НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ

І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри

комп’ютерних наук

_____ Голуб Б.Л.

підпис ПІБ

“___” _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

СИСТЕМА РОЗПІЗНАВАННЯ І ВИЗНАЧЕННЯ ХАРАКТЕРИСТИК РУХОМИХ ОБ’ЄКТІВ З ВИКОРИСТАННЯМ МЕТОДІВ КОМП’ЮТЕРНОГО ЗОРУ

Спеціальність 121 – «Інженерія програмного забезпечення»

ОП «Інженерія програмного забезпечення»

Гарант освітньої програми

_____к.т.н., доцент_____ / Голуб Б. Л. /

Науковий ступінь та вчене звання підпис ПІБ

Керівник бакалаврської кваліфікаційної роботи

__асистент_____ / Віннічук Д. О. /

Науковий ступінь та вчене звання підпис ПІБ

Консультант бакалаврської кваліфікаційної роботи

__д.т.н, професор_____ / Семко В. В. /

Науковий ступінь та вчене звання підпис ПІБ

Виконав_____ / Арцибарський І. В. /

підпис ПІБ студента

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ

Завідувач кафедри

комп'ютерних наук

_____ / Голуб Б.Л., доцент, к.т.н /

підпис

"__"_____2024 р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

студенту Арцибарському Івану Володимировичу

Спеціальність 121 - «Інженерія програмного забезпечення»

ОП «Інженерія програмного забезпечення»

1. Тема роботи: Система розпізнавання і визначення характеристик рухомих об'єктів з використанням методів комп'ютерного зору

   Затвепджена наказом ректора НУБіП України №2248 "С" від 16.12.2024

2. Термін подання завершеної роботи на кафедру 2025 . 06 . 02
   рік, місяць, число
3. Вихідні дані до роботи: опис програмного забезпечення
4. Перелік питань, що розглядаються:
   1. Аналіз предметної області.
   2. Вибір та обґрунтування засобів для розробки системи.
   3. Проектування інформаційної системи.
   4. Висновки

Керівник бакалаврської кваліфікаційної роботи_____ / Віннічук Д. О. /
підпис ініціали та прізвище

Консультант бакалаврської кваліфікаційної роботи_____ / Семко В. В. /
підпис ініціали та прізвище

Завдання прийняв до виконання _____ / Арцибарський І. В. /
підпис ініціали та прізвище

Дата отримання завдання 2024 . 12 . 16

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

1. YOLO – You Only Look Once.
2. CV – Computer Vision.
3. NBA – National Basketball Association.
4. UEFA – Union of European Football Associations.
5. MOT – Multi-Objects Traking.
6. DL – Deep Learning.
7. AI – Artificial Intelligence.
8. TRACAB – Tracking and Control Application for Broadcast.
9. GPU – Graphics Processing Unit.
10. CPU – Central Processing Unit.
11. SQL – Structured Query Language.
12. UML – Unified Modeling Language.
13. ER – Entity-Relationship.
14. UI – User Interface.
15. GUI – Graphical User Interface.
16. ACID – Atomicity, Consistency, Isolation, Durability.
17. MVCC – Multi-Version Concurrency Control.
18. DDL – Data Definition Language.
19. DML – Data Manipulation Language.
20. NMS – Non-Maximum Suppression.
21. ID – Identifier.
22. CUDA – Compute Unified Device Architecture.
23. БД – База даних.
24. СУБД – Система управління базами даних.

ВСТУП

Актуальність теми. Аналітика присутня у будь якому виді спорту та стрімко розвивається. Ручний збір та аналіз даних є неефективним та вимагає багато часу, особливо на фоні зростаючого обсягу відеоінформації, який необхідно аналізувати. Комп'ютерний зір та нейронні мережі використовуються доволі розповсюджено у спортивній аналітиці. Тому розробка системи для виявлення та аналізу рухомих об'єктів є досить важливим завданням.

Метою дипломної роботи є створення системи, що автоматично виявляє гравців на полі та м'яч, зберігає та надає їх статистичні дані для полегшення та підвищення ефективності подальшого аналізу.

Методи та технології дослідження. Під час розробки було використано методи глибокого навчання, а саме нейронні мережі YOLO для розпізнавання об'єктів, технологію комп'ютерного зору для обробки відео та компенсації руху камери та метод геометричних перетворень для отримання точних координат об'єктів шляхом перетворення перспективи камери на площину футбольного поля.

Апробація програмного додатку. За результатами виконання роботи було підготовлено тези доповідей для конференції "Теоретичні та прикладні аспекти розробки комп'ютерних систем "2025"", яка проходила 24 квітня 2025 року.

Структура записки. Записка складається із 4 основних розділів, складається з 88 сторінок, 14 використаних джерел та 8 додатків. У 1 розділі описується аналіз предметної області, вимог до системи та існуючих рішень. У 2 розділі відбувається проектування програмного забезпечення за допомогою UML-діаграм та ER-моделювання бази даних. 3 розділ охоплює саме опис технічних аспектів реалізації системи та її модулів. 4 розділ присвячений тестуванню системи, визначення апаратних вимог техніки та опис складу інсталяційного пакету додатку.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Спортивна аналітика - це галузь, яка стрімко розвивається. Вона займається збором і аналізом даних про спортивні змагання з метою підвищення ефективності команд і окремих спортсменів. Традиційно, у цій сфері аналітики покладались на ручний збір статистики з матчів та розмітку відеозаписів, але такий метод вимагав значний об'єм трудових затрат. У сучасності обсяги відеоданих стрімко зростають - від записів і трансляцій спортивних ігор до записів із камер відеоспостереження, тому автоматизація їх обробки отримала критичний статус. Новітні досягнення в галузі комп'ютерного зору (CV) суттєво вплинули на розвиток аналізу вмісту відеоданих, зокрема на спортивну аналітику, оскільки вони дозволяють автоматизувати збір, аналіз та інтерпретацію важливої інформації з відеоматеріалів змагань. Завдяки алгоритмам CV стає можливим витягувати із відео змістовну та структуровану інформацію про гру, що забезпечує детальні відомості про переміщення гравців і командну тактику. Замість затратної та виснажливої ручної розмітки, яка раніше була єдиним шляхом отримання даних, комп'ютерний зір пропонує більш ефективну альтернативу. Це підвищує оперативність і точність аналізу одночасно знижуючи витрати часу та людського ресурсу[1].

Роль автоматизованої обробки відеоданих у спорті підтверджується зростаючими інвестиціями спортивних клубів і ліг у відповідні технології. Команди все активніше впроваджують системи відстеження гравців, м'яча та подій на полі, щоб в результаті отримати об'єктивні дані для прийняття рішень щодо тренувань і тактики[2]. Розвинені спортивні ліги (наприклад, NBA, UEFA) вже використовують аналітичні платформи на основі комп'ютерного зору, які збирають показники ефективності гравців, командні статистичні показники і навіть автоматично

генерують звіти або графіки на телебаченні. Зростає попит і з боку вболівальників та медіа: аудиторія прагне глибшого розуміння стратегічних компонентів гри та індивідуальної результативності спортсменів[3]. Це стимулює розвиток комп'ютерного зору для спорту, адже тільки автоматизовані засоби здатні обробляти та інтерпретувати великі масиви відеоданих у реальному часі.

Предметна область даної роботи охоплює застосування комп'ютерного зору в футбольній аналітиці. Футбол є однієї із найпопулярніших ігор у світі, що характеризується динамічністю та складністю колективних дій 22 гравців на великому полі. Специфіка футбольного відео полягає в тому, що гра майже безперервна, гравці переміщуються великим простором, групуються, перекривають один одному оглядовість, а м'яч може швидко змінювати своє позиціонування на полі. Крім того, гравці мають схожий вигляд форми, що ускладнює їх розрізнення на зображенні. Ці фактори створюють труднощі для автоматизованого аналізу - швидкий темп гри, спотворення картинки через рух камери або через зміну перспективи, взаємні перекриття (оклюзії) гравців, а також однорідність форми - все це ускладнює задачу багатооб'єктного трекінгу (MOT)[3].

Комп'ютерний зір відіграє критичну роль у вирішенні вищезгаданих проблем. Сучасні моделі глибокого навчання (DL) у поєднанні з потужними графічними процесорами просунули у перед можливості аналізу відео. Системи комп'ютерного зору успішно застосовують у різних задачах: детекція об'єктів, відстеження траєкторії об'єктів у часі, розпізнавання дій (наприклад, удар по м'ячу, гол), зміна перспективи на геометричну площину (так звана прив'язка до поля або homography alignment) тощо[1]. За допомогою таких технологій можна з легкістю отримувати детальні дані про розташування гравців та м'яча протягом матчу, їх швидкість, пройдену дистанцію, учать у ігрових епізодах, що дає можливість аналізувати тактику команд і індивідуальні характеристики гравців. Наприклад, система комп'ютерного зору може розпізнавати розміри футбольного поля, ідентифікувати

гравців та зіставляти їх положення на відео з реальними координатами на полі[1] - таким чином генерується набір точних статистичних показників про гру.

Варто зазначити, що комп'ютерний зір розглядається як необхідний інструмент у сучасному футболі, оскільки альтернативні методи відстеження мають певні обмеження. Наприклад, застосування носимих GPS-трекерів на гравцях дає корисні метрики, але в офіційних матчах топ-рівня такі пристрої часто не дозволяються, або використовуються дуже обмежено. Натомість, камери здатні неінвазивно фіксувати положення гравців під час гри[4]. Дійсно, комп'ютерний зір на даний момент часу є найкращим із можливих рішень для здійснення трекінгу гравців, не зважаючи на вищезгадані виклики. Без автоматичного трекінгу неможливо отримати точні кількісні показники, необхідні для сучасного наукового підходу до управління командою.

Таким чином, предметна область розроблюваної системи є надзвичайно актуальною. Йдеться про використання методі комп'ютерного зору для розв'язання завдань автоматичного виявлення, відстеження та обрахування статистичних даних об'єктів (насамперед, гравців та м'яча) у сценах футбольного матчу. Це включає створення технологій для обробки відеопотоку, а саме для виділення об'єктів в кадрі та отримання статистичних даних. У підсумку подібні системи надають можливість тренерам й аналітикам новий рівень для аналізу своїх футбольних команд та тактики гри суперників.

1.2 Аналіз вимог до програмної системи

При розробці системи розпізнавання і аналізу рухомих об’єктів варто визначити перелік функціональних та нефункціональних вимог до неї. Ці вимоги базуються на потребах користувачів системи та особливостей предметної області.

Функціональні вимоги описують те, що саме має виконувати система, її безпосередні функції та послуги:

1. Виявлення об’єктів на відео – система повинна автоматично розпізнавати на кожному кадрі відеозапису всю необхідну інформацію – футболістів обох команд, кольори їх форми та м’яч. Для цього використовуються алгоритми розпізнавання об’єктів, здатні окреслювати гравці та м’яч обмежувальними рамками. Модель повинна вміти визначати різні класи об’єктів і визначати їх положення в кадрі.
2. Трекінг (відстеження) об’єктів – після виявлення об’єктів система має відстежувати їх переміщення від кадру до кадру, призначаючи кожному об’єкту ідентифікатор. Це дозволяє будувати траєкторію руху об’єктів протягом часу, обчислювати пройдену відстань, середню швидкість об’єкта тощо. Сучасні системи комп’ютерного зору надають багатооб’єктне відстеження на полі, зокрема здатні прогнозувати позиціонування об’єкту навіть при тимчасові його втраті із поля зору.
3. Розпізнавання індивідуальності гравців – система повинна уміти ідентифікувати, який саме гравець відображений. Для цього при виявлення об’єктів їм надається унікальний ідентифікатор, який буде використовуватись протягом всього аналізу відеозапису. Ідентифікація гравців важлива, щоб пов’язати зібрані траєкторії та статистику з конкретними футболістами та надалі аналізувати індивідуальні показники.
4. Прив’язка до координат реального поля – необхідно, щоб система могла трансформувати координати об’єктів з пікселів відео у координати на площині футбольного поля. Для цього використовується модель поля і методи геометричної трансформації (гомографії). Прив’язка дозволяє згодом будувати точні метрики та зручні. Деякі системи додають калібрувальні мітки або відомі орієнтири (наприклад, лінії на полі) для автоматичного встановлення відповідності між відео та реальною системою координат[1].

5. Обчислення динамічних характеристик – система повинна вміти обчислювати характеристики руху об’єктів на основі їх положення. Сучасні рішення на базі AI-камер вже зараз здатні відстежувати рух гравців, аналізувати траєкторію польоту м’яча, генерувати теплові карти та контролювати дистанцію, швидкість і прискорення гравців[5].
6. Надання результатів у зручній формі – результати роботи системи мають бути доступні користувачам. Це може бути відображення даних на інтерфейсі користувача та інтерактивна візуалізація: накладення графічних елементів на відео тощо. Наприклад, система може інтегруватися з аналітичним програмним забезпеченням, яке в реальному часі будує графіку для телетрансляції (такі елементи як лінії офсайду, шляхи передач). У контексті дипломної роботи достатньо, щоб дані зберігалися в базі даних і могли бути інтерпретовані в зрозумілому для користувача системи форматі.

Окрім функціоналу, необхідно визначити нефункціональні вимоги до системи, які характеризують якість роботи системи:

1. Точність – система повинна забезпечувати високу точність розпізнавання та трекінгу. Помилки у виявленні та відстеженні мають бути мінімізовані. Наприклад, показник ідентифікованих гравців у сучасних системах розпізнавання досягає ~87% точності визначення гравців[6].
2. Продуктивність – бажано забезпечити обробку відеопотоку системою зі швидкістю не меншою за частоту кадрів у відео (зазвичай 25-30 кадрів/с). Це особливо актуально, якщо система використовується безпосередньо під час матчу в режимі трансляції в реальному часі. Наприклад, відомі професійні рішення видають аналітичні дані практично миттєво під час гри, що дозволяє використовувати їх у телетрансляціях[7]. У межах нашого проекту допускається офлайн-обробка, але всеодно алгоритми мають бути оптимізовані. Для досягнення високої швидкодії можуть використовуватися

обчислення графічним процесором, багатопоточність та оптимізовані бібліотеки CV.

3. Надійність та стійкість – система повинна стабільно працювати протягом усього процесу обробки відеопотоку, не допускаючи збоїв. Вона має коректно поводитися у випадку тимчасової втрати відеосигналу або якщо об’єкти зникли з кадру. Не менш важливою є узгодженість даних – наприклад, якщо двоє гравців зіткнулися і тимчасово алгоритм міг сплутати їх треки, система має вжити заходів для виправлення цього.

Отже, сформульовані вимоги визначають вектор розробки: система має автоматично розпізнавати та відстежувати гравців і м’яч на футбольному відео, забезпечувати високу точність і продуктивність, надавати статистичні показники руху в зручному вигляді. Як приклад орієнтовного функціоналу можна навести сучасні AI-камери, що вже зараз вміють відстежувати рух гравців, аналізувати траєкторії м’яча та генерувати показники швидкості і прискорення для атлетів[5]. Наша система покликана реалізувати подібні можливості у рамках академічного проекту, ґрунтуючись на передових підходах комп’ютерного зору.

1.3 Моделювання предметної області

Проектування системи вимагає глибокого розуміння предметної області та побудови її моделі. Моделювання предметної області – це процес створення абстрактного уявлення про об’єкти реального світу, їх властивості та взаємозв’язки, з темою спрощення подальшою розробки програмного забезпечення. Згідно з визначенням, моделлю предметної області є структуроване візуальне представлення взаємопов’язаних концепцій або об’єктів реального світу, що включає ключові сутності, їх поведінку, атрибути та відносини між ними[8].

Іншими словами, модель предметної області вводить формальні терміни для всіх важливих елементів системи і показує, як вони пов’язані.

На першому етапі моделювання необхідно проаналізувати акторів та їх взаємодію із предметною областю. Тут корисно будувати діаграми випадків використання (Use Case Diagrams) (рис. 1) та діаграми послідовності (рис. 2). Діаграма випадків використання відображає основні сценарії взаємодії користувачів із системою (наприклад, розпізнати гравців чи м’яч на відео, перегляд відео тощо). Діаграми послідовності, своєю чергою, показують покрокову взаємодію об’єктів системи або акторів у рамках предметної області, для нашої системи було змодельовано взаємодію між тренером футбольної команди та аналітиком футбольних матчів за для отримання статистичної інформації про гравців та матч.

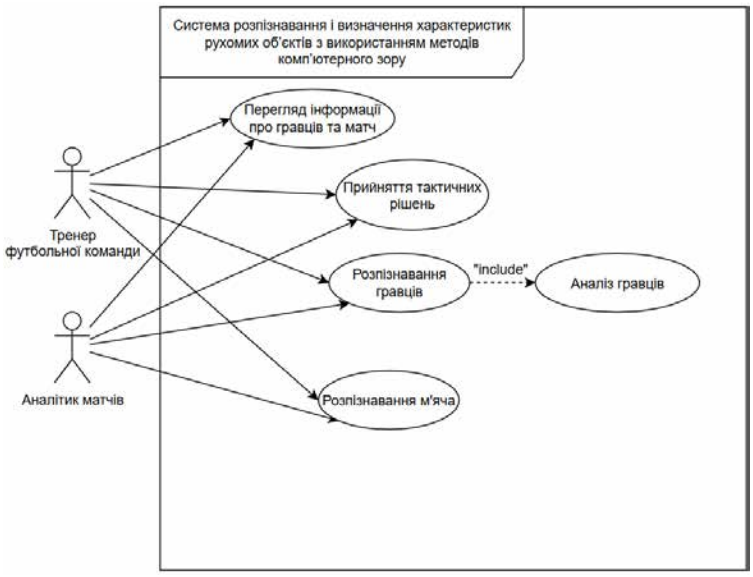

Рис. 1. Діаграма випадків використання предметної області системи

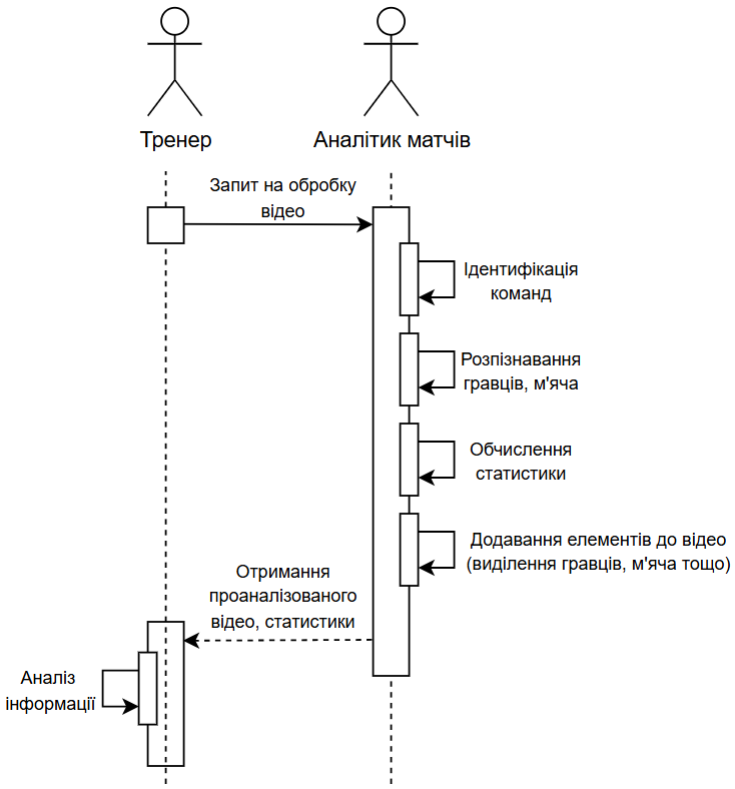

Рис. 2. Діаграма послідовності предметної області системи

Також для повноти картини була створена діаграма розгортання (рис. 3), яка відображає розташування компонентів системи на апаратних вузлах. У випадку нашої системи, всі необхідні компоненти системи будуть розгортатися на локальній машині: клієнтський додаток, на якому буде відбуватись вся взаємодія користувача із системою, сервіс для обробки та аналізу відео, модель нейронної мережі для розпізнавання гравців чи м’яча, та база даних, у якій сервіс буде зберігати та надавати доступ до даних через клієнтський додаток.

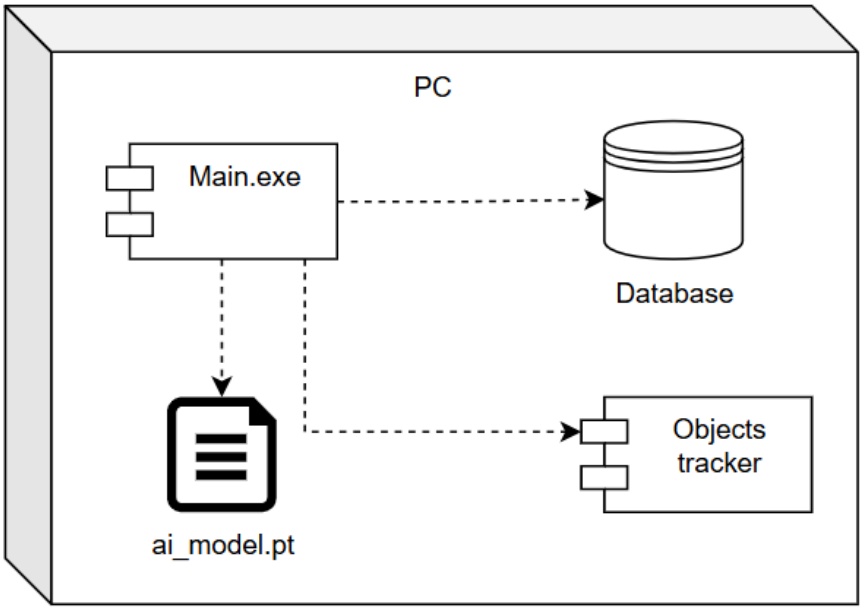

Рис. 3. Діаграма розгортання системи

1.4 Огляд інформаційних джерел та існуючих рішень

Для створення ефективної системи комп'ютерного зору в футбольній аналітиці важливо врахувати досвід існуючих рішень у цій галузі. Було розглянуто кілька актуальних підходів та рішень, що поєднують трекінг об'єктів, розпізнавання та статистичний аналіз у відеозаписі матчу.

1. Оптичні багатокамерні системи відстеження (TRACAB).

Ця система є одним із провідних промислових рішень, що встановлена на багатьох стадіонах провідних ліг світу. TRACAB використовує кілька синхронних камер, розташованих по периметру поля, та комп'ютерний зір для визначення координат гравців і м'яча в будь який момент матчу. Фактично, це створює повну тривимірну картинку гри: програмне забезпечення аналізує зображення з усіх камер, виділяє об'єкти і обчислює їх X, Y та Z координати, досягаючи тривимірного

трекінгу в реальному часі[7]. Перевагою такого підходу є висока точність та повнота даних – за результатами валідації, похибка визначення гравця складає лише близько 8 см, швидкості - ~0.08м/с[9]. TRACAB вже сьогодні надає нові метрики для футболу – визначення скелетів моделі гравців (положення кінцівок) для моделювання рухів у 3D. Недоліки багатокамерних систем – це їх висока вартість та складність розгортання. Потрібно встановлювати обладнання на стадіоні, калібрувати камери, забезпечувати надійну передачу даних. Часто такі системи потребують залучення операторів. Наприклад, у ранніх версіях TRACAB потрібно було двоє технічних спеціалістів на стадіоні для контролю роботи системи, хоча зараз багато процесів автоматизовано та перенесено в хмару[7]. Такі рішення, як правило, недоступні для аматорських команд або персонального використання через ціну.

2. Однокамерні системи на основі глибокого навчання (на прикладі FootyVision).

В останні роки, в академічних дослідженнях набуває популярності підхід, що дозволяє отримувати дані зі звичайної трансляції футбольного матчу, тобто з однієї камери, як бачить глядач по ТВ. Система FootyVision пропонує комплексний підхід – використовується одна трансляційна камера, а для аналізу застосовано сучасну модель нейронної мережі для виявлення об’єктів на зовбаженні, спеціально донавчену під футбольні об’єкти а також модуль перспективної трансформації для проекції координат гравців на площину поля. Цей підхід дозволив досягти вражаючих результатів точності детекції - ~95.7% на відкритому датасеті SoccerNet[3], що наближається до рівня багатокамерних системи. Переваги даного підходу – доступність та гнучкість: достатньо мати відеозапис матчу, і система зможе отримати дані без жодного додаткового обладнання. Це дає можливість для аналізу ігор нижчих ліг чи аналізу архівних матчів. Недоліки такого однокамерного підходу полягають у його обмеженнях у покритті і в реальному часі. По-перше, телевізійна трансляція не завжди покриває весь простір поля: якщо об’єкт

знаходиться поза кадром то дані про нього втрачаються. По-друге, глибокі нейронні моделі вимагаються високих обчислювальних ресурсів машини і досягти їх роботи в режимі live важко – система позиціонується як near-real-time, але на практиці для високої якості може знадобитися потужна GPU.

Ще один важливий аспект – генерація статистичного аналізу на основі відстежених даних. Існуючі платформи (наприклад, Second Spectrum, InStat, Wyscout) інтегрують комп'ютерний зір з аналітичними модулями, що обчислюють показники ефективності. Second Spectrum, офіційний трекінг-провайдер NBA не лише відстежує координати гравців, а й рахує складні метрики: інтенсивність пресингу, ймовірності передач і ударів, оцінку внеску гравця в результат і т.п. – ці дані надаються клубам і навіть виводяться як інсайти під час трансляцій матчів[10]. Перевагою таких систем є повний цикл: від сирого відео до змістовних показників та графіки, що дозволяє приймати рішення (заміни, тактичні перебудови) безпосередньо по ходу гри. Недоліком, правда, є те, що алгоритми цих метрик зазвичай закриті і не завжди зрозуміло, як саме вони розраховані. До того ж, повна система вимагає значних обчислювальних ресурсів та ретельної валідації кожної метрики, щоб команди їй довіряли.

Розглянуті рішення демонструють два полюси сучасних підходів: з одного боку – апаратно інтенсивні системи, з іншого – алгоритмічно інтенсивні системи. Багатокамерні системи типу TRACAB є золотим стандартом точності: вони надають максимально повні дані, на основі яких можна будувати будь-яку аналітику, включно з тривимірним відтворенням матчу. Однокамерні DL-рішення прагнуть демократизувати спортивну аналітику – вони дешевші, але поки що трохи поступаються в надійності і вимогливі до якості відео. Втім, прогрес у цій галузі дуже швидкий: за останні 3–5 років точність детекторів і трекерів значно зросла, про що свідчать досягнення на кшталт FootyVision[3]. Статистичний аналіз значною мірою залежить від якості вихідних трекінгових даних: чим вони повніші й точніші, тим складніші й цікавіші метрики можна обчислити. Таким чином, оптимальним

шляхом є комбінування найкращих практик: використання глибоких моделей для детекції/трекінгу, доповнення їх спеціалізованими алгоритмами (наприклад, для розпізнавання номерів чи особливих подій), а також ретельна валідація отриманих даних. У нашому проекті, беручи до уваги ці напрацювання, доцільно реалізувати підхід на основі однієї камери (як найбільш універсальний), використовуючи нейромережеві детектори і існуючі бібліотеки трекінгу.

1.5 Постановка завдання

Метою даної дипломної роботи є розробка програмної системи, яка здатна автоматично виявляти футболістів і м’яч на відеозаписі матчу, відстежувати їх переміщення у часі та надавати статистичні показники їх руху і взаємодії. Для досягнення поставленої мети потрібно вирішити такі задачі:

1. Аналіз предметної області та технологій. Провести огляд сучасних підходів до комп’ютерного зору в спорті, вивчити алгоритми детекції та трекінгу об’єктів, доступні датасети і інструменти. На основі цього обґрунтувати вибір методів та платформ для реалізації системи.
2. Розробка концепції системи та моделі даних. Сформувати логічну архітектуру рішення: визначити компоненти (модуль обробки відео, модуль розпізнавання, модуль трекінгу, база даних результатів, інтерфейс для користувача), розробити модель даних (структури для зберігання статистики, прив’язки до гравців).
3. Реалізація модуля розпізнавання об’єктів. Обрати та налаштувати алгоритм/модель для детекції гравців і м’яча на кадрах відео. В якості такого алгоритму доцільно використати згорткову нейронну мережу типу YOLO[11], які забезпечують швидке і точне виявлення об’єктів. Планується

використати попередньо навчену модель та донавчити її на прикладах футбольних кадрів, щоб підвищити точність для нашого сценарію.

4. Реалізація модуля багатооб’єктного трекінгу. Для реалізації обрано алгоритм трекінгу, що на основі послідовності детекцій формуватиме траєкторії об’єктів ByteTrack[12], який добре зарекомендував себе для відстеження множинних об’єктів. Алгоритм буде асоціювати нові детекції з існуючими треками, оновлювати їх або ініціювати/завершувати треки відповідно до того, як гравці з’являються чи зникають з кадру.
5. Розрахунок статистичних даних. На основі отриманих даних реалізувати обчислення потрібних статистичних показників. Зокрема, для кожного гравця розрахувати: загальну дистанцію, яку він пробіг за матч та середню швидкість; час володіння м’ячем командами у відсотковому співвідношенні (якщо можливо ідентифікувати, хто володіє м’ячем, через близькість гравця до м’яча).

Вибір технологій. Для реалізації поставлених завдань було використано інструментарій, який найкраще зарекомендував себе у сфері комп’ютерного зору. Мовою програмування обрано Python завдяки наявності потужних бібліотек комп’ютерного зору та глибокого навчання. Зокрема, була використана бібліотеку OpenCV для базової обробки відео, а також популярний фреймворк для глибокого навчання нейромережі RoboFlow для виконання аналізу з допомогою алгоритму YOLO. Сам алгоритм YOLO було обрано через його високу швидкість та точність визначення об’єктів[13]. Для трекінгу та передбачення траєкторії руху об’єктів оптимальним є ByteTracker. Для присвонєння та визначення кольорів команди гравця було обрано бібліотеку Scikit-learn, а саме її алгоритм K-Means Clustering[14]. Його суть полягає в тому, що ми виділяємо верхню частину гравця, щоб визначити колір форми його команди.

Для розробки інтерфейсу додатку обрано фреймворк PyQt5 – потужний інструмент для розробки користувацького інтерфейсу, з допомогою якого можна

створити зручний, інтуїтивно зрозумілий та гарний інтерфейс для взаємодії із системою.

Система зберігатиме дані в структурованому вигляді, для цього обрано реляційну базу даних PostgreSQL – це обумовлено зручністю подальшого аналізу. Щодо апаратної платформи, для навчання моделей або їх донавчання може бути використано GPU серверів Google (запуск скриптів навчання нейромережі в Google Collab).

Отже, в межах дипломної роботи було реалізовано повний цикл – від аналізу вимог і вибору методів до створення і тестування прототипу системи комп’ютерного зору, що розпізнає рухомі об’єкти на футбольному полі та визначає їх характеристики. Успішне виконання поставлених завдань дозволить продемонструвати можливості сучасних алгоритмів CV в спортивній аналітиці та стане підґрунтям для подальшого вдосконалення таких систем.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Діаграма класів та кооперацій

На етапі аналізу вимог виокремлено основні сутності нашої системи: Відеопотік, Матч, М’яч, Команда та Гравець. Побудова моделі передбачає уточнення цих сутностей, визначення їх атрибутів, властивостей та зв’язків. Для початку необхідно виділити класи об’єктів та їхні атрибути: клас Відеопотік з атрибутами назви відео та самим відео, клас Матч з атрибутами ідентифікатор матчу, дата проведення матчу, локація та рахунок матчу, клас М’яч із атрибутами колір м’яча, координати м’яча у просторі – X та Y та швидкість м’яча, клас Команда з атрибутами назва та колір команди та клас Гравець з атрибутами номер, команда, роль, швидкість та пройдена дистанція гравця.

Для наочної візуалізації сутностей та їх взаємозв’язків було використано UML-діаграму класів (рис. 4). Діаграма класів є основним компонентом об’єктно-орієнтованого моделювання. Вона використовується як для моделювання на рівні предметної області, так і на рівні проектування, коли моделі перетворюються на програмний код[8]. На діаграмі класи зображуються у вигляді прямокутників із зазначеними іменем класу, його атрибутами та за потреби операціями. Зв’язки між класами (асоціації, агрегації, композиції, успадкування) показують, як об’єкти взаємодіють. Наприклад, Команда містить багато гравців, у Матчі приймають участь 2 команди тощо.

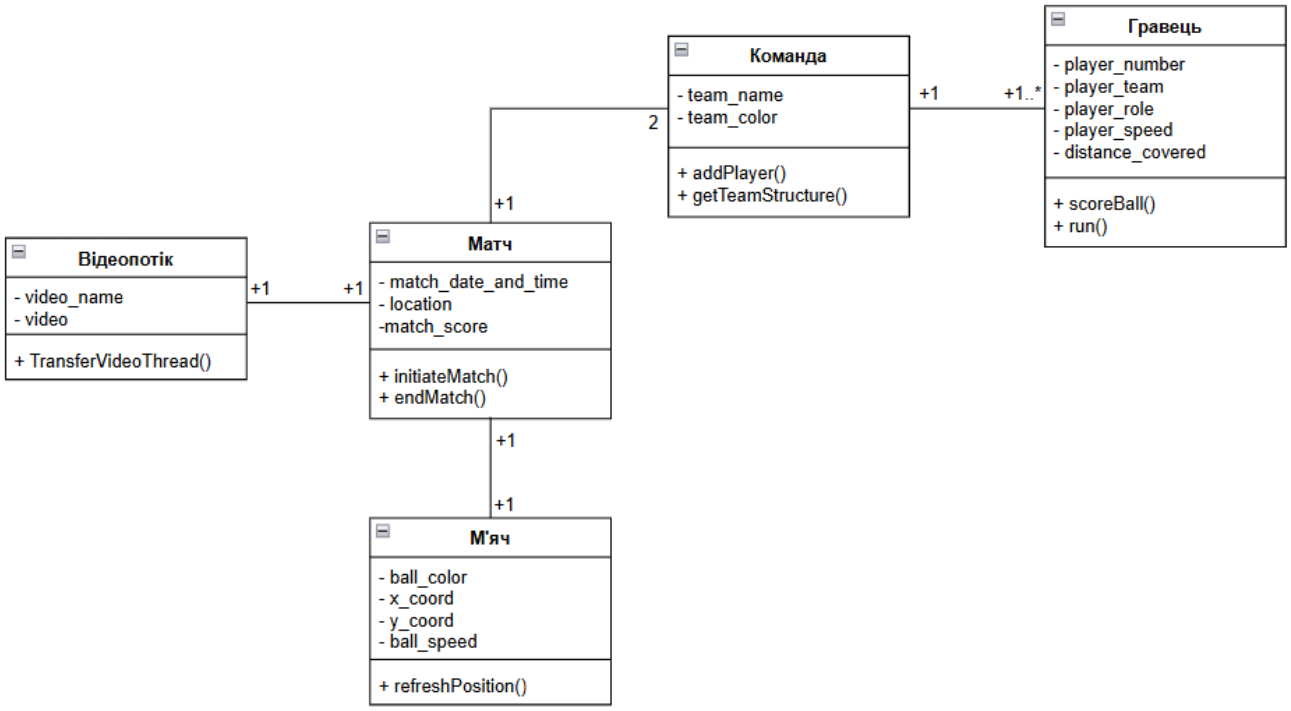

Рис. 4. Діаграма класів предметної області системи

Після побудови загальної діаграми класів було виділено основні кооперації класів – групи класів, що взаємодіють для виконання окремих підзадач системи. Зокрема, одну кооперацію складають класи, що відповідають за формування команди та її учасників, а іншу – класи, що забезпечують передачу відеопотоку для початку аналізу матчу. Нижче описано основні діаграми кооперацій системи.

Кооперація 1 – “Відеопотік - Матч” (рис. 5). Ця кооперація моделює взаємодію між об’єктом Відеопотік та об’єктом Матч. Основна ідея полягає у ініціалізації матчу з використанням конкретного відеофайлу, що представляє собою футбольну гру. У діаграмі відображено агрегаційний зв’язок між класами, де Відеопотік включає Матч як частину своєї структури.

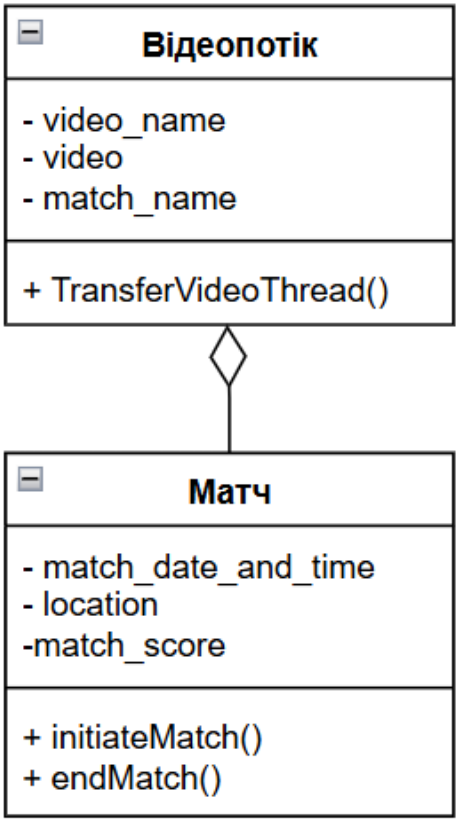

Рис. 5. Діаграма кооперацій “Відеопотік-Матч” системи

Кооперація 2 – “Команда - Гравець” (рис. 6). Ця кооперація демонструє внутрішню організацію команди в контексті об’єктної моделі. Між класами встановлено зв’язок композиція – це означає, що Команда делегує один або декілька об’єктів Гравець, і в той ж час, Гравець не може існувати без об’єкту Команда. Ця кооперація відображає повсякденну ситуацію у футбольному матчі — гравці формують команду, команда управляє своїм складом.

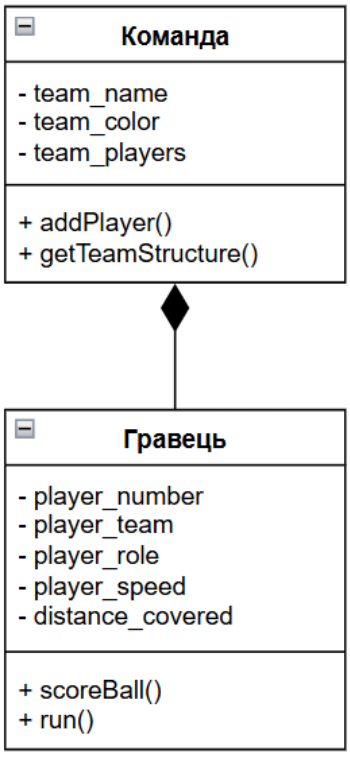

Рис. 6. Діаграма кооперацій “Команда-Гравець” системи

Кооперація 3 – “Матч” (рис. 7). Ця діаграма показує головну кооперацію всієї системи –тут Матч агрегує інші об’єкти – М’яч та Команда. Діаграма демонструє складові частини матчу, які співпрацюють під час гри.

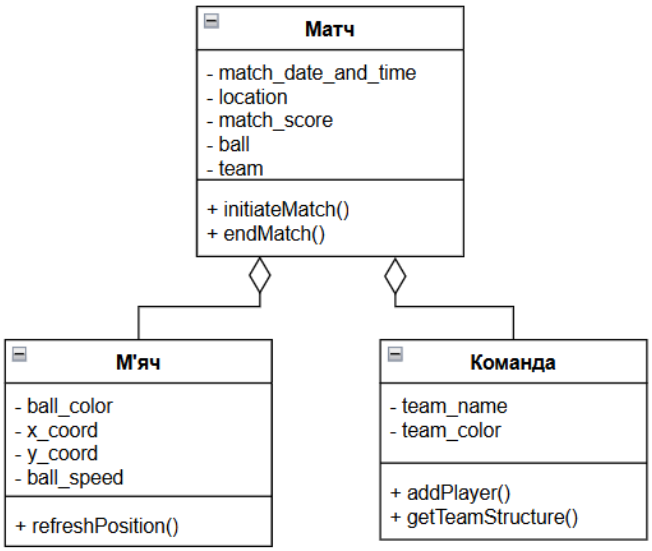

Рис. 7. Діаграма кооперацій “Матч” системи

2.2 Логічна модель даних у вигляді ER-діаграми

Іншим підходом для моделювання предметної області є побудова ER-діаграми (діаграми “сутність-зв’язок”) для моделювання структури бази даних системи (рис. 8). ER-діаграма подібна до діаграми класів, але фокусується на даних: сутності відповідають таблицями в базі даних, їх атрибути – полям, а зв’язки – зовнішнім ключам. Для нашої системи було визначено ряд таких таблиць:

- Match з полями match_ID (ключове поле) та processed_match_video (поле для збереження байтового потоку обробленого відеозапису матчу);
- Team з полями team_ID (ключове поле), match_ID (зовнішній ключ) та team color (поле для збереження кольору команди);
- TeamStats з полями team_ID та match_ID (зовнішні ключі) та полем ball_possesion (для зберігання відсоткового значення володіння м’ячем командою впродовж матчу);
- Player з полями player_ID (ключове поле), team_ID та match_ID (зовнішні ключі) та player_number;
- PlayerStats з полями player_ID, team_ID та match_ID (зовнішні ключі), та поля для зберігання статистики про гравця (distance та avg_speed).

Проектування здійснено відповідно до вимог реляційних баз даних, зокрема 3-ї нормальної форми, щоб уникнути надлишковості даних та забезпечити цілісність інформації. Ключовими сутностями (майбутніми таблицями БД) в моделі є наступні:

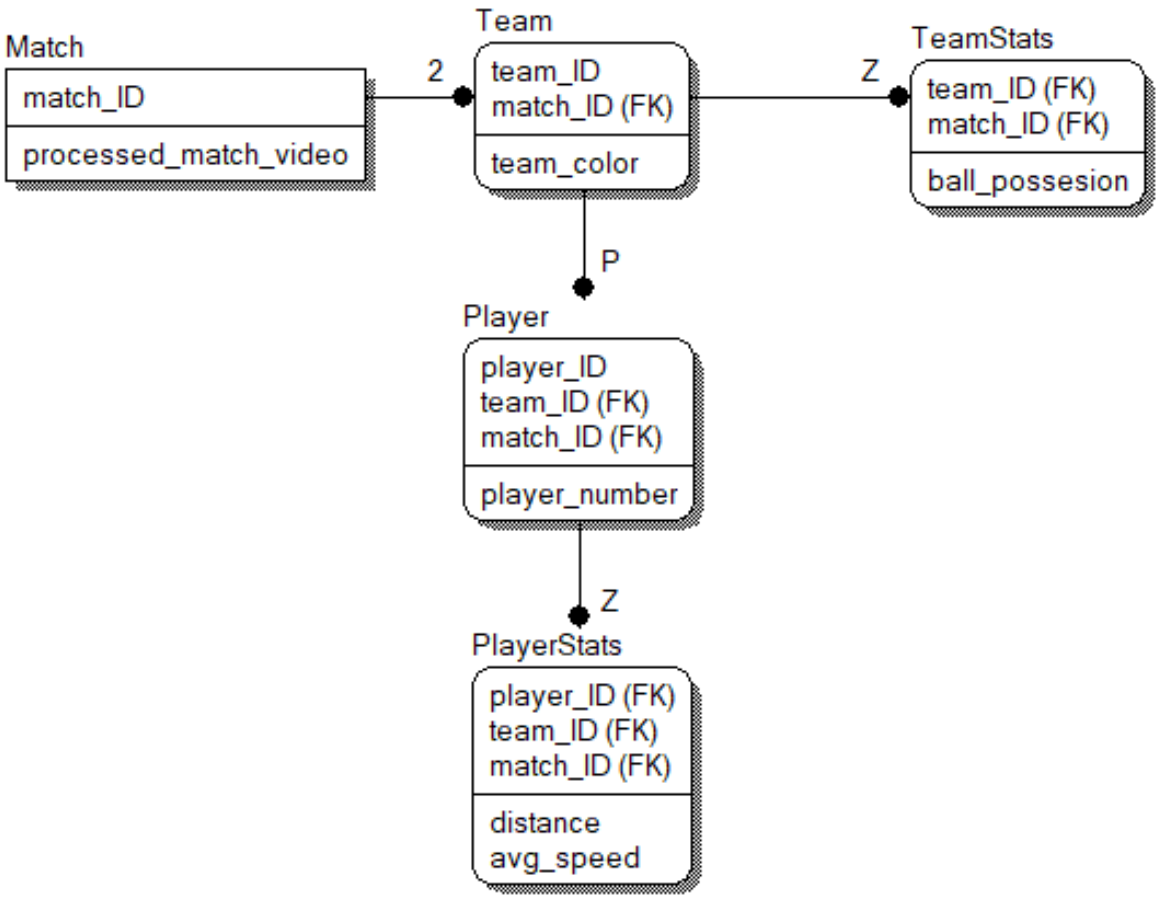

Рис. 8. ER-діаграма системи

2.3 Діаграма пакетів

Для відображення архітектури програмної системи на більш високому рівні абстракції побудовано діаграму пакетів. На ній класи згруповані в логічні модулі, що відповідають функціональним підсистемам. Діаграма пакетів допомагає показати, як система розбита на частини та як ці частини взаємодіють, а також відобразити розміщення кооперацій класів усередині відповідних модулів. Розбиття на пакети було виконано згідно з принципом відокремлення сфер відповідальності – класи, що вирішують схожі завдання або стосуються однієї

підобласті, об’єднані в один пакет. Така модульна структура спрощує навігацію в проекті та потенційно дозволяє повторно використовувати цілі пакети в інших проектах або незалежно модифікувати їх. На рис. 9 зображено діаграму пакетів для власної системи. Вона побудована з урахуванням основних функціональних компонентів: розпізнавання об’єктів, перетворення перспективи та коригування координат об’єктів, обчислення статистичних даних, генерація обробленого відео та візуалізація результатів. Нижче розглянуто кожен із пакетів окремо.

Пакет “Трекінг об’єктів” призначений для виявлення та відстеження об’єктів на кожному кадрі відеопотоку. У контексті поставленої задачі йдеться про відстеження об’єктів матчу (наприклад, гравців та м’яча на спортивному полі) – цей модуль реалізує ядро системи комп’ютерного зору. Він отримує з пакету обробки відео послідовність кадрів і на кожному кадрі здійснює виявлення цікавих об’єктів, визначає їхні координати та ідентифікує об’єкти між кадрами, аби прослідкувати їхні траєкторії у часі. Пакет відстеження надає вихідні дані про траєкторії та стани об’єктів, які споживаються далі за ланцюжком. Важливо, що логіка відстеження ізольована в окремому пакеті, оскільки це спрощує заміну або вдосконалення алгоритмів виявлення/трекінгу без внесення змін у інші частини системи. Наприклад, можна покращити модель детекції (для підвищення точності розпізнавання гравців) або оптимізувати продуктивність трекера, не зачіпаючи при цьому модуль обробки відео чи інтерфейс – достатньо внести зміни лише в пакет “Трекінг об’єктів”. Така модульність підвищує гнучкість системи і полегшує тестування: пакет можна відлагоджувати, подаючи на вхід тестові відеодані, і оцінювати якість трекінгу незалежно від візуалізації чи розрахунку статистик.

Пакет “Перетворення перспективи, коригування координат” забезпечує роботу з геометричними перетвореннями, пов’язаними з характеристиками камери та просторовими координатами. Його основне призначення – встановити зв’язок між координатами об’єктів на зображенні (пікселями кадру) та реальними координатами в просторі реального світу (наприклад, координатами на площині

спортивного поля в метрах). У задачах аналізу спортивного відео це особливо актуально: знаючи параметри камери та перспективну проекцію, можна обчислити, яку відстань на полі відповідає зміщенню об’єкта на певну кількість пікселів у кадрі, або визначити реальну швидкість руху гравця. Пакет може включати в себе модулі калібрування та обчислення матриці гомографії або інших перетворень для приведення координат до площини поля. В сучасних умовах для цього зазвичай користуються засобами OpenCV: бібліотека має готові функції для калібрування та функції для обчислення і застосування проективних перетворень.

Пакет “Обчислення статистичних даних” відповідає за обчисленнях даних необхідних для системи – середня швидкість гравця, його пройдена дистанція та відсоток володіння м’ячем командою. Після обчислення ці дані зберігаються в базі даних, для цього в цьому пакеті існує модуль Бази даних, який надає необхідний функціонал для взаємодії із таблицями.

Пакет “Генерація обробленого відео” слугує за конвертацію байтового потоку у відеофайл, доступний для перегляду користувачеві. Він використовує вбудовані у бібліотеку OpenCV методи, які дозволяють реалізувати це.

Пакет “UI” відповідає за взаємодію системи з користувачем та візуалізацію результатів роботи інших модулів. Він забезпечує графічну оболонку, через яку оператор може завантажувати необхідні файли для старту аналізу. Інтерфейс користувача також відображає вихідні дані системи у зручному вигляді: відеоплеєр, який дозволяє переглядати оброблене відео, та інформаційні блоки, у яких розміщені дані про кожного гравця та матч.

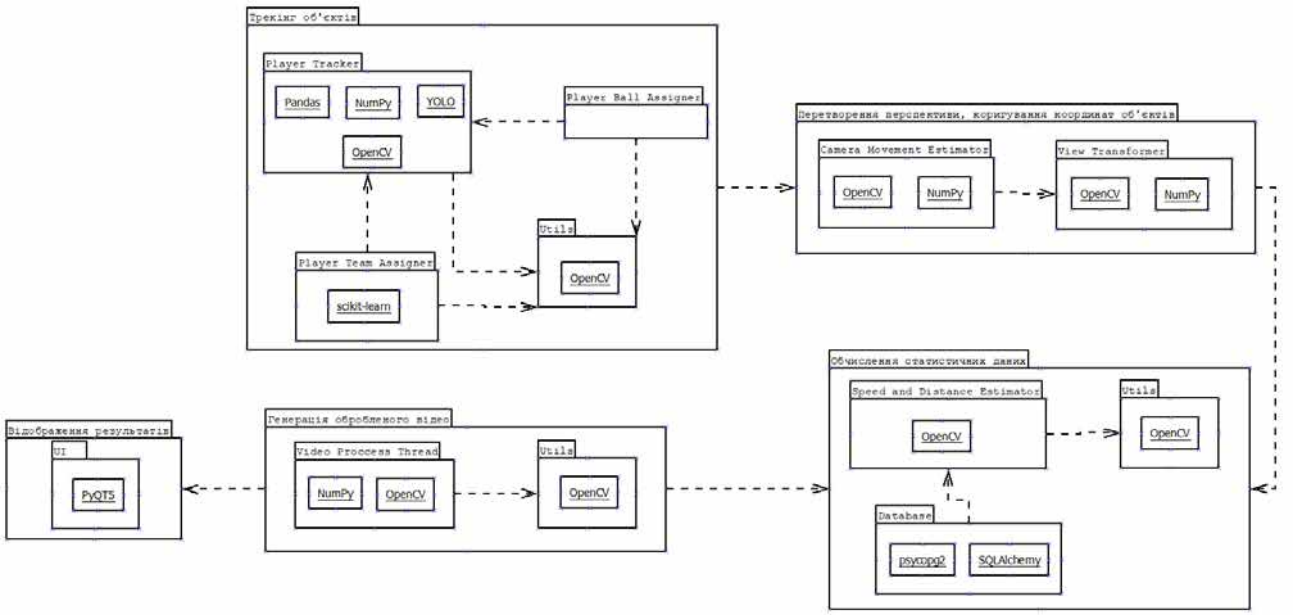

Рис. 9. Діаграма пакетів системи

2.4 Діаграма компонентів

Одним із ключових етапів архітектурного проєктування програмного забезпечення є побудова діаграми компонентів, яка демонструє архітектуру системи у вигляді окремих функціональних блоків, що взаємодіють між собою через інтерфейси. На рис. 10 представлено діаграму компонентів для розробленої системи розпізнавання та визначення характеристик рухомих об’єктів. Вона відображає взаємозв’язки між ключовими модулями, які реалізують інтерфейс користувача, обробку відео, комп’ютерний зір, статистичний аналіз, взаємодію з базою даних, а також використання спеціалізованої моделі глибокого навчання.

Компонент Main.exe є ядром системи. Він виконує роль координатора, через який здійснюється запуск усіх процесів і ініціалізація взаємодії між модулями. Усередині нього міститься два модулі: Window та VideoProcessThread.

Модуль Window відповідає за реалізацію графічного інтерфейсу користувача. Саме тут описані всі елементи GUI: кнопки, списки, меню, індикатори прогресу. Через нього користувач може обирати відеофайл, запускати аналіз, переглядати результати, а також взаємодіяти з іншими частинами системи.

Модуль VideoProcessThread реалізує обробку відео в окремому потоці, що дозволяє забезпечити плавну роботу інтерфейсу навіть під час ресурсоємного аналізу. Саме в цьому класі зосереджена основна логіка системи: запуск аналізу відео, передача кадрів до модуля розпізнавання об’єктів, обчислення статистики, збереження результатів у базі даних та відображення інформації на інтерфейсі.

Інтерфейс video_utils надає базові утиліти для взаємодії з відеофайлами: метод read_video() використовується для зчитування вхідного відео, а save_video() — для збереження обробленого матеріалу після завершення аналізу. Його функції активно використовуються у класі VideoProcessThread.

Інтерфейс bbox_utils містить набір методів, що забезпечують взаємодію з обмежувальними рамками (bounding boxes) об’єктів, які є результатами роботи алгоритмів розпізнавання. Зокрема, тут реалізовано обчислення центра рамки, ширини рамки, відстані до інших, а також визначення положення ніг об’єкта, що особливо актуально у спортивному аналізі. Ці функції активно використовуються в модулях оцінки руху, прив’язки м’яча до гравця, вимірювання швидкості та інших.

Компонент Tracker реалізує механізм розпізнавання об’єктів на кожному кадрі відео. Він використовує модель глибокого навчання, що зберігається у вигляді файлу ai_model.pt. Дана модель спеціально натренована для задачі спортивного аналізу, зокрема, розпізнавання гравців та м’яча на футбольному полі. Tracker є базою для формування координат, які згодом використовуються іншими модулями.

Компонент TeamAssigner відповідає за автоматичне віднесення гравців до команди на основі кольору форми. Це досягається шляхом кластеризації колірних ознак, отриманих із зображення, що дозволяє ідентифікувати командну приналежність кожного гравця без необхідності ручного маркування.

Модуль PlayerBallAssigner визначає, який саме гравець володіє м’ячем у поточний момент. Він отримує координати гравців та м’яча, обчислює відстані між ними та встановлює зв’язок на основі мінімальної дистанції. Для точних розрахунків активно використовуються методи з bbox_utils.

Компонент CameraMovementEstimator використовується для компенсації руху камери під час матчу. Якщо камера здійснює панорамування або змінює кут огляду, це може вплинути на точність вимірювання координат. Даний модуль аналізує зміщення зображення між кадрами та дозволяє стабілізувати обчислення.

ViewTransformer перетворює координати об’єктів з піксельного простору у метричну систему координат футбольного поля. Це дає змогу отримувати реальні показники швидкості, відстані, зони перебування гравця тощо. Він працює у зв’язці з CameraMovementEstimator, щоб забезпечити високу точність геометричних розрахунків.

Модуль SpeedAndDistanceEstimator відповідає за обчислення пройденої дистанції гравця та його швидкості руху. Для цього використовується поточне положення гравця, а також обчислені зміни координат за певний проміжок часу. Ці показники надалі передаються до бази даних для накопичення статистики.

Дана архітектура компонентів системи демонструє чітке розмежування функціональності між незалежними модулями, що забезпечує модульність, розширюваність та зручність у супроводі системи.

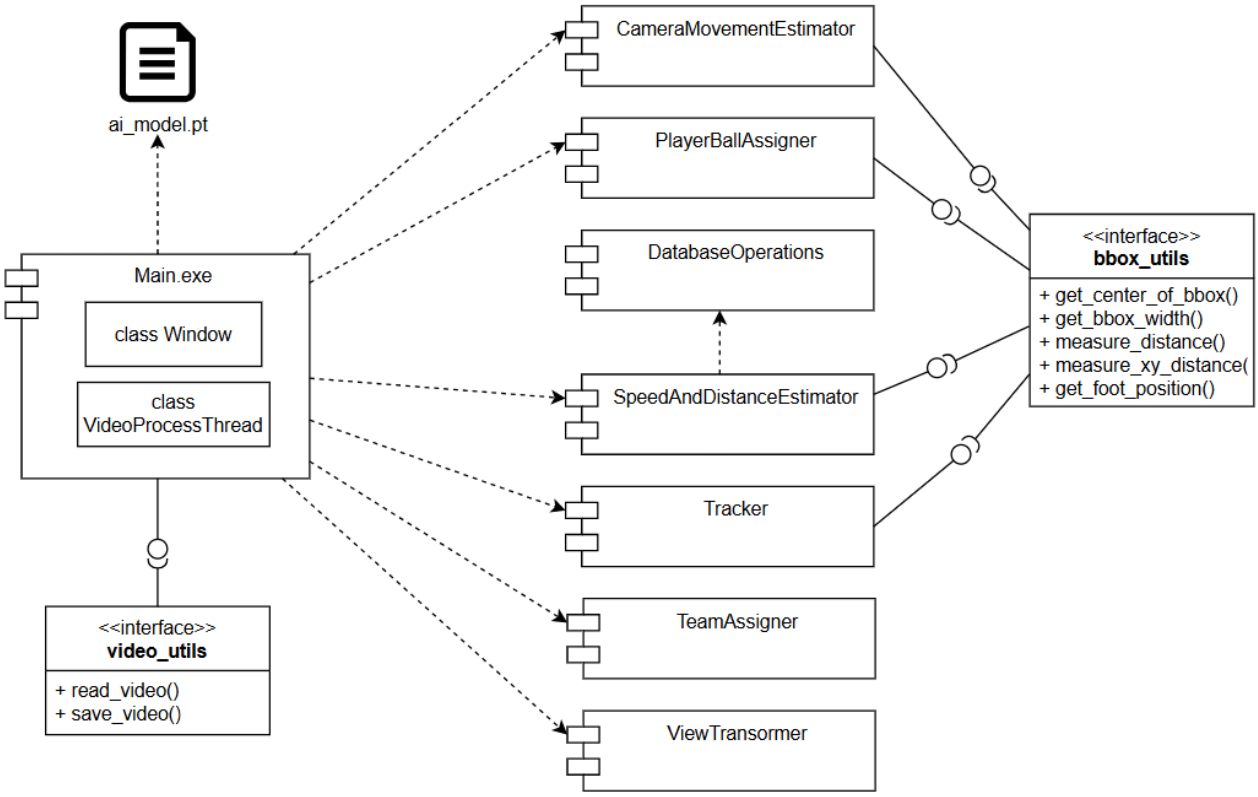

Рис. 10. Діаграма компонентів

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління інформаційною базою

Для зберігання результатів аналізу футбольного матчу було обрано систему управління базами даних PostgreSQL – сучасну об’єктно-реляційну СУБД. PostgreSQL є програмним забезпеченням з відкритим кодом, яке поєднує реляційну модель даних з підтримкою об’єктних розширень. Такий тип СУБД забезпечує гнучкість у роботі з табличними даними, дозволяючи, за потреби, визначати власні типи та методи і працювати з ними у запитах. PostgreSQL відома своєю надійністю та суворим дотриманням принципів ACID. Це досягається завдяки багатоверсійній системі керування паралелізмом MVCC, що гарантує цілісність даних при одночасному зверненні кількох транзакцій. СУБД підтримує складні зв’язки між таблицями (зовнішні ключі, каскадні операції, тригери), які необхідні для забезпечення референційної цілісності даних у системі. PostgreSQL легко масштабувати для роботи з великими обсягами інформації, вона підтримує реплікацію та балансування навантаження, що є важливим на випадок розширення системи. Не менш важливою перевагою є гнучкість та розширюваність: PostgreSQL дозволяє розробникам створювати власні функції (у тому числі мовою Python), що полегшує інтеграцію з прикладним програмним забезпеченням. Перераховані можливості стали вирішальними при виборі PostgreSQL як основи інформаційної бази проєкту. Для управління даними використовується мова структурованих запитів SQL – стандартний засіб взаємодії з реляційними СУБД. У межах проєкту SQL застосовано як для створення структури бази (DDL), так і для маніпулювання даними (DML). Зокрема, у файлі database_utils.py (див. ДОДАТОК А) сформовано SQL-скрипти створення необхідних таблиць із визначенням ключів та зв’язків між

ними. Також за допомогою SQL реалізовано операції додавання записів – результати роботи модулів аналізу заносяться до БД шляхом виконання відповідних INSERT-запитів. Підключення до PostgreSQL здійснюється через бібліотеку psycopg2, що забезпечує виконання SQL-запитів з коду Python. Таким чином, вибір PostgreSQL та використання мови SQL обґрунтовується вимогами до надійного зберігання та структурованої обробки даних в системі. На основі обраної СУБД було спроєктовано структуру інформаційної бази.

3.2 Розробка інформаційної бази

Розроблена інформаційна база даних містить п’ять взаємопов’язаних таблиць, що відображають основні сутності предметної області: матч, команда, гравець, статистика команди, статистика гравця. Логічна структура бази даних визначає зв’язки між цими сутностями, а фізична структура – реалізацію у вигляді таблиць PostgreSQL з відповідними полями і ключами.

1. match_info – таблиця матчів. Містить загальну інформацію про матч: первинний ключ match_id (унікальний ідентифікатор матчу) і поле для зберігання шляху до обробленого відео або іншого ідентифікатора вхідних даних. Таблиця матчу є батьківською для таблиць команд і статистичних даних, тому match_id виступає зовнішнім ключем в інших таблицях. Нижче зображено SQL-запит для створення цієї таблиці.

```
CREATE TABLE IF NOT EXISTS match_info (
    match_id SERIAL PRIMARY KEY,
    match_video_name VARCHAR(50),
    processed_match_video BYTEA
);
```

2. team – таблиця команд, що беруть участь у матчі. Первинний ключ композиційний, складається з team_id та match_id. Поле team_id ідентифікує

команду в межах конкретного матчу, а зовнішній ключ match_id посилається на запис у таблиці матчів, встановлюючи зв’язок «матч–команди». Неключовий атрибут color зберігає колір форми команди, визначений під час аналізу (це дозволяє розрізняти команди за кольором екіпірування). Нижче зображено SQL-запит для створення цієї таблиці.

```
CREATE TABLE IF NOT EXISTS team (
    team_id SERIAL PRIMARY KEY,
    match_id INT NOT NULL,
    team_color VARCHAR(50),
    CONSTRAINT fk_team_match
        FOREIGN KEY (match_id)
        REFERENCES match_info(match_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

3. player – таблиця гравців. Містить інформацію про кожного виявленого гравця у матчі. Первинний ключ – складений з полів player_id, team_id та match_id. Таким чином, кожен гравець однозначно ідентифікується номером гравця в команді та прив’язкою до конкретної команди і матчу. Поле player_id відповідає умовному внутрішньому індексу, team_id і match_id є зовнішніми ключами на таблиці команд та матчів відповідно. Неключовий атрибут number зберігає ігровий номер гравця (якщо доступний). Нижче зображено SQL-запит для створення цієї таблиці.

```
CREATE TABLE IF NOT EXISTS player (
    player_id SERIAL PRIMARY KEY,
    team_id INT NOT NULL,
    match_id INT NOT NULL,
    player_number INT,
    CONSTRAINT fk_player_team
        FOREIGN KEY (team_id)
        REFERENCES team(team_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    CONSTRAINT fk_player_match
        FOREIGN KEY (match_id)
        REFERENCES match_info(match_id)
        ON DELETE CASCADE
```

```
        ON UPDATE CASCADE
);
```

4. teamstats – таблиця статистики команд. Містить по одному запису для кожної команди в матчі. Ключем є складена комбінація team_id + match_id, які одночасно є і зовнішніми ключами на таблицю team. Основний атрибут – possession (відсоток володіння м’ячем), що показує, яку частку часу матчу володіла м’ячем дана команда. Зв’язок між таблицями team і teamstats – 1:1 (кожна команда має рівно один запис статистики). Нижче зображено SQL-запит для створення цієї таблиці.

```
CREATE TABLE IF NOT EXISTS teamstats (
    team_id INT NOT NULL,
    match_id INT NOT NULL,
    ball_possession INT,
    PRIMARY KEY (team_id, match_id),
    CONSTRAINT fk_teamstats_team
        FOREIGN KEY (team_id)
        REFERENCES team(team_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    CONSTRAINT fk_teamstats_match
        FOREIGN KEY (match_id)
        REFERENCES match_info(match_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

5. playerstats – таблиця статистики гравців. Містить показники для кожного гравця у межах матчу. Первинний ключ складений з трьох полів: player_id, team_id, match_id (одночасно ці поля є зовнішніми ключами, що посилаються на відповідного гравця з таблиці player). Основні атрибути: distance – загальна дистанція, яку пробіг гравець за матч, та avg_speed – його середня швидкість. Зв’язок player–playerstats є 1:1 (кожному гравцю відповідає один запис його статистики). Нижче зображено SQL-запит для створення цієї таблиці.

```
CREATE TABLE IF NOT EXISTS playerstats (
    player_id INT NOT NULL,
    team_id INT NOT NULL,
    match_id INT NOT NULL,
    distance FLOAT,
```

```
    avg_speed FLOAT,
    PRIMARY KEY (player_id, team_id, match_id),
    CONSTRAINT fk_playerstats_player
        FOREIGN KEY (player_id)
        REFERENCES player(player_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    CONSTRAINT fk_playerstats_team
        FOREIGN KEY (team_id)
        REFERENCES team(team_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    CONSTRAINT fk_playerstats_match
        FOREIGN KEY (match_id)
        REFERENCES match_info(match_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

Отримавши в результаті відпрацювання SQL-запитів нашу базу даних, нам необхідно записувати та витягувати дані з неї. Після завершення аналізу матчу програмні модулі вставляють отримані дані в БД за допомогою SQL. Приклад SQL-запиту вставки даних: припустимо, розраховано дистанцію та швидкість для гравця з ID 15 у команді 1 (матч №1). Для збереження цього результату використовується команда:

```
INSERT INTO playerstats (match_id, team_id, player_id, distance, avg_speed)
VALUES (1, 1, 15, 1120.0, 6.8);
```

Такий запит додає до таблиці playerstats новий запис із ключем (1,1,15) та показниками: пройдена дистанція 1120 метрів і середня швидкість 6.8 м/с. Наведений приклад відповідає фрагменту коду модуля роботи з базою даних (див. ДОДАТОК А). Аналогічним чином формуються й інші SQL-запити для внесення статистики команд (володіння м’ячем) та загальної інформації про матч і команди. Отже, інформаційна база даних спроектована і реалізована таким чином, щоб ефективно зберігати усі необхідні дані для подальшого аналізу та відображення.

3.3 Вибір інструментарію для створення прикладного ПЗ

Розробка прикладного забезпечення системи виконана мовою програмування Python з використанням низки спеціалізованих бібліотек. Вибір інструментарію обґрунтовано вимогами до функціоналу системи (комп'ютерний зір, обробка відео в реальному часі, зручний інтерфейс користувача, зберігання даних тощо) та доступністю перевірених рішень для цих завдань. Коротко розглянемо основні засоби, використані у проєкті:

Python 3.11 – інтерпретована мова високого рівня, обрана як основна мова розробки. Переваги Python для даного проєкту: велика кількість готових бібліотек для комп'ютерного зору та машинного навчання, простий синтаксис, що прискорює розробку, і крос-платформеність. Мова Python дозволила легко інтегрувати між собою різні компоненти системи, реалізувати математичні обчислення та роботу з базою даних у стислі терміни.

OpenCV – відкрита бібліотека комп'ютерного зору, використана для обробки відео та зображень. OpenCV надає широкий набір функцій, необхідних для нашого проекту: засоби для зчитування та декодування відеопотоку, попіксельної обробки кадрів, виділення контурів та ключових точок, обчислення оптичного потоку, перетворення перспективи тощо. Завдяки OpenCV реалізовано алгоритми стабілізації зображення та вимірювання відстаней (див. підрозділи 3.4.2 і 3.4.3). Бібліотека добре сумісна з Python через модуль cv2 і оптимізована для швидкої роботи з відеоданими.

Ultralytics YOLO – модель глибокого навчання для детекції об'єктів (нейронна мережа сімейства YOLO). Версія YOLOv5 була обрана як одна з найсучасніших на момент розробки, що забезпечує високу точність розпізнавання об'єктів при збереженні швидкодії реального часу. Архітектура YOLO (You Only Look Once) виконує пошук всіх цільових об'єктів за один прогін нейронної мережі по зображенню, завдяки чому придатна для аналізу відеопотоку на рівні 20–30 кадрів за секунду. У проєкті модель YOLOv5 використовується для виявлення футболістів, м'яча та інших об'єктів на полі в кожному кадрі (детально процес

описано у підрозділі 3.4.1). Модель була попередньо навчена на великому наборі даних з футбольними сценами (використано датасет FootballPlayersDetection та інші, що містять сотні зображень різних матчів), що дозволило адаптувати її до умов спортивного відео. Застосування YOLOv5 значно спрощує вирішення задачі комп'ютерного зору, оскільки більшість складної логіки розпізнавання реалізована всередині архітектури моделі.

Scikit-learn (алгоритм KMeans) – бібліотека Python для машинного навчання, з якої використано алгоритм кластеризації K-Means. Цей алгоритм застосовано для автоматичної класифікації гравців за командами на основі кольору форми (див. підрозділ 3.4.6). Перевага KMeans полягає в тому, що він без вказівки з боку розробника розбиває вибірку на k кластерів – у нашому випадку k=2 (перший – це сама форма гравця, другий – задній фон або ж інші непотрібні деталі). Бібліотека scikit-learn надає ефективну реалізацію KMeans, що легко інтегрується з даними, отриманими з OpenCV (колірні ознаки пікселів).

PyQt5 – бібліотека для створення графічного інтерфейсу користувача (GUI) в Python. PyQt5 є обгорткою для популярного фреймворку Qt, що забезпечує крос-платформений інтерфейс та широкий набір графічних елементів. У даному проєкті PyQt5 використано для розробки зручного інтерфейсу, який дозволяє користувачу взаємодіяти із системою: завантажувати відеофайли, запускати аналіз, переглядати результати тощо. Візуальна частина реалізована у модулі main.py (див. ДОДАТОК Б). Інтерфейс містить вікно відтворення відео з накладеними на нього графічними анотаціями (рамки навколо гравців і м'яча, текстові позначки швидкості тощо), а також панелі або поля для відображення зведеної статистики матчу. Завдяки PyQt5 забезпечено інтерактивність: під час обробки відео користувач бачить прогрес, а після завершення – може переглянути результати у зручному вигляді.

Psycopg2 – як було зазначено, для зберігання результатів використовується СУБД PostgreSQL. Всі операції з базою даних із боку програми реалізовані через драйвер psycopg2, який інтегрує PostgreSQL з Python-кодом. Це дозволило, після

завершення обчислень, автоматично зберігати отримані дані (відео, статистику) у базі для подальшого доступу. Таким чином, поєднання PostgreSQL та psycopg2 відповідає за надійне довготривале зберігання інформації поза межами сеансу роботи застосунку.

Вибір перелічених інструментів підтверджено їхньою популярністю та ефективністю у відповідних задачах. Поєднання Python з бібліотеками OpenCV та PyQt5 дало змогу побудувати застосунок, що інтегрує алгоритми комп'ютерного зору з зручним GUI, а використання YOLOv5 та KMeans забезпечило реалізацію складних інтелектуальних методів (детекція, класифікація) без потреби розробляти їх «з нуля». У результаті вдалося зосередитися на логіці роботи системи, використовуючи готові оптимізовані рішення.

3.4 Алгоритмізація та програмування програмних модулів

Основні функції системи аналізу футбольного відео реалізовано у вигляді окремих модулів, кожен з яких відповідає за виконання певного завдання. Нижче представлено ключові модулі та описано алгоритми їх роботи, використані структури даних і методи. Також наведено важливі фрагменти реалізації (повні лістинги коду – у ДОДАТКАХ В–З).

3.4.1 Модуль Tracker – розпізнавання об'єктів на основі YOLO

Модуль Tracker відповідає за автоматичне виявлення рухомих об'єктів на полі – передусім гравців та м'яча – на кожному кадрі відео. У його основі лежить нейромережева модель YOLO, за допомогою якої здійснюється розпізнавання об'єктів. Під час опрацювання кожного чергового кадру відео модель YOLO виконує прогнозування обмежувальних рамок (bounding boxes), в яких присутні цільові об'єкти, та визначає їх належність до певного класу (наприклад, гравець, м'яч, арбітр). Алгоритм YOLO використовує поділ зображення на сітку і оцінює

для кожної комірки ймовірності присутності об’єктів різних класів, а також координати рамок об’єктів. Завдяки цьому підходу система отримує множину прямокутних областей з високою ймовірністю містити потрібні об’єкти. Після отримання початкових результатів детекції проводиться їх додаткова обробка для підвищення точності. По-перше, застосовується алгоритм NMS: якщо модель видала кілька рамок для одного об’єкта (що трапляється при близькому розташуванні гравців чи швидкому русі), NMS залишає лише одну – з найвищим коефіцієнтом впевненості, а інші відкидає. Це запобігає дублюванню та “миготінню” рамок навколо одного й того ж гравця. По-друге, модуль здійснює відстеження об’єктів між кадрами (трекінг). Для цього результати детекції на поточному кадрі порівнюються з об’єктами, знайденими на попередньому кадрі: обчислюється коефіцієнт перекриття або відстань між рамками, і якщо нова рамка знаходиться поблизу попередньої позиції об’єкта, їй призначається той самий ідентифікатор track_id. Якщо ж з’являється рамка у новому місці, що не відповідає жодному з раніше бачених об’єктів, модуль створює для неї новий трек з новим унікальним ID. Таким чином, Tracker формує безперервні траєкторії руху для кожного виявленого гравця та м’яча. Кожна така траєкторія зберігає актуальні координати об’єкта, його попередні координати, унікальний ідентифікатор та інші необхідні атрибути (наприклад, історію швидкостей). Логіка відстеження реалізована за допомогою структури даних на кшталт списку активних треків, що оновлюється на кожному кадрі. Ключові атрибути та методи модуля Tracker включають: model (завантажена модель YOLO), метод detect_frames, що повертає список передбачених рамок та класів об’єктів, метод для збереження поточних треків get_object_tracks. Реалізація модуля Tracker наведена у ДОДАТОК В. Завдяки цьому модулю система отримує на кожному кроці часу актуальні координати всіх гравців і м’яча, що надалі використовуються іншими компонентами аналізу.

3.4.2 Модуль CameraMovementEstimator – компенсація руху камери

Під час зйомки футбольного матчу камера зазвичай рухається (у нашому випадку приймаються рухи камери по горизонталі, без туму і тд.), що призводить до зміщення всіх об’єктів на зображенні навіть при їх відносній нерухомості на полі. Для коректного обчислення траєкторій і швидкостей гравців необхідно відокремити власне рух об’єктів від руху камери. Цю задачу виконує модуль CameraMovementEstimator, який оцінює глобальне зміщення зображення між послідовними кадрами і компенсує його. Основний алгоритм, застосований у модулі, - обчислення оптичного потоку Лукас-Канаде. Оптичний потік являє собою поле векторів, що описують, на скільки і куди змістився кожен піксель (або ключова точка зображення) при переході від одного кадру до наступного. Метод Лукаса-Канаде обчислює ці вектори для вибраних характерних точок зображення (наприклад, кутів або текстурованих областей) на основі розв’язання систем рівнянь інтенсивності для невеликого околу кожної точки. У реалізації використано функцію cv2.calcOpticalFlowPyrLK з бібліотеки OpenCV, яка повертає набір векторів зміщення. Щоб зосередитися саме на русі камери, модуль намагається відфільтрувати рух об’єктів: для цього аналізуються переважно точки фону (наприклад, лінії розмітки поля, трибуни) або ж обчислений оптичний потік усереднюється по всьому кадру. В результаті отримується приблизний вектор зсуву камери між двома кадрами, наприклад $\Delta x = +5$ пікселів, $\Delta y = +2$ пікселі (означає, що новий кадр зміщений праворуч і трохи вгору відносно попереднього). Отримані значення глобального зсуву застосовуються для коригування координат об’єктів. На практиці, перед оновленням треків (у модулі Tracker), до кожної координати гравця та м’яча додається зворотний вектор зміщення камери. Наприклад, якщо камера посунулася праворуч на 5 px, то всі координати об’єктів зміщуються вліво на 5 px, ніби ми «вирівнюємо» кадр

до спільної системи відліку. Такий підхід нівелює вплив руху камери на траєкторії об’єктів. У результаті траєкторії відображають реальні переміщення гравців по полю. Ключові етапи роботи модуля CameraMovementEstimator: вибір опорних точок на кадрі (використовується cv2.goodFeaturesToTrack для визначення куточків зображення), обчислення оптичного потоку для цих точок на наступному кадрі (calcOpticalFlowPyrLK), розрахунок середнього або медіанного вектора зміщення camera_movement та передача цього вектора у модуль трекінгу для компенсації. Реалізація модуля CameraMovementEstimator наведена у ДОДАТОК Д. Застосування цього алгоритму підвищує точність подальших вимірювань, особливо коли камера рухається по горизонталі під час гри.

3.4.3 Модуль ViewTransformer – перетворення перспективи

Для обчислення реальних величин (дистанцій, швидкостей) необхідно переводити координати об’єктів з площини зображення (пікселів) у площину реального футбольного поля (метрів). Модуль ViewTransformer виконує перспективне перетворення координат, прив’язуючи їх до моделі поля. Іншими словами, цей модуль «вирівнює» вид з камери до ортогональної проекції на площину поля. Задача зводиться до пошуку відповідного проектування між двома системами координат: системою координат відеокадру (x,y в пікселях) та системою координат поля (X,Y в умовних метрах або відносних одиницях поля). Перетворення перспективи математично описується матрицею гомографії 3×3. Щоб її визначити, зазвичай потрібно знати координати кількох контрольних точок як на зображенні, так і на полі. В нашому випадку такими точками можуть бути, наприклад, кути футбольного поля або точки на лініях розмітки (центр поля, кути штрафного майданчика тощо). Координати цих точок у відеокадрі отримують автоматично (наприклад, шляхом виявлення розмітки), а їх реальні координати відомі з геометрії поля (стандартні розміри поля). На основі щонайменше чотирьох пар відповідних точок функція отримуємо матрицю проекції H. Після отримання

матриці перетворення модуль ViewTransformer застосовує її до координат кожного об’єкта, використовуючи функцію cv2.perspectiveTransform. Таким чином, для кожного гравця (та м’яча) визначаються координати (X,Y) на полі – наприклад, у метрах відносно одного з кутів поля. Це дозволяє надалі оперувати реальними відстанями. Зокрема, можна точно виміряти, яку відстань пробіг гравець, або яка відстань між гравцем і м’ячем у будь-який момент. Перетворення перспективи також дає змогу врахувати спотворення від ракурсів камери: наприклад, якщо гравець рухається по горизонталі з точки зору камери, на полі це може бути діагональний рух – гомографія це врахує. У нашій системі перспективне перетворення виконується для кожної точки траєкторії гравців після компенсації руху камери. На виході модуль ViewTransformer формує набір координат у системі поля. Важливо зазначити, що правильність цього перетворення значною мірою залежить від точності визначення контрольних точок та стабільності камери (перетворення актуальне, поки не зміниться кут огляду камери). Реалізація модуля ViewTransformer наведена у ДОДАТОК Е. За допомогою OpenCV вдалося досягти коректного зіставлення координат зображення із реальними метричними координатами поля, що створило основу для точного обчислення фізичних показників гравців.

3.4.4 Модуль SpeedAndDistanceEstimator – обчислення швидкості та дистанції

Модуль SpeedAndDistanceEstimator використовує результати, отримані попередніми модулями (треки гравців з урахуванням руху камери і переведені у координати поля), для розрахунку статистичних показників кожного гравця: пройдена дистанція та середня швидкість. Алгоритм роботи модуля по суті є аналізом траєкторії руху кожного гравця. Для обчислення дистанції модуль послідовно сумує відстані між точками траєкторії гравця. Траєкторія представлена дискретною послідовністю координат на полі для кожного кадру. Відстань між двома послідовними положеннями гравця обчислюється за формулою евклідової

відстані. Потім всі ці малі відрізки додаються, отримана сума і є загальною дистанцією, яку пробіг гравець за матч. Важливо, що координати на полі в метрах, тож і результуюча дистанція буде виражена в метрах. Для підвищення точності можуть застосовуватися фільтри: наприклад, ігнорувати дуже малі зміщення, що можуть бути шумом від детекції, або згладжувати траєкторію перед обчисленням. Для визначення швидкості використовується час, за який гравець проходить відповідні відстані. У контексті нашої системи, для спрощення, середня швидкість може розраховуватися як відношення загальної дистанції до тривалості аналізованого відеофрагменту. Отримане значення конвертується в обрану одиницю. Програмно модуль SpeedAndDistanceEstimator реалізований як функції, що обробляють масив координат для кожного треку гравця. Після обчислень модуль передає результати до компонента збереження даних, де формуються SQL-запити для вставки у таблицю playerstats (наприклад, як показано у підрозділі 3.2). Реалізація модуля SpeedAndDistanceEstimator наведена у ДОДАТОК Ж. На основі цих розрахунків система надає тренерам та аналітикам корисні кількісні показники ефективності гравців.

3.4.5 Модуль PlayerBallAssigner – визначення володіння м’ячем

Модуль PlayerBallAssigner визначає, який гравець володіє м’ячем у той чи інший момент часу, і на основі цього розраховує загальний відсоток володіння м’ячем для кожної з команд. Алгоритм базується на аналізі відстані між м’ячем та гравцями на полі на кожному кадрі. Вхідні дані для модуля – це координати м’яча та координати всіх гравців в кожний момент часу t. На кожному кроці модуль обчислює відстань від м’яча до кожного гравця. Якщо знайдено гравця, що знаходиться ближче за інших до м’яча і ця відстань менша за певний поріг (наприклад, радіус 1 м), вважається, що саме цей гравець володіє м’ячем у даний момент. Його ідентифікатор і команда фіксуються як поточний власник м’яча. Якщо ж м’яч ні до кого не наближений (наприклад, в польоті або вільно котиться), можна вважати, що наразі жодна з команд не володіє м’ячем. У нашій системі для

спрощення приймаємо, що володіння миттєво переходить лише при безпосередньому зближенні м’яча з гравцем іншої команди. Модуль накопичує часові інтервали володіння для кожної команди.

Отримані значення (наприклад, 55% vs 45%) відображають, яка частка часу м’яч була під контролем першої та другої команди відповідно. Ці показники заносяться до таблиці teamstats у поле possession. В процесі визначення володіння м’ячем можуть виникати короткі періоди невизначеності або швидкої зміни власника (наприклад, під час перехоплення). Щоб уникнути осциляцій, модуль може вводити невелику затримку або вимагати, щоб гравець перебував близько до м’яча принаймні n послідовних кадрів, перш ніж зафіксувати зміну володіння. В нашому проєкті припускаємо, що частота кадрів досить висока, тому один кадр контакту можна вважати достатнім. Реалізація модуля PlayerBallAssigner наведена у ДОДАТОК И. У результаті роботи цього модуля система отримує дані про володіння м’ячем, які є важливим командним показником і можуть бути проаналізовані тренерським штабом.

3.4.6 Модуль TeamAssigner – класифікація гравців за командами

Модуль TeamAssigner автоматично визначає, до якої команди належить кожен виявлений гравець, на основі кольору його форми. Це дозволяє системі розрізняти гравців двох команд без ручного введення інформації про колір форми чи номери гравців. Задача вирішується методом кластеризації кольорів за допомогою алгоритму K-Means. На вході модуль отримує фрагмент із зображенням гравця (рис. 13). З цього фрагменту виділяється колірна інформація – переважний колір футболки. Зокрема, можна взяти верхню частину рамки гравця (рис. 14) (де зазвичай видно футболку). Далі алгоритм K-Means розбиває зображення на 2 кластери – із формою гравця та іншими деталями відповідно (рис. 15). Після цього береться середнє значення кольору у відповідному кластері і зберігаєтсья у форматі RGB. Реалізація використовує функцію KMeans з бібліотеки scikit-learn, де

необхідно задати кількість кластерів і початкові умови. Після виконання кластеризації кожному гравцеві (його кольоровому вектору) присвоюється мітка кластера – 0 або 1. Ці мітки безпосередньо відповідають двом командам. Наприклад, якщо кластер 0 характеризується кластером без необхідної інформації про колір команди, то кластер 1 відповідає саме за той кластер, який нам необхідний. Надалі ця інформація використовується іншими модулями: зокрема, PlayerBallAssigner при підрахунку володіння оперує саме командами гравців, а SpeedAndDistanceEstimator може групувати статистику по командах. Варто зауважити, що перед кластеризацією кольорів доцільно виконати певну нормалізацію: відфільтрувати фон, врахувати освітлення. У нашому випадку використано простий підхід – усереднення кольору всередині області гравця автоматично згладжує дрібні деталі і дає основний тон форми. Алгоритм KMeans достатньо стійкий, щоб розділити навіть подібні відтінки, якщо вони утворюють дві щільні групи. Реалізація модуля TeamAssigner наведена у ДОДАТОК К. У підсумку система автоматично визначає командну приналежність футболістів, що дозволяє коректно зіставляти статистику (наприклад, дистанції) з конкретними командами, а також відображати інформацію в інтерфейсі (накладати різнокольорові рамки для різних команд тощо).

Усі розглянуті програмні модулі інтегруються в загальну систему через головний сценарій (файл main.py, ДОДАТОК Б). Під час запуску аналізу відео модулі працюють у конвеєрі: кадри послідовно проходять через Tracker, який знаходить об’єкти; потім застосовується CameraMovementEstimator для корекції координат; далі ViewTransformer переводить координати в площину поля. Отримані дані накопичуються, і по завершенні аналізу викликаються SpeedAndDistanceEstimator, PlayerBallAssigner та TeamAssigner, які розраховують фінальні статистичні показники та додають атрибути команди. Результати (списки гравців, їхні показники, відсоток володіння) відображаються у GUI та записуються до бази даних.

Таким чином, в межах даного розділу було спроєктовано інформаційну базу даних та реалізовано програмне забезпечення системи, включаючи всі необхідні алгоритмічні модулі. Синергія вибраних інструментів та розроблених алгоритмів дозволила створити цілісну систему, що відповідає поставленим вимогам: автоматично розпізнає гравців і м’яч на відео, відстежує їх переміщення, компенсує рух камери для точності вимірювань, класифікує гравців за командами та обчислює ключові показники матчу. Результати роботи кожного з модулів коректно зберігаються у базі даних та надаються користувачу через зручний графічний інтерфейс. Це створює основу для подальшого аналізу матчів та прийняття рішень на основі даних, згенерованих системою.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Тестування системи є невід'ємним етапом впровадження, що дозволяє виявити та усунути можливі недоліки перед експлуатацією. Системне тестування – це перевірка повністю інтегрованої системи в цілому, з метою її оцінки відповідності початковим вимогам. На цьому етапі всі компоненти програми працюють разом як єдиний комплекс, і перевіряється коректність їх взаємодії та функціональність системи в реальних умовах використання.

У процесі розробки системи було виконано системне тестування, оскільки після реалізації окремих модулів (виявлення об'єктів, трекінгу, інтерфейсу та бази даних) необхідно було переконатися, що інтегрована програма працює стабільно та правильно. Тестування проводилося на реальних даних – відеофрагментах футбольних матчів, щоб наблизити умови до експлуатаційних. Було підготовлено тестовий набір відео із заздалегідь відомими очікуваними результатами (наприклад, відома кількість гравців на полі, наявність м'яча та арбітра у кадрі тощо) для перевірки коректності роботи всіх підсистем.

Основні аспекти, перевірені під час системного аналізу включають:

1. Функціонал виявлення та трекінгу об'єктів. Перевірено, що система правильно розпізнає всі цільові об'єкти на полі (футболістів обох команд, м'яч, арбітрів) на кожному кадрі відео, а модуль трекінгу коректно відстежує переміщення цих об'єктів між кадрами. Особливу увагу приділено ідентифікації гравців та безперервності їхнього відстеження протягом усього відеофрагменту (гравець повинен отримувати постійний унікальний ідентифікатор на всьому проміжку присутності в кадрі).

2. Коректність класифікації та підрахунку статистичних показників. Перевірено роботу алгоритму кластеризації KMeans для розподілу гравців між командами за кольором форми. Тестування підтвердило, що всі гравці правильно віднесені до своїх команд (рис. 11).

Рис. 11. Перевірка присвоєння гравців до відповідної команди

3. Інтеграція з базою даних. Проведено перевірку збереження результатів обробки у базі даних. Після обробки тестових відео система автоматично вносила зібрані дані (оброблені відеофайли з їх назвами, всю необхідно статистику по гравця та матчу) до бази даних (рис. 12). Тестування включало перевірку цілісності даних – наприклад, чи всі записи додано без помилок, чи правильно пов’язані таблиці (наприклад, таблиці для зберігання інформації про гравців та їх статистики). Було виконане вибіркове вилучення даних з бази даних (SQL-запити до БД) для впевненості, що збережені дані можна успішно прочитати та використовувати надалі (рис. 13).

	player_id [PK] uuid	team_id [PK] uuid	match_id [PK] uuid	distance double precision	avg_speed double precision
1	75c0ab91...	ed5e1c4d-0ad4-4111-b6f9-e3ea43cae3...	8e1e0c19-62d0-4ad0-8451-d96563ec80...	22.6781933522034	39.6585779704043
2	abd72dcc...	ed5e1c4d-0ad4-4111-b6f9-e3ea43cae3...	8e1e0c19-62d0-4ad0-8451-d96563ec80...	15.1658432058886	26.0713165363104
3	8a1533be...	1ac053f6-7ece-4212-8ae3-7e1d3944c3...	8e1e0c19-62d0-4ad0-8451-d96563ec80...	24.3483727513894	17.1032472009759
4	9c8f7569...	1ac053f6-7ece-4212-8ae3-7e1d3944c3...	8e1e0c19-62d0-4ad0-8451-d96563ec80...	10.8971163365697	33.6253875528438
5	e474b17f...	ed5e1c4d-0ad4-4111-b6f9-e3ea43cae3...	8e1e0c19-62d0-4ad0-8451-d96563ec80...	4.88997126340696	18.3692833547113

Рис. 12. Зібрані дані по кожному гравцеві.

Query Query History

```
1  SELECT * FROM player
```

Data output Messages Notifications

	player_id [PK] uuid	team_id uuid	match_id uuid	player_number integer
1	75c0ab91-1f6a-4b08-b995-0d7a48f38...	ed5e1c4d-0ad4-4111-b6f9-e3ea43cae3...	8e1e0c19-62d0-4ad0-8451-d96563ec80...	6
2	abd72dcc-b83b-4642-a0df-51ca7c085...	ed5e1c4d-0ad4-4111-b6f9-e3ea43cae3...	8e1e0c19-62d0-4ad0-8451-d96563ec80...	12
3	8a1533be-3812-41d3-8df5-4756e8e59...	1ac053f6-7ece-4212-8ae3-7e1d3944c3...	8e1e0c19-62d0-4ad0-8451-d96563ec80...	1
4	9c8f7569-7b4e-4e45-9e5d-d8652c6bf...	1ac053f6-7ece-4212-8ae3-7e1d3944c3...	8e1e0c19-62d0-4ad0-8451-d96563ec80...	55
5	e474b17f-60f9-4fc5-9984-e1da0d17b6...	ed5e1c4d-0ad4-4111-b6f9-e3ea43cae3...	8e1e0c19-62d0-4ad0-8451-d96563ec80...	83

Рис. 13. Запит для отримання всіх записів у таблиці player

4. Тестування користувацького інтерфейсу. Проаналізовано роботу графічного інтерфейсу (GUI), розробленого за допомогою бібліотеки PyQt5. Перевірено всі основні елементи GUI – завантаження відео файлу користувачем через відповідний діалог, запуск аналізу, відображення процесу опрацювання (повідомлення статусу обробки) (рис. 14), показ результатів – анотованого відео з накладеною інформацією у відеоплеєрі, та статистичних даних після аналізу. Інтерфейс протестовано на зручність взаємодії та відсутність збоїв. Наприклад, перевірено реакцію системи на некоректні діх користувача і коректність повідомлень про помилки.

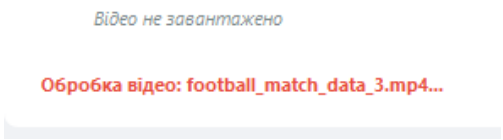

Рис. 14. Перевірка відображення статусу обробки відео

За результатами системного аналізу встановлено, що розроблена система відповідає заданим функціональним вимогам і готова до експлуатації. На рис. 15 та рис. 16 зображено загальний вигляд користувацького інтерфейсу, який був розроблений та доповнений за результатами тестування:

Рис. 15. Демонстрація користувацького інтерфейсу

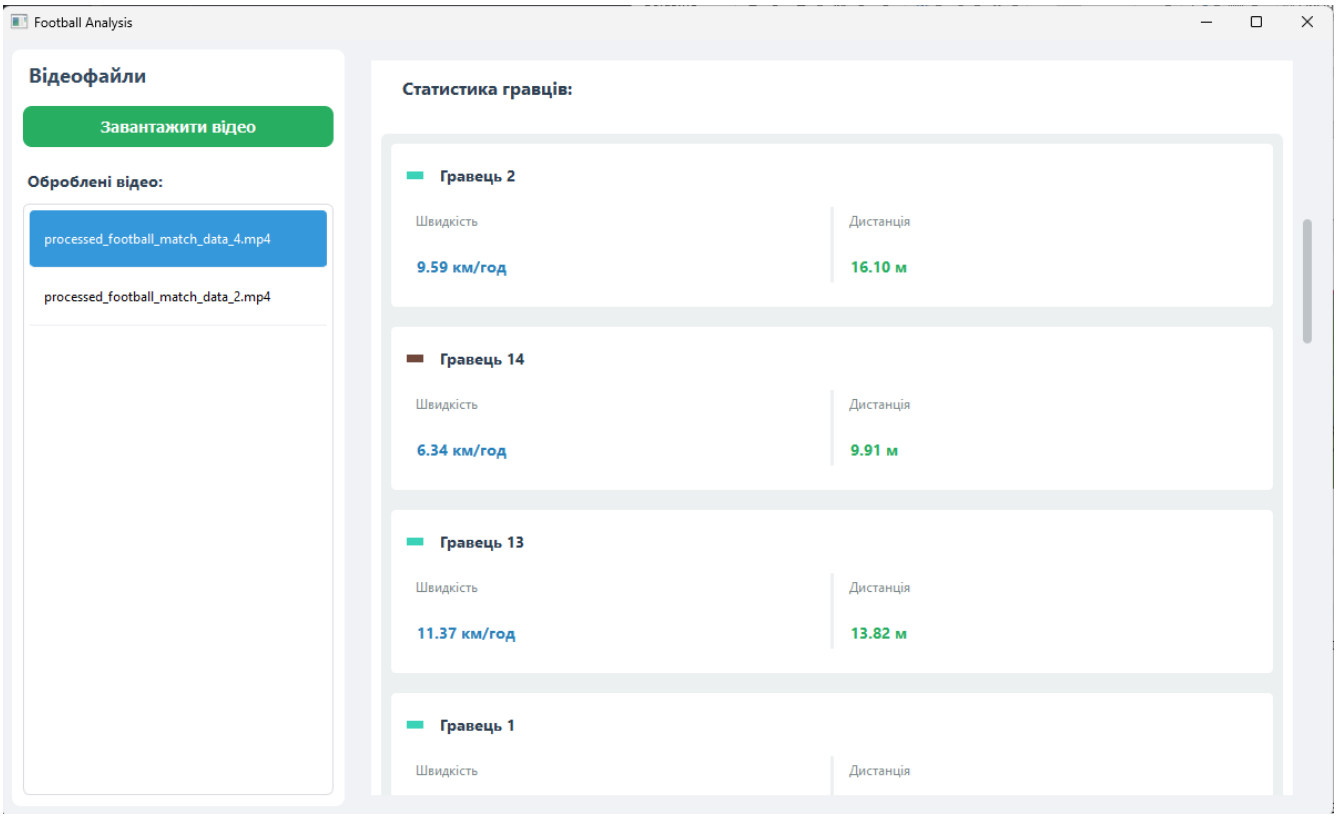

Рис. 16. Демонстрація користувацького інтерфейсу

4.2 Вимоги до апаратного та програмного забезпечення

Перед розгортанням системи необхідно визначити мінімальні вимоги до обладнання та програмної платформи, які забезпечать коректну роботу всіх компонентів. На діаграмі розгортання (рис. 3) представлена топологія рішення – усі складові системи встановлюються локально на одному комп’ютері. Застосунок Main.exe (головний виконуваний файл програми) працює на персональному комп'ютері користувача і взаємодіє із локально розгорнутою базою даних, використовуючи попередньо навчену модель нейронної мережі та модуль трекінгу об’єктів. Така одновузлова конфігурація означає, що обробка відео, збереження даних та відображення результатів виконуються в межах одного фізичного комп’ютера, без необхідності мережевої взаємодії з віддаленими серверами під час роботи. Відповідно, апаратні ресурси цього комп’ютера мають бути достатніми для виконання задач, такими як керування базою даних та графічним інтерфейсом.

Мінімальні апаратні вимоги для локального розгортання системи наступні:

- процесор. Багатоядерний процесор з тактовою частотою не менше 2.5 ГГц.

  Бажано використання як мінімум 4-ядерного процесора класу Intel Core i5 6-го покоління чи еквівалентного AMD. Потужніший CPU (наприклад, Intel Core i7/i9 або AMD Ryzen 7/9) рекомендований для швидшої обробки, але мінімально система здатна працювати і на середньопродуктивному процесорі за рахунок використання апаратного прискорення на GPU;

- оперативна пам’ять. Обробка відеофайлів потребує значного обсягу пам’яті.

  8 ГБ є мінімальним порогом для запуску системи, забезпечуючи роботу моделі та СУБД одночасно. Для опрацювання відео високої роздільної здатності або для підвищення стабільності бажано 16 ГБ і більше;

- відеокарта (GPU). Для ефективної роботи системи рекомендується наявність дискретної GPU з підтримкою технології CUDA. Мінімально підходить графічний процесор рівня NVIDIA GeForce GTX 1050 Ti або вище (з обсягом відеопам’яті від 4 ГБ). Наявність GPU значно прискорює інференс нейронної мережі (розпізнавання об’єктів на кадрах). У разі відсутності GPU модель може виконуватися на CPU, але продуктивність буде обмеженою (аналіз відео відбуватиметься значно повільніше);
- місце на диску. Для інсталяції програмних компонентів потрібен дисковий простір 1 – 2 ГБ. Сам файл моделі може займати кілька сотень мегабайт, також слід врахувати простір для СУБД (база даних зберігатиме проаналізовані відео та статистичні дані). Рекомендується мати щонайменше 10 ГБ вільного диску, щоб забезпечити збереження кількох відеозаписів та відповідних даних без браку пам’яті.

Мінімальні програмні вимоги для роботи системи такі:

- операційна система: Windows 10 або Windows 11 (64-розрядна версія). Розробку та тестування проведено в середовищі Windows 11 x64, тому гарантується сумісність саме з цією платформою. Теоретично система може бути розгорнута і на інших ОС (Linux) за наявності необхідних бібліотек, проте інсталяційний пакет підготовлено для Windows;
- платформа Python: інтерпретатор Python версії 3.11 або вище. Хоча кінцевому користувачу достатньо запускати готовий виконуваний файл, для запуску скриптів потрібно середовище Python 3.x. В інсталяційний пакет, зібраний у вигляді Main.exe, вже включено необхідний інтерпретатор і бібліотеки, але

якщо запускати систему з вихідних кодів – потрібне попереднє встановлення Python і залежностей;

- бібліотеки та фреймворки: програма використовує ряд зовнішніх бібліотек Python, які мають бути наявні в системі (вони ж включені у збірку .exe). Серед основних: бібліотека OpenCV для обробки зображень та роботи з відеофайлами, бібліотека scikit-learn (використовується алгоритм KMeans для кластеризації гравців за кольорами форми), бібліотека PyQt5 для реалізації графічного інтерфейсу користувача, а також драйвер psycopg2 для підключення до бази даних PostgreSQL. Усі ці залежності повинні відповідати версіям, з якими проводилась задля забезпечення сумісності. В рамках готового інсталяційного пакету всі необхідні бібліотеки постачаються разом з програмою, тому додатково встановлювати їх кінцевому користувачу не потрібно;
- система управління базами даних: PostgreSQL (рекомендована версія 14 або новіша). Для роботи системи необхідно встановити локально сервер СУБД PostgreSQL та створити порожню базу даних для додатку. Вимоги до СУБД стандартні: PostgreSQL повинна працювати у середовищі операційної системи, мати запущений сервіс і приймати підключення (через локальний хост) від програми. Інсталяційний пакет містить скрипти для ініціалізації структури бази (створення таблиць), але саме ПЗ PostgreSQL встановлюється окремо. Мінімальна конфігурація PostgreSQL достатня для роботи (типово виділяється кілька сотень МБ оперативної пам’яті, що входить у зазначені вище 8 ГБ RAM мінімальної системи). База даних не потребує спеціальних налаштувань, окрім створення користувача та схеми згідно з інструкціями, наведеними в інсталяційному керівництві.

Виходячи з наведених вимог, можна зробити висновок, що для успішного розгортання системи потрібен сучасний персональний комп’ютер середнього рівня з 64-розрядною ОС Windows та встановленим середовищем виконання Python і PostgreSQL. Дотримання мінімальних вимог гарантує працездатність програмного комплексу, тоді як рекомендовані вищі характеристики апаратної частини підвищать швидкодію та стабільність роботи при аналізі відео.

4.3 Склад інсталяційного пакету

Для розгортання системи на локальному комп’ютері підготовлено інсталяційни пакет, який містить всі необхідні компоненти програмного забезпечення (окрім зовнішньої залежності – СУБД). До складу пакету входять такі файли:

1. Виконуваний файл застосунку. Головний файл Main.exe, що є скомпільованою версією програми. Саме цей файл запускається користувачем для старту роботи системи. У ньому інтегровані всі ключові логічні модулі, графічний інтерфейс та необхідні бібліотеки. Main.exe забезпечує взаємодію з користувачем, обробку відеоданих із залученням моделі та запис результатів.
2. Файл моделі штучного інтелекту – файл нейронної мережі ai_model.pt, який містить попередньо навчену модель глибинного навчання для розпізнавання об’єктів (футболісти, м’яч, судді). Цей файл завантажується програмою при старті і використовується модулем інференсу для визначення об’єктів на кожному кадрі. Модель збережена у форматі .pt і була отримана в результаті навчання на відповідному наборі даних; вона є невід’ємною частиною інтелектуальної підсистеми програми.

3. Модуль трекінгу об’єктів – компонент програми, відповідальний за відстеження знайдених об’єктів між послідовними кадрами відео. У пакеті він може бути представлений у вигляді інтегрованого Python-скрипту, включеного до складу виконуваного файлу. У контексті інсталяційного пакету важливо, що всі необхідні для трекінгу бібліотеки вже включені, тож користувачу не потрібно встановлювати їх окремо.
4. Конфігураційні файли – файли налаштувань, що дозволяють змінювати параметри системи без модифікації коду. До них належить файл конфігурації застосунку, в якому вказані основні параметри налаштування підключення до бази даних (адреса сервера, ім’я користувача, пароль, назва БД).
5. Скрипти ініціалізації бази даних – текстові файли з SQL-скриптами для створення необхідної структури бази даних PostgreSQL. Як правило, надається файл, який містить команди CREATE TABLE для створення таблиць, що будуть використовуватися системою. Встановлення системи передбачає запуск цього скрипту в середовищі PostgreSQL, щоб розгорнути порожню базу даних відповідно до логічної моделі даних, спроєктованої в рамках системи.
6. Документація та інструкції – до пакету додаються текстові матеріали, зокрема, користувацька інструкція або файл README. В інструкції описано процес встановлення системи (наприклад, послідовність встановлення PostgreSQL, виконання скрипту створення БД, запуску головного застосунку), а також наведено короткий посібник користувача щодо роботи з програмою. Документація допомагає забезпечити правильне впровадження та ефективну експлуатацію системи кінцевим користувачем.

Таким чином, інсталяційний пакет містить повний набір компонентів, необхідних для локального запуску системи. Після встановлення всі складові розміщуються на комп’ютері користувача і взаємодіють згідно з діаграмою розгортання. Виконання файлу Main.exe ініціює завантаження моделі та необхідних

модулів, після чого користувач може користуватися системою без додаткових налаштувань, лише забезпечивши наявність встановленої СУБД та початкової ініціалізації бази даних. Перелічені в цьому підрозділі файли й компоненти гарантують, що розгортання проходить максимально просто, а експлуатація системи – надійна та зручна для користувача.

ВИСНОВОК

У результаті виконання бакалаврської кваліфікаційної роботи було розроблено додаток для автоматизованого аналізу футбольних відео з використанням методів комп’ютерного зору. Додаток дозволяє розпізнавати гравців та м’яч на полі, класифікувати гравців за їх командами, обчислювати статистичні показники, такі як середня швидкість гравця, пройдена дистанція гравцем, володіння м’ячем обома командами у відсотковому співвідношенні.

Розробка системи комп’ютерного зору для розпізнавання та аналізу рухомих об’єктів на футбольному полі забезпечує розвиток сучасних методів аналітики футбольних матчів. Це сприяє переходу від ручного аналізу матчів до автоматизованого, підвищуючи точність та оперативність аналізу.

Робота з відео та статистикою допомагає розвитку цифрових навичок та аналітичного мислення у користувача, також дозволяє тренерам чи аналітикам приймати більш обґрунтовані стратегічні рішення. Система є прикладом практичного застосування інтелектуальних технології у спорті.

Використання методів комп’ютерного зору у спортивній аналітиці значно покращує якість аналізу гри, дозволяє зменшити навантаження на тренерів чи аналітиків та забезпечує глибше розуміння подальших дій для спортивного росту гравців і командної тактики. Результати впровадження показують, що система є зручною, ефективною, та має потенціал для подальшого розвитку.

Постійне використання подібних систем сприятиме більш систематизованому підходу до підготовки команд, прийняттю рішень на основі даних та підвищенню загального рівня обізнаності під час використання цифрових технологій у спортивному середовищі.

СПИСКИ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Fujii K. Computer Vision for Sports Analytics. URL: https://link.springer.com/chapter/10.1007/978-981-96-1445-5_2.
2. Tracking Systems in Team Sports: A Narrative Review of Applications of the Data and Sport Specific Analysis - Sports Medicine - Open / L. Torres-Ronda et al. URL: https://sportsmedicine-open.springeropen.com/articles/10.1186/s40798-022-00408-z.
3. Andrews P., Borch N., Fjeld M. FootyVision: Multi-Object Tracking, Localisation, and Augmentation of Players and Ball in Football Video. URL: https://dl.acm.org/doi/10.1145/3665026.3665029.
4. Thulasya Naik B., Farukh Hashmi M., Dhanraj Bokde N. A Comprehensive Review of Computer Vision in Sports: Open Issues, Future Trends and Research Directions. URL: https://arxiv.org/abs/2203.02281.
5. Smirnov A., Worobjow A. How to Choose AI Cameras for Sports. Step by Step Guide from Engineers. URL: https://promwad.com/news/ai-in-sports-industry.
6. Koshkina M., Elder J. A General Framework for Jersey Number Recognition in Sports Video. URL: https://arxiv.org/abs/2405.13896.
7. Tracab Case Study. Amazon Web Services, Inc. URL: https://aws.amazon.com/ru/solutions/case-studies/tracab/.
8. Rodríguez (Norbs) N. Domain Modeling: What you need to know before coding. URL: https://www.thoughtworks.com/insights/blog/agile-project-management/domain-modeling-what-you-need-to-know-before-coding.
9. Linke D., Link D., Lames M. Football-specific validity of TRACAB's optical video tracking systems. URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0230179.

10. Second Spectrum. URL: https://www.secondspectrum.com.
11. Bo C. Research Overview of YOLO Series Object Detection Algorithms Based on Deep Learning | Journal of Computing and Electronic Information Management. URL: https://drpress.org/ojs/index.php/jceim/article/view/28340.
12. Zhou J. PitchEye: YOLO v5 and ByteTrack Football Player Tracking System in Football Videos.
URL: https://www.ijeast.com/papers/12-25,%20Tesma0901,IJEAST.pdf.
13. Bhoopati K. AI/ML Football Analysis system with YOLO, OpenCV, and Python. URL: https://medium.com/@krish.bhoopati556/ai-ml-football-analysis-system-with-yolo-opencv-and-python-778aa679501e.
14. He H. Clustering algorithm based on Hopkins statistics and K-means. Multimedia Tools and Applications. 2025. URL: https://doi.org/10.1007/s11042-025-20693-6.

ДОДАТКИ

ДОДАТОК А

```
import os
import uuid
from dotenv import load_dotenv
import psycopg2

load_dotenv()


def get_connection():
    return psycopg2.connect(
        dbname=os.getenv('dbname'),
        user=os.getenv('user'),
        password=os.getenv('password'),
        host=os.getenv('host'),
        port=os.getenv('port')
    )


def insert_match(video_bytes, video_name):
    conn = get_connection()
    cur = conn.cursor()
    try:
        match_id = str(uuid.uuid4())
        insert_query = """
            INSERT INTO match_info (match_id, match_video_name,
processed_match_video)
            VALUES (%s, %s, %s);
        """
        cur.execute(insert_query, (match_id, video_name,
psycopg2.Binary(video_bytes)))
        conn.commit()
        return match_id
    except Exception as e:
        conn.rollback()
        raise e
    finally:
        cur.close()
        conn.close()


def insert_team_and_stats(match_id, team_color, ball_possession):
    conn = get_connection()
    cur = conn.cursor()
    try:
        team_id = str(uuid.uuid4())
        team_query = """
            INSERT INTO team (team_id, match_id, team_color)
            VALUES (%s, %s, %s);
        """
        cur.execute(team_query, (team_id, match_id, team_color))
```

```
        teamstats_query = """
            INSERT INTO teamstats (team_id, match_id, ball_possession)
            VALUES (%s, %s, %s);
        """
        cur.execute(teamstats_query, (team_id, match_id, ball_possession))

        conn.commit()
        return team_id
    except Exception as e:
        conn.rollback()
        raise e
    finally:
        cur.close()
        conn.close()


def insert_player_and_stats(team_id, match_id, player_number, distance, avg_speed):
    conn = get_connection()
    cur = conn.cursor()
    try:
        player_id = str(uuid.uuid4())
        player_query = """
            INSERT INTO player (player_id, team_id, match_id, player_number)
            VALUES (%s, %s, %s, %s);
        """
        cur.execute(player_query, (player_id, team_id, match_id, player_number))

        playerstats_query = """
            INSERT INTO playerstats (player_id, team_id, match_id, distance,
avg_speed)
            VALUES (%s, %s, %s, %s, %s);
        """
        cur.execute(playerstats_query, (player_id, team_id, match_id, distance,
avg_speed))

        conn.commit()
        return player_id
    except Exception as e:
        conn.rollback()
        raise e
    finally:
        cur.close()
        conn.close()


def fetch_all_matches():
    conn = get_connection()
    cur = conn.cursor()
    try:
        cur.execute("SELECT match_id, match_video_name FROM match_info;")
        rows = cur.fetchall()
        return [{'match_id': row[0], 'match_video_name': row[1]} for row in rows]
    finally:
        cur.close()
        conn.close()


def fetch_match(match_id):
```

```
    conn = get_connection()
    cur = conn.cursor()
    try:
        cur.execute("SELECT processed_match_video FROM match_info WHERE match_id =
%s", (match_id,))
        row = cur.fetchone()
        return row
    finally:
        cur.close()
        conn.close()


def fetch_all_teams():
    conn = get_connection()
    cur = conn.cursor()
    try:
        cur.execute("SELECT team_id, match_id, team_color FROM team;")
        rows = cur.fetchall()
        return [{'team_id': row[0], 'match_id': row[1], 'team_color': row[2]} for
row in rows]
    finally:
        cur.close()
        conn.close()


def fetch_all_teamstats():
    conn = get_connection()
    cur = conn.cursor()
    try:
        cur.execute("SELECT team_id, match_id, ball_possession FROM teamstats;")
        rows = cur.fetchall()
        return [{'team_id': row[0], 'match_id': row[1], 'ball_possession': row[2]}
for row in rows]
    finally:
        cur.close()
        conn.close()


def fetch_all_players():
    conn = get_connection()
    cur = conn.cursor()
    try:
        cur.execute("SELECT player_id, team_id, match_id, player_number FROM
player;")
        rows = cur.fetchall()
        return [{'player_id': row[0], 'team_id': row[1], 'match_id': row[2],
'player_number': row[3]} for row in rows]
    finally:
        cur.close()
        conn.close()


def fetch_all_playerstats():
    conn = get_connection()
    cur = conn.cursor()
    try:
        cur.execute("SELECT player_id, team_id, match_id, distance, avg_speed FROM
playerstats;")
        rows = cur.fetchall()
```

```
        return [
            {
                'player_id': row[0], 'team_id': row[1], 'match_id': row[2],
'distance': row[3], 'avg_speed': row[4]
            } for row in rows
        ]
    finally:
        cur.close()
        conn.close()


def fetch_team_colors(match_id):
    conn = get_connection()
    cur = conn.cursor()

    cur.execute("SELECT team_id, team_color FROM team WHERE match_id = %s",
(match_id,))
    results = cur.fetchall()

    team_colors = {}
    for row in results:
        team_id, color_str = row
        try:
            rgb = tuple(int(c) for c in color_str.strip("()").split(','))
            team_colors[team_id] = rgb
        except ValueError:
            team_colors[team_id] = (0, 0, 0)

    cur.close()
    conn.close()
    return team_colors
```

ДОДАТОК Б

```
from PyQt5.QtGui import QIcon
from PyQt5.QtWidgets import (
    QApplication,
    QMainWindow,
    QListWidget,
    QPushButton,
    QVBoxLayout,
    QHBoxLayout,
    QWidget,
    QFileDialog,
    QLabel,
    QFrame,
    QListWidgetItem, QScrollArea
)
from PyQt5.QtCore import QThread, pyqtSignal, QUrl, Qt
from PyQt5.QtMultimedia import QMediaPlayer, QMediaContent
from PyQt5.QtMultimediaWidgets import QVideoWidget
from database_utils import insert_match, insert_team_and_stats,
insert_player_and_stats, fetch_all_matches, \
    fetch_all_teamstats, fetch_all_players, fetch_all_playerstats, fetch_match,
fetch_all_teams, fetch_team_colors
import sys
import os
from utils import read_video
from trackers import Tracker
import cv2
import numpy as np
from team_assigner import TeamAssigner
from player_ball_assigner import PlayerBallAssigner
from camera_movement_estimator import CameraMovementEstimator
from view_transformer import ViewTransformer
from speed_and_distance_estimator import SpeedAndDistanceEstimator


class VideoProcessThread(QThread):
    finished = pyqtSignal(str, dict)
    progress = pyqtSignal(int)

    def __init__(self, video_path):
        super().__init__()
        self.video_path = video_path

    def run(self):
        try:
            video_frames = read_video(self.video_path)

            tracker = Tracker('models/best.pt')
            tracks = tracker.get_object_tracks(video_frames, read_from_stub=False,
stub_path='stubs/track_stubs.pkl')

            tracker.add_position_to_tracks(tracks)

            camera_movement_estimator = CameraMovementEstimator(video_frames[0])
            camera_movement_per_frame =
camera_movement_estimator.get_camera_movement(
```

```
                video_frames, read_from_stub=False,
stub_path='stubs/camera_movement_stub.pkl'
            )
            camera_movement_estimator.add_adjust_positions_to_tracks(tracks,
camera_movement_per_frame)

            view_transformer = ViewTransformer()
            view_transformer.add_transformed_position_to_tracks(tracks)

            tracks['ball'] = tracker.interpolate_ball_positions(tracks['ball'])

            speed_and_distance_estimator = SpeedAndDistanceEstimator()
            speed_and_distance_estimator.add_speed_and_distance_to_tracks(tracks)

            team_assigner = TeamAssigner()
            team_assigner.assign_team_color(video_frames[0], tracks['players'][0])
            for frame_num, player_track in enumerate(tracks['players']):
                for player_id, track in player_track.items():
                    team = team_assigner.get_player_team(video_frames[frame_num],
track['bbox'], player_id)
                    tracks['players'][frame_num][player_id]['team'] = team
                    tracks['players'][frame_num][player_id]['team_color'] =
team_assigner.team_colors[team]

            player_assigner = PlayerBallAssigner()
            team_ball_control = []
            for frame_num, player_track in enumerate(tracks['players']):
                ball_bbox = tracks['ball'][frame_num][1]['bbox']
                assigned_player =
player_assigner.assign_ball_to_player(player_track, ball_bbox)
                if assigned_player != -1:
                    tracks['players'][frame_num][assigned_player]['has_ball'] =
True

team_ball_control.append(tracks['players'][frame_num][assigned_player]['team'])
                else:
                    if len(team_ball_control) > 0:
                        team_ball_control.append(team_ball_control[-1])
                    else:
                        team_ball_control.append(0)

            team_ball_control = np.array(team_ball_control)

            output_video_frames = tracker.draw_annotations(video_frames, tracks,
team_ball_control)

speed_and_distance_estimator.draw_speed_and_distance(output_video_frames, tracks)

            stats = self.calculate_statistics(tracks, team_ball_control)

            filename = os.path.basename(self.video_path)
            base_name, ext = os.path.splitext(filename)
            output_path = os.path.join('output_videos',
f'processed_{base_name}.mp4')
            os.makedirs('output_videos', exist_ok=True)

            fourcc = cv2.VideoWriter_fourcc(*'mp4v')
            height, width = output_video_frames[0].shape[:2]
            out = cv2.VideoWriter(output_path, fourcc, 30.0, (width, height))
```

```
            for frame in output_video_frames:
                out.write(frame)
            out.release()

            with open(output_path, 'rb') as f:
                video_bytes = f.read()

            match_id = insert_match(video_bytes, os.path.basename(output_path))

            team_colors = {1: (0, 0, 0), 2: (0, 0, 0)}

            for player_stats in stats['players'].values():
                team = player_stats['team']
                team_color = tuple(int(round(c)) for c in
player_stats['team_color'])
                if team in team_colors and team_colors[team] == (0, 0, 0):
                    team_colors[team] = team_color

            team1_poss = stats['team_possession']['team1']
            team2_poss = stats['team_possession']['team2']

            team1_color_str = ','.join(map(str, team_colors[1]))
            team2_color_str = ','.join(map(str, team_colors[2]))

            team1_id = insert_team_and_stats(match_id, team1_color_str,
int(team1_poss))
            team2_id = insert_team_and_stats(match_id, team2_color_str,
int(team2_poss))

            for player_id, player_stats in stats['players'].items():
                team = player_stats['team']
                team_id = team1_id if team == 1 else team2_id
                distance = player_stats['total_distance']
                speed = player_stats['avg_speed']
                insert_player_and_stats(team_id, match_id, int(player_id),
distance, speed)

            self.finished.emit(output_path, stats)

        except Exception as e:
            print(f'Помилка обробки відео: {e}')
            self.finished.emit('', {})

    @staticmethod
    def calculate_statistics(tracks, team_ball_control):
        total_frames = len(team_ball_control)
        if total_frames > 0:
            team1_percentage = (team_ball_control == 1).sum() / total_frames * 100
            team2_percentage = (team_ball_control == 2).sum() / total_frames * 100
        else:
            team1_percentage = 0
            team2_percentage = 0

        stats = {
            'team_possession': {
                'team1': team1_percentage,
                'team2': team2_percentage
            },
```

```
            'players': {}
        }

        for player_id in tracks['players'][-2].keys():
            player_speeds = []
            for frame in tracks['players']:
                if player_id in frame and 'speed' in frame[player_id]:
                    player_speeds.append(frame[player_id]['speed'])

            if player_speeds:
                avg_speed = sum(player_speeds) / len(player_speeds)
            else:
                avg_speed = 0

            last_frame = tracks['players'][-2]
            if player_id in last_frame:
                team = last_frame[player_id]['team']
                team_color = last_frame[player_id]['team_color'] if 'team_color' in
last_frame[player_id] else (0, 0, 0)
                total_distance = last_frame[player_id].get('distance', 0)

                stats['players'][player_id] = {
                    'avg_speed': avg_speed,
                    'total_distance': total_distance,
                    'team': team,
                    'team_color': team_color
                }

        return stats


class Window(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Football Analysis')
        self.setGeometry(500, 250, 1200, 700)

        self.setStyleSheet("""
            QMainWindow {
                background-color: #f0f2f5;
            }
            QLabel {
                font-family: 'Segoe UI', Arial, sans-serif;
                color: #2c3e50;
            }
            QPushButton {
                background-color: #3498db;
                color: white;
                border: none;
                padding: 8px 16px;
                border-radius: 4px;
                font-weight: bold;
            }
            QPushButton:hover {
                background-color: #2980b9;
            }
            QPushButton:pressed {
                background-color: #1c6ea4;
            }
```

```
        QListWidget {
            background-color: white;
            border: 1px solid #dcdde1;
            border-radius: 4px;
            padding: 5px;
        }
        QListWidget::item {
            padding: 8px;
            border-bottom: 1px solid #f0f0f0;
        }
        QListWidget::item:selected {
            background-color: #3498db;
            color: white;
        }
        QScrollBar:vertical {
            border: none;
            background: #f0f2f5;
            width: 8px;
            margin: 0px 0px 0px 0px;
        }
        QScrollBar::handle:vertical {
            background: #bdc3c7;
            min-height: 20px;
            border-radius: 4px;
        }
        QScrollBar::handle:vertical:hover {
            background: #95a5a6;
        }
        QScrollBar::add-line:vertical, QScrollBar::sub-line:vertical {
            height: 0px;
        }
        QScrollBar::add-page:vertical, QScrollBar::sub-page:vertical {
            background: none;
        }
        QScrollArea {
            border: none;
            background-color: transparent;
        }
    """)

    self.processed_videos = {}
    self.current_video = None

    self.upload_button = None
    self.video_list = None
    self.video_container = None
    self.media_player = None
    self.video_widget = None
    self.play_button = None
    self.pause_button = None
    self.player_status = None
    self.progress_label = None
    self.stats_container = None
    self.team1_layout = None
    self.team1_label = None
    self.team1_color = None
    self.team1_possession = None
    self.team2_layout = None
    self.team2_label = None
```

```
        self.team2_color = None
        self.team2_possession = None
        self.players_stats_container = None
        self.processing_thread = None

        self.init_ui()
        self.load_video_history()

    def load_video_history(self):
        self.video_list.clear()
        matches = fetch_all_matches()
        for match in matches:
            video_name = match['match_video_name']
            match_id = match['match_id']

            item = QListWidgetItem(video_name)
            item.setData(Qt.UserRole, match_id)
            self.video_list.addItem(item)

    def init_ui(self):
        main_layout = QHBoxLayout()
        main_layout.setSpacing(15)

        left_panel = QVBoxLayout()
        left_panel.setContentsMargins(10, 10, 10, 10)

        left_title = QLabel('Відеофайли')
        left_title.setStyleSheet('font-size: 18px; font-weight: bold; margin-
bottom: 10px; color: #34495e;')
        left_panel.addWidget(left_title)

        self.upload_button = QPushButton('Завантажити відео')
        self.upload_button.setStyleSheet("""
            QPushButton {
                background-color: #27ae60;
                color: white;
                font-size: 14px;
                padding: 10px;
                margin-bottom: 15px;
            }
            QPushButton:hover {
                background-color: #2ecc71;
            }
            QPushButton:pressed {
                background-color: #219653;
            }
        """)
        self.upload_button.setIcon(QIcon.fromTheme("document-open"))
        self.upload_button.clicked.connect(self.upload_video)
        left_panel.addWidget(self.upload_button)

        list_container = QVBoxLayout()
        list_label = QLabel('Оброблені відео:')
        list_label.setStyleSheet('font-size: 14px; font-weight: bold; margin-
bottom: 5px;')
        list_container.addWidget(list_label)

        self.video_list = QListWidget()
        self.video_list.setStyleSheet("""
```

```
            QListWidget {
                background-color: white;
                border-radius: 6px;
                padding: 5px;
            }
            QListWidget::item {
                height: 35px;
                padding-left: 10px;
                border-bottom: 1px solid #ecf0f1;
            }
            QListWidget::item:selected {
                background-color: #3498db;
                color: white;
                border-radius: 4px;
            }
            QListWidget::item:hover {
                background-color: #d6eaf8;
            }
        """)
        self.video_list.currentItemChanged.connect(self.video_selected)
        list_container.addWidget(self.video_list)
        left_panel.addLayout(list_container)

        left_widget = QWidget()
        left_widget.setLayout(left_panel)
        left_widget.setFixedWidth(300)
        left_widget.setStyleSheet("background-color: white; border-radius: 8px;")
        main_layout.addWidget(left_widget)

        right_panel = QVBoxLayout()
        right_panel.setContentsMargins(0, 10, 10, 10)

        right_scroll_area = QScrollArea()
        right_scroll_area.setWidgetResizable(True)
        right_scroll_area.setHorizontalScrollBarPolicy(Qt.ScrollBarAlwaysOff)
        right_scroll_area.setVerticalScrollBarPolicy(Qt.ScrollBarAsNeeded)
        right_scroll_area.setStyleSheet("border: none;")

        right_scroll_widget = QWidget()
        right_scroll_layout = QVBoxLayout(right_scroll_widget)
        right_scroll_layout.setSpacing(15)

        self.video_container = QWidget()
        self.video_container.setStyleSheet("background-color: white; border-radius:
8px; padding: 10px;")
        video_layout = QVBoxLayout(self.video_container)

        video_title = QLabel('Перегляд відео')
        video_title.setStyleSheet('font-size: 18px; font-weight: bold; margin-
bottom: 10px; color: #34495e;')
        video_layout.addWidget(video_title)

        self.media_player = QMediaPlayer(None, QMediaPlayer.VideoSurface)
        self.media_player.error.connect(self.handle_player_error)
        self.media_player.stateChanged.connect(self.media_state_changed)

        self.video_widget = QVideoWidget()
        self.video_widget.setMinimumHeight(360)
        self.video_widget.setStyleSheet("background-color: #2c3e50; border-radius:
```

```
4px;")
        self.media_player.setVideoOutput(self.video_widget)
        video_layout.addWidget(self.video_widget)

        player_controls = QHBoxLayout()
        player_controls.setContentsMargins(0, 10, 0, 10)

        self.play_button = QPushButton('Відтворити')
        self.play_button.setIcon(QIcon.fromTheme("media-playback-start"))
        self.play_button.clicked.connect(self.play_video)
        self.play_button.setStyleSheet("""
            QPushButton {
                background-color: #27ae60;
                color: white;
                font-size: 14px;
                padding: 8px 16px;
                border-radius: 4px;
                min-width: 100px;
            }
            QPushButton:hover {
                background-color: #2ecc71;
            }
        """)
        player_controls.addWidget(self.play_button)

        self.pause_button = QPushButton('Пауза')
        self.pause_button.setIcon(QIcon.fromTheme("media-playback-pause"))
        self.pause_button.clicked.connect(self.pause_video)
        self.pause_button.setStyleSheet("""
            QPushButton {
                background-color: #e74c3c;
                color: white;
                font-size: 14px;
                padding: 8px 16px;
                border-radius: 4px;
                min-width: 100px;
            }
            QPushButton:hover {
                background-color: #c0392b;
            }
        """)
        player_controls.addWidget(self.pause_button)

        player_controls.addSpacing(20)
        player_controls.addStretch()

        video_layout.addLayout(player_controls)

        status_container = QHBoxLayout()
        status_icon = QLabel()
        status_icon.setPixmap(QIcon.fromTheme("dialog-information").pixmap(16, 16))
        status_container.addWidget(status_icon)

        self.player_status = QLabel('Відео не завантажено')
        self.player_status.setStyleSheet("color: #7f8c8d; font-style: italic;")
        status_container.addWidget(self.player_status)
        status_container.addStretch()

        video_layout.addLayout(status_container)
```

```
        progress_container = QVBoxLayout()
        self.progress_label = QLabel('Відео обробляється, будь ласка зачекайте...')
        self.progress_label.setStyleSheet("color: #e74c3c; font-weight: bold;")
        progress_container.addWidget(self.progress_label)
        self.progress_label.hide()
        video_layout.addLayout(progress_container)

        right_scroll_layout.addWidget(self.video_container)

        stats_widget = QWidget()
        stats_widget.setStyleSheet("background-color: white; border-radius: 8px;
padding: 15px;")
        self.stats_container = QVBoxLayout(stats_widget)

        stats_title = QLabel('Статистика матчу')
        stats_title.setStyleSheet('font-size: 18px; font-weight: bold; margin-
bottom: 15px; color: #34495e;')
        self.stats_container.addWidget(stats_title)

        possession_container = QWidget()
        possession_container.setStyleSheet("background-color: #ecf0f1; border-
radius: 6px; padding: 10px;")
        possession_layout = QVBoxLayout(possession_container)

        possession_title = QLabel('Володіння м\'ячем:')
        possession_title.setStyleSheet('font-size: 16px; font-weight: bold; margin-
bottom: 10px; color: #2c3e50;')
        possession_layout.addWidget(possession_title)

        self.team1_layout, self.team1_color, self.team1_possession =
self.create_team_layout('Команда 1:')
        possession_layout.addLayout(self.team1_layout)

        separator = QFrame()
        separator.setFrameShape(QFrame.HLine)
        separator.setFrameShadow(QFrame.Sunken)
        separator.setStyleSheet("background-color: #bdc3c7; margin: 5px 0;")
        possession_layout.addWidget(separator)

        self.team2_layout, self.team2_color, self.team2_possession =
self.create_team_layout('Команда 2:')
        possession_layout.addLayout(self.team2_layout)

        self.stats_container.addWidget(possession_container)

        players_title = QLabel('Статистика гравців:')
        players_title.setStyleSheet(
            'font-size: 16px; font-weight: bold; margin-top: 15px; margin-bottom:
10px; color: #2c3e50;')
        self.stats_container.addWidget(players_title)

        players_container = QWidget()
        players_container.setStyleSheet("background-color: #ecf0f1; border-radius:
6px; padding: 10px;")
        self.players_stats_container = QVBoxLayout(players_container)
        self.players_stats_container.setSpacing(10)

        self.stats_container.addWidget(players_container)
```

```
        right_scroll_layout.addWidget(stats_widget)

        right_scroll_area.setWidget(right_scroll_widget)
        right_panel.addWidget(right_scroll_area)

        right_widget = QWidget()
        right_widget.setLayout(right_panel)
        main_layout.addWidget(right_widget)

        central_widget = QWidget()
        central_widget.setLayout(main_layout)
        self.setCentralWidget(central_widget)

    def create_team_layout(self, team_name):
        layout = QHBoxLayout()

        team_info = QHBoxLayout()
        label = QLabel(team_name)
        label.setStyleSheet('font-weight: bold; min-width: 100px;')

        color_frame = QFrame()
        color_frame.setFixedSize(20, 20)
        color_frame.setStyleSheet("border: 1px solid #bdc3c7; border-radius: 3px;")

        team_info.addWidget(label)
        team_info.addWidget(color_frame)
        team_info.addStretch()
        layout.addLayout(team_info, 1)

        possession_layout = QHBoxLayout()
        possession_label = QLabel('0%')
        possession_label.setStyleSheet('font-size: 16px; font-weight: bold; color:
#2980b9;')
        possession_layout.addWidget(possession_label)
        possession_layout.addStretch()
        layout.addLayout(possession_layout, 1)

        return layout, color_frame, possession_label

    def upload_video(self):
        file_dialog = QFileDialog()
        file_path, _ = file_dialog.getOpenFileName(
            self, 'Завантажити відео', '', 'Video Files (*.mp4 *.avi *.mov *.mkv)'
        )

        if file_path:
            self.progress_label.show()
            self.progress_label.setText(f'Обробка відео:
{os.path.basename(file_path)}...')

            self.processing_thread = VideoProcessThread(file_path)
            self.processing_thread.finished.connect(self.video_processed)
            self.processing_thread.start()

    def video_processed(self, output_path, stats):
        self.progress_label.hide()

        if output_path:
```

```
            video_name = os.path.basename(output_path)

            matches = fetch_all_matches()
            match = next((m for m in matches if m['match_video_name'] ==
video_name), None)

            if match:
                item = QListWidgetItem(video_name)
                item.setData(Qt.UserRole, match['match_id'])
                self.video_list.addItem(item)
                self.video_list.setCurrentItem(item)


    def video_selected(self, current):
        if current:
            video_name = current.text()
            match_id = current.data(Qt.UserRole)

            video = fetch_match(match_id)

            if not video:
                self.show_error_message('Відео не знайдено в базі даних.')
                return

            video_bytes = video[0]
            temp_path = os.path.join("temp", f"{video_name}")
            os.makedirs("temp", exist_ok=True)
            with open(temp_path, 'wb') as f:
                f.write(video_bytes)

            self.current_video = temp_path
            self.load_video(temp_path)

            team_colors_map = fetch_team_colors(match_id)

            teamstats = fetch_all_teamstats()
            teamstats = [t for t in teamstats if t['match_id'] == match_id]

            players = fetch_all_players()
            players = [p for p in players if p['match_id'] == match_id]

            playerstats = fetch_all_playerstats()
            playerstats = [s for s in playerstats if s['match_id'] == match_id]

            stats = {
                'team_possession': {},
                'players': {}
            }

            for t in teamstats:
                color = t['team_id'][-1]
                key = 'team1' if color in ('1', 'a', 'b') else 'team2'
                stats['team_possession'][key] = t['ball_possession']

            for p in players:
                stat = next((s for s in playerstats if s['player_id'] ==
p['player_id']), None)
                if stat:
                    stats['players'][p['player_number']] = {
```

```
                        'avg_speed': stat['avg_speed'],
                        'total_distance': stat['distance'],
                        'team': 1 if p['team_id'] == teamstats[0]['team_id'] else
2,
                        'team_color': tuple(int(round(float(c))) for c in
team_colors_map.get(p['team_id'], (0, 0, 0)))
                    }

            self.display_statistics(stats)


    def show_error_message(self, message):
        self.player_status.setText(f'ПОМИЛКА: {message}')
        self.player_status.setStyleSheet("color: #e74c3c; font-weight: bold;")

    def load_video(self, video_path):
        abs_path = os.path.abspath(video_path)

        if not os.path.exists(abs_path):
            self.show_error_message(f'Файл не знайдено: {abs_path}')
            return

        url = QUrl.fromLocalFile(abs_path)

        content = QMediaContent(url)
        self.media_player.setMedia(content)

        self.media_player.play()
        self.video_widget.show()

    def play_video(self):
        if self.media_player.mediaStatus() == QMediaPlayer.NoMedia:
            self.player_status.setText('Немає завантаженого відео')
            self.player_status.setStyleSheet("color: #f39c12; font-style: italic;")
            return

        self.media_player.play()
        self.player_status.setText('Відтворення...')
        self.player_status.setStyleSheet("color: #27ae60; font-weight: bold;")

    def pause_video(self):
        self.media_player.pause()
        self.player_status.setText('Пауза')
        self.player_status.setStyleSheet("color: #7f8c8d; font-style: italic;")

    def handle_player_error(self, error):
        error_messages = {
            QMediaPlayer.NoError: 'Немає помилок',
            QMediaPlayer.ResourceError: 'Ресурс не знайдено або недоступний',
            QMediaPlayer.FormatError: 'Формат не підтримується',
            QMediaPlayer.NetworkError: 'Помилка мережі',
            QMediaPlayer.AccessDeniedError: 'Доступ заборонено',
            QMediaPlayer.ServiceMissingError: 'Сервіс не знайдено'
        }

        error_text = error_messages.get(error, f'Невідома помилка: {error}')
        self.show_error_message(f'Помилка відтворення: {error_text}')
        print(f'Помилка плеєра: {error_text}')
```

```
    def media_state_changed(self, state):
        states = {
            QMediaPlayer.StoppedState: 'Зупинено',
            QMediaPlayer.PlayingState: 'Відтворення',
            QMediaPlayer.PausedState: 'Пауза'
        }

        status_text = states.get(state, f'Невідомий стан: {state}')
        self.player_status.setText(status_text)

        if state == QMediaPlayer.PlayingState:
            self.player_status.setStyleSheet("color: #27ae60; font-weight: bold;")
        else:
            self.player_status.setStyleSheet("color: #7f8c8d; font-style: italic;")

    def display_statistics(self, stats):
        self.clear_layout(self.players_stats_container)

        if 'team_possession' in stats:
            team1_possession = stats['team_possession'].get('team1', 0)
            team2_possession = stats['team_possession'].get('team2', 0)

            self.team1_possession.setText(f'{team1_possession:.1f}%')
            self.team2_possession.setText(f'{team2_possession:.1f}%')

            match_id = self.video_list.currentItem().data(Qt.UserRole)
            team_colors_raw = fetch_team_colors(match_id)

            if 'players' in stats and stats['players']:
                team_colors = {}
                for team_id, color_tuple in team_colors_raw.items():
                    color = tuple(int(round(float(c))) for c in color_tuple)
                    key = 1 if team_id == list(team_colors_raw.keys())[0] else 2
                    team_colors[key] = color

                if 1 in team_colors:
                    color = team_colors[1]
                    self.team1_color.setStyleSheet(
                        f'background-color: rgb({color[0]}, {color[1]},
{color[2]}); '
                        f'border: 1px solid #bdc3c7; border-radius: 3px;'
                    )

                if 2 in team_colors:
                    color = team_colors[2]
                    self.team2_color.setStyleSheet(
                        f'background-color: rgb({color[0]}, {color[1]},
{color[2]}); '
                        f'border: 1px solid #bdc3c7; border-radius: 3px;'
                    )

        if 'players' in stats:
            for player_id, player_stats in stats['players'].items():
                player_card = QFrame()
                player_card.setStyleSheet("""
                    background: white;
                    border-radius: 4px;
                    padding: 5px;
                    margin-bottom: 8px;
```

```
                """)
                player_layout = QVBoxLayout(player_card)

                header_layout = QHBoxLayout()

                team_color_frame = QFrame()
                team_color_frame.setFixedSize(15, 15)
                if 'team_color' in player_stats:
                    color = player_stats['team_color']
                    team_color_frame.setStyleSheet(
                        f'background-color: rgb({color[0]}, {color[1]},
{color[2]}); border-radius: 7px;')
                header_layout.addWidget(team_color_frame)

                player_id_label = QLabel(f'Гравець {player_id}')
                player_id_label.setStyleSheet('font-weight: bold; font-size:
14px;')
                header_layout.addWidget(player_id_label)
                header_layout.addStretch()

                player_layout.addLayout(header_layout)

                stats_layout = QHBoxLayout()

                speed_container = QVBoxLayout()
                speed_title = QLabel('Швидкість')
                speed_title.setStyleSheet('color: #7f8c8d; font-size: 12px;')
                speed_container.addWidget(speed_title)

                avg_speed = player_stats.get('avg_speed', 0)
                speed_value = QLabel(f'{avg_speed:.2f} км/год')
                speed_value.setStyleSheet('font-weight: bold; font-size: 14px;
color: #2980b9;')
                speed_container.addWidget(speed_value)
                stats_layout.addLayout(speed_container)

                line = QFrame()
                line.setFrameShape(QFrame.VLine)
                line.setFrameShadow(QFrame.Sunken)
                line.setStyleSheet('background-color: #ecf0f1;')
                stats_layout.addWidget(line)

                distance_container = QVBoxLayout()
                distance_title = QLabel('Дистанція')
                distance_title.setStyleSheet('color: #7f8c8d; font-size: 12px;')
                distance_container.addWidget(distance_title)

                total_distance = player_stats.get('total_distance', 0)
                distance_value = QLabel(f'{total_distance:.2f} м')
                distance_value.setStyleSheet('font-weight: bold; font-size: 14px;
color: #27ae60;')
                distance_container.addWidget(distance_value)
                stats_layout.addLayout(distance_container)

                player_layout.addLayout(stats_layout)

                self.players_stats_container.addWidget(player_card)

    def clear_layout(self, layout):
```

```
        if layout is not None:
            while layout.count():
                item = layout.takeAt(0)
                widget = item.widget()
                if widget is not None:
                    widget.deleteLater()
                else:
                    self.clear_layout(item.layout())


def application():
    app = QApplication(sys.argv)
    window = Window()
    window.show()
    sys.exit(app.exec_())


if __name__ == '__main__':
    application()
```

ДОДАТОК В

```
import os

import numpy as np
import pandas as pd
from ultralytics import YOLO
from utils import get_center_of_bbox, get_bbox_width, get_foot_position
import supervision as sv
import pickle
import cv2
import sys
sys.path.append('../')


class Tracker:
    def __init__(self, model_path):
        self.model = YOLO(model_path)
        self.tracker = sv.ByteTrack()

    def add_position_to_tracks(self, tracks):
        for object, object_tracks in tracks.items():
            for frame_num, track in enumerate(object_tracks):
                for track_id, track_info in track.items():
                    bbox = track_info['bbox']
                    if object == 'ball':
                        position = get_center_of_bbox(bbox)
                    else:
                        position = get_foot_position(bbox)
                    tracks[object][frame_num][track_id]['position'] = position

    def interpolate_ball_positions(self, ball_positions):
        ball_positions = [x.get(1, {}).get('bbox', []) for x in ball_positions]
        df_ball_positions = pd.DataFrame(ball_positions, columns=['x1', 'y1', 'x2',
'y2'])

        df_ball_positions = df_ball_positions.interpolate()
        df_ball_positions = df_ball_positions.bfill()

        ball_positions = [{1: {'bbox': x}} for x in
df_ball_positions.to_numpy().tolist()]

        return ball_positions

    def detect_frames(self, frames):
        batch_size = 20
        detections = []
        for i in range(0, len(frames), batch_size):
            detections_batch = self.model.predict(frames[i:i + batch_size],
conf=0.1)
            detections += detections_batch
        return detections

    def get_object_tracks(self, frames, read_from_stub=False, stub_path=None):
        if read_from_stub and stub_path is not None and os.path.exists(stub_path):
            with open(stub_path, 'rb') as f:
                tracks = pickle.load(f)
```

```
            return tracks

        detections = self.detect_frames(frames)

        tracks = {
            'players': [],
            'referees': [],
            'ball': []
        }

        for frame_num, detection in enumerate(detections):
            cls_names = detection.names
            cls_names_inv = {v: k for k, v in cls_names.items()}

            detection_supervision = sv.Detections.from_ultralytics(detection)

            for object_ind, class_id in enumerate(detection_supervision.class_id):
                if cls_names[class_id] == 'goalkeeper':
                    detection_supervision.class_id[object_ind] =
cls_names_inv['player']

            detection_with_tracks =
self.tracker.update_with_detections(detection_supervision)

            tracks['players'].append({})
            tracks['referees'].append({})
            tracks['ball'].append({})

            for frame_detection in detection_with_tracks:
                bbox = frame_detection[0].tolist()
                cls_id = frame_detection[3]
                track_id = frame_detection[4]

                if cls_id == cls_names_inv['player']:
                    tracks['players'][frame_num][track_id] = {'bbox': bbox}

                if cls_id == cls_names_inv['referee']:
                    tracks['referees'][frame_num][track_id] = {'bbox': bbox}

            for frame_detection in detection_supervision:
                bbox = frame_detection[0].tolist()
                cls_id = frame_detection[3]

                if cls_id == cls_names_inv['ball']:
                    tracks['ball'][frame_num][1] = {'bbox': bbox}

        if stub_path is not None:
            with open(stub_path, 'wb') as f:
                pickle.dump(tracks, f)

        return tracks

    @staticmethod
    def draw_ellipse(frame, bbox, color, track_id=None):
        y2 = int(bbox[3])
        x_center, _ = get_center_of_bbox(bbox)
        width = get_bbox_width(bbox)

        cv2.ellipse(
```

```
            frame,
            center=(x_center, y2),
            axes=(int(width), int(0.35 * width)),
            angle=0.0,
            startAngle=-45,
            endAngle=235,
            color=color,
            thickness=2,
            lineType=cv2.LINE_4
        )

        rectangle_width = 40
        rectangle_height = 20
        x1_rect = x_center - rectangle_width // 2
        x2_rect = x_center + rectangle_width // 2
        y1_rect = (y2 - rectangle_height // 2) + 15
        y2_rect = (y2 + rectangle_height // 2) + 15

        if track_id is not None:
            cv2.rectangle(frame, (int(x1_rect), int(y1_rect)), (int(x2_rect),
int(y2_rect)), color, cv2.FILLED)

            x1_text = x1_rect + 12
            if track_id > 99:
                x1_text -= 10

            cv2.putText(
                frame,
                f'{track_id}',
                (int(x1_text), int(y1_rect + 15)),
                cv2.FONT_HERSHEY_SIMPLEX,
                0.6,
                (0, 0, 0),
                2
            )

        return frame

    @staticmethod
    def draw_triangle(frame, bbox, color):
        y = int(bbox[1])
        x, _ = get_center_of_bbox(bbox)

        triangle_points = np.array([
            [x, y],
            [x - 10, y - 20],
            [x + 10, y - 20],
        ])
        cv2.drawContours(frame, [triangle_points], 0, color, cv2.FILLED)
        cv2.drawContours(frame, [triangle_points], 0, (0, 0, 0), 2)

        return frame

    def draw_team_ball_control(self, frame, frame_num, team_ball_control):
        overlay = frame.copy()
        cv2.rectangle(overlay, (1350, 850), (1900, 970), (255, 255, 255), -1)
        alpha = 0.4
        cv2.addWeighted(overlay, alpha, frame, 1 - alpha, 0, frame)
```

```
        team_ball_control_till_frame = team_ball_control[:frame_num + 1]
        team_1_num_frames =
team_ball_control_till_frame[team_ball_control_till_frame == 1].shape[0]
        team_2_num_frames =
team_ball_control_till_frame[team_ball_control_till_frame == 2].shape[0]

        total = team_1_num_frames + team_2_num_frames
        if total == 0:
            team_1 = team_2 = 0
        else:
            team_1 = team_1_num_frames / total
            team_2 = team_2_num_frames / total

        cv2.putText(
            frame,
            f'Team 1 Ball Control: {team_1 * 100:.2f}%',
            (1400, 900),
            cv2.FONT_HERSHEY_SIMPLEX,
            1,
            (0, 0, 0),
            3
        )
        cv2.putText(
            frame,
            f'Team 2 Ball Control: {team_2 * 100:.2f}%',
            (1400, 950),
            cv2.FONT_HERSHEY_SIMPLEX,
            1,
            (0, 0, 0),
            3
        )

        return frame

    def draw_annotations(self, video_frames, tracks, team_ball_control):
        output_video_frames = []
        for frame_num, frame in enumerate(video_frames):
            frame = frame.copy()

            player_dict = tracks['players'][frame_num]
            ball_dict = tracks['ball'][frame_num]
            referee_dict = tracks['referees'][frame_num]

            for track_id, player in player_dict.items():
                color = player.get('team_color', (0, 0, 255))
                frame = self.draw_ellipse(frame, player['bbox'], color, track_id)

                if player.get('has_ball', False):
                    frame = self.draw_triangle(frame, player['bbox'], (0, 0, 255))

            for _, referee in referee_dict.items():
                frame = self.draw_ellipse(frame, referee['bbox'], (0, 255, 255))

            for track_id, ball in ball_dict.items():
                frame = self.draw_triangle(frame, ball['bbox'], (0, 255, 0))

            frame = self.draw_team_ball_control(frame, frame_num,
team_ball_control)
```

```
        output_video_frames.append(frame)

    return output_video_frames
```

ДОДАТОК Д

```
from utils import measure_distance, measure_xy_distance
import pickle
import cv2
import numpy as np
import os
import sys

sys.path.append('../')


class CameraMovementEstimator:
    def __init__(self, frame):
        self.minimum_distance = 5

        self.lk_params = dict(
            winSize=(15, 15),
            maxLevel=2,
            criteria=(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03)
        )

        first_frame_grayscale = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        mask_features = np.zeros_like(first_frame_grayscale)
        mask_features[:, 0:20] = 1
        mask_features[:, 900:1050] = 1

        self.features = dict(
            maxCorners=100,
            qualityLevel=0.3,
            minDistance=3,
            blockSize=7,
            mask=mask_features
        )

    def add_adjust_positions_to_tracks(self, tracks, camera_movement_per_frame):
        for object, object_tracks in tracks.items():
            for frame_num, track in enumerate(object_tracks):
                for track_id, track_info in track.items():
                    position = track_info['position']
                    camera_movement = camera_movement_per_frame[frame_num]
                    position_adjusted = (position[0] - camera_movement[0],
position[1] - camera_movement[1])
                    tracks[object][frame_num][track_id]['position_adjusted'] =
position_adjusted

    def get_camera_movement(self, frames, read_from_stub=False, stub_path=None):
        if read_from_stub and stub_path is not None and os.path.exists(stub_path):
            with open(stub_path, 'rb') as f:
                return pickle.load(f)

        camera_movement = [[0, 0]] * len(frames)

        old_gray = cv2.cvtColor(frames[0], cv2.COLOR_BGR2GRAY)
        old_features = cv2.goodFeaturesToTrack(old_gray, **self.features)

        for frame_num in range(1, len(frames)):
```

```
            frame_gray = cv2.cvtColor(frames[frame_num], cv2.COLOR_BGR2GRAY)
            new_features, _, _ = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray,
old_features, None, **self.lk_params)

            max_distance = 0
            camera_movement_x, camera_movement_y = 0, 0

            for i, (new, old) in enumerate(zip(new_features, old_features)):
                new_features_point = new.ravel()
                old_features_point = old.ravel()

                distance = measure_distance(new_features_point, old_features_point)
                if distance > max_distance:
                    max_distance = distance
                    camera_movement_x, camera_movement_y =
measure_xy_distance(old_features_point, new_features_point)

            if max_distance > self.minimum_distance:
                camera_movement[frame_num] = [camera_movement_x, camera_movement_y]
                old_features = cv2.goodFeaturesToTrack(frame_gray, **self.features)

            old_gray = frame_gray.copy()

        if stub_path is not None:
            with open(stub_path, 'wb') as f:
                pickle.dump(camera_movement, f)

        return camera_movement
```

ДОДАТОК Е

```
import numpy as np
import cv2


class ViewTransformer:
    def __init__(self):
        court_width = 68
        court_length = 23.32

        self.pixel_vertices = np.array([
            [110, 1035],
            [265, 275],
            [910, 260],
            [1640, 915]
        ])

        self.target_vertices = np.array([
            [0, court_width],
            [0, 0],
            [court_length, 0],
            [court_length, court_width]
        ])

        self.pixel_vertices = self.pixel_vertices.astype(np.float32)
        self.target_vertices = self.target_vertices.astype(np.float32)

        self.perspective_transformer =
cv2.getPerspectiveTransform(self.pixel_vertices, self.target_vertices)

    def transform_point(self, point):
        p = (int(point[0]), int(point[1]))
        is_inside = cv2.pointPolygonTest(self.pixel_vertices, p, False) >= 0
        if not is_inside:
            return None

        reshaped_point = point.reshape(-1, 1, 2).astype(np.float32)
        transform_point = cv2.perspectiveTransform(reshaped_point,
self.perspective_transformer)
        return transform_point.reshape(-1, 2)

    def add_transformed_position_to_tracks(self, tracks):
        for object, object_tracks in tracks.items():
            for frame_num, track in enumerate(object_tracks):
                for track_id, track_info in track.items():
                    position = track_info['position_adjusted']
                    position = np.array(position)
                    position_transformed = self.transform_point(position)
                    if position_transformed is not None:
                        position_transformed =
position_transformed.squeeze().tolist()
                    tracks[object][frame_num][track_id]['position_transformed'] =
position_transformed
```

ДОДАТОК Ж

```
from utils import measure_distance, get_foot_position
import cv2
import sys

sys.path.append('../')


class SpeedAndDistanceEstimator:
    def __init__(self):
        self.frame_window = 5
        self.frame_rate = 24

    def add_speed_and_distance_to_tracks(self, tracks):
        total_distance = {}

        for object, object_tracks in tracks.items():
            if object == 'ball' or object == 'referees':
                continue
            number_of_frames = len(object_tracks)
            for frame_num in range(0, number_of_frames, self.frame_window):
                last_frame = min(frame_num + self.frame_window, number_of_frames -
1)

                for track_id, _ in object_tracks[frame_num].items():
                    if track_id not in object_tracks[last_frame]:
                        continue

                    start_position =
object_tracks[frame_num][track_id]['position_transformed']
                    end_position =
object_tracks[last_frame][track_id]['position_transformed']

                    if start_position is None or end_position is None:
                        continue

                    distance_covered = measure_distance(start_position,
end_position)
                    time_elapsed = (last_frame - frame_num) / self.frame_rate

                    if time_elapsed == 0:
                        speed_km_per_hour = 0.0
                    else:
                        speed_meters_per_second = distance_covered / time_elapsed
                        speed_km_per_hour = speed_meters_per_second * 3.6

                    if object not in total_distance:
                        total_distance[object] = {}

                    if track_id not in total_distance[object]:
                        total_distance[object][track_id] = 0

                    total_distance[object][track_id] += distance_covered

                    for frame_num_batch in range(frame_num, last_frame):
                        if track_id not in tracks[object][frame_num_batch]:
                            continue
```

```
                        tracks[object][frame_num_batch][track_id]['speed'] =
speed_km_per_hour
                        tracks[object][frame_num_batch][track_id]['distance'] =
total_distance[object][track_id]

    def draw_speed_and_distance(self, frames, tracks):
        output_frames = []
        for frame_num, frame in enumerate(frames):
            for object, object_tracks in tracks.items():
                if object == 'ball' or object == 'referees':
                    continue
                for _, track_info in object_tracks[frame_num].items():
                    if 'speed' in track_info:
                        speed = track_info.get('speed', None)
                        distance = track_info.get('distance', None)
                        if speed is None or distance is None:
                            continue

                        bbox = track_info['bbox']
                        position = get_foot_position(bbox)
                        position = list(position)
                        position[1] += 40

                        position = tuple(map(int, position))
                        cv2.putText(
                            frame,
                            f'{speed:.2f} km/h',
                            position,
                            cv2.FONT_HERSHEY_SIMPLEX,
                            0.5,
                            (0, 0, 0),
                            2
                        )
                        cv2.putText(
                            frame,
                            f'{distance:.2f} m',
                            (position[0], position[1] + 20),
                            cv2.FONT_HERSHEY_SIMPLEX,
                            0.5,
                            (0, 0, 0),
                            2
                        )
            output_frames.append(frame)

        return output_frames
```

ДОДАТОК И

```
from utils import get_center_of_bbox, measure_distance
import sys
sys.path.append('../')


class PlayerBallAssigner:
    def __init__(self):
        self.max_player_ball_distance = 70

    def assign_ball_to_player(self, players, ball_bbox):
        ball_position = get_center_of_bbox(ball_bbox)

        minimum_distance = 99999
        assigned_player = -1

        for player_id, player in players.items():
            player_bbox = player['bbox']

            distance_left = measure_distance((player_bbox[0], player_bbox[-1]),
ball_position)
            distance_right = measure_distance((player_bbox[2], player_bbox[-1]),
ball_position)
            distance = min(distance_left, distance_right)

            if distance < self.max_player_ball_distance:
                if distance < minimum_distance:
                    minimum_distance = distance
                    assigned_player = player_id

        return assigned_player
```

ДОДАТОК К

```
from sklearn.cluster import KMeans


class TeamAssigner:
    def __init__(self):
        self.kmeans = None
        self.team_colors = {}
        self.player_team_dict = {}

    @staticmethod
    def get_clustering_model(image):
        image_2d = image.reshape(-1, 3)

        kmeans = KMeans(n_clusters=2, init='k-means++', n_init=1)
        kmeans.fit(image_2d)

        return kmeans

    def get_player_color(self, frame, bbox):
        image = frame[int(bbox[1]):int(bbox[3]), int(bbox[0]):int(bbox[2])]

        top_half_image = image[0:int(image.shape[0] / 2), :]

        kmeans = self.get_clustering_model(top_half_image)

        labels = kmeans.labels_

        clustered_image = labels.reshape(top_half_image.shape[0],
top_half_image.shape[1])

        corner_clusters = [
            clustered_image[0, 0], clustered_image[0, -1], clustered_image[-1, 0],
clustered_image[-1, -1]
        ]
        non_player_cluster = max(set(corner_clusters), key=corner_clusters.count)
        player_cluster = 1 - non_player_cluster

        player_color = kmeans.cluster_centers_[player_cluster]

        return player_color

    def assign_team_color(self, frame, player_detections):
        player_colors = []
        for _, player_detection in player_detections.items():
            bbox = player_detection['bbox']
            player_color = self.get_player_color(frame, bbox)
            player_colors.append(player_color)

        kmeans = KMeans(n_clusters=2, init='k-means++', n_init=10)
        kmeans.fit(player_colors)

        self.kmeans = kmeans

        self.team_colors[1] = kmeans.cluster_centers_[0]
        self.team_colors[2] = kmeans.cluster_centers_[1]
```

```
def get_player_team(self, frame, player_bbox, player_id):
    if player_id in self.player_team_dict:
        return self.player_team_dict[player_id]

    player_color = self.get_player_color(frame, player_bbox)

    team_id = self.kmeans.predict(player_color.reshape(1, -1))[0]
    team_id += 1

    self.player_team_dict[player_id] = team_id

    return team_id
```