя до центру кадру [5]. Це усуває різкі рухи та забезпечує плавне динамічне відео.

Під час тестування система продемонструвала хороші результати (рисунок 1). Тестування показало, що система здатна коректно розрізняти обличчя, не втрачала ціль при появі інших людей у кадрі та успішно відновлювала стеження після тимчасового перекриття обличчя. Програму протестовано на семи різних обличчях, помилкову ідентифікацію було зафіксовано лише один раз.

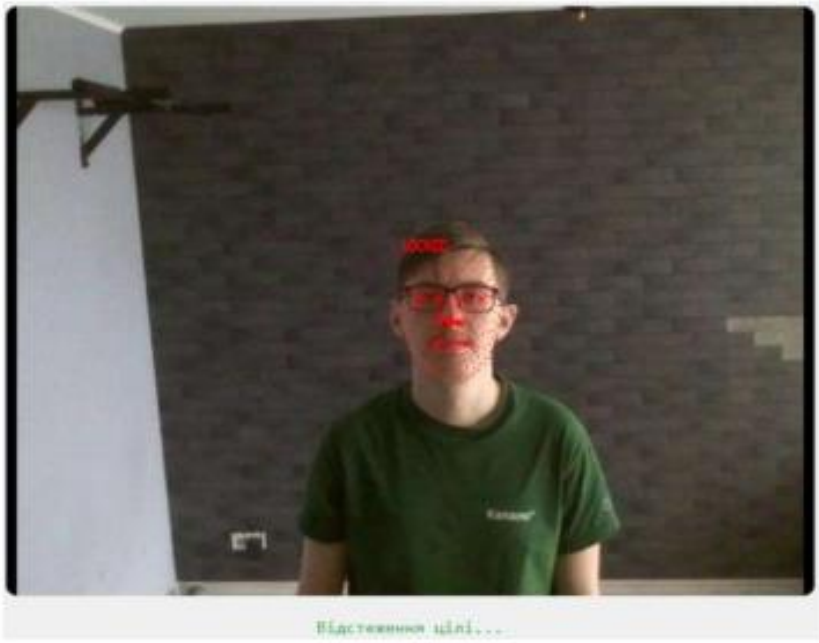

Рисунок 1 – Результат роботи програмного модуля

Список використаних джерел

1. Viola P., Jones M. Rapid Object Detection Using a Boosted Cascade of Simple Features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. (CVPR 2001)*, Kauai, 8-14 December 2001, I-511-I-518. https://doi.org/10.1109/CVPR.2001.990517

2. Lugaresi, C., Tang, J., Nash, H., McClanahan, C., Uboweja, E., Hays, M., Zhang, F., Chang, C. L., Yong, M. G., Lee, J., Chang, W. T., Hua, W., Georg, M., Grundmann, M. MediaPipe: A Framework for Building Perception Pipelines. 2019. *arXiv:1906.08172*. https://arxiv.org/abs/1906.08172.

3. Fuentes, D. DJITelloPy: Python library for DJI Tello drones. GitHub, 2021. https://github.com/damiafuentes/DJITelloPy.

4. Deza, M. M., Deza, E. Encyclopedia of Distances. Springer, 2009. https://doi.org/10.1007/978-3-642-00234-2.

5. Ang, K. H., Chong, G., & Li, Y. (2005). PID control system analysis, design, and technology. *IEEE Transactions on Control Systems Technology*. 2005. Vol. 13(4). P. 559–576. https://doi.org/10.1109/TCST.2005.847331.