ШИФР: PoR

Конкурсна робота

на тему:

«ПРОГРАМНИЙ КОМПЛЕКС УПРАВЛІННЯ ІНТЕРАКТИВНИМ ГРАФІЧНИМ ІНТЕРФЕЙСОМ ЗА ТЕХНОЛОГІЯМИ EYE TRACKING»

ЗМІСТ

ВСТУП

Зір – важливий інструмент для людей під час сприйняття інформації з навколишнього середовища. Тому багато дослідників намагаються знайти ефективні рішення з використанням людського зору. Технологія Eye Tracking (також відома як «околуграфія», хоча цей термін не прижився) відносно нова і постійно розвивається. В Україні дана технологія поки не досить поширена.

Основне завдання технології Eye Tracking – відстеження та запис рухів очей для вивчення зорової уваги та поведінки. Дані, що надходять з інфрачервоних датчиків або камери для відстеження положення та рухів очей, дають змогу дослідникам та розробникам зрозуміти поведінку користувачів, зручність використання та когнітивні процеси для різного ПЗ.

Технологія Eye Tracking набуває популярності. Таке стрімке поширення новітньої технології можна пояснити необхідністю адаптації зростаючої кількості людей, які мають проблеми з моторикою або втратили кінцівки рук в непередбачуваній життєвій ситуації (під час ДТП, війни, травми тощо). У людей, що мають проблеми такого характеру, здебільшого виникає проблема керування пристроями вводу комп’ютера, на кшталт мишею або клавіатурою. Вищезазначені обставини – суттєва перешкода на шляху інтеграції людей з інвалідністю з сучасним світом. Технології Computer Vision дозволяють розв’язати актуальну проблему сьогодення шляхом створення програмного забезпечення для керування графічним інтерфейсом, зокрема рухом курсора миші за допомогою мови Python та інтегрованих пакетів, наприклад, OpenCV.

Практичний аспект використання технології Eye Tracking доволі широкий: від дослідження людської поведінки та когнітивних функцій до створення допоміжних технологій для людей з обмеженими можливостями.

За допомогою технології Eye Tracking можна визначити, що люди бачать та сприймають під час спостереження за курсором миші, взаємодії з посиланнями, елементами керування. Дана технологія допомагає у точному визначенні привабливості візуальних елементів для користувачів (інтерфейс

навігації, графіка, посилання, текст, мультимедійний контент тощо). Отже, Eye Tracking дає відповідь на запитання, як користувачі сприймають online-контент.

Системи, що дозволяють інтегрувати технологію Eye Tracking в розроблене ПЗ, використовують програмні та апаратні засоби, які є важкодоступними в побутових умовах.

Метою програмної частини та *новизною* проекту є впровадження вищезазначеної технології, використовуючи доступні інструменти – звичайну web-камеру.

Незначне поширення на ранніх етапах розвитку технології відстеження очей можна пояснити тим, що спочатку системи Eye Tracking були інвазивними (ті, які втручаються в організм) та нерухомими. Але сучасні технологічні досягнення дозволяють впроваджувати дану технологію в ПЗ, незважаючи на певні обмеження, що досі має Eye Tracking.

Метою дослідження є автоматизація процесу управління інтерактивним графічним інтерфейсом із впровадженням технологій Eye Tracking.

Предмет дослідження: методи ідентифікації, стеження та управління результатами об’єктового моніторингу з використанням можливостей Computer Vision.

Об’єкт дослідження: спостереження та аналіз положення зіниці ока за допомогою технологій Computer Vision.

Дана робота складається зі вступу, трьох розділів, висновків та пропозицій, списку використаних джерел, додатків, що розміщуються на 109 сторінках, і включає: 33 рисунки, 6 додатків та список використаних джерел, який налічує 38 найменувань.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ВІДОМИХ ПІДХОДІВ ДЛЯ ПРОЕКТУВАННЯ ТА ВПРОВАДЖЕННЯ ПРОГРАМНИХ КОМПЛЕКСІВ УПРАВЛІННЯ ІНТЕРАКТИВНИМ ГРАФІЧНИМ ІНТЕРФЕЙСОМ

1.1. Сфери застосування програмних методів управління інтерактивним графічним інтерфейсом

Очі – основна складова зорового аналізатора людини. Приблизно 90 % інформації надходить у мозок через зоровий канал. Вони забезпечують сприйняття форми, розмірів та кольорів предметів, що оточують людину.

Для взаємодії з будь-яким ПЗ потрібно мати зовнішній інтерфейс. Графічний інтерфейс користувача (GUI) – засіб зручної взаємодії користувача з інформаційною системою (ІС).

Для взаємодії з графічним інтерфейсом використовуються пристрої вводу комп’ютера, що забезпечують безпосередньо передачу інформації. Серед таких пристроїв найважливішим є маніпулятор «миша», адже він перетворює механічні рухи в рухи курсора на екрані.

За допомогою технологій Eye Tracking можливо створити програмний комплекс, який допоміг би людям з обмеженими можливостями без перешкод замінити пристрої вводу. Актуальність даного рішення можна пояснити зростанням кількості людей вищезгаданої категорії внаслідок непередбачуваної життєвої ситуації: війни, ДТП, вроджені захворювання тощо.

Eye Tracking може використовуватись для моніторингу лікування, реєстрації медичних записів або фармакологічного контролю. Рухи очей можуть вказувати на різні захворювання мозку та очей, такі як черепно-мозкова травма, хвороба Альцгеймера та глаукома. Процес Eye Tracking фіксує майже непомітні зміни в рухах очей і допомагає визначити хвороби на ранніх етапах їхнього розвитку. Дана технологія може стати корисною у сфері військової медицини: при розробці системи, що визначає рівень втоми солдатів. Показниками втоми виступають частота та тривалість кліпання. Варто

зазначити, що рухи очей важливі в сфері аеронавтики: за допомогою Eye Tracking можна проаналізувати фізичний та психологічний стан пілота.

Американські льотні інструктори використовують Eye Tracker для стеження за рухами очей пілотів в авіасимуляторах. Відстеження очей можливо використовувати для наведення зброї, просто дивлячись на ціль. Розробники винищувачів BAE Systems створили технологію «wearable cockpit» («віртуальна кабіна пілота»), згідно якої пілот може швидко отримувати важливу інформацію за допомогою віртуального дисплею на шоломі (Рисунок 1.1)[1].

Eye Tracking допоможе оптимізувати розміщення товарів або рекламних банерів. Вона визначить місця, які насамперед привертають увагу споживачів. За допомогою даної технології можна отримати інформацію про те, які продукти залишаються непоміченим. Це слугує своєрідним сигналом для виробників, яку продукцію потрібно виробляти, аби максимізувати власний прибуток.

В сучасному автомобілі, за допомогою Eye Tracking можна визначати зони небезпеки та втрати контролю водія. Завдяки фіксації погляду можна зрозуміти, як водії взаємодіють з своїм автомобілем, покращити умови керування, забезпечити комфорт та безпеку.

1.2. Аналіз відомих методологічних основ управління інтерактивним графічним інтерфейсом в стеці технологій Computer Vision

Зазвичай для керування інтерактивним інтерфейсом використовується миша як пристрій вводу. За її допомогою користувач може вибирати будь-яку операцію, гортати сторінки тощо. Але за допомогою Eye Tracking, що використовує відомі стеки технологій Computer Vision, розроблено систему керування інтерактивним інтерфейсом за допомогою очей.

Електромагнітна склеральна пошукова котушка – одна з перших електромагнітних систем. Ці котушки вбудовувались в контактну лінзу, що

[1] Рисунки до Розділу 1 наведені в Додатку А.

щільно прилягала до ока, та за допомогою дроту підключались до записуючого пристрою (Рисунок 1.2). Використання таких котушок забезпечувало високу точність та швидкість. Такий спосіб вважався найточнішим для визначення напрямку погляду людини.

Eye Tracking на основі відео складається з камери, що чутлива до інфрачервоного (ІЧ) випромінювання, оскільки останнє практично невидиме для учасника, а артефакти від штучних джерел освітлення можна легко відфільтрувати за довжиною хвилі.

Системи зі світлою зіницею використовують джерело ІЧ-променів на тій же осі, що й камера (Рисунок 1.3). Вони відстежують світлове відображення від сітківки через зіницю. Даний підхід може допомогти компенсувати камери нижчої якості.

Також технології Eye Tracking включають методику, яка відстежує відображення рогівки або рефлекс (CR) – яскравий «блиск» освітлення від сферичної поверхні рогівки. Такий механізм використовується для розрізнення рухів очей та голови

Існують системи Eye Tracking, що не потребують спеціального обладнання, наприклад, використання ІЧ-освітлення. Такі системи дозволяють використовувати звичайні веб-камери (Рисунок 1.5) та проводити Eye Tracking з великою кількістю учасників. Однак ці системи мають недолік, що проявляється в точності та якості даних.

Для систем Eye Tracking важливе калібрування – обчислення систематичних і біометричних характеристик людського ока на основі некаліброваної точки погляду (PoR). Необхідність калібрування в системах Eye Tracking можна пояснити тим, що існує різниця розміру очей, положення ямки та особливості фізіології кожної людини. Калібрування допомагає скорегувати погляд користувача системи Eye Tracking на фіксовані відомі точки в полі зору, що відображаються на екрані комп’ютера для системи Eye Tracking. Системи відстеження очей (мобільні та настільні) обчислюють відносне 3D-положення

ока (по відношенню до пристрою відстеження очей) і напрямок, у якому дивиться зіниця.

Існує ряд методів калібрування. Один з них використовує одну центральну точку, але він доволі рідко використовується. Зазвичай достатньо 5-9 точок для адекватної роботи системи Eye Tracking. Під час цього процесу відбувається послідовний збір даних для однієї або кількох точок у різних положеннях на екрані. Після того, як дані для калібрування зібрані, відбувається математичне перетворення між положенням очей (без врахування CR) та положенням погляду на кожну точку, що використовує система калібрування. На основі отриманих даних, алгоритм будує матрицю, що покриває всю область калібрування з інтерполяцією між кожною точкою. Чим більше точок використовується для калібрування, тим краща точність.

Незважаючи на існування передових методологій, системи Eye Tracking все ще мають певні функціональні обмеження, які варто враховувати при розробці цих систем. Дана система повинна мати безперешкодний огляд зіниці. Повіки та вії можуть перекривати простір зіниці, закриваючи огляд камери стеження за очима. Однак сучасні алгоритми виявлення зіниці можуть визначати центр, навіть якщо зіниця частково закрита.

Сонячне світло має широкий ІЧ-компонент, який може затемнювати зіницю та засліплювати камеру стеження за очима. Лише деякі системи Eye Tracking здатні працювати з сонячним світлом, хоча вони все ще потребують певного способу затемнення очей. Якщо користувач змушений примружитись через яскраве сонячне світло – система Eye Tracking втратить зіниці та не зможе їх відстежити через обструкцію (закупорку) зіниці.

1.3. Аналіз відомих технологічних рішень управління інтерактивним графічним інтерфейсом

Мова програмування Python має багато інструментів для створення графічних інтерфейсів.

Модуль Tkinter – популярна бібліотека Python для створення GUI з відкритим вихідним кодом. Це – один з основних інструментів для створення GUI.

Застосування методів Machine Learning (ML) – актуальне серед розробок сучасного ПЗ для розпізнавання облич. Існує багато технологічних методів, що використовують ML, серед яких OpenCV та Mediapipe.

OpenCV – популярна бібліотека з відкритим вихідним кодом, що надає широкий вибір інструментів для комп'ютерного зору та обробки зображень. Завдяки своїм потужним можливостям і зручному інтерфейсу OpenCV якнайкраще підходить для реалізації задач даного проекту.

Mediapipe – фреймворк машинного навчання для обробки часових рядів, наприклад відео, аудіо тощо. Mediapipe вимагає мінімальних ресурсів на відміну від інших енергоємних фреймворків машинного навчання.

Аналізуючи вищевикладене, OpenCV надає оптимізований набір алгоритмів та інструментів для обробки медіа включно з моделями Machine Learning (ML) та штучних нейронних мереж. В свою чергу, Mediapipe – гарний інструмент для обробки потокових медіа. До можливостей фреймворку входить розпізнавання обличчя. Для цієї задачі Mediapipe використовує так званий facemesh, що містить 468 орієнтирів обличчя (Рисунок 1.6).

Згідно офіційної документації Mediapipe [27], завдання Face Landmarker – виявлення орієнтирів обличчя на зображеннях та відео. Face Landmarker використовує моделі ML, що можуть працювати як із статичними зображеннями, так із відеопотоком. Після роботи вищезазначеного інструменту повертається об'єкт для кожного запуску. Цей об'єкт містить сітку обличчя (facemesh) для кожного виявленого в кадрі обличчя з відповідними координатами.

Для реалізації програмного комплексу, що забезпечує вільну інтеграцію особливих людей з сучасним ІТ, використано бібліотеку Tkinter як одну з найпопулярніших бібліотек Python для створення графічного інтерфейсу,

OpenCV для доступу до відеокамери та Mediapipe, оскільки два останніх інструменти утворюють гарний тандем.

Для демонстрації практичного застосування Eye Tracking на прикладі взаємодії графічного інтерфейсу засобами Tkinter використано Pyautogui – бібліотеку автоматизації Python, яка дозволяє керувати мишею та клавіатурою, що використовується для полегшення автоматизації руху миші та клавіатури для встановлення взаємодії з іншою програмою за допомогою сценаріїв Python.

1.4. Формалізація задачі досліджень та постановка часткових задач досліджень

Аналізуючи відомі підходи, що існують на даний момент щодо впровадження технології Eye Tracking, має вийти програмний комплекс, що поєднує відомі методологічні основи розробки програмних інтерфейсів та новітні технології Eye Tracking. Програмна система використовуватиме побутові засоби – веб-камеру, яка не містить інфрачервоного датчику на відміну від Eye Tracker від шведської компанії Tobii [38], що розробляє ПЗ для відстеження погляду людини.

Частковими задачами в даному дослідженні є:

1. Аналіз відомих методологічних основ управління інтерактивним графічним інтерфейсом в стеці технологій Computer Vision.
2. Розробка методики управління інтерактивним графічним інтерфейсом за технологіями Eye Tracking.
3. Розробка програмного комплексу управління інтерактивним графічним інтерфейсом за технологіями Eye Tracking, що включає такі складові:

інженерія вимог;

архітектурне проектування програмного комплексу;

програмна реалізація програмного комплексу.

Комплексне вирішення вищенаведених питань у поєднанні з новітніми технологіями Computer Vision роблять можливим розробку програмного

комплексу управління інтерактивним графічним інтерфейсом, що використовує технології Eye Tracking.

Програмний комплекс, що розробляється, має забезпечити основні дії з графічним інтерфейсом: рух віртуального курсора вздовж робочого вікна графічного інтерфейсу; активізація дії типу «натискання». Аби розширити практичну спрямованість розробленого ПЗ, передбачено додаткові функції, серед яких перебір вкладок інтерфейсу та відкривання контекстної інформації.

Технологічною основою для виконання проекту є відомий стек технологій Computer Vision, що поєднує теоретичні основи та алгоритми машинного навчання.

В результаті має бути розроблена методика управління інтерактивним графічним інтерфейсом з використанням технологій Eye Tracking, що входять до стеку технологій Computer Vision. Розроблена методологія повинна застосовуватись до конкретного програмного продукту.

Розроблений програмний продукт може мати практичне застосування у сферах управління зі звільненням/відсутності можливості застосування функцій моторики рук, серед яких:

– складні динамічні рухомі об’єкти – автомобілі, бойові комплекси тощо;

– виробниче устаткування із складним переліком органів управління;

– забезпечення належного рівня функціональності людей із особливими потребами, поранених тощо.

Програмна компонента має забезпечити безперешкодну інтеграцію людей з особливими потребами з сучасним світом ІТ, що відповідає основним цілям проекту.

РОЗДІЛ 2. РОЗРОБКА ПРОГРАМНОГО КОМПЛЕКСУ УПРАВЛІННЯ ІНТЕРАКТИВНИМ ГРАФІЧНИМ ІНТЕРФЕЙСОМ ЗА ТЕХНОЛОГІЯМИ EYE TRACKING

2.1. Розробка методики управління інтерактивним графічним інтерфейсом за технологіями Eye Tracking

Методи та технології Computer Vision дають змогу створити програмний комплекс керування інтерактивним графічним інтерфейсом за допомогою технологій Eye Tracking. Однак сучасні Eye Tracker, на кшталт від шведської компанії Tobii [22], використовують ІЧ-освітлення. Воно невидиме для людини та краще відбивається від зіниць, що підвищує точність роботи приладу.

Завдання проекту – розробити програмний комплекс без використання спеціалізованого обладнання. Веб-камера – єдине джерело, що надає координати положення очей для Eye Tracker. Побутові веб-камери не вимагають використання ІЧ-освітлення, але це впливає на точність роботи Eye Tracker.

Для кожної програми, що використовує взаємодію користувача з графічним інтерфейсом, калібрування – один із найважливіших етапів. Без нього неможливо правильно керувати GUI. Під час цього процесу обчислюються систематичні та біометричні характеристики ока людини на основі некаліброваної точки погляду (PoR). Отже, процес калібрування – це послідовний збір даних для однієї або кількох точок в різних положеннях на екрані. На основі записаних даних обчислюється функція корекції, що застосовується до некаліброваної PoR.

Зазвичай у процесі калібрування приймає участь 5-9 точок. Незалежно від кількості точок, яку використовує етап калібрування, кожна точка має відображатись досить довго, аби зібрати достатньо даних. Керуючись твердженням, що якість калібрування значно не покращиться за умови використання більше дев’яти точок, використано методику калібрування саме за

дев’ятьма точками (Рисунок 2.1)[2]. Дана методологія застосовує проективне перетворення, що використовується в гомографії.

Проективне перетворення – це геометричне перетворення, що відображає плоскі об’єкти на зображенні [14, с. 19-21]. Воно проектує кожний об’єкт зображення в еквівалентний об’єкт, залишаючи всі проективні властивості незмінними. Евклідова геометрія описує кути та форми об’єктів. Точка в евклідовому просторі задається парою координат *(x, y)*. До цієї пари можна додати масштабний коефіцієнт *k*, наприклад 1 *(x, y, 1)*, але значення точки від цього не зміниться. Додавання коефіцієнта до пари координат *(x, y)* призводить до гомогенності координат, що використовується при обчисленні нової координати за допомогою матриці гомографії (Рівняння 2.1).

В евклідовій геометрії всі точки рівнозначні, а весь простір – однорідний. При додаванні точки в евклідовий простір за початок координат обирається будь-яка точка серед наявних. Аби перетворити координати гомогенного простору, необхідно застосувати лінійне перетворення однорідних координат. Лінійне перетворення представлено множенням матриці на гомогенні координати точки (Рівняння 2.2 - 2.4).

Калібрування застосовує принцип гомографії. Вона характеризується матрицею *H*, яка має вигляд:

$$H = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{pmatrix}, \qquad (2.1)$$

де a, e – масштабування по *Ox* та *Oy* відповідно;

b, d – зсув по осях (разом із коефіцієнтами a, e впливають на обертання);

c, f – зсув по осях;

g, h –зміна перспективи.

Як видно з рівняння 2.1, матриця гомографії *H* має 8 ступенів свободи (DoF) – це значення, які обчислюються на основі двох наборів даних, що використовуються при розрахунку матриці гомографії:

[2] Рисунки до Розділу 2 наведені в Додатку Б.

1. точки одного зображення – $x_i \in P^2$, де P^n – проективний простір із гомогенними координатами;

2. відповідні точки (corresponding points) $x_i' \in P^{2\prime}$.

Крайній правий елемент третього рядка матриці H завжди дорівнює 1.

Використовуючи матрицю H, можна описати зміну координат одного об'єкта між двома фреймами, якщо вона викликана рухом камери. Нові координати обчислюються за формулою в канонічному вигляді:

$$x' = \frac{ax+by+c}{gx+hy+1}\ ,\ y' = \frac{dx+ey+f}{gx+hy+1} \qquad (2.2)$$

або в матричному вигляді:

$$k\cdot(x', y', 1) = H\cdot(x, y, 1) \qquad (2.3)$$

де k – масштабний коефіцієнт, $k \neq 0$;

(x', y') – нові координати точки;

H – матриця гомографії;

(x, y) – координати точки Евклідового простору;

чи

$$x_i' = H\cdot x_i, \qquad (2.4)$$

де x_i' – координати внаслідок проективного перетворення;

H – матриця гомографії 3x3;

x_i – координати точок, що підлягають геометричному перетворенню.

Для обчислення нової координати гомографія, як і афінні перетворення, використовує однорідні (гомогенні) координати: до координат точки x_i додається масштабний коефіцієнт, переважно 1:

$$x_i\ (x, y, 1) \qquad (2.5)$$

Для обчислення координат x_i' використовуються гомогенні координати (формула 2.5). Варто зазначити, що вектори x_i' та $H\cdot x_i$ не рівні, хоча мають однаковий напрямок. Два вектори можуть відрізнятись за величиною масштабних коефіцієнтів, що не дорівнюють 0.

Рисунок 2.2 демонструє, що гомографія – це проективне перетворення на площині, що умовно складається з таких операцій:

1. вхідне зображення, що підлягатиме подальшому гомографічному перетворенню;
2. зображення після корекції перспективи зображення;
3. корекція зсувної деформації;
4. поворот зображення;
5. розтягнення та масштабування.

Тобто, матриця гомографічного перетворення дозволяє однією дією виконати ряд перетворень, що відносяться до корекції перспективи та афінних перетворень , в результаті яких паралельні прямі, що на площині, переходять самі в себе. Матриця *H* містить всі необхідні значення, як то, вектор переміщення, масштабування, повороту, деформації.

Результат алгоритму для обчислення гомографії залежить від положення камери, що використовується під час проведення Eye Tracking.

Алгоритм калібрування застосовує структуру часового слота (time slot), тобто запис координат кожної точки відповідає номеру слота. Кожен часовий слот триває чотири секунди, до появи наступної точки в певному положенні екрана. Після закінчення часу слота відбувається переміщення реперної точки. Реперна точка – це певна позиція об’єкта в координатах монітора. Отримані дані накопичуються в масив, значення медіани якого буде використано в матриці перетворень.

Візуалізація даних демонструє, що результат калібрування системи Eye Tracking з використанням матриці гомографії не є досконалим. Викривлення сформованих точок викликано дисторсією, що властиво оптичним системам дешевих веб-камер. Причиною виникнення дисторсії є відмінність лінійного збільшення різних частин зображення [15]. З курсу оптики відомо, що викривлення зображення предмета залежить від коефіцієнта лінійного збільшення з віддаленням елементів зображення від оптичної осі лінзи. Дисторсія буває бочкоподібною (опуклою – лінії, особливо по краях кадру, вигинаються назовні) або подушкоподібною (увігнутою – вигин ліній

направлений до центра кадру) (Рисунок 2.3). Такі викривлення неможливо виправити, лише застосувавши матрицею гомографії.

Для накопичення даних, що формуються в процесі розрахунку положення курсора, обрано кільцеву чергу. Принцип роботи кільцевої черги – FIFO (First In – First Out). Відмінність кільцевої черги полягає у записі нового елемента на місце першого. Кільцева черга – структура даних, що широко використовується в різних алгоритмах.

Така властивість кільцевої черги є основою для реалізації алгоритму, що зменшує некеровані рухи курсора, зумовлені постійними рухами людської зіниці. Значення погляду користувача заноситься в кільцеву чергу. Такий підхід дозволяє робити аналіз та прогноз наступного значення, використовуючи алгоритми інтерполяції (наприклад, МНК) на основі попередніх значень веб-камери.

Для підвищення точності руху курсора впроваджено методику порівняння на основі алгоритмів найменших квадратів (МНК) та центроїда, що посідають провідне місце в статистичних методах фільтрації випадкових коливань.

Один з алгоритмів задля досягнення більш плавного руху курсору шукає центроїд черги. В евклідовій геометрії центроїд (центр мас) – середнє арифметичне скінченної множини точок.

Інший спосіб для зменшення аномальних рухів курсору – МНК. Це математичний метод, який мінімізує суму квадратів різниць між дискретними значеннями та згладженими параметрами [4]. Даний фільтр дозволяє спостерігати за зашумленим процесом та передбачати його поведінку.

Головна задача МНК – визначення коефіцієнтів лінійної залежності, при яких функція приймає найменше значення:

$$F(a, b) = \sum_{i=1}^{n} (y_i - (ax_i + b))^2 \Rightarrow min \qquad (2.6)$$

де a, b – коефіцієнти прямої, що екстраполює експериментальні дані;

y_i – функція, що описує характер експериментальних даних.

Алгоритм МНК в матричній формі [4] має такий вигляд:

$$\overline{Y} = (A^T A)^{-1} A^T \overline{B} \qquad (2.7)$$

де $\overline{Y}$ – вектор параметрів апроксимуючої функції;

A – матриця значень модельних функцій;

$\overline{B}$ – вектор виміряних параметрів.

За допомогою такого алгоритму можна знайти пряму, яка рівновіддалена від усіх точок. Лінійна апроксимація методом найменших квадратів – це інструмент, який дозволяє передбачати поведінку зашумленого значення за певний період, адже в результаті МНК утворюється рівняння прямої.

Отже, методика розробки програмного комплексу управління графічним інтерфейсом за технологіями Eye Tracking включає низку послідовних етапів:

1. дев’ятиточкове калібрування системи з використанням матриці гомографії (вирази (2.1)-(2.5));
2. відстеження PoR людини, використовуючи інструменти OpenCV, наприклад, підготовка кадра для розпізнавання форми, що схожа на шуканий об’єкт. Процес відстеження ока використовує FaceMesh від Mediapipe. Дана технологія використовує конвеєр із двох нейронних мереж для визначення 3D-координат. Координати, що виявив FaceMesh на людському обличчі, нормалізуються в межах [0, 1];
3. розробка методики кільцевої черги (вирази (2.6)-(2.7));
4. вибір алгоритму фільтрації на основі кільцевої черги (МНК, центроїд), що зменшує некеровані рухи курсора, шляхом пошуку мінімального середньоквадратичного відхилення.

Розроблена методологічна основа, що поєднує технології Computer Vision та Data Science, є основою для реалізації програмного комплексу управління інтерактивним інтерфейсом за технологіями Eye Tracking.

2.2. Розробка програмного комплексу управління інтерактивним графічним інтерфейсом за технологіями Eye Tracking

2.2.1. Інженерія вимог

Інженерія вимог програмного комплексу для управління інтерактивним графічним інтерфейсом за допомогою технологій Eye Tracking базується на аналізі функціональних вимог до нього.

Функціональні вимоги мають забезпечити безперешкодну взаємодію людей з особливими потребами з інтерактивним інтерфейсом, що не вимагає застосування традиційних засобів вводу, на кшталт маніпулятора «миші».

Виходячи з цього можна виділити такі функціональні вимоги програмного комплексу для управління інтерактивним графічним інтерфейсом за допомогою Eye Tracking, що базуються на відомому стеці технологій Computer Vision:

- врахування фізіологічних особливостей кожної людини;
- отримання та обробка зображення інструментами OpenCV;
- фільтрація зображення інструментами Numpy;
- забезпечення швидкого та точного процесу калібрування;
- збір даних для аналізу погляду людини;
- ідентифікація обличчя інструментами Mediapipe;
- точне відстеження погляду очей користувача інструментами Mediapipe;
- співставлення координат від FaceMesh Mediapipe та реальних координат;
- забезпечення точності, достатньої для адекватної роботи розробленого комплексу із застосуванням технологій Eye Tracking;
- розрахунок Евклідової відстані як основної метрики для визначення найкращого алгоритму фільтрації системи Eye Tracking;
- відображення результатів калібрування задля подальшої роботи;
- забезпечення швидкодії програмного комплексу;
- вибір бажаної операції за допомогою технологій Eye Tracking;

- рух віртуального курсора вздовж робочого вікна графічного інтерфейсу;
- активізація дії типу «натискання»;
- перебір вкладок інтерфейсу;
- відображення контекстної інформації;
- масштабованість до різного виду ПЗ;
- кросплатформенність додатку.

Використовуючи відомі методології та технології Computer Vision, можна розробити прикладний продукт – програмний комплекс для управління інтерактивним графічним інтерфейсом за допомогою технологій Eye Tracking відповідно до сформованого функціоналу.

2.2.2. Архітектурне проектування програмного комплексу

Архітектура програмного комплексу, що використовує технологію Eye Tracking, має модульну структуру. Під час розробки будь-якого ПЗ модульна архітектура має переваги: від простоти та динамічності коду до його високого рівня безпеки. Модульний підхід до розробки програмного забезпечення дозволяє розбити всі компоненти програми на модулі – незалежні компоненти. Кожний модуль виконує окрему функцію програмної системи, при цьому маючи власну логіку та інтерфейс для взаємодії з іншими модулями. Однією з переваг модульної архітектури є можливість повторного використання коду. Наприклад, тренувальна гра «Sunny Bunny», яка надає порівняльну характеристику алгоритмів фільтрації, та основний додаток, що дає можливість керувати Інтернет-ресурсами однієї з чотирьох категорій, використовують модуль, що відповідає за етап калібрування. За допомогою модульної архітектури розроблений програмний комплекс може легко масштабуватись шляхом додавання нових функцій.

Програмний комплекс складається з п’яти основних модулів, що відображають практичне застосування Eye Tracking. Структурна схема на рисунку 2.4 відображає архітектурне проектування програмної системи.

Етап калібрування – найважливіший модуль програмного комплексу Eye Tracking. Оскільки керування проводиться очима, потрібно визначити правильне місцерозташування PoR користувача. Без даного етапу програмний комплекс не працюватиме адекватно. Його складовими елементами є:

– формування поля для дев'ятиточкового калібрування – відповідає за візуалізацію поля з маркером, що з’являється в одному з дев’яти положень кожні чотири секунди;

– формування маркера, що буде змінювати положення під час калібрування;

– налаштування веб-камери та FaceMesh для отримання координат погляду;

– формування матриці гомографії для визначення фактичного положення курсору. Вона використовується в наступних модулях програмного комплексу.

На даному етапі користувач системи Eye Tracking користується традиційними засобами управління комп’ютером, наприклад мишею, адже калібрування передбачає підготовку даних для проведення процедури Eye Tracking. Необхідно зазначити, що калібрування включає в себе технологію визначення координат погляду користувача.

Перед проведенням тренувальної гри користувачу надається можливість обрати її параметри. Етап вибору параметрів включає:

– формування полей для вибору параметрів гри "Catch Sunny Bunny";

– налаштування веб-камери та FaceMesh для отримання координат погляду;

– вибір параметрів гри "Catch Sunny Bunny" – колір фону гри та довжина кільцевої черги за допомогою технології Eye Tracking;

– запуск основного циклу гри "Catch Sunny Bunny".

Таким чином, обравши дані параметри, можна розпочати гру з сонячним зайчиком.

Гра «Catch Sunny Bunny» надає можливість отримати порівняльну характеристику алгоритмів фільтрації на основі кільцевої черги технологіями Data Science. Вона складається з таких кроків:

– формування поля для гри "Catch Sunny Bunny";

– формування «сонячного зайчика» у випадковій точці екрану;

– формування «кошенят», що відображають результат одного з алгоритмів фільтрації – МНК або Центроїд;

– налаштування веб-камери та FaceMesh для отримання координат погляду;

– аналіз і вибір алгоритму фільтрації та довжини черги.

Перед запуском основної програми користувачам системи надається можливість обрати алгоритм фільтрації за результатами тренувальної гри. Етап вибору параметрів програми включає:

– формування полей для вибору параметрів програми "Eye Tracker";

– налаштування веб-камери та FaceMesh для отримання координат погляду;

– вибір параметрів програми "Eye Tracker" за допомогою технології Eye Tracking – алгоритми фільтрації, що застосовується під час керування GUI;

– запуск основного циклу програми "Eye Tracker".

Етап вибору операцій демонструє практичне застосування Eye Tracking. Користувач розробленої системи має змогу керувати Інтернет-ресурсами однієї з чотирьох категорій за допомогою даної технології. Він складається з:

– формування графічного інтерфейсу, що відображає чотири операції для забезпечення життєдіяльності людини;

– калібрування системи Eye Tracking з використанням матриці гомографії;

– формування даних для здійснення процесу Eye Tracking – відбувається нормалізація даних з використанням алгоритмів фільтрації;

– вибір операції за допомогою технології Eye Tracking.

Слід зазначити, що всі етапи використовують інструменти для визначення координат погляду.

Механізм роботи програмного комплексу Eye Tracking наведений на блок-схемі алгоритму, що на рисунку 2.5.

Виділений фрагмент схеми виконує калібрування програмної системи. Індентифікація та відстеження PoR відбувається в безперервному циклі за допомогою FaceMesh від Mediapipe, що використовує високоточні орієнтири обличчя. Дані, які надходять з камери, накопичуються в двох масивах *Ox* та *Oy*, що містять відповідні *x* та *y* для кожної точки. Останні зберігають нормалізовані

значення в межах [0, 1]. Після закінчення чергового часового відрізка знаходиться медіана, яка використовується при обчисленні матриці гомографії, методом numpy.median.

Особливістю алгоритму калібрування є запис координат, що починається з другої секунди після демонстрації маркера, хоча Mediapipe аналізує відеопотік безперервно. Це дає змогу перевести погляд з попередньої точки, не спотворюючи дані проміжними результатами.

Результатом роботи даного алгоритму є матриця *H*, що використовується іншими модулями системи. Вони застосовують подібну методологію для визначення та використання координат зіниці користувача.

2.2.3. Програмна реалізація програмного комплексу

Використовуючи архітектуру, розроблену на етапі архітектурного проектування програмної компоненти Eye Tracking, ПЗ створено мовою високого рівня програмування Python з використанням парадигм об'єктно-орієнтованого програмування. Структура програми у вигляді функціональної схеми (діаграми класів) представлена на рисунку 2.6.

ПЗ використовує зовнішні бібліотеки, що надає Python. Для використання даного програмного комплексу необхідно інсталювати потрібні залежності. Після встановлення всіх бібліотек, що використовує даний проект, технології Eye Tracking стануть доступними. Для цього необхідно використати командну стрічку:

```
PYTHON RUN.PY
```

де `RUN.PY` – файл запуску всіх класів, що є GUI додатками на основі бібліотеки Tkinter. Вони запускаються послідовно відповідно до логіки функціонування програмного комплексу. Програмна система не використовує зовнішні ключі, що вводяться за допомогою консольної утиліти Windows.

Система Eye Tracking складається з модулів, що на рисунку 2.6, в порядку їх застосування (Додаток В).

РОЗДІЛ 3. ПРИКЛАДНІ АСПЕКТИ ЗАСТОСУВАННЯ ПРОГРАМНОГО КОМПЛЕКСУ УПРАВЛІННЯ ІНТЕРАКТИВНИМ ГРАФІЧНИМ ІНТЕРФЕЙСОМ ЗА ТЕХНОЛОГІЯМИ EYE TRACKING

3.1. Демонстрація практичних можливостей розробленого програмного комплексу

Інтерактивний інтерфейс є складовою програмного комплексу за технологіями Eye Tracking, що базується на відомому стеці технологій Computer Vision. Основне завдання розробленого ПЗ – дати можливість людям, що мають проблеми з моторикою рук, повноцінно керувати комп’ютером, не використовуючи традиційні пристрої вводу, на кшталт емулятора «миші». В даному випадку, очі – головний «пристрій вводу» задля керування програмною системою.

Програмний комплекс – це проект, написаний мовою високого рівня Python. Його архітектура відповідає правилам архітектурного проектування програмних продуктів. Проект «Eye Tracking», написаний з використанням відомих парадигм об’єктно-орієнтованого програмування, має модульну структуру. Вона містить такі складові (Рисунок 3.1)[3]:

– інфраструктурне оточення, що надає Python;

– модулі для обчислень;

– каталоги для збереження зображень, що використовуються як піктограми для основних операцій GUI.

Файл `RUN.PY` призначений для запуску програмного комплексу, що використовує технології Eye Tracking (Рисунок 3.2). Програмний скрипт визначає порядок запуску всіх класів відповідно до логіки функціонування програмної системи.

Клас CalibrationApp, що відповідає за етап калібрування – найважливіший у функціонуванні програми (Рисунок 3.3). Без нього програмний комплекс не

[3] Рисунки до Розділу 3 наведені в Додатку Г.

працюватиме коректно. Даний етап визначає координати положення зіниці людини та перетворює їх в координати монітора для правильної роботи програмного комплексу з використанням технологій Eye Tracking.

Метод calibrate() здійснює процес калібрування за дев’ятьма точками. Воно має структуру часового слота. За замовчуванням, маркер для калібрування, що змінює своє положення дев’ять разів, жовтого кольору (Рисунок 3.4). Зміна кольору маркера на червоний свідчить про початок запису координат для калібрування. Збір даних для даного процесу починається з другої секунди time slot. Дві секунди відведено на переведення погляду користувача додатку.

Клас Controller відповідає за вибір параметрів для гри «Catch Sunny Bunny» (Рисунок 3.5). Користувач може керувати додатком за допомогою як традиційних засобів вводу, так і за допомогою технології Eye Tracking.

Користувачеві Eye Tracking доступні три види черг довжиною 5 елементів, 15, 21 (Рисунок 3.6). Вибір черги впливає на чутливість та швидкість реакції курсора. Використовуючи механізм кільцевої черги, можна зменшити некеровані рухи курсора, зумовлені постійними рухами людської зіниці. Також користувач може обрати колір фону гри, який не впливає на її хід.

Засобами Canvas забезпечено візуалізацію станів операцій:

– зелена рамка – операція активна;
– червона рамка – операція неактивна.

Кнопка «START Game» резюмує обрані параметри (Рисунок 3.7). Повідомлення може бути:

– помилковим, якщо користувач не обрав жодного або один з запропонованих параметрів;

– схвальним, якщо користувач обрав обидва параметри, що необхідні для проведення гри «Catch Sunny Bunny». Після цього тренувальна гра починається.

Клас GameSB формує поле для гри «Catch Sunny Bunny» на основі параметрів, що отримані від попереднього етапу (Рисунок 3.8).

Основна задача гри – слідкувати за «сонячним зайчиком», що з’являється у випадковому місці на екрані (Рисунок 3.9). Гра пропонує три «кошеняти»:

1. нефільтроване значення;
2. центроїд;
3. МНК.

Два останніх демонструють роботу алгоритмів фільтрації на основі кільцевої черги.

Результатом пройденої гри є визначення найкращого алгоритму фільтрації шляхом проведення статистичного аналізу інструментами Data Science (Рисунок 3.10). Статистичний аналіз даних полягає в обчисленні мінімального середньоквадратичного відхилення для трьох алгоритмів фільтрації. Серед них обирається мінімальне.

Клас EyeTrackController відповідає за вибір алгоритму фільтрації для GUI, що використовує Eye Tracking (Рисунок 3.11). Механізм взаємодії з користувачем подібний до вибору параметрів для гри «Catch Sunny Bunny».

Функціонал даного контролера пропонує два алгоритми фільтрації на основі кільцевої черги (Рисунок 3.12):

1. центроїд;
2. МНК.

За домогою цих алгоритмів можна зменшити помилкові рухи курсора, викликані постійними рухами людської зіниці.

Вибір алгоритму фільтрації є важливим. Фільтруючи значення координат погляду PoR, можна досягти плавного руху курсора миші. Таким чином, керування поглядом стає подібним до керування мишею. Також контролер передбачає повідомлення, якщо алгоритм фільтрації не обраний (Рисунок 3.13).

Клас EyeTracker реалізує логіку GUI, що керується технологіями Eye Tracking (Рисунок 3.14).

GUI пропонує користувачеві чотири базових операції для нормального способу життя (Рисунок 3.15):

1. «Харчування»;

2. «Навчання»;
3. «Побут»
4. «Розваги».

За замовчуванням, всі операції – неактивні.

Активна операція характеризується зеленим кольором відповідного фрейма (Рисунок 3.16). Неактивна – червоного кольору відповідно.

Після двох секунд відбувається «фліп» активної операції. Обравши кнопку з відповідним Інтернет-ресурсом, можна ним керувати, використовуючи технології Eye Tracking (Рисунок 3.17).

Етап тестування розробленого ПЗ в процесі експлуатації програмної системи довів, що:

1. за допомогою фреймворків Mediapipe та OpenCV система може правильно визначати положення PoR в режимі реального часу;
2. використовуючи один з двох алгоритмів фільтрації можна зменшити аномальні рухи курсора, зумовлені постійним рухом людської зіниці.

Отже, керування інтерактивним інтерфейсом за допомогою технологій Eye Tracking нагадує керування звичайною мишею.

Демонстрація практичних можливостей та показники ефективності (у вигляді СКВ) свідчать про успішну реалізацію програмного комплексу управління інтерактивним графічним інтерфейсом за технологіями Eye Tracking.

Використовуючи програмну систему можна здійснити автоматизацію процесу управління інтерактивним графічним інтерфейсом із впровадженням технологій Eye Tracking, що є основною метою досліджень.

3.2. Програмна документація на програмне забезпечення

Проектування ПЗ дозволяє дотримуватись певного графіку та послідовності створення проекту задля отримання якісного продукту на вимогу замовника.

Документація має бути структурованою та відповідати вимогам міжнародних стандартів. На сьогодні відомий стандарт в Україні ДСТУ4302:2004 (ISO/IEC 6592:2000 MOD – міжнародний стандарт даного документа) [3], що визначає вимоги щодо розробки якісної програмної документації.

Технічна документація вимагає безперервного моніторингу її якості та наповнення. Правильно задокументоване ПЗ впливає на зручність та доступність послуг користувачам системи. Ідентифікація всіх типів користувачів – головний принцип документації ПЗ. Технічна документація програмної системи розроблена відповідно до вимог користувачів, зважаючи на їх досвід роботи з програмою. Документація до програмного комплексу управління інтерактивним графічним інтерфейсом за технологіями Eye Tracking характеризується: точністю, предметністю, об’єктивністю, конкретністю.

Документація до ПЗ чітко структурована, написана простою і доступною мовою та має логічну зв’язаність. Технічна документація до ПЗ за технологіями Eye Tracking зберігає:

– стиль (зрозуміле для усіх формулювання вимог);
– чітко визначену структуру документа;
– гнучкість (просте додавання нових вимог).

Оформлення є важливою рисою написання документації для розробленого програмного комплексу. Її текст поділяється на розділи та підрозділи. Для наочності застосовано різні форми шрифту, виділення. Також використано графічні форми подачі інформації у вигляді графіків та малюнків (зокрема, у формі скріншотів).

Розроблено посібник користувача, в якому описуються можливості програми та принципи роботи з нею (Додаток Д). До роботи додається лістинг програми (Додаток Ж).

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Складність сучасних комп’ютерних систем призвела до зростання зацікавленості в альтернативних методах керування графічними інтерфейсами.

Існуючі методологічні та технологічні рішення допомогли визначити головні вимоги щодо використаного устаткування. Це – звичайна веб-камера, яка не потребує ІЧ-освітлення на відміну від інших сучасних Eye Tracker. Однак її використання має недоліки, що позначаються на точності системи. Для покращення точності позиціювання, використано калібрування з використанням дев'яти точок, що базується на гомографічному перетворенні. Це дає змогу покращити якість керування до прийнятного рівня.

Вивчення предметної області магістерського проекту визначило основні функціональні вимоги до програмного комплексу: 1) врахування фізіологічних особливостей кожної людини; 2) ідентифікація обличчя інструментами Mediapipe; 3) точне відстеження погляду очей користувача інструментами Mediapipe; 4) забезпечення швидкодії програмного комплексу; 5) вибір бажаної операції за допомогою технологій Eye Tracking; 6) рух віртуального курсора вздовж робочого вікна графічного інтерфейсу; 7) активізація дії типу «натискання»; 8) перебір вкладок інтерфейсу; 9) відображення контекстної інформації.

Програмний комплекс забезпечує процес калібрування за дев’ятьма точками, розробки кільцевої черги, тренувальної гри, в результаті якої відбувається статистичний аналіз алгоритмів фільтрації за технологіями Data Science.

Програмна система реалізує процеси виявлення та ідентифікації погляду людини шляхом обробки відеопотоку в режимі реального часу. Використовуючи нейронну мережу, що надає Mediapipe, формуються дані для калібрування.

Запропонована методологія нормалізації місцерозташування зіниці людини базується на проективному перетворенні, що включає в себе матрицю гомографії. Такий метод робить можливим застосування дев'ятиточкового

калібрування, що може легко модифікуватись у разі збільшення або зменшення кількості точок калібрування.

Завдяки кільцевій черзі зменшуються аномальні рухи курсора, що зумовлені постійним рухом людської зіниці.

При загальному тестуванні програмного комплексу управління інтерактивним графічним інтерфейсом за технологіями Eye Tracking виявилось, що: 1) за допомогою фреймворків Mediapipe та OpenCV система здатна правильно визначати положення PoR в режимі реального часу; 2) можна зменшити аномальні рухи курсора, зумовлені постійним рухом людської зіниці, застосовуючи алгоритми фільтрації.

Для супроводу ПЗ з використанням технологій Eye Tracking оформлено якісну документацію, що є запорукою подальшого розвитку проекту. Документація структурована, точна, предметна, об’єктивна та конкретна. Використовуючи її, користувачі, що мають різний досвід роботи, можуть з легкістю керувати ним або інтегрувати в інші проекти.

Розроблений програмний комплекс управління інтерактивним графічним інтерфейсом з використанням Eye Tracking, що базується на технологіях Computer Vision, використовує останні досягнення в галузі нейронних мереж та обробки зображень.

Програмна система буде корисна людям з обмеженими можливостями інтегруватись в сучасний світ.

На основі даного проекту можна розробити систему керування не лише графічним інтерфейсом. Інтегрування Eye Tracking в технології розумних речей, інтернет-речей (SmartThings, IoT) тощо. Вони нададуть нових властивостей звичним пристроям. Одна з таких ідей полягає в створенні «розумної» смарт-системи, що буде використовувати технологію Eye Tracking. Побутові та розважальні пристрої отримають змогу керуватися та реагувати на погляд користувача, краще планувати режими роботи. Зрештою, це призведе до економії ресурсів, покращенню умов життя та емоційного стану людей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Глібко О. А. Створення моделей та сцен у тривимірному середовищі : навч. посіб. / ред.: М. О. Максимова, І. П. Гречка. Харків : НТУ «ХПІ», 2018. 132 с.

2. Конвенція Організації Об’єднаних Націй про права людей з інвалідністю (неофіційний стислий виклад) - Посібник з освіти в області прав людини за участі молоді - www.coe.int. *Посібник з освіти в області прав людини за участі молоді*. URL: https://www.coe.int/uk/web/compass/convention-on-the-rights-of-persons-with-disabilities (дата звернення: 03.12.2023).

3. Настанови щодо документування комп'ютерних програм (ISO\IEC 6592:2000, MOD) ДСТУ 4302:2004. Київ : Держспоживстандардат України, 2005. 34 с.

4. Писарчук О.О., Харченко В.П. Нелінійне та багатокритеріальне моделювання процесів у системах керування рухом: монографія. Київ: Нац. авіац. ун-т, 2014. 372 с.

5. Скиба О. П. Комп'ютерна графіка : конспект лекцій. Тернопіль : Терноп. нац. техн. ун-т ім. Ів. Пулюя, 2019. 88 с.

6. Тотосько О. В., Микитишин А. Г., Стухляк П. Д. Комп'ютерна графіка : навч. посіб. в 2-х кн. Тернопіль : Терноп. нац. техн. ун-т ім. Ів. Пулюя, 2017. 304 с.

7. A Guide to Selecting Eye Tracking Solutions : Електрон. посіб. Smart Eye / ebook. 18 с. URL: https://smarteye.se/technology/eye-tracking/ (дата звернення: 07.12.2023).

8. Gonzalez R. C. Digital image processing. Reading, Mass : Addison-Wesley Pub. Co., Advanced Book Program, 1977.

9. Harezlak K., Kasprowski P., Stasch M. Towards accurate eye tracker calibrations – methods and procedures. м. Gliwice. Gliwice, 2014. С. 1073–1078.

10. Jia S. Real-time movement analysis for biometrics and driving safety : Dissertation. Boston, 2019.

11. Krishna R. Computer Vision: Foundations and Applications. Stanford : Stanford University, 2017. 213 с.

12. Shapiro L. Computer Vision and Image Processing. Elsevier Science & Technology Books, 1992. 638 с.

13. Solem J. E. Programming Computer Vision with Python: Tools and Algorithms for Analyzing Images. O'Reilly Media, Incorporated, 2012. 264 с.

14. Zisserman A., Hartley R. Multiple View Geometry in Computer Vision. 2-ге вид. Cambridge University Press, 2004. 672 с.

15. Дисторсія. Приклад на автопортретах. – Цікаво про фотографію. Цікаво про фотографію. URL: https://martatrotsiuk.com/archives/1245 (дата звернення: 17.11.2023).

16. Інфрачервоне підсвічування: чи потрібне сучасній камері? – Новоград.City. *Новоград.City*. URL: https://novograd.city/articles/252306/infrachervone-pidsvichuvannya-chi-potribne-suchasnij-kameri (дата звернення: 16.11.2023).

17. Eye Gaze Tracking: Applications, Techniques, and Key Metrics - Datagen. *Datagen*. URL: https://datagen.tech/guides/face-recognition/eye-gaze-tracking/ (дата звернення: 16.11.2023).

18. Eye Tracking: What Is It For And When To Use It - Usability Geek. *Usability Geek*. URL: https://usabilitygeek.com/what-is-eye-tracking-when-to-use-it/ (дата звернення: 16.11.2023).

19. Eye Tracking: What Is It, Usage, Benefits + How It Works?. *QuestionPro*. URL: https://www.questionpro.com/blog/eye-tracking/ (дата звернення: 16.11.2023).

20. GitHub - PacktPublishing/Artificial-Intelligence-with-Python: Code repository for Artificial Intelligence with Python, published by Packt. *GitHub*. URL: https://github.com/PacktPublishing/Artificial-Intelligence-with-Python (дата звернення: 16.12.2023).

21. GitHub - rasbt/python-machine-learning-book-3rd-edition: The "Python Machine Learning (3rd edition)" book code repository. *GitHub*. URL:

https://github.com/rasbt/python-machine-learning-book-3rd-edition (дата звернення: 20.12.2023).

22. Global leader in eye tracking for over 20 years. *Global leader in eye tracking for over 20 years - Tobii*. URL: https://www.tobii.com/ (дата звернення: 16.11.2023).

23. Home. *OpenCV*. URL: https://opencv.org/ (дата звернення: 16.12.2023).

24. Kaggle: Your Machine Learning and Data Science Community. *Kaggle: Your Machine Learning and Data Science Community*. URL: https://www.kaggle.com/ (дата звернення: 16.12.2023).

25. Keras: Deep Learning for humans. *Keras: Deep Learning for humans*. URL: https://keras.io/ (дата звернення: 16.12.2023).

26. Learn Object Tracking in OpenCV Python with Code Examples - MLK - Machine Learning Knowledge. *MLK - Machine Learning Knowledge*. URL: https://machinelearningknowledge.ai/learn-object-tracking-in-opencv-python-with-code-examples/ (дата звернення: 16.12.2023).

27. MediaPipe Solutions API reference | Google for Developers. *Google for Developers*. URL: https://developers.google.com/mediapipe/api/solutions (дата звернення: 16.11.2023).

28. Mento M. A. Learn here what are the different eye-tracking techniques and methods to record eye movements accurately, how to calibrate, and what are the limitations. | Bitbrain. *Bitbrain*. URL: https://www.bitbrain.com/blog/eye-tracking-technology (дата звернення: 16.11.2023).

29. Mikac M. EyeLogic - Blog - What is Calibration and why we need it. *EyeLogic*. URL: https://www.eyelogicsolutions.com/what-is-calibration-and-why-we-need-it/ (дата звернення: 16.11.2023).

30. NumPy documentation – NumPy v1.26 Manual. *NumPy*. URL: https://numpy.org/doc/stable/index.html (дата звернення: 17.11.2023).

31. Object Tracking OpenCV Python | Hack The Developer. *Hack The Developer*. URL: https://hackthedeveloper.com/object-tracking-opencv-python/ (дата звернення: 16.12.2023).

32. OR-Tools Google for Developers. *Google for Developers*. URL: https://developers.google.com/optimization (дата звернення: 16.12.2023).

33. Python pyautogui Library - Javatpoint. *www.javatpoint.com*. URL: https://www.javatpoint.com/python-pyautogui-library (дата звернення: 16.11.2023).

34. Rivkin M. A Complete Review of the OpenCV Object Tracking Algorithms. *BroutonLab Blog*. URL: https://broutonlab.com/blog/opencv-object-tracking (дата звернення: 16.12.2023).

35. Scapy. *Scapy*. URL: https://scapy.net/ (дата звернення: 16.12.2023).

36. TensorFlow. *TensorFlow*. URL: https://www.tensorflow.org/ (дата звернення: 16.12.2023).

37. The Global Leader in Human Insight AI. *Smart Eye*. URL: https://smarteye.se/ (дата звернення: 07.12.2023).

38. Tobii - The global leader in eye tracking since 2001. *Global leader in eye tracking for over 20 years - Tobii*. URL: https://www.tobii.com/company/this-is-tobii (дата звернення: 16.11.2023).

ДОДАТОК А

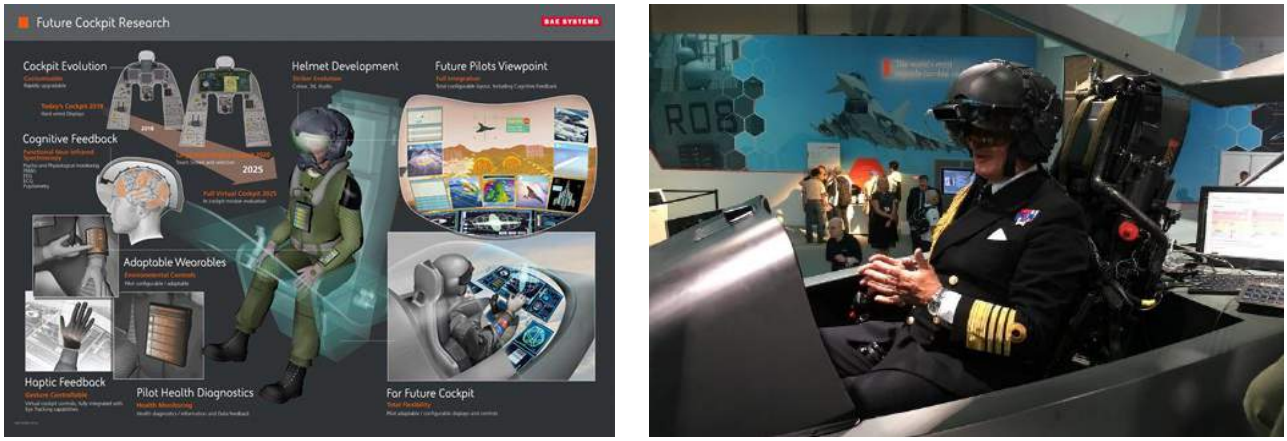


Рисунок 1.1. Технологія «wearable cockpit»

Рисунок 1.2. Електромагнітна склеральна котушка

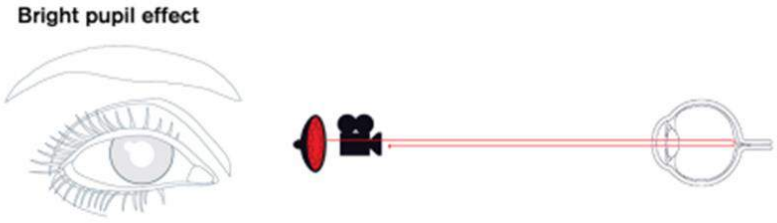


Рисунок 1.3. Eye Tracker світлої зіниці

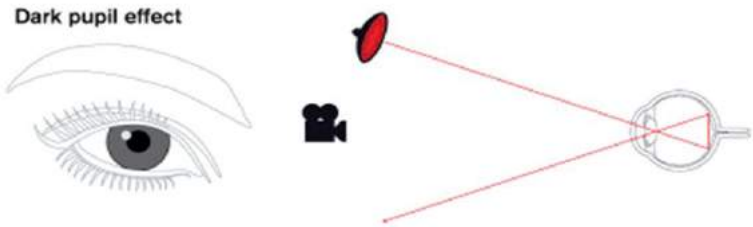


Рисунок 1.4. Eye Tracking темної зіниці

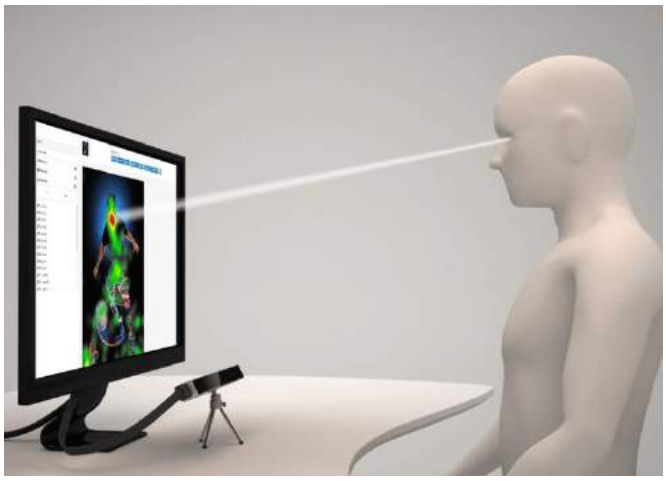


Рисунок 1.5. Системи веб-камер

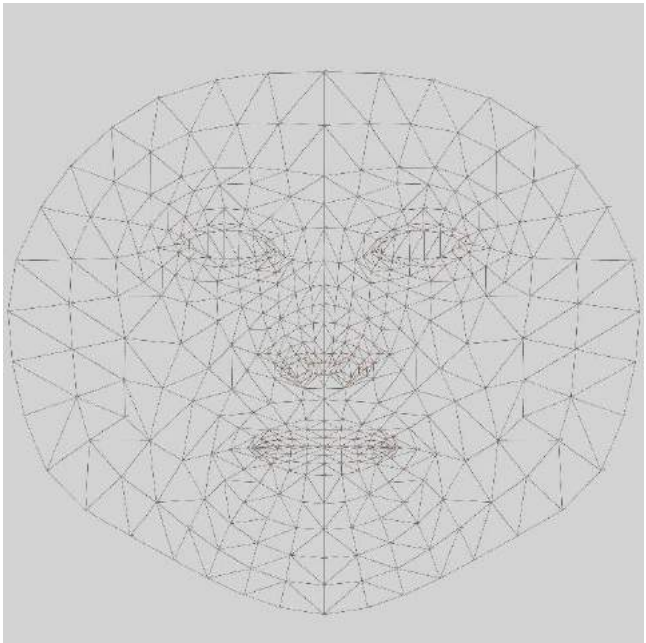


Рисунок 1.6. Facemesh від Mediapipe

ДОДАТОК Б

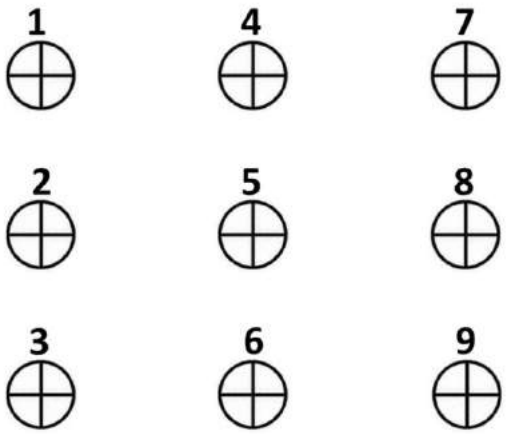


Рисунок 2.1. Калібрування
на основі 9 точок

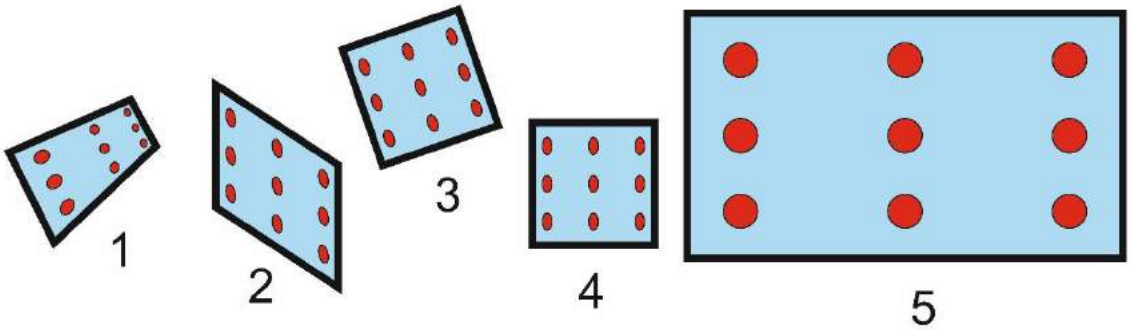


Рисунок 2.2. Перетворення зображень

Рисунок 2.3. Явище дисторсії

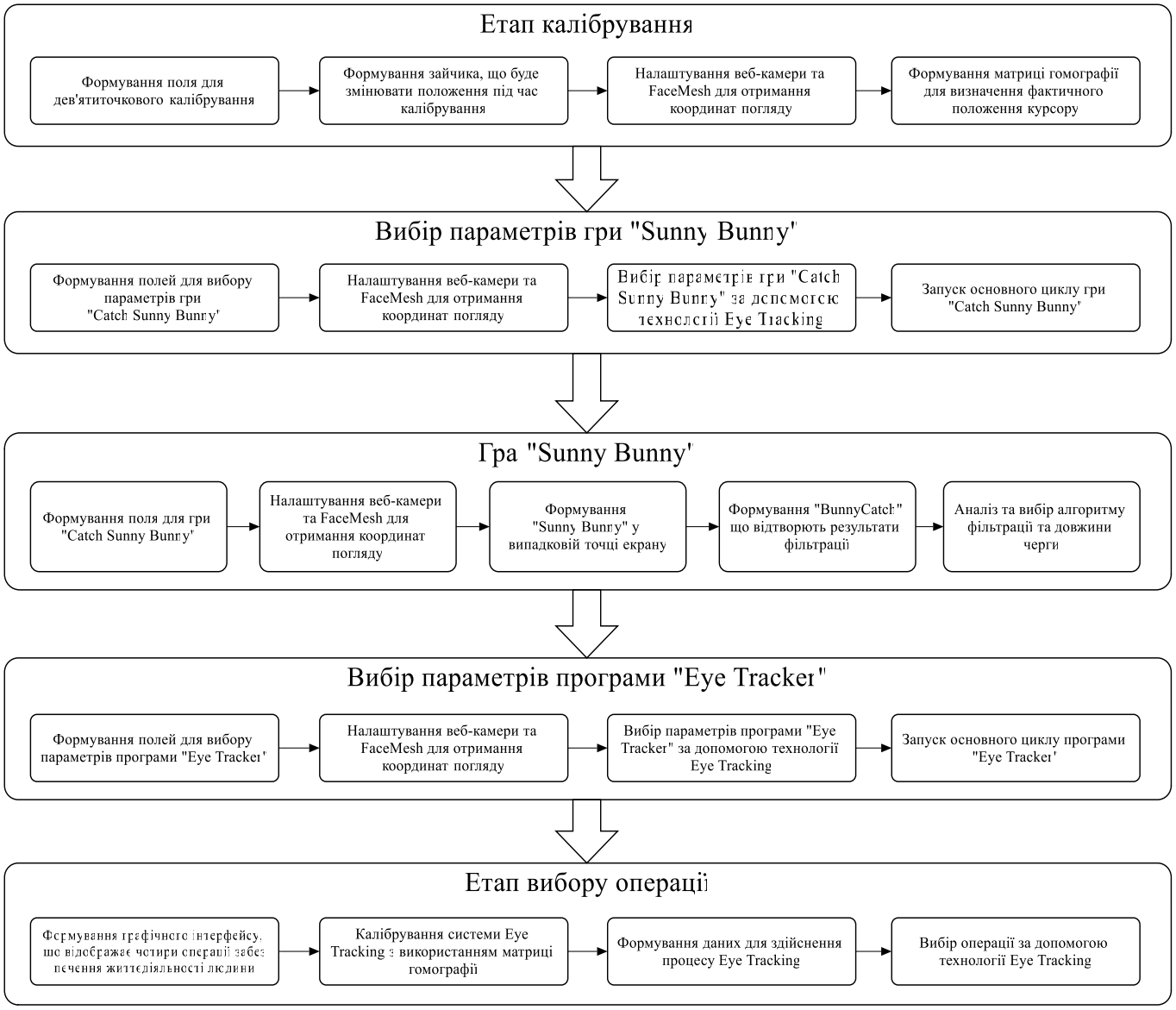


Рисунок 2.4. Структурна схема проекту

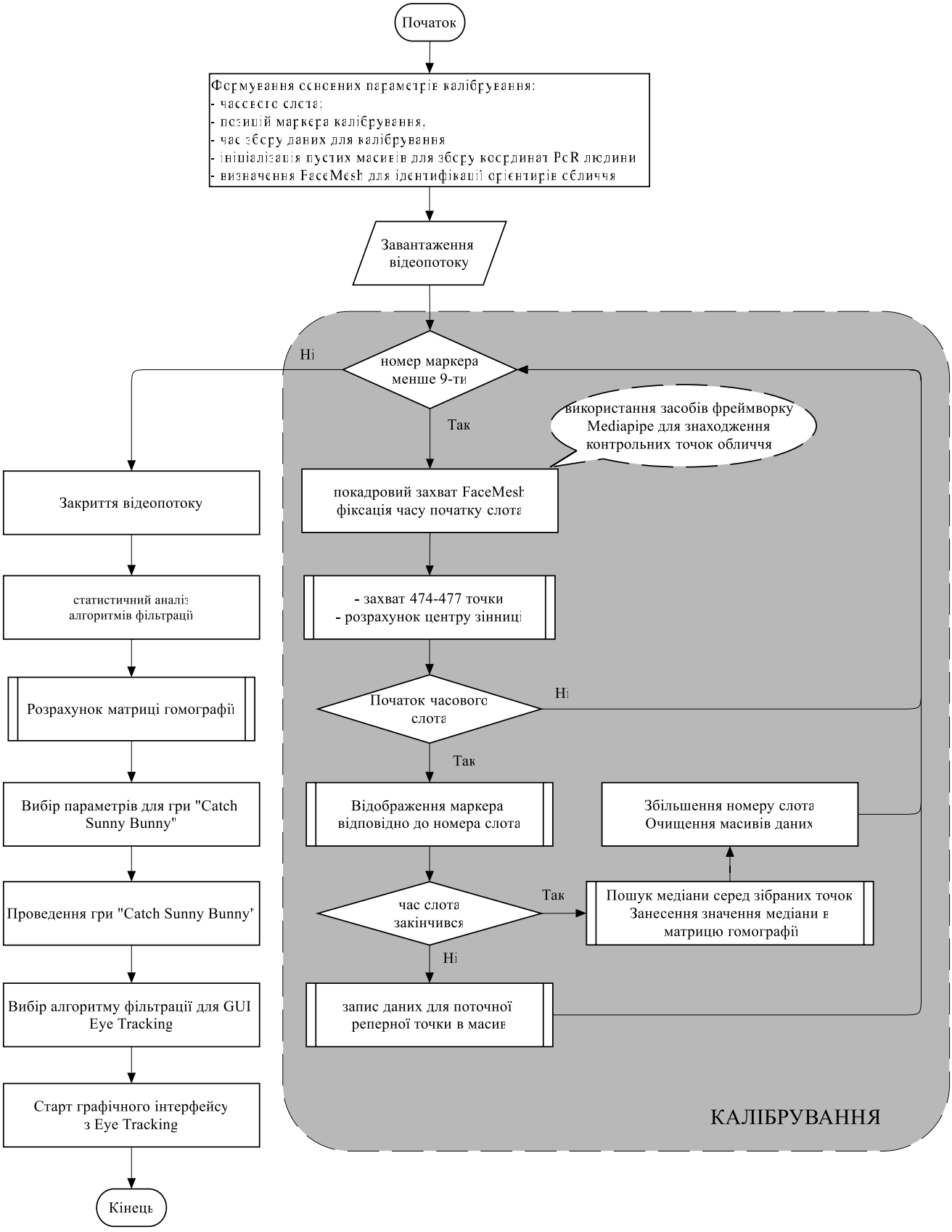


Рисунок 2.5. Блок-схема функціонування
програмного комплексу

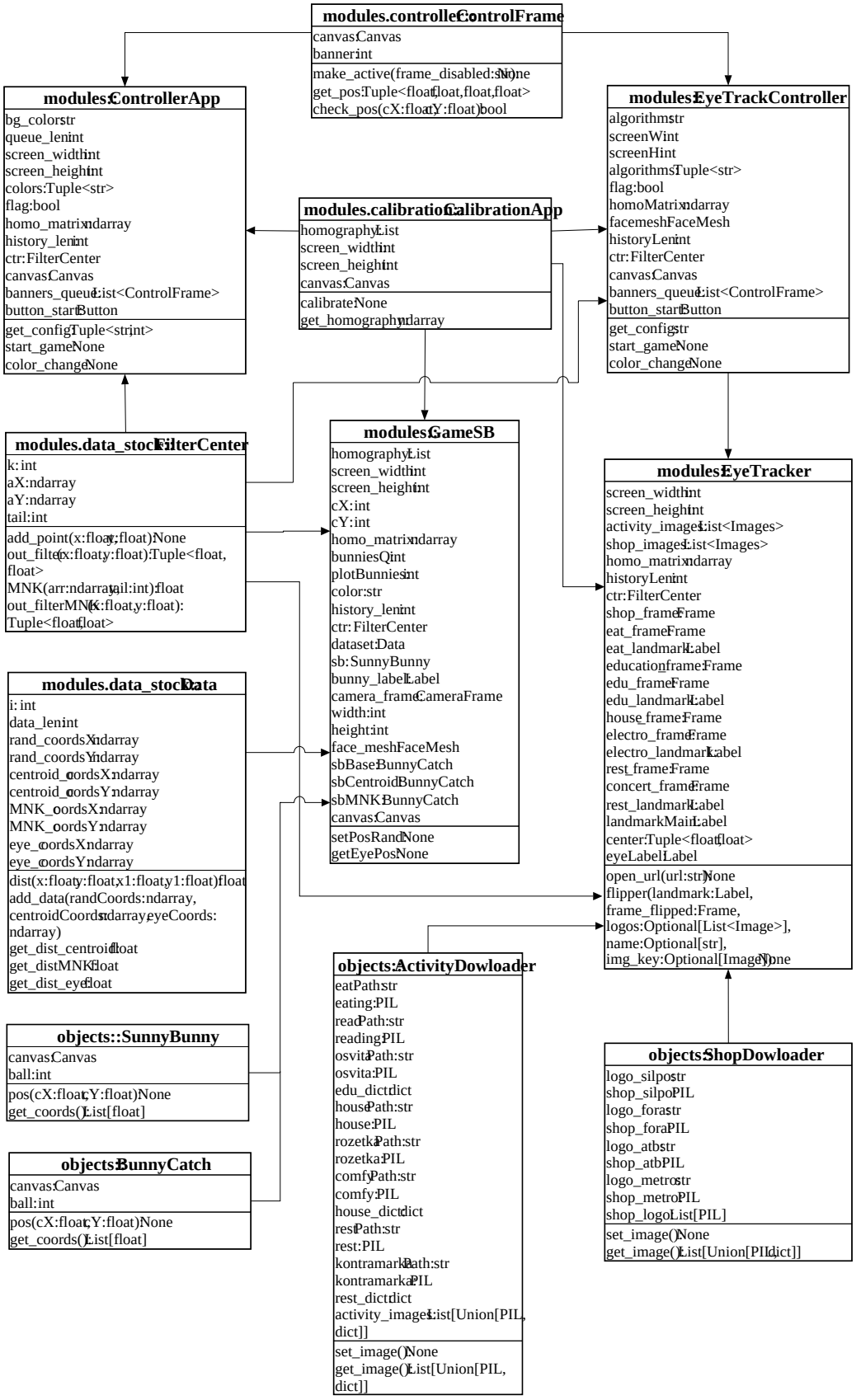


Рисунок 2.6. Функціональна схема програмної компоненти
(діаграма класів)

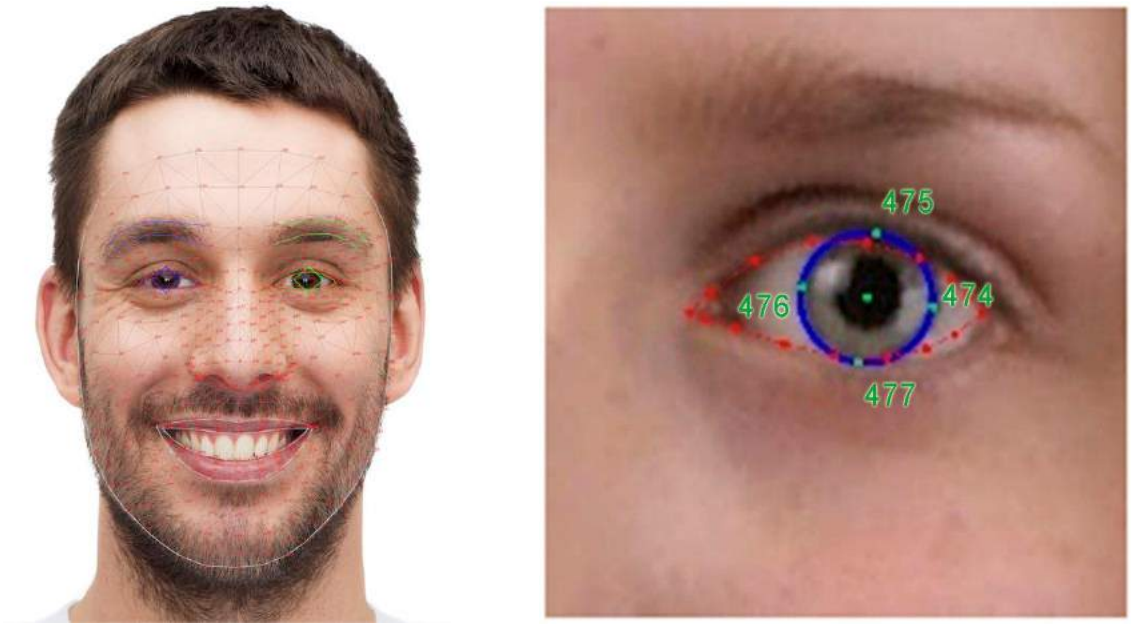
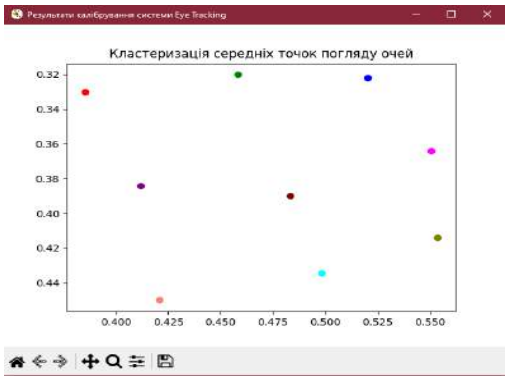


Рисунок 2.7. Розширений FaceMesh від Medaiapipe




```
Вхідна матриця для обчислення гомографії
[[0.38505745 0.3301734 ]
 [0.45803058 0.32029317]
 [0.52008843 0.32208849]
 [0.55036831 0.36399315]
 [0.55369224 0.41383249]
 [0.49819657 0.43437522]
 [0.42094241 0.44984576]
 [0.41195035 0.38415407]
 [0.48327072 0.39009957]]
```

Рисунок 2.8. Візуалізація результатів калібрування

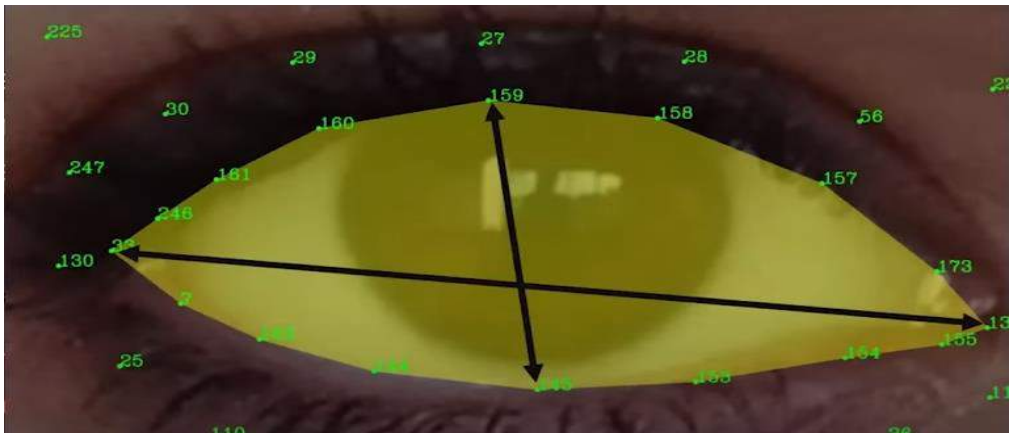


Рисунок 2.9. Необхідні точки для процесу кліпання ока
як вибору нової операції

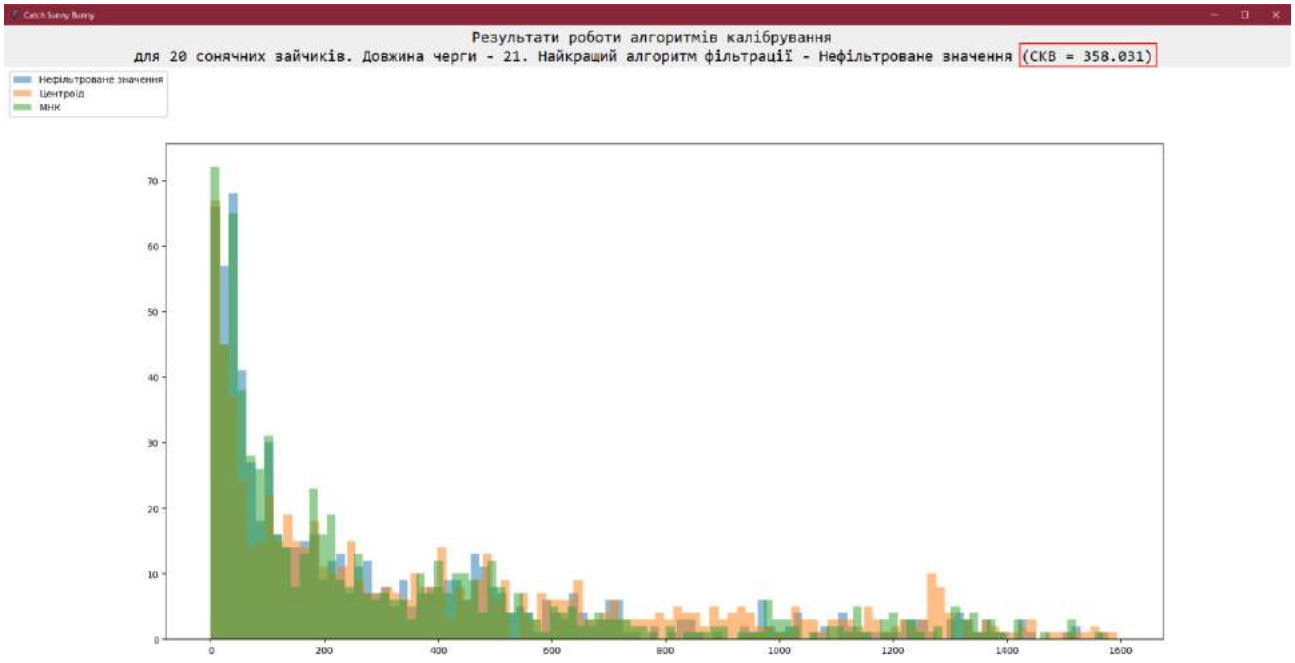


Рисунок 2.10. Використання технологій Data Science
для аналізу найкращого алгоритму фільтрації

ДОДАТОК В

ОПИС ПРОГРАМНОГО КОМПЛЕКСУ

Клас CalibrationApp, що в директорії modules\calibration\calibration_app.py, відповідає за етап калібрування системи Eye Tracking. Можливостями Tkinter, Numpy, Mediapipe та OpenCV розроблено алгоритм калібрування на основі дев'яти точок із застосуванням методології, описаної в розділі 2.1. Мета розробленого алгоритму калібрування – зібрати дані для матриці гомографії.

Для збору даних засобами Tkinter створено додаток, що відображає точку в одній з дев'ятьох позицій протягом чотирьох секунд. Збір даних для матриці гомографії починається з другої секунди, оскільки дві секунди відведено на переведення погляду (для адекватності результатів). Серед масивів, що утворюються під час погляду на одну з дев'яти точок, засобами Numpy обчислюється медіана, що визначає середні точки. Перший аргумент – масив середніх точок. Він подається на вхід методу OpenCV findHomography(). Даний метод за допомогою алгоритму RANSAC (Random Sample Consensus) [11, с. 69, 71-78] обчислює матрицю гомографії.

Другий аргумент методу findHomography() – масив координат точок екрана:

- для Ox – [0, 1920];
- для Oy – [0, 1080].

Після розрахунку матриці H вищезазначеним методом, обчислюється добуток гомогенних координат та матриці гомографії. Координати погляду обличчя обчислюються на основі FaceMesh від Mediapipe. Для обчислення координат центра зіниці необхідно взяти половину від різниці координат точок з індексами 474 та 476 для Ox, 477 і 475 для Oy. Ці індекси відповідають крайнім точкам зіниці правого ока. Потім потрібно змістити обчислені значення на точки з індексами 476 та 475 відповідно (Рисунок 2.7).

Налаштування алгоритму калібрування мають структуру часового слота, тобто запис медіани кожної точки відповідає номеру слота. Вбудований модуль datetime забезпечує процес формування початку та закінчення time slot. Кожен часовий слот має сталу довжину, під час якої відображається реперна точка в одному з дев'яти положень. Реперні точки для маркера калібрування визначаються масивом *m9*. Якщо порядок поточного слота не перевищує загальну кількість слотів – відбувається переміщення реперної точки. Реперні точки в координатах екрана зберігаються в масиві screenPoints.

Формування даних для матриці гомографії відбувається з другої секунди. Дані заносяться до масивів *mX* та *mY*, які призначені для збору координат погляду *x* та *y* відповідно, що створюються на початку кожного слота. Засобами Numpy знаходяться медіани масивів, що утворились після закінчення слота.

Для довідки, було проведено кластеризацію середніх точок погляду, яка демонструє, що результат калібрування системи Eye Tracking з використанням матриці гомографії – не ідеальний (Рисунок 2.8). Такий результат пояснюється явищем дисторсії, яке описане в розділі 2.1 даної роботи.

Клас Controller, що в директорії modules\controller_app.py, дозволяє вибрати параметри тренувальної гри «Catch Sunny Bunny»: колір фону та довжину кільцевої черги. Це забезпечується інструментами Tkinter, що надають полотно Canvas. Даний етап використовує кільцеву чергу засобами Python. Користувачу системи Eye Tracking запропоновано на вибір три черги довжиною 5 елементів, 15 та 21. Від цього залежить швидкість реакції курсора, що є необхідним мінімумом для даного програмного комплексу.

Властивість кільцевої черги, згадана в розділі 2.1, є основою для реалізації алгоритму, що зменшує некеровані рухи курсора, зумовлені постійними рухами людської зіниці. Значення координат погляду заносяться в кільцеву чергу. Такий підхід дозволяє робити аналіз та прогноз наступного значення, використовуючи алгоритми інтерполяції на основі попередніх значень веб-камери. В даній роботі використовується алгоритм МНК та центроїд.

Клас FilterCenter, що в директорії modules\data\queue_setup.py, створює кільцеву чергу, яка зберігає історію позицій очей користувача. Основа розробленої черги – масив Numpy, чиї методи дозволяють виконувати основні арифметичні операції для пошуку центроїда чи здійснення МНК-прогнозування. Причинами використання бібліотеки є її швидкодія та малий об’єм пам’яті Numpy-масивів.

Одним з алгоритмів для досягнення більш плавного руху курсору є пошук центроїда черги. Центроїд розробленої системи Eye Tracking, – це центральна точка одного з «кластерів», чиї дані надходять з кільцевої черги. Спираючись на те, що він є середнім арифметичним координат точок кластера, засобами Numpy знайдено центроїд утворених «кластерів» як середнє арифметичне *Ox* та *Oy* відповідно. Знайдені центроїди для кожної з черг вважатимуться новими координатами положення курсору.

Іншим способом для зменшення аномальних рухів курсору є МНК. Застосовуючи даний алгоритм та можливості масивів Numpy, цей фільтр використовується для плавності рухів курсору, що дає можливість імітувати роботу емулятора «миші». Вхідна черга потребує «впорядкування» елементів за їх надходженням до неї перед МНК-згладжуванням запропонованим методом. Використовуючи поточне значення хвоста черги, її елементи впорядковуються згідно їхнього надходження до неї.

Для підтвердження вибору операції застосовано механізм кліпання ока як імітація кліку «миші». Останній використовує різницю ординат нижньої та верхньої повік ока, адже кліпання ока – вертикальний процес. Цим ординатам відповідають точки 145 та 159 FaceMesh (Рисунок 2.9) від Mediapipe.

Клас GameSB, що в директорії modules\game_bunny.py, містить тренувальну гру «Catch Sunny Bunny», аби об’єктивно визначити найкращий алгоритм фільтрації задля плавного керування курсором.

Основна задача гри – слідкувати за «сонячним зайчиком» що з’являється різних точках екрана. За допомогою можливостей полотна Canvas, що надає бібліотека Tkinter, створено поле для руху «сонячного зайчика», чиї координати

генеруються випадково, та «кошенят» (зелений – «Центроїд», синій – «МНК»), які керуються відповідними фільтрами на основі кільцевої черги.

Класи SunnyBunny та BunnyCatch розташовані в директорії objects\sunny_bunny.py та objects\bunny_catch.py. Вони формують «сонячний зайчик» та «кошенят». Процес гри дозволяє зібрати dataset з відфільтрованими значеннями. Слід зазначити про формування нормалізованих координат, що піддаються обробці фільтрів на основі кільцевої черги. Оскільки основою алгоритму є матриця гомографії, необхідно результати добутку гомогенних координат на матрицю H перевести в світові координати. Для цього потрібно поділити відповідні значення координат на масштабний коефіцієнт k [14]:

$$P\ (kx/k,\ ky/k) \qquad (2.8)$$

де P – світові координати погляду людини;

k – масштабний коефіцієнт.

На результатах такого перетворення випробувано два алгоритми, що використовують кільцеву чергу Python.

Клас Data, що в директорії modules\data\data_stock.py, займається формуванням dataset для аналізу даних. Метою такого аналізу є визначення найкращого алгоритму керування курсором. Це можна визначити за відстанню між «зайчиком» та кожним з «кошенят», розрахувавши Евклідову відстань. Позиція сонячного зайчика генерується випадково з використанням вбудованого модулю random. «Кошенята» керуються поглядом людини з використанням фільтрів центроїда та МНК кільцевої черги. Формула Евклідової відстані для 2D простору має вигляд:

$$d\ (a,\ b) = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2} \qquad (2.9)$$

де a – початкова точка відрізка з координатами (x_a, y_a);

b – кінцева точка відрізка з координатами (x_b, y_b).

Методи статистичного навчання засобами Numpy (побудова частотного аналізу та розрахунок статистичних показників) допомагають обрати найкращий алгоритм фільтрації на основі кільцевої черги для керування курсором.

Клас MatplotlibFrame, що в директорії modules\visual\matplotlib_frame.py, використовує інструменти бібліотеки Matplotlib. Цей клас допомагає здійснити графічний аналіз з використанням технологій Data Science. Вони стають у нагоді для статистичного аналізу шляхом знаходження мінімального середньоквадратичного відхилення. Це один з найпоширеніших показників розсіювання (розкиду) значень, які надходять від кільцевої черги, відносно центру розкиду (Рисунок 2.10). Мінімальний показник даної метрики дозволяє визначити найкращий алгоритм фільтрації.

Клас CameraFrame, що розташований в директорії modules\camera_frame.py, забезпечує відеопотік для гри «Catch Sunny Bunny» інструментами Tkinter та OpenCV.

Клас EyeTrackController, що в директорії modules\eye_track_controller.py, дозволяє обрати алгоритм фільтрації за допомогою інтерактивного інтерфейсу відповідно до результату пройденої гри. Логіка роботи даного модулю подібна до програмної реалізації контролера для гри «Catch Sunny Bunny». Відмінність полягає лише у виборі алгоритму фільтрації.

Клас ActivityDownloader знаходиться в директорії objects\activity_downloader.py та призначений для завантаження картинок для GUI. Інструментами бібліотеки Pillow забезпечується візуалізація картинок на екран. Цей клас має властивість get_image, що повертає сформований список картинок. Кожна операція додатка оформлена у вигляді «плитки», що містить піктограму. Вона відображає основну суть операції. Наприклад, для категорії «Освіта» обрано піктограму книжок.

Властивості Python дозволяють створювати методи, які вважаються атрибутами класу. Використання цих властивостей допомагає позбавитись getter та setter, що роблять код перенавантаженим. За допомогою property код набуває стилю Python.

Клас EyeTracker міститься в директорії modules\eye_track.py та використовує Tkinter. Він характеризує основний етап програми для відображення графічного інтерфейсу, що керується за допомогою технологій

Eye Tracking. Цей клас використовує вже розроблені модулі, що є однією з переваг модульної структури проекту. Необхідно відзначити, що додаток, як і інші програмні компоненти системи, використовує результати калібрування, сформовані на початковому етапі користування програмним комплексом.

Основна мета додатку «Eye Tracker» полягає в демонстрації використання технології Eye Tracking та тренуванні потенційних користувачів в керуванні GUI.

Наступним етапом прикладного використання отриманих навичок є додаток EyeTrackController.

Концептуально додаток побудований на виборі чотирьох базових операцій, що забезпечують нормальний спосіб життя людини:

- «Харчування»;
- «Освіта»;
- «Побут»;
- «Розваги».

Наприклад, під категорією «Харчування» маються на увазі продуктові та продовольчі магазини. Використовуючи модуль pyautogui та webbrowser, що надає доступ до веб-сайтів інструментами Python, дана категорія містить посилання на інтернет-ресурси чотирьох найпопулярніших магазинів, що забезпечують населення України харчовими та побутовими товарами:

- «Сільпо»;
- «Фора»;
- «АТБ»;
- «МЕТРО».

За допомогою запропонованих алгоритмів, можливо розширити функціонал розробленого Eye Tracker, додавши більше операцій для керування очима та розпізнавання.

Клас ShopDownloader в директорії objects\shop_downloader.py, який за своєю логікою роботи подібний до класу ActivityDownloader, створений для

збереження піктограм з логотипами вищезазначених магазинів. Клас містить властивість get_image, що повертає список логотипів всіх чотирьох магазинів.

За допомогою інструментів Mediapipe відбувається аналіз координат погляду людини, які нормалізуються шляхом множення гомогенних координат на матрицю гомографії. Отримані координати перевіряються, чи вони знаходяться в межах одного з чотирьох квадрантів. Порівняння проводиться за центральною точкою на екрані, що є перетином чотирьох фреймів. Якщо координати потрапляють в межі зазначених фреймів – відбувається активація відповідного фрейму та його «фліп»(обертання). Всі операції за замовчуванням – неактивні. Про це свідчить червоний колір рамок фреймів. Зелений колір сигналізує про вибір однієї з доступних операцій. Інші операції знаходяться в неактивному стані.

Після «обертання» фрейму під назвою «Харчування» користувачеві пропонується на вибір чотири кнопки. Вони відповідають Інтернет-ресурсам одного із зазначених магазинів. Клікнувши на одну з них, користувач системи Eye Tracking отримує можливість керувати обраним сайтом, використовуючи дану технологію.

Категорія «Освіта» доступна користувачам Eye Tracker завдяки можливостям словника Python. Він міститься в класі ActivityDownloader. Ключ зовнішнього словника – умовний ідентифікатор, за яким розрізняються операції (для освіти – «edu»), значення – вкладений словник. Ключем вкладеного словника є піктограма для освіти, значенням – логотип Інтернет-ресурсу з освітньою тематикою.

Графічний інтерфейс містить категорію «Побут». Користувачам надається доступ до Інтернет-ресурсів двох найпопулярніших магазинів, що забезпечують населення України побутовими товарами:

- «Rozetka»;
- «Comfy».

Як і в попередній категорії для магазинів побутової техніки використовується словник Python, що знаходиться в класі ActivityDownloader.

Він містить вкладені словники. Ключ основного словника – унікальний ідентифікатор «house» операції «Побут», значення – вкладений словник з логотипом побутових товарів в якості ключа. Значенням виступає ще один вкладений словник. Ключем останнього є назва Інтернет-магазину, а значенням – відповідний логотип.

Програмний комплекс містить категорію «Розваги». Механізм словника для збереження необхідних картинок цієї категорії аналогічний процесу для категорії «Освіта». У разі вибору операції «Розваги» користувач має можливість керувати Інтернет-ресурсом kontramarka.ua, який надає афіші різноманітних подій: від концертів та театральних вистав до виставок в різних куточках світу.

Практичне використання технології Eye Tracking не обмежується лише керуванням графічним інтерфейсом 2x2, що містить базові операції життєдіяльності людини. Розроблений програмний комплекс Eye Tracking може стати основою для подальшого розвитку в галузі інформаційних технологій.

ДОДАТОК Г

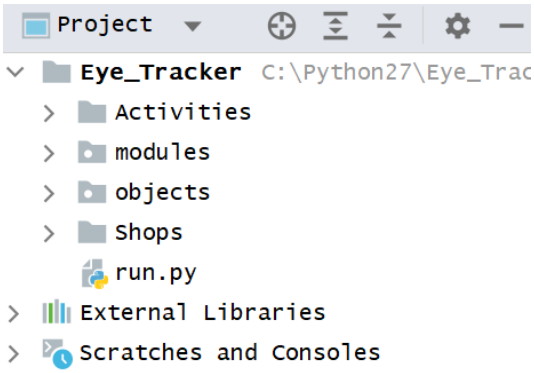


Рисунок 3.1. Структура програми

run.py

```
"""
Програмний комплекс
управління графічним інтерфейсом
за допомогою технологій Eye Tracking
Кондратюк Наталії
ПІ-22-1м
"""

from modules.calibration import CalibrationApp
from modules.controller_app import Controller
from modules.game_bunny import GameSB
from modules.eye_track_controller_app import EyeTrackController
from modules.eye_track import EyeTracker

def main() :
    CalibrationApp()()
    Controller()()
    GameSB()()
    EyeTrackController()()
    EyeTracker()()

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

Рисунок 3.2. Структура файлу, що визначає
порядок запуску класів

calibration_app.py

```python
import time
from datetime import datetime

import tkinter as tk
import cv2
import matplotlib.pyplot as plt
import numpy as np
import mediapipe as mp

class CalibrationApp(tk.Tk) :
    """
    Механізм калібрування
    системи Eye Tracking
    """

    homography = []

    def __init__(self, *args, **kwargs):
        tk.Tk.__init__(self, *args, **kwargs)
        self.wm_title("Calibrate Eye Tracking" )

        self.screen_width, self.screen_height = self.winfo_screenwidth()-15, self.winfo_screenheight()-65
        self.wm_geometry('%dx%d+0+0' % (self.screen_width, self.screen_height))
        self.canvas = tk.Canvas(self, width=self.screen_width, height=self.screen_height)
        self.canvas.pack()
        self.bind('<Escape>', lambda e: self.destroy())

        self._calibrate()
```

Калібрування системи Eye Tracking

Рисунок 3.3. Структура класу, що відповідає за етап калібрування

Рисунок 3.4. Стани маркера калібрування:
жовтий (ліворуч) – підготовка до запису, червоний (праворуч) – запис координат

controller_app.py

```python
import tkinter as tk
from tkinter import messagebox

import numpy as np
import cv2
import mediapipe as mp
import pyautogui

from modules.data.queue_setup import FilterCenter
from modules.calibration import CalibrationApp
from modules.controller import ControlFrame

class Controller(tk.Tk) :
    """
    Користувацькі налаштування
    для гри "Catch Sunny Bunny"
    """

    bg_color: str = ""
    queue_len: int = 0

    def __init__( self, *args, **kwargs ) :
        tk.Tk.__init__( self, *args, **kwargs )
        tk.Tk.wm_title( self, "Controller App" )
        self.screen_width, self.screen_height = self.winfo_screenwidth()-15, self.winfo_screenheight()-65
        self.wm_geometry('%dx%d+0+0' % (self.screen_width, self.screen_height))
        self.cX, self.cY = int(self.screen_width/2), int(self.screen_height/2)
        self.bind('<Escape>', lambda e: self.destroy())

            self.button_start = tk.Button(self, text="START Game",
                                          width=20, height=3,
                                          font=('Consolas', 25), command=self._start_game)
            self.button_start.place(relx=0.36, rely=0.72)

            self._color_change()
```

Контролер для гри "Catch Sunny Bunny"

Рисунок 3.5. Структура класу Controller

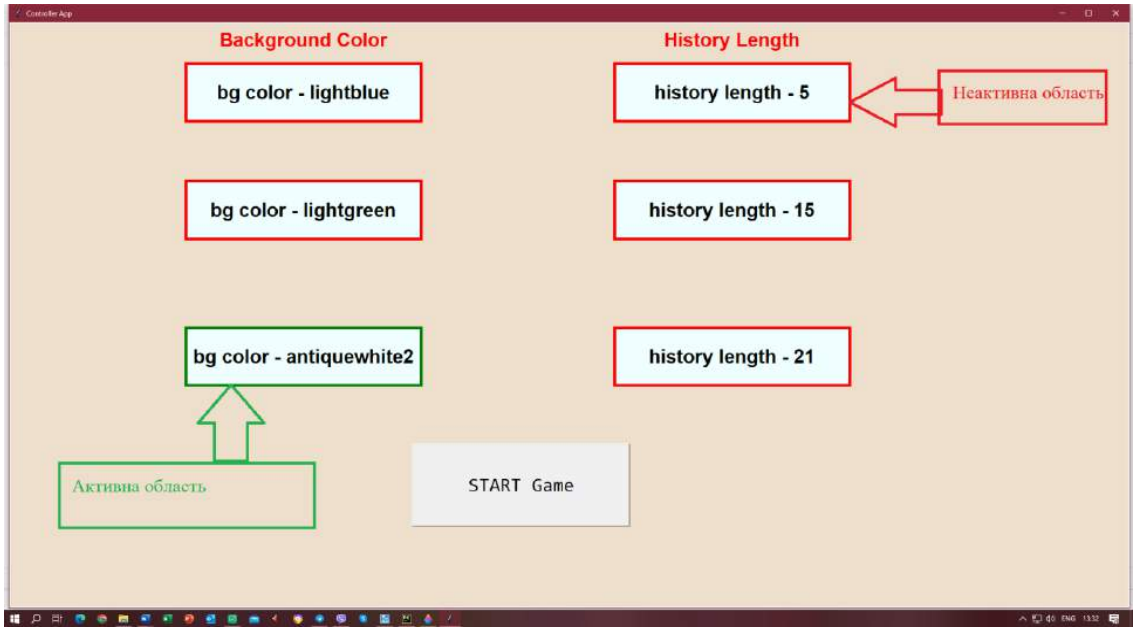


Рисунок 3.6. Загальний вигляд контролера гри «Catch Sunny Bunny»

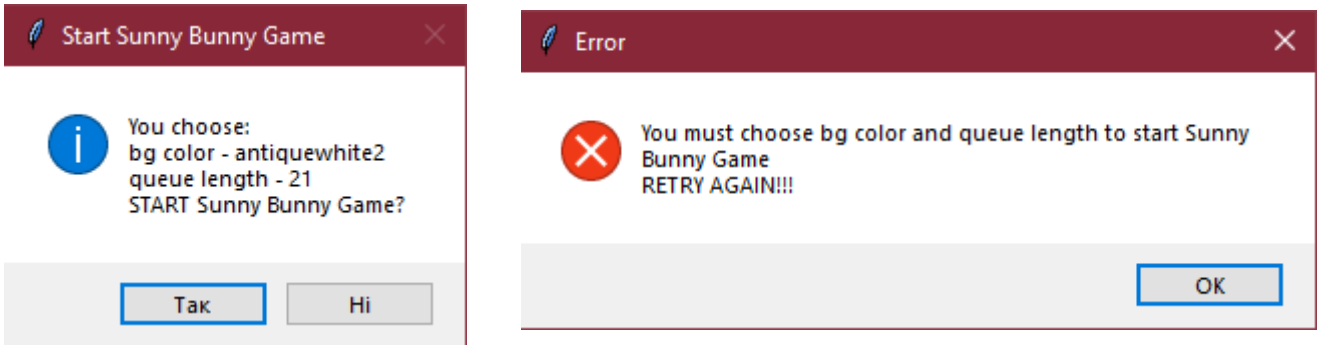


Рисунок 3.7. Два види повідомлень контролера гри «Catch Sunny Bunny»: підтвердженння (ліворуч), відхилення (праворуч)

game_bunny.py

```
import tkinter as tk
from PIL import Image, ImageTk

import numpy as np
import math as mt

import cv2
import mediapipe as mp

from modules.controller_app import Controller
from modules.calibration import CalibrationApp
from modules.data import FilterCenter, Data
from modules.visual import CameraFrame, MatplotlibFrame

from objects.sunny_bunny import SunnyBunny
from objects.bunny_catch import BunnyCatch

class GameSB( tk.Tk ) :
    """
    Тренувальна гра
    "Catch Sunny Bunny"
    для визначення найкращого алгоритму
    фільтрації на основі кільцевої черги
    """

    def __init__(self, *args, **kwargs):
        tk.Tk.__init__(self, *args, **kwargs)
        tk.Tk.wm_title(self, "Catch Sunny Bunny" )
        self.screen_width, self.screen_height = self.winfo_screenwidth()-15, self.winfo_screenheight()-65
        self.wm_geometry('%dx%d+0+0' % (self.screen_width, self.screen_height))
        self.cX, self.cY = int(self.screen_width/2), int(self.screen_height/2)
            self.set_pos_rand() Генерація сонячного зайчика
            self.get_eye_pos() Основний цикл гри
```

Рисунок 3.8. Структура класу гри «Catch Sunny Bunny»

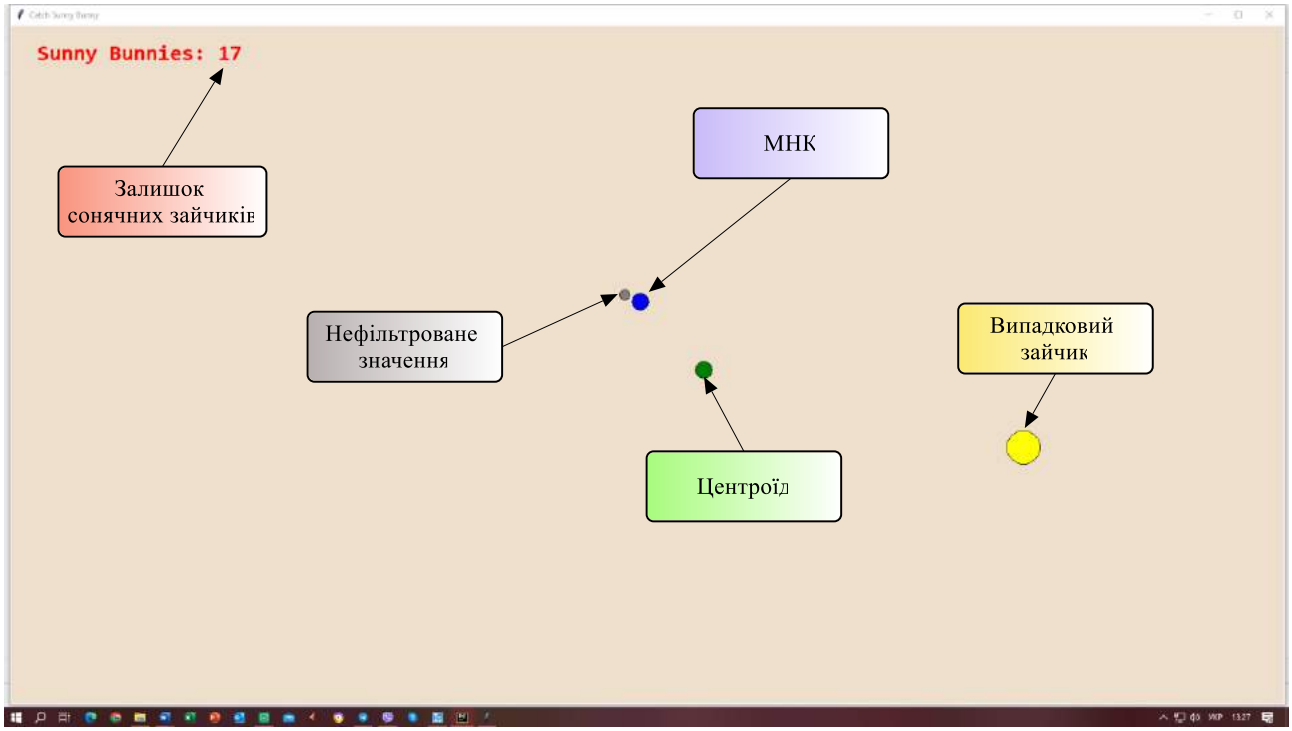


Рисунок 3.9. Загальний вигляд гри «Catch Sunny Bunny»

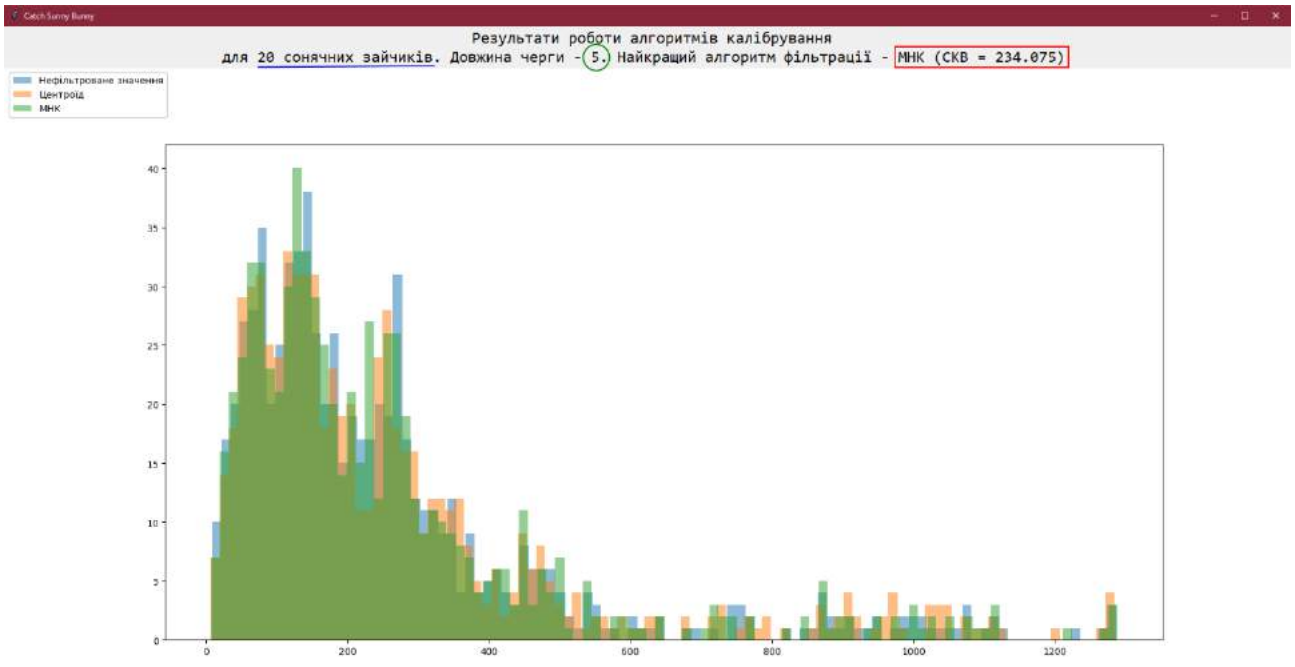


Рисунок 3.10. Результати гри «Catch Sunny Bunny»

eye_track_controller_app.py

```
import tkinter as tk
from tkinter import messagebox

import numpy as np
import cv2
import mediapipe as mp
import pyautogui

from modules.data.queue_setup import FilterCenter
from modules.calibration import CalibrationApp
from modules.controller_app import Controller
from modules.controller import ControlFrame

class EyeTrackController(tk.Tk) :
    """
    Користувацькі налаштування
    для процесу Eye Tracking
    """

    algorithm: str = ""

    def __init__( self, *args, **kwargs ) :
        tk.Tk.__init__( self, *args, **kwargs )
        tk.Tk.wm_title( self, "Controller App" )
        self.screen_width, self.screen_height = self.winfo_screenwidth()-15, self.winfo_screenheight()-65
        self.wm_geometry('%dx%d+0+0' % (self.screen_width, self.screen_height))
        self.cX, self.cY = int(self.screen_width/2), int(self.screen_height/2)
        self.bind('<Escape>', lambda e: self.destroy())
        self.button_start = tk.Button(self, text="START Program",
                                      width=20, height=3,
                                      font=('Consolas', 25), command=self._start_game)
        self.button_start.place(relx=0.36, rely=0.52)

        self._color_change()
```

Контролер для Eye Tracker

Рисунок 3.11. Структура класу EyeTrackController

Рисунок 3.12. Загальний вигляд контролера для Eye Tracker

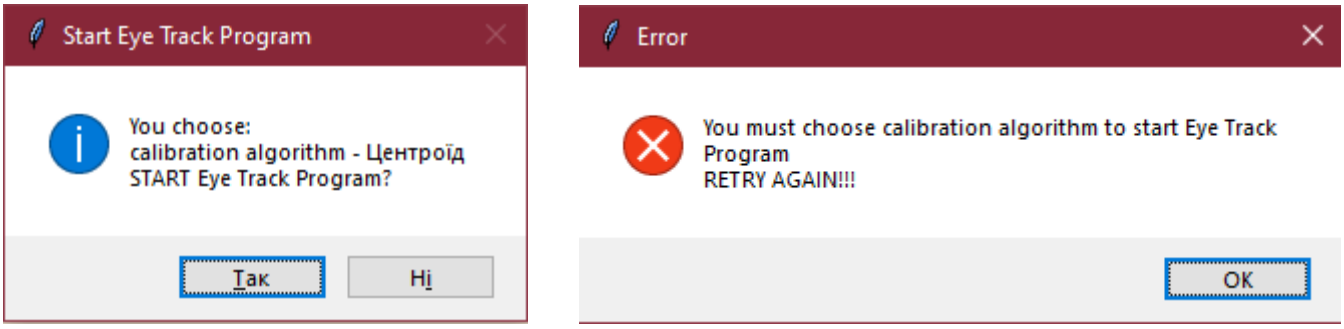


Рисунок 3.13. Види повідомлень контролера програми «Eye Tracker»:
вибір алгоритму фільтрації (ліворуч) та відсутність вибору алгоритму фільтрації

eye_track.py

```
import webbrowser
import tkinter as tk

import cv2
import numpy as np
import mediapipe as mp

import pyautogui
pyautogui.FAILSAFE = False

from modules.data.queue_setup import FilterCenter
from modules.calibration import CalibrationApp
from modules.eye_track_controller_app import EyeTrackController
from modules.controller_app import Controller

from objects.activity_downloader import ActivityDownloader
from objects.shop_downloader import ShopDownloader

class EyeTracker(tk.Tk) :
    """
    GUI, що відображає практичне застосування
    технології Eye Tracking на прикладі
    керування Інтернет-ресурсами однієї
    з чотирьох категорій
    """

    def __init__(self, *args, **kwargs) :
        tk.Tk.__init__(self, *args, **kwargs)
        tk.Tk.wm_title(self, "Eye Tracker")
        self._screen_width, self._screen_height = self.winfo_screenwidth(), self.winfo_screenheight()
        tk.Tk.wm_state( self, 'zoomed' )
        self.bind('<Escape>', lambda e: self.destroy())
            self._shop_frame.grid(row=0, column=0, sticky=tk.NW)
            self._education_frame.grid(row=0, column=1, sticky=tk.NE)
            self._house_frame.grid(row=1, column=0, sticky=tk.SW)
            self._rest_frame.grid(row=1, column=1, sticky=tk.SE)

            self._main() Основний процес Eye Tracking
```

Рисунок 3.14. Структура класу EyeTracker

Рисунок 3.15. Загальний вигляд Eye Tracker

Рисунок 3.16. Індикатори стану операцій

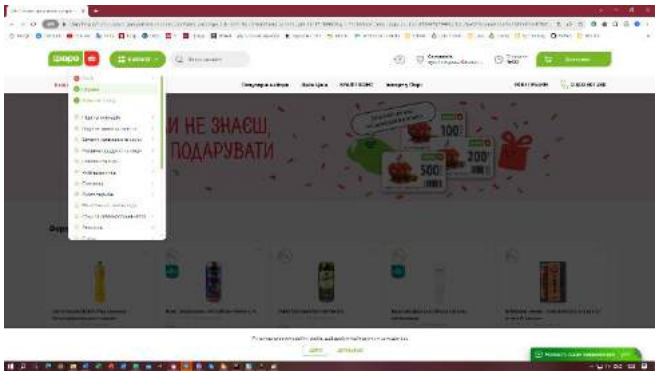


Рисунок 3.17. Практичне застосування технології Eye Tracking

ДОДАТОК Д

ІНСТРУКЦІЯ КОРИСТУВАЧА

до наукової роботи

на тему: «Програмний комплекс управління інтерактивним графічним інтерфейсом за технологіями Eye Tracking»

Київ - 2024

ЗМІСТ

ВСТУП

Даний документ призначений для опису та надання інформації щодо інсталяції, користування та налаштування програмного комплексу управління інтерактивним графічним інтерфейсом за технологіями Eye Tracking.

Зазначене програмне забезпечення створено мовою програмування високого рівня Python. Користувацьке ПЗ має бути сумісним з налаштуваннями, на яких розроблено програмну систему Eye Tracking.

Система Eye Tracking надає можливість:

– отримувати показники користувацького PoR в режимі реального часу за допомогою інструменту Mediapipe;

– аналізувати інструменти фільтрації даних за допомогою технологій Data Science;

– керувати інтерфейсом комп’ютера без використання традиційних засобів керування, на кшталт емулятора миші.

Метою розробленої програмної системи є демонстрація практичного використання технології відстеження очей.

Користувач повинен мати комфортні умови для користування GUI із використанням передових технологій Eye Tracking. Веб-камера, що використовується як Eye Tracker, має бути розташована по центру монітора. Таке розташування забезпечує кращі результати калібрування за допомогою матриці гомографії. Роздільна здатність монітора, на якому використовується програмний комплекс, має бути не менше 1920x1080 пікселів. На моніторах меншої роздільної здатності можуть виникнути певні проблеми з візуалізацією етапу калібрування.

МІНІМАЛЬНІ ТЕХНІЧНІ ВИМОГИ ДО ПК

- процесор — Intel(R) Core(TM) i7-4765T CPU @ 2.00GHz 2.00 GHz;
- оперативна пам’ять об’ємом 8-16 Гб;
- веб-камера Venus USB2.0 Camera;
- монітор Full HD з підтримкою HDMI 1920x1080.

Примітка: зазначені технічні вимоги є мінімально необхідними для функціонування програмного забезпечення.

СУМІСНІСТЬ З ОПЕРАЦІЙНИМ СИСТЕМАМИ

- 32-бітні ОС: Windows 10;
- 64-бітні ОС: Windows 10.

Система також сумісна з ОС родини Linux.

ІНСТРУКЦІЯ З ІНСТАЛЯЦІЇ ТА НАЛАШТУВАННЯ ПРОГРАМНОГО КОМПЛЕКСУ УПРАВЛІННЯ ІНТЕРАКТИВНИМ ГРАФІЧНИМ ІНТЕРФЕЙСОМ ЗА ТЕХНОЛОГІЯМИ EYE TRACKING

Програмний комплекс управління інтерактивним графічним інтерфейсом (надалі – GUI) за технологіями Eye Tracking використовує сторонні бібліотеки, що надає Python. Тому перед використанням даного програмного комплексу потрібно встановити всі необхідні залежності, які він використовує. Після інсталяції всіх бібліотек, можливе практичне застосування технології Eye Tracking, використавши для цього командну стрічку:

```
PYTHON RUN.PY
```

де `RUN.PY` – файл запуску всіх класів, що є GUI додатками на основі бібліотеки Tkinter. Програмний комплекс не використовує зовнішні ключі, що вводяться за допомогою консольної утиліти Windows.

Після встановлення всіх необхідних модулів та запуску `RUN.PY` за допомогою командної стрічки можна розпочати застосування технології Eye Tracking. Вона складається з низки послідовних етапів:

1) етап калібрування;
2) вибір параметрів гри «Catch Sunny Bunny»;
3) гра «Catch Sunny Bunny»;
4) вибір параметрів Eye Tracker;
5) вибір операції.

ЕТАП № 1. КАЛІБРУВАННЯ

Калібрування – найважливіший етап роботи розробленого GUI, без якого ним неможливо керувати. Процес калібрування – це послідовний збір даних для однієї або кількох точок в різних положеннях на екрані. Програмний комплекс використовує дев’ятиточкове калібрування. Під час процесу калібрування слід враховувати фізіологічні властивості, що індивідуальні для кожної людини. Суть процесу калібрування полягає в стеженні за наступним положенням маркера.

Рисунок 1 – Підготовка до запису координат
під час калібрування

Механізм формування даних для калібрування – часовий слот тривалістю чотири секунди. Маркер для калібрування жовтого кольору свідчить про підготовку до запису координат, що триває дві секунди задля стабілізації погляду користувача (Рисунок 1). У разі змінення кольору маркера на червоний відбувається запис даних координат погляду тривалістю дві секунди (Рисунок 2).

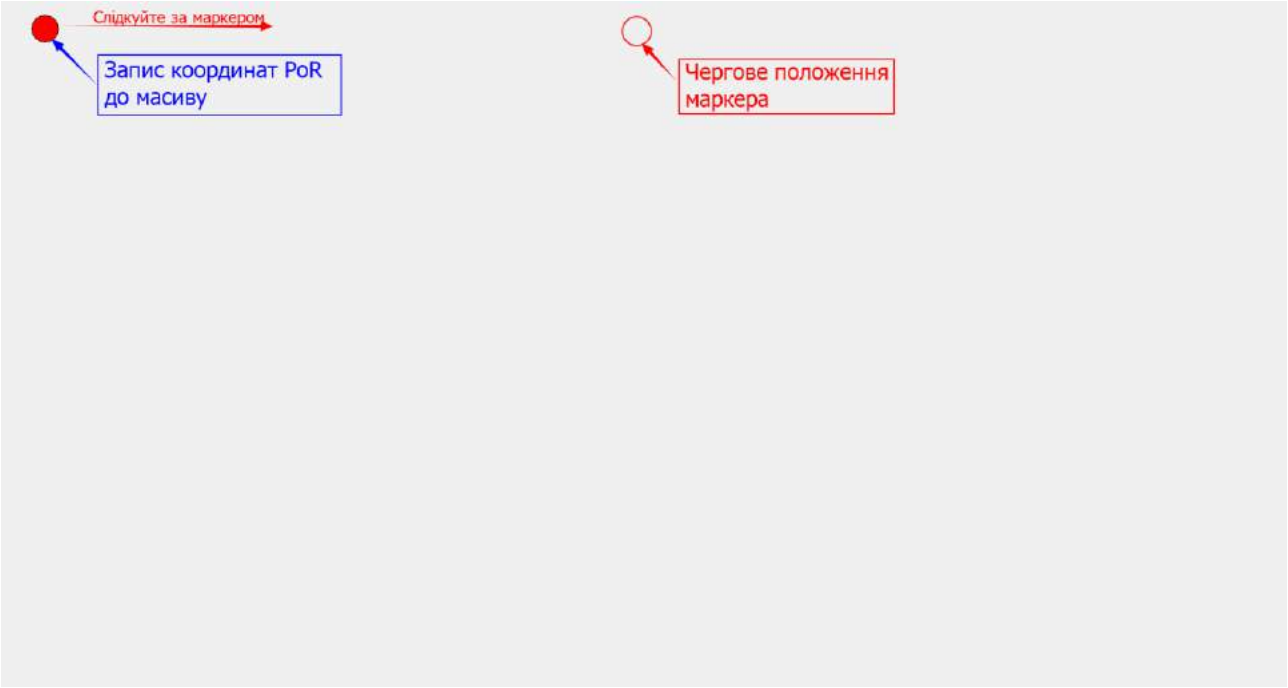


Рисунок 2 – Запис координат погляду користувача для калібрування

Після закінчення тривалості слота, під час якого програма калібрування формує необхідні дані для подальших розрахунків, особа має слідкувати за новим положенням маркера на полотні. Під час руху маркера запис координат погляду не відбувається.

ВАЖЛИВО! Для якісного калібрування необхідно уважно дивитись на червоний маркер, щоб Eye Tracker накопичив коректні дані.

Для аналізу сформованих даних під час калібрування особа може переглянути результати пройденого процесу. У разі задовільного проходження етапу калібрування має вийти кластеризація, точки якої майже повторюють місцерозташування маркера (Рисунок 3). Вона має форму неправильного чотирикутника, подібну до трапеції.

```
Вхідна матриця для обчислення гомографії
[[0.38505745 0.3301734 ]
 [0.45803058 0.32029317]
 [0.52008843 0.32208849]
 [0.55036831 0.36399315]
 [0.55369224 0.41383249]
 [0.49819657 0.43437522]
 [0.42094241 0.44984576]
 [0.41195035 0.38415407]
 [0.48327072 0.39009957]]
```

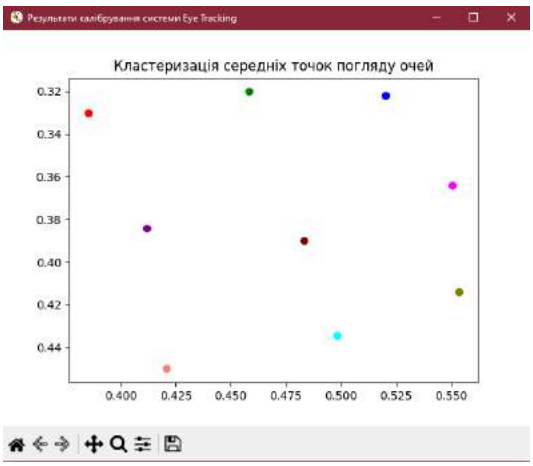


Рисунок 3 – Результати кластеризації
при дев’ятиточковому калібруванні

Під час калібрування слід враховувати той факт, що результати калібрування можуть бути викривлені внаслідок *дисторсії* – властивістю лінзи камери, що має спотворення, переважно радіальні та тангенціальні. Причина її виникнення – відмінність лінійного збільшення різних частин зображення. Вона буває бочкоподібною або подушкоподібною. Зміщена центральна точка – наочний приклад дисторсії.

Для завершення калібрування необхідно натиснути клавішу `<Escape>`.

ЕТАП № 2. ВИБІР ПАРАМЕТРІВ ДЛЯ ГРИ «CATCH SUNNY BUNNY»

Гра «Catch Sunny Bunny» дозволяє визначити найкращий алгоритм фільтрації для зменшення некерованих рухів курсора технологіями Data Science. Параметри тренувальної гри можна обрати індивідуально, скориставшись додатком «Controller App», що використовує технологію Eye Tracker.

Для запуску гри «Catch Sunny Bunny» особі необхідно обрати фон гри (background color) та довжину кільцевої черги (history length).

На основі кільцевої черги побудовано алгоритми фільтрації для зменшення некерованих рухів курсора, зумовлені постійними рухами людської зіниці.

Для запуску гри необхідно натиснути кнопку «START Game», після чого гра розпочнеться (Рисунок 4).

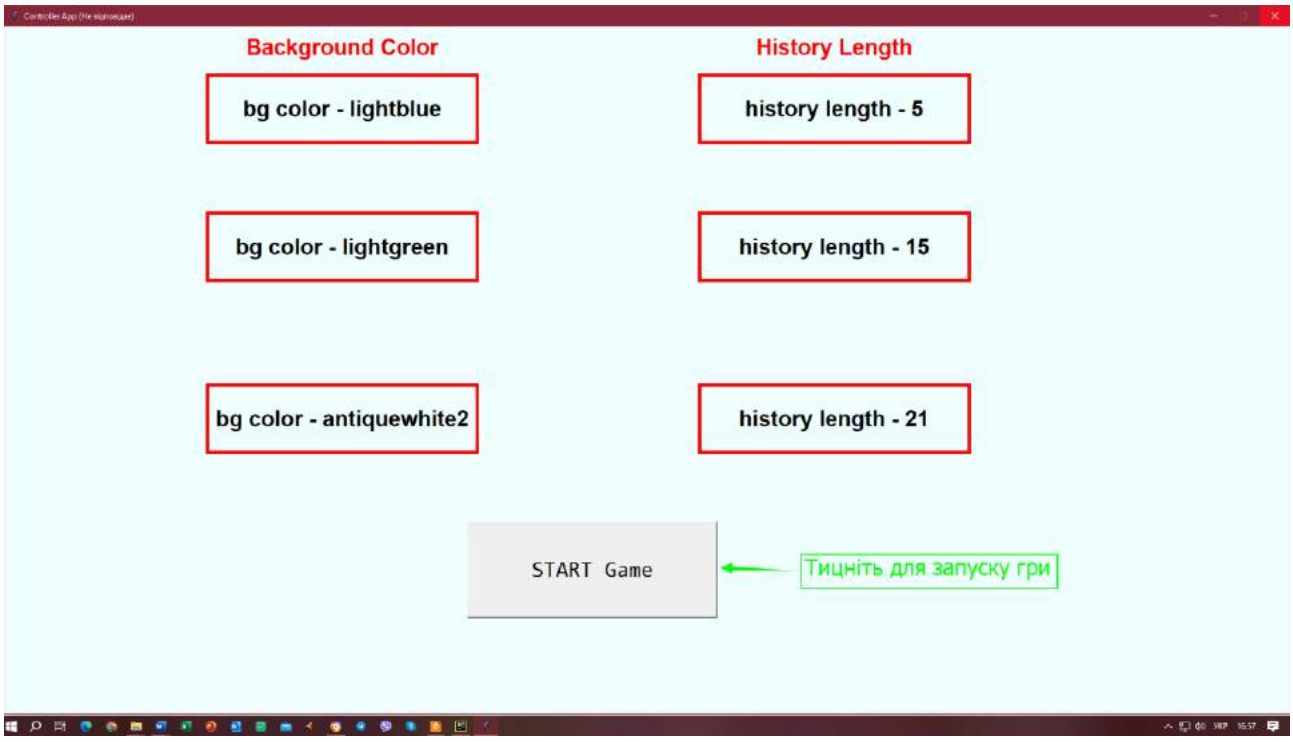


Рисунок 4 – Загальний вигляд контролера
для гри «Catch Sunny Bunny»

Вибір параметрів гри важливий: без нього не можна керувати наступними етапами програмного комплексу. У разі невиконанні вимог контролера на гравця очікує помилка та попереджувальне повідомлення про необхідність вибору параметрів «Catch Sunny Bunny» (Рисунок 5).

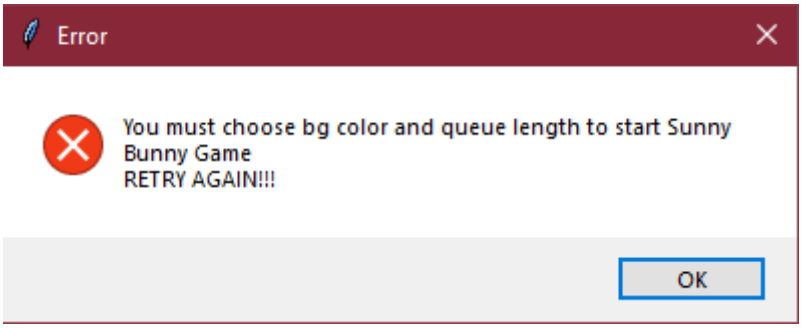


Рисунок 5 – Попереджувальне повідомлення
в разі не вибору необхідних параметрів

Під час вибору, активований параметр виділяється зеленою рамкою, а неактивний має рамку червоного кольору (Рисунок 6).

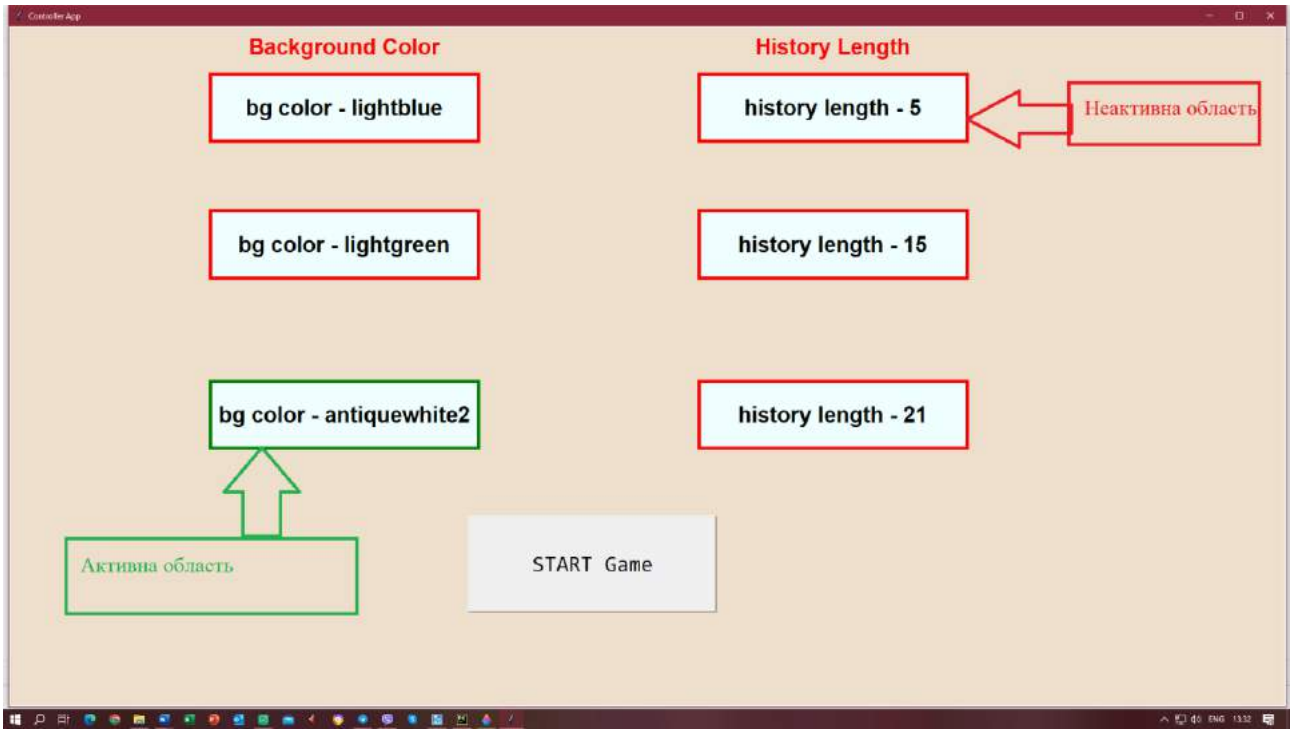


Рисунок 6 – Умовні позначення вибору параметрів

Після вибору всіх необхідних параметрів для гри перед особою з’явиться діалогове вікно для підтвердження операції (Рисунок 7). Натиснувши «Так», гра «Catch Sunny Bunny» розпочнеться, інакше – можна обрати інші параметри гри. Для виходу з додатка необхідно натиснути клавішу <Escape>.

ВАЖЛИВО! Для підтвердження/відхилення параметрів гри необхідно скористатись традиційним пристроєм вводу, наприклад за допомогою емулятора «миша» або клавіші `<Enter>`.

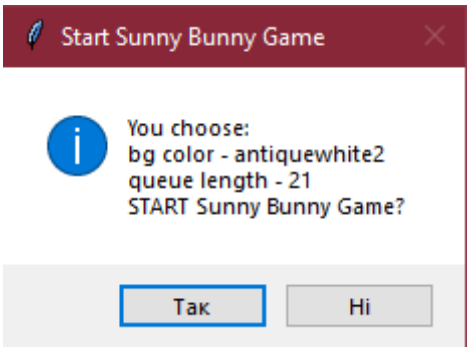


Рисунок 7 – Діалогове вікно підтвердження/відхилення операції

ЕТАП № 3. ГРА «CATCH SUNNY BUNNY»

Після успішного вибору параметрів гри особа може зіграти в тренувальну гру «Catch Sunny Bunny». Дана гра дає можливість оцінити, який алгоритм фільтрації для зменшення некерованих рухів курсора на основі кільцевої черги найкращий. Вона носить тренувальний характер, оскільки під час гри можна поліпшити навички в керуванні за допомогою Eye Tracking. Гра «Catch Sunny Bunny» розрахована для 20 сонячних зайчиків.

ПРАВИЛА ГРИ «CATCH SUNNY BUNNY»

На полотні Canvas, з обраним кольором фону після успішного проходження попереднього етапу, з’являється жовтий зайчик, чиї координати генеруються випадковим чином кожні п’ять секунд. Також на полотні рухаються три «кошеняти», два з яких позначають алгоритми фільтрації на основі кільцевої черги (Рисунок 8).

Кольори та алгоритми фільтрації «кошенят»:

1. сірий – нефільтроване значення (дійсне положення, що надійде у кільцеву чергу);
2. зелений – центроїд кільцевої черги;
3. синій – метод найменших квадратів (МНК) кільцевої черги.

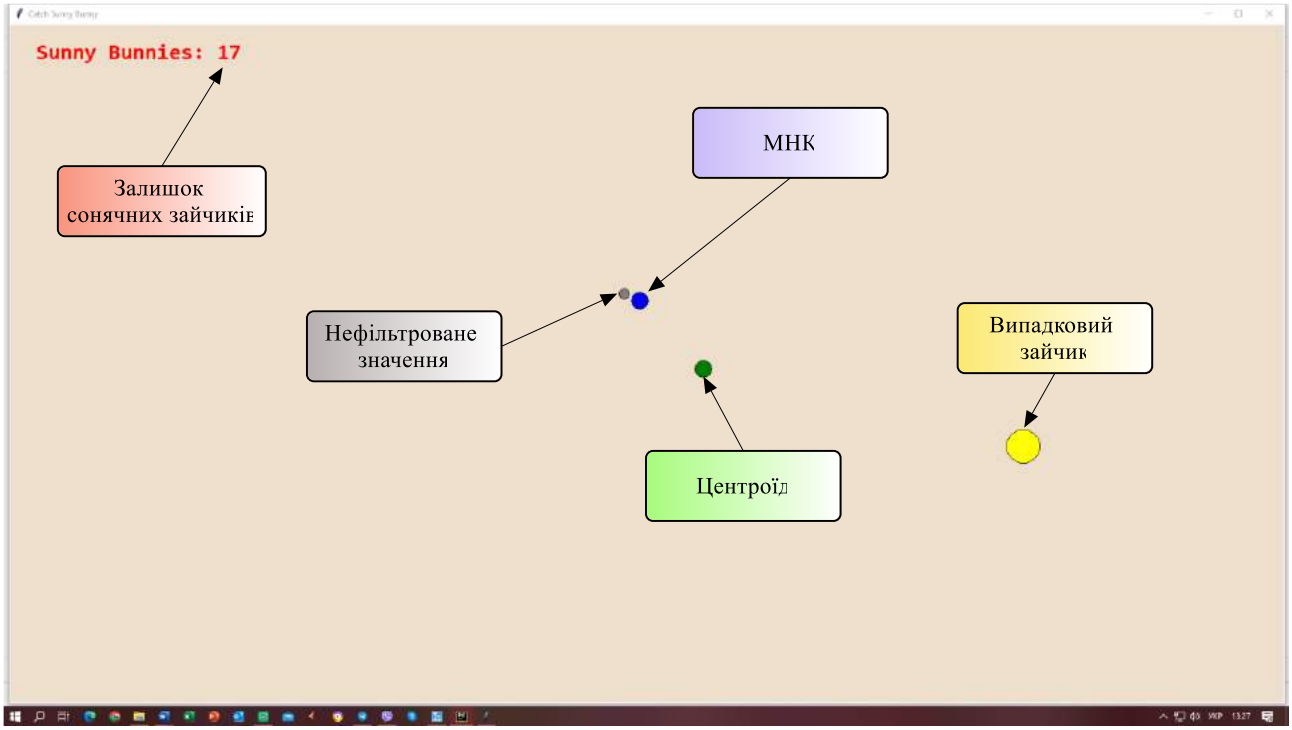


Рисунок 8 – Умовні позначення
для гри «Catch Sunny Bunny»

Для довідки, банер допоможе дізнатись, скільки зайчиків залишилось.

Після закінчення гри особа може отримати гістограму розподілу кожного алгоритму: дані для статистичного аналізу формуються внаслідок розрахунку *Евклідової відстані* між сонячним зайчиком та кожним «кошенятком» (Рисунок 9). Шляхом знаходження ***min*** середньоквадратичного відхилення можна отримати дані для аналізу щодо найкращого алгоритму фільтрації для певної довжини черги, яку пропонує програмний комплекс.

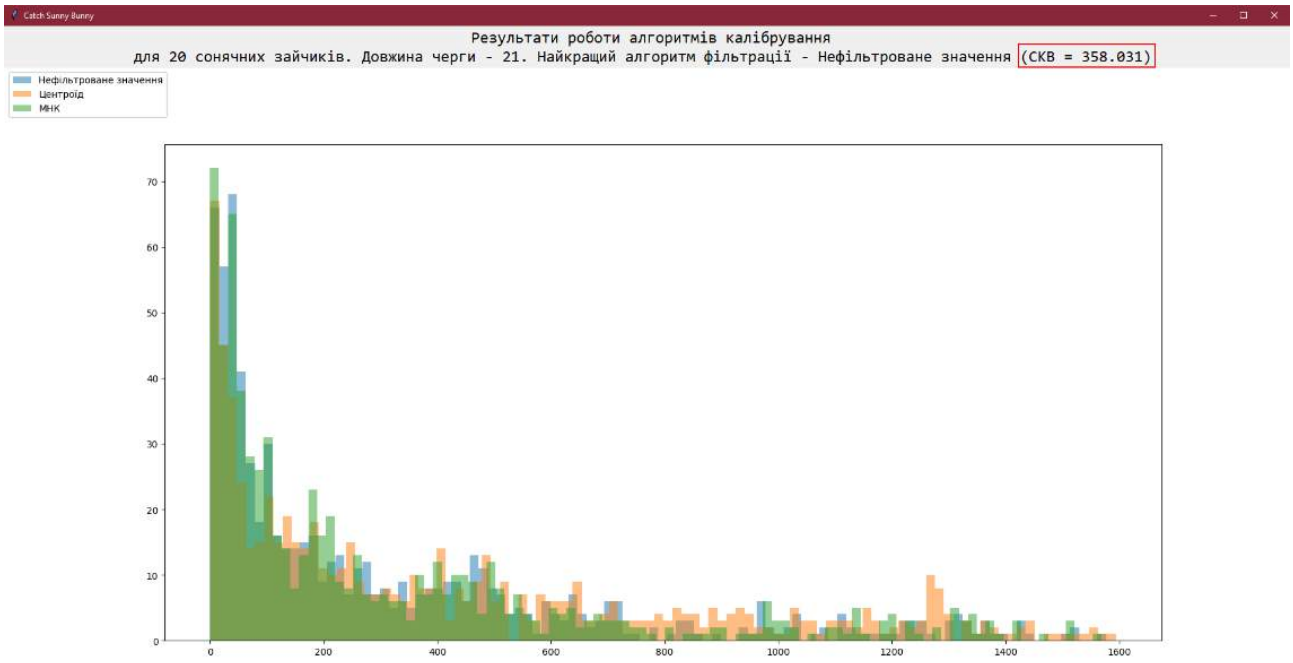


Рисунок 9 – Технології Data Science для визначення
допомагають визначити найкращий алгоритм фільтрації

ЕТАП № 4. ВИБІР ПАРАМЕТРІВ EYE TRACKER

Аналізуючи результат пройденої гри «Catch Sunny Bunny» особа може обрати один з двох алгоритмів фільтрації для зменшення некерованих рухів курсора, зумовлених постійним рухом людської зіниці:

1. центроїд;
2. МНК.

Механізм дії контролера щодо вибору алгоритму фільтрації для фінального етапу програмного комплексу подібний до використання контролера для гри «Catch Sunny Bunny» (див. ЕТАП № 2). Користувачеві пропонується вибрати алгоритм який виявився зручнішим або більш точним під час гри (Рисунок 10).

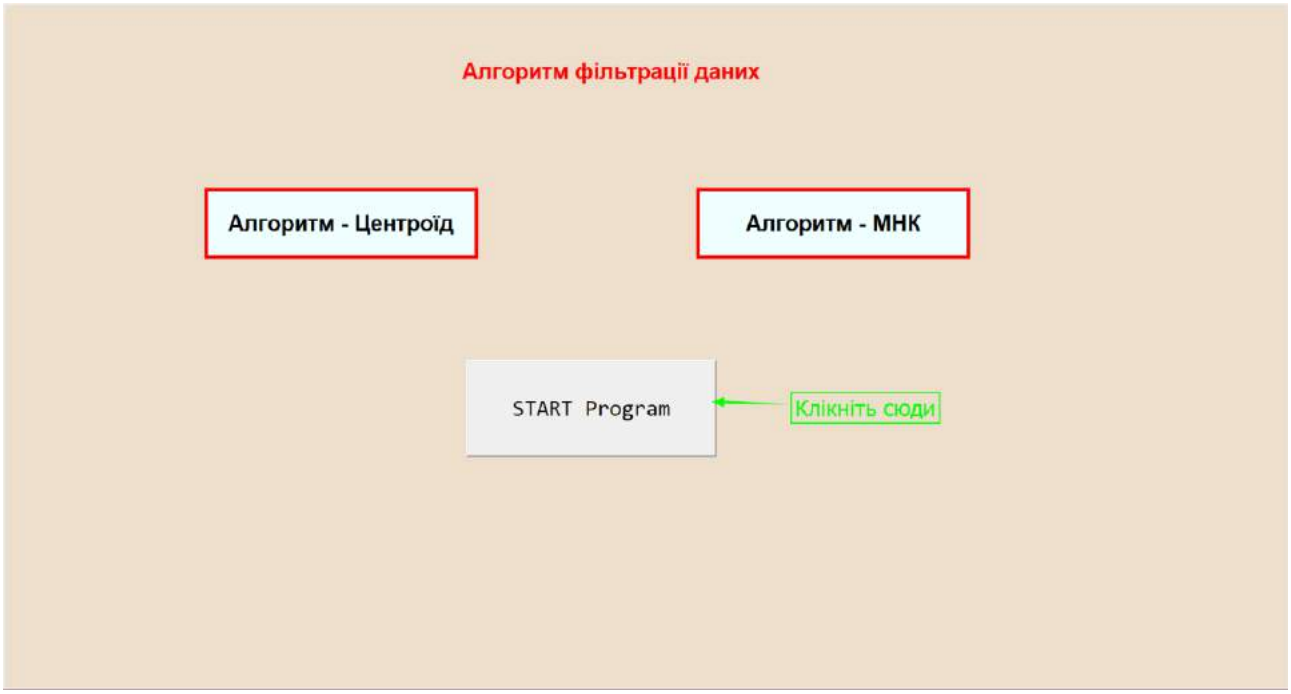


Рисунок 10 – Загальний вигляд
контролера для програми Eye Tracker

У разі не обрання алгоритму фільтрації програма видасть попереджувальне повідомлення про необхідність його вибору (Рисунок 11).

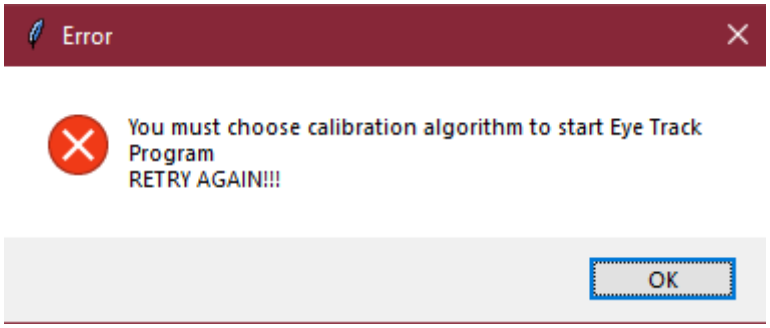


Рисунок 11 – Попереджувальне повідомлення
про необхідність вибору алгоритму фільтрації

Після вибору алгоритму фільтрації на основі кільцевої черги особі буде запропоновано підтвердити свій вибір (Рисунок 12). Інакше – надається можливість обрати інший алгоритм для зменшення некерованих рухів курсора.

Для виходу з додатка необхідно натиснути клавішу `<Escape>`.

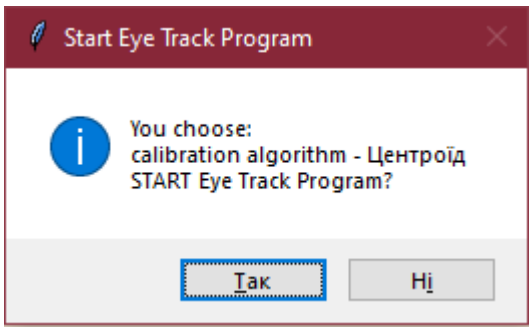


Рисунок 12 – Діалогове вікно
для підтвердження/відхилення

ЕТАП № 5. ВИБІР ОПЕРАЦІЇ

За допомогою GUI, що використовує технологію Eye Tracking, особа може побачити її практичне застосування, керуючи веб-сайтами однієї з чотирьох категорій (Рисунок 13):

1) «Харчування»;

2) «Освіта»;

3) «Побут»;

4) «Розваги».

Рисунок 13 – Загальний вигляд
Eye Tracker

Для довідки, банер допоможе дізнатись координати погляду користувача. Координати погляду варіюються в межах монітора:

- ширина – [0, 1920];
- висота – [0, 1080].

Кліпання ока в межах програмного комплексу, що використовує технологію Eye Tracking – імітація натискання лівої кнопки емулятора «миша» для вибору бажаної операції. Аби програма реагувала на кліпання очей, встановлено поріг, який дорівнює 0,01 одиниць (нормалізованих) на розсуд розробника. Якщо різниця ординат лівого ока (кліпання ока – вертикальний процес) менше зазначеного порога, то це означає, що особа кліпнула.

ВАЖЛИВО! Поріг кліпання ока – індивідуальний для кожної людини. Його зміна може підвищити чутливість Eye Tracker. Зміна порогу для кліпання відбувається в програмному коді. Таке рішення не ускладнює програмний код. В наступних версіях передбачено етап калібрування кліпання ока окремо.

Обробка події «кліпання» відбувається в одному потоці з головним циклом програми, що призводить до затримок відпрацювання. Для усунення подібних помилок необхідно створити окремий потік. Це буде реалізовано в наступних версіях ПЗ.

Після вибору бажаної операції відповідний банер змінює свій колір на зелений (Рисунок 14).

Рисунок 14 – Кольори банерів активності
активної/неактивної операції Eye Tracker

Після двох секунд відбувається «фліп» обраної операції, при чому всі інші залишаються неактивними (Рисунок 15).

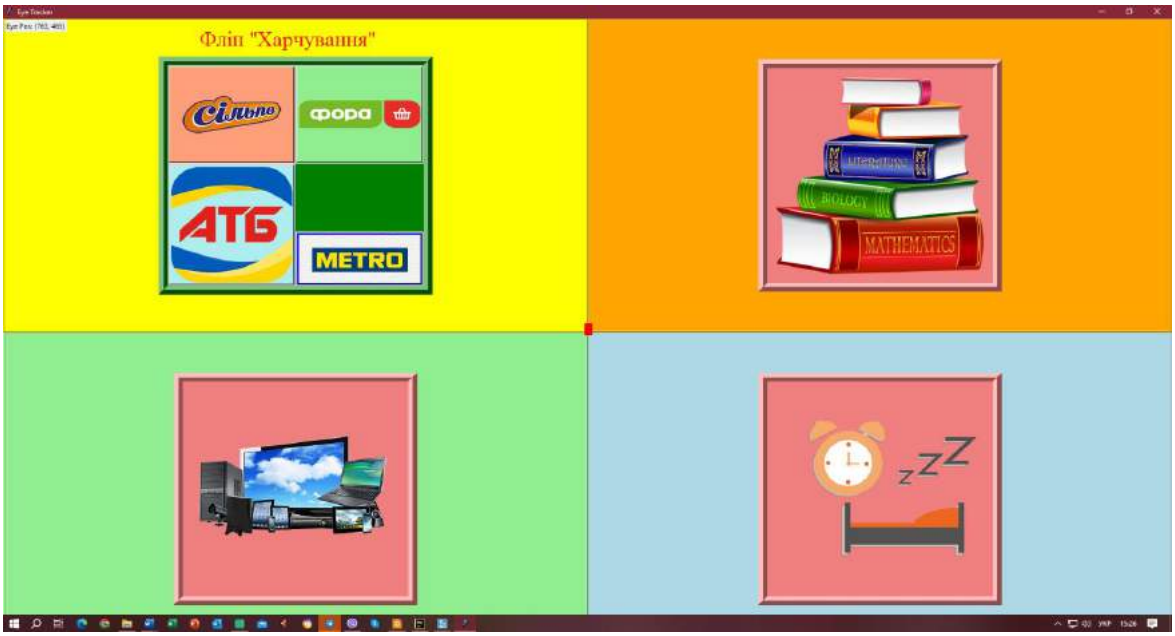


Рисунок 15 – «Фліп» обраної операції

Після «фліпу» обраної операції з’являються кнопки з Інтернет-ресурсами, з якими асоціюється кожна категорія. *Наприклад*, категорія «Харчування» включає чотири найпопулярніших магазини, що забезпечують населення України харчовими та побутовими товарами:

1. «Сільпо»;
2. «Фора»;
3. «АТБ»;
4. «МЕТРО».

Обравши один з чотирьох магазинів, особа має можливість керувати його веб-сторінкою. На рисунку 16 продемонстровано керування Інтернет-ресурсом магазину «Фора».

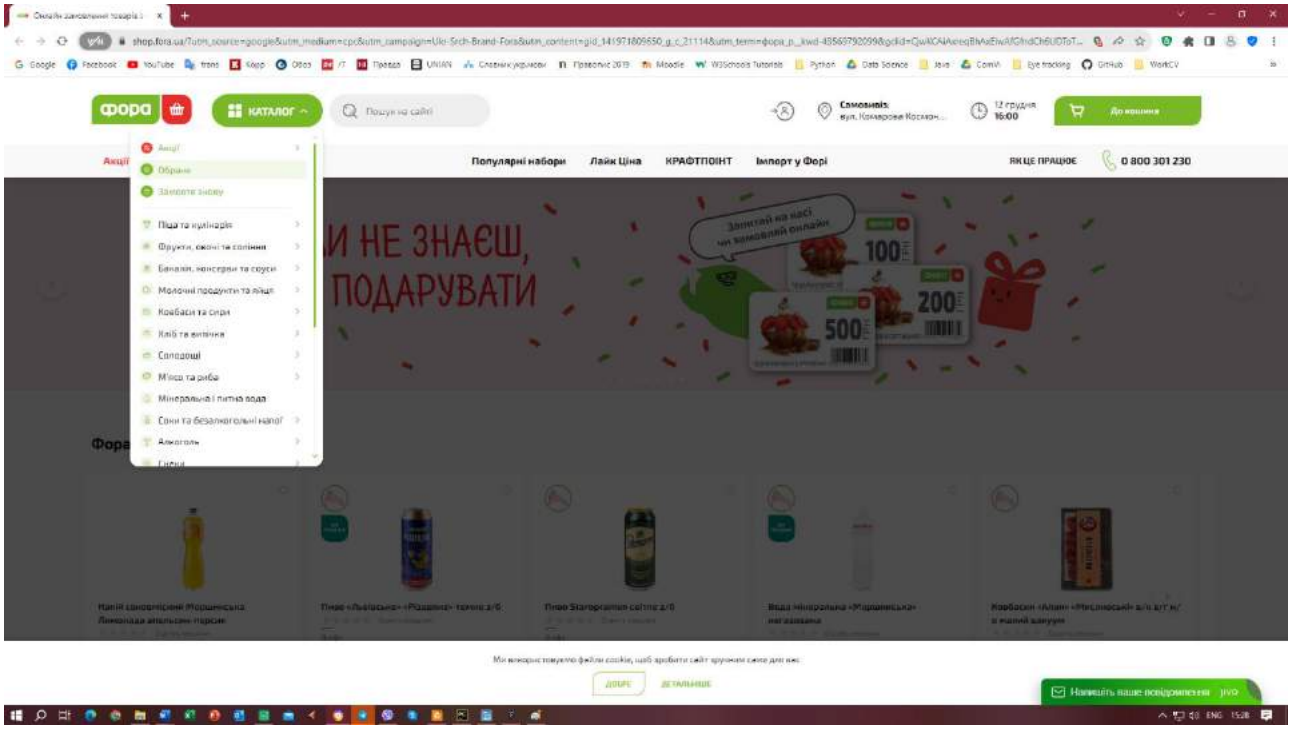


Рисунок 16 – Практичне застосування технології Eye Tracking
на прикладі керування Інтернет-ресурсами

Після завершення керування Інтернет-ресурсом однієї категорії існує можливість переключитись на іншу: банер попередньої категорії змінить свій колір на червоний, натомість банер новообраної категорії стане зеленим (Рисунок 17).

Рисунок 17 – Вибір іншої операції
за допомогою технології Eye Tracking

Для виходу з додатка необхідно натиснути клавішу `<Escape>`.

Це – завершальний етап, що пропонує програмний комплекс, який використовує технології Eye Tracking.

Демонстрація на прикладі керування Інтернет-ресурсами – лише одна з можливостей технології, чиї принципи базуються на відомому стеці технологій Computer Vision.

Дана інструкція буде доповнюватись з розширенням функціоналу програмного комплексу.

ДОДАТОК Ж

ТЕКСТ ПРОГРАМНОГО КОДУ

до наукової роботи

на тему: «Програмний комплекс управління інтерактивним графічним інтерфейсом за технологіями Eye Tracking»

Київ – 2024

run. py

```
"""
Програмний комплекс
управління графічним інтерфейсом
за допомогою технологій Eye Tracking
Кондратюк Наталії
ПІ-22-1м
"""

from modules.loggers import LoggerSetup

from modules.calibration import CalibrationApp
from modules.controller_app import Controller
from modules.game_bunny import GameSB
from modules.eye_track_controller_app import EyeTrackController
from modules.eye_track import EyeTracker

def main() :
    LoggerSetup().set_warning_logs().set_console_logs().set_logs_timetable()

    CalibrationApp()()
    Controller()()
    GameSB()()
    EyeTrackController()()
    EyeTracker()()

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

calibration_app. py

```
import logging
import time
from datetime import datetime

import tkinter as tk
import cv2
import matplotlib.pyplot as plt
import numpy as np
import mediapipe as mp

class CalibrationApp(tk.Tk) :
    """
    Механізм калібрування
    системи Eye Tracking
    """

    homography = []

    def __init__(self, *args, **kwargs):
        tk.Tk.__init__(self, *args, **kwargs)
        self.wm_title("Calibrate Eye Tracking" )

        self._logger : logging.Logger = logging.getLogger(type(self).__name__)

        self.screen_width, self.screen_height = self.winfo_screenwidth()-15,
self.winfo_screenheight()-65
        self.wm_geometry('%dx%d+0+0' % (self.screen_width, self.screen_height))
        self.canvas = tk.Canvas(self, width=self.screen_width,
```

```
height=self.screen_height)
        self.canvas.pack()
        self.bind('<Escape>', lambda e: self.destroy())

        self._logger.warning("Calibration started")
        self._calibrate()

    def _calibrate(self) :
        '''Часові налаштування калібрування на основі 9 точок'''
        num_slot = 1
        duration_slot = 4
        record_time = 2

        '''9 точок - позиція зайчика (матриця 3x3)'''
        m9 = [
            (64, 40),
            (948, 40),
            (1812, 40),
            (1812, 518),
            (1812, 968),
            (948, 968),
            (64, 968),
            (64, 518),
            (948, 518)
        ]

        circle_radius = 20  # розмір зайчика

        '''
        root = tk.Tk()
        screenW, screenH = root.winfo_screenwidth(), root.winfo_screenheight()
        root.wm_geometry("%dx%d" % (screenW, screenH))

        canvas = tk.Canvas(root, width=screenW, height=screenH)
        canvas.pack()'''
        mark = self.canvas.create_oval(m9[0][0] - circle_radius, m9[0][1] -
circle_radius, m9[0][0] + circle_radius, m9[0][1] + circle_radius,
                                       fill='white')
        countdown = self.canvas.create_text(m9[0][0], m9[0][1], text='',
font=('Helvetica 15 bold'))
        self.update()
        # time.sleep(3)
        for i in range(3, -1, -1):
            self.canvas.itemconfig(countdown, text=str(i))
            self.update()
            time.sleep(1)
        self.canvas.delete(countdown)

        start_time = datetime.now()

        mark_points = []  # для гомографії

        cX, cY = .5, .5  # початкова позиція очей

        mX = []
        mY = []

        face_mesh = mp.solutions.face_mesh.FaceMesh(
            refine_landmarks=True)  # виявлення 3-ох координат в просторі 468
точок
        cam = cv2.VideoCapture(0)

        while True and len(m9) >= num_slot:
            _, frame = cam.read()  # ігнорування першого аргументу
```

```
            frame = cv2.flip(frame, 1)  # вертикальний поворот
            ''''
            winCv = "Calibration Frame"
            cv2.namedWindow(winCv)
            cv2.moveWindow(winCv, 100, 510)
            cv2.imshow(winCv, frame)
            '''
            rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            detection = face_mesh.process(rgb_frame)  # виявлення обличчя
            landmarks = detection.multi_face_landmarks  # виявлення орієнтирів
обличчя
            now_time = datetime.now()
            slot_time = (now_time - start_time).total_seconds()

            if landmarks:
                single_face = detection.multi_face_landmarks[0].landmark

                cX = single_face[476].x + (single_face[474].x -
single_face[476].x) / 2
                cY = single_face[475].y + (single_face[477].y -
single_face[475].y) / 2

            if slot_time / duration_slot < num_slot:
                self.canvas.coords(mark,
                                   m9[num_slot - 1][0] - circle_radius,
                                   m9[num_slot - 1][1] - circle_radius,
                                   m9[num_slot - 1][0] + circle_radius,
                                   m9[num_slot - 1][1] + circle_radius)
                self.canvas.itemconfig(mark, fill='yellow')
                self.update()
                if slot_time > num_slot * duration_slot - record_time:
                    self.canvas.itemconfig(mark, fill='red')
                    self.update()

                    mX.append(cX)
                    mY.append(cY)
            else:
                mark_points.append((np.median(mX), np.median(mY)))
                mX = []
                mY = []
                num_slot += 1

        cam.release()
        cv2.destroyAllWindows()
        self._logger.warning("Calibration has been finished")

        CalibrationApp.homography = np.array(mark_points, dtype=float)

        '''Кластеризація середніх точок погляду'''
        plt.figure("Результати калібрування системи Eye Tracking")
        plt.title("Кластеризація середніх точок погляду очей")

        colors = ['red', 'green', 'blue', 'magenta', 'olive', 'cyan', 'salmon',
'purple', 'maroon']
        for i, p in enumerate(CalibrationApp.homography):
            plt.scatter(p[0], p[1], color=colors[i])

        plt.gca().invert_yaxis()
        plt.show()

    @staticmethod
    def get_homography() :
        '''Формування corresponding points для обчислення матриці гомографії'''
        screen_points = np.array([[0, 0],
```

```
                                    [960, 0],
                                    [1920, 0],
                                    [1920, 540],
                                    [1920, 1080],
                                    [960, 1080],
                                    [0, 1080],
                                    [0, 540],
                                    [960, 540]], dtype=float)


        '''Обчислення матриці гомографії'''
        homo_matrix, _ = cv2.findHomography(CalibrationApp.homography,
screen_points)
        return homo_matrix
    def __call__(self) : tk.mainloop()
```

controller_app. py

```
import logging

import tkinter as tk
from tkinter import messagebox

import numpy as np
import cv2
import mediapipe as mp
import pyautogui

from modules.data.queue_setup import FilterCenter
from modules.calibration import CalibrationApp
from modules.controller import ControlFrame

class Controller(tk.Tk) :
    """
    Користувацькі налаштування
    для гри "Catch Sunny Bunny"
    """

    bg_color: str = ""
    queue_len: int = 0

    def __init__( self, *args, **kwargs ) :
        tk.Tk.__init__( self, *args, **kwargs )
        tk.Tk.wm_title( self, "Controller App" )
        self._logger: logging.Logger = logging.getLogger(type(self).__name__)

        self.screen_width, self.screen_height = self.winfo_screenwidth()-15,
self.winfo_screenheight()-65
        self.wm_geometry('%dx%d+0+0' % (self.screen_width, self.screen_height))
        self.cX, self.cY = int(self.screen_width/2), int(self.screen_height/2)
        self.bind('<Escape>', lambda e: self.destroy())

        self.colors = ('lightblue', 'lightgreen', 'antiquewhite2')
        self.history_len = (5, 15, 21)
        self._flag = False

        self.homo_matrix = CalibrationApp.get_homography()

        '''Mouse Controll Settings'''
        self.history_length = 21  # довжина черги (історія попередніх координат
від веб-камери)
```

```
        self.ctr = FilterCenter(self.history_length)

        self.canvas = tk.Canvas(self, width=self.screen_width,
height=self.screen_height, bg='azure1')
        self.canvas.pack()

        self.bg_title = self.canvas.create_text(500, 30, text='Background
Color', fill='red', font=('Helvetica 25 bold'))

        self.banners_color = [
            ControlFrame(self.canvas, 300, 71, f'bg color - {self.colors[0]}'),
            ControlFrame(self.canvas, 300, 275, f'bg color - {self.colors[1]}'),
            ControlFrame(self.canvas, 300, 529, f'bg color - {self.colors[2]}')
        ]

        self.history_title = self.canvas.create_text(1229, 30, text='History
Length', fill='red', font=('Helvetica 25 bold'))
        self.banners_queue = [
            ControlFrame(self.canvas, 1029, 71, f'history length -
{self.history_len[0]}'),
            ControlFrame(self.canvas, 1029, 275, f'history length -
{self.history_len[1]}'),
            ControlFrame(self.canvas, 1029, 529, f'history length -
{self.history_len[2]}')
        ]

        self.button_start = tk.Button(self, text="START Game", width=20,
height=3, font=('Consolas', 25), command=self._start_game)
        self.button_start.place(relx=0.36, rely=0.72)

        self._color_change()

    @staticmethod
    def get_config() : return Controller.bg_color, Controller.queue_len

    def _start_game(self) :
        if Controller.bg_color == "" or Controller.queue_len == 0 :
            messagebox.showerror("Error",
                                 "You must choose bg color and queue length "
                                 "to start Sunny Bunny Game\n"
                                 "RETRY AGAIN!!!")
            self._logger.warning('You must choose bg color and queue length '
                                 'to start Sunny Bunny Game')
        else:
            start = messagebox.askyesno("Start Sunny Bunny Game",
                                        f"You choose:\nbg color -
{Controller.bg_color}\nqueue length - {Controller.queue_len}\n"
                                        f"START Sunny Bunny Game?", icon='info')
            self._logger.warning(f'You have chose bg color -
{Controller.bg_color} & queue length - {Controller.queue_len}')

            if start == True : self._flag = True

    def _color_change(self):
        face_mesh = mp.solutions.face_mesh.FaceMesh(refine_landmarks=True)  #
виявлення 3-ох координат в просторі 468 точок
        cam = cv2.VideoCapture(0)

        while True and not self._flag:
            _, frame = cam.read()  # ігнорування першого аргументу
            frame = cv2.flip(frame, 1)  # вертикальний поворот
            rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            face_detector = face_mesh.process(rgb_frame)  # розпізнавання
обличчя людини
```

```
            landmarks = face_detector.multi_face_landmarks  # виявлення
орієнтирів обличчя

            if landmarks:
                single_face = face_detector.multi_face_landmarks[0].landmark

                cX = single_face[476].x + (single_face[474].x -
single_face[476].x) / 2
                cY = single_face[475].y + (single_face[477].y -
single_face[475].y) / 2
                self.update()

                '''Calculate eye position'''
                eye_pos = (cX, cY)
                eye_pos_homo = np.append(eye_pos, 1)
                posNorm = self.homo_matrix.dot(eye_pos_homo)  # tx, ty, tz -
гомогенні координати
                '''
                Для переведення з однорідних координат в світові
                дві перших координати діляться на w != 0
                '''

                '''Sunny Bunny Centroid'''
                xCentroid, yCentroid = self.ctr.out_filter(int(posNorm[0] /
posNorm[2]), int(posNorm[1] / posNorm[2]))

                if xCentroid < 0: xCentroid = 0
                if xCentroid > self.screen_width: xCentroid = self.screen_width

                if yCentroid < 0: yCentroid = 0
                if yCentroid > self.screen_height: yCentroid =
self.screen_height

                pyautogui.moveTo(xCentroid, yCentroid)

                '''Кліпання лівого ока - вибір нової операції'''
                left_eye = [single_face[145], single_face[159]]
                blink = left_eye[0].y - left_eye[1].y  # різниця між краями
лівого ока - признак кліпання ока
                blink_threshold = 0.01  # все, що менше порогу - ознака кліпання
ока
                if blink < blink_threshold:
                    for i, ban in enumerate (self.banners_color):
                        if ban.check_pos(xCentroid,yCentroid):
                            ban.make_active(self.colors[i])
                            Controller.bg_color = self.colors[i]
                        else : ban.make_disabled()

                    for i, ban in enumerate(self.banners_queue):
                        if ban.check_pos(xCentroid, yCentroid):
                            ban.make_active()
                            Controller.queue_len = self.history_len[i]
                        else: ban.make_disabled()
                    pyautogui.click()

                    self.config(cursor='hand2')
                    self.update()

                self.config(cursor='arrow')
                self.update()

        cam.release()
        cv2.destroyAllWindows()
```

```
    def __call__(self): tk.mainloop()
```

game_bunny. py

```
import logging
import os
from datetime import datetime

import tkinter as tk
from PIL import Image, ImageTk

import numpy as np
import math as mt

import cv2
import mediapipe as mp

from modules.controller_app import Controller
from modules.calibration import CalibrationApp
from modules.data import FilterCenter, Data
from modules.visual import CameraFrame, MatplotlibFrame

from objects.sunny_bunny import SunnyBunny
from objects.bunny_catch import BunnyCatch

class GameSB( tk.Tk ) :
    """
    Тренувальна гра
    "Catch Sunny Bunny"
    для визначення найкращого алгоритму
    керування курсором
    """

    def __init__(self, *args, **kwargs):
        tk.Tk.__init__(self, *args, **kwargs)
        tk.Tk.wm_title(self, "Catch Sunny Bunny" )
        self._logger: logging.Logger = logging.getLogger(type(self).__name__)

        self.screen_width, self.screen_height = self.winfo_screenwidth()-15,
self.winfo_screenheight()-65
        self.wm_geometry('%dx%d+0+0' % (self.screen_width, self.screen_height))
        self.cX, self.cY = int(self.screen_width/2), int(self.screen_height/2)

        self.color, self.history_len = Controller.get_config()

        self.homo_matrix = CalibrationApp.get_homography()

        self.canvas = tk.Canvas( self, width=self.screen_width,
height=self.screen_height, bg=self.color )
        self.canvas.pack()

        self.ctr = FilterCenter(self.history_len)
        self.bunniesQ = self.plot_bunnies = 20
        self.dataset = Data( self.bunniesQ * 35 ) # розмір датасету для аналізу
(~35fps)

        self.sb = SunnyBunny(self.canvas, 50,'yellow')
        self.bunny_label = tk.Label( self, text=f"Sunny Bunnies:
{self.bunniesQ}", font= ('Consolas 24 bold'), bg=self.color, fg='red' )
        self.bunny_label.place(relx=0.02, rely=0.02)
```

```
        '''Налаштування камери для роботи'''
        self.camera_frame = CameraFrame(self)
        self.cam = cv2.VideoCapture(0)
        self.width, self.height = 800, 600

        self.cam.set(cv2.CAP_PROP_FRAME_WIDTH, self.width)
        self.cam.set(cv2.CAP_PROP_FRAME_HEIGHT, self.height)

        self.face_mesh = mp.solutions.face_mesh.FaceMesh(refine_landmarks=True)

        self.sbBase = BunnyCatch(self.canvas, 15, 'gray')
        self.sbCentroid = BunnyCatch(self.canvas, 25, 'green')
        self.sbMNK = BunnyCatch(self.canvas, 25, 'blue')

        self._logger.warning("Game Sunny Bunny started")
        self.set_pos_rand()
        self.get_eye_pos()

    def set_pos_rand(self):
        """
        Генерування сонячного зайчика у випадковому місці
        :return:
        """

        pos = self.sb.get_coords()
        border = 100 # сонячний зайчик не "прилипає" до країв робочого вікна
        x, y = np.random.randint(border, self.screen_width-border),
np.random.randint(border, self.screen_height-border)

        dx = x - pos[0]
        dy = y - pos[1]

        self.sb.pos(dx, dy)
        self.bunniesQ -= 1
        self.bunny_label.configure(text=f"Sunny Bunnies: {self.bunniesQ}")

        self.after(5000, self.set_pos_rand) # швидкість сонячного зайчика

    def set_pos_cam(self, sb, x, y):
        """
        Керування очима сонячним зайчиком
        :param sb: сонячний зайчик, що використовує один з алгоритмів
        :param x: Ox точки робочої області екрана
        :param y: Ox точки робочої області екрана
        :return:
        """

        pos = sb.get_coords()

        dx = x - pos[0]
        dy = y - pos[1]

        sb.pos(dx, dy)

    def get_eye_pos(self):
        """
        Основний цикл програми
        :return:
        """

        '''Крок № 1 - налаштування камери'''
        _, frame = self.cam.read()  # ігнорування першого аргументу
        if isinstance(frame, np.ndarray) :
```

```
            frame = cv2.flip(frame, 1)  # вертикальний поворот
            rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

            '''Вікно OpenCV у вікні Tkinter'''
            captured_image = Image.fromarray(rgb_frame)
            photo_image = ImageTk.PhotoImage(image=captured_image)

            self.camera_frame.camLabel.photo_image = photo_image
            self.camera_frame.camLabel.configure(image=photo_image)
            self.camera_frame.camLabel.after(10, self.get_eye_pos)

            '''Основний алгоритм роботи програми'''
            face_detector = self.face_mesh.process(rgb_frame)  # розпізнавання
обличчя людини
            landmarks = face_detector.multi_face_landmarks

            if landmarks:
                single_face = face_detector.multi_face_landmarks[0].landmark

                cX = single_face[476].x + (single_face[474].x -
single_face[476].x) / 2
                cY = single_face[475].y + (single_face[477].y -
single_face[475].y) / 2

                '''Calculate eye position'''
                eye_pos = (cX, cY)
                eye_pos_homo = np.append(eye_pos, 1)
                pos_norm = self.homo_matrix.dot(eye_pos_homo)  # tx, ty, tz -
гомогенні координати
                '''
                Для переведення з однорідних координат в світові
                дві перших координати діляться на w != 0
                '''

                '''Sunny Bunny Base'''
                xScreen, yScreen = int(pos_norm[0] / pos_norm[2]),
int(pos_norm[1] / pos_norm[2])

                '''
                Погляд поза межами екрана -
                значення курсору миші - нижня/верхня межа
                висоти та ширини екрана
                '''

                if xScreen < 0: xScreen = 0
                if xScreen > self.screen_width: xScreen = self.screen_width

                if yScreen < 0: yScreen = 0
                if yScreen > self.screen_height: yScreen = self.screen_height

                self.set_pos_cam( self.sbBase, xScreen, yScreen )

                '''Sunny Bunny Centroid'''
                xCentroid, yCentroid = self.ctr.out_filter(int(pos_norm[0] /
pos_norm[2]), int(pos_norm[1] / pos_norm[2]))

                if xCentroid < 0: xCentroid = 0
                if xCentroid > self.screen_width: xCentroid = self.screen_width

                if yCentroid < 0: yCentroid = 0
                if yCentroid > self.screen_height: yCentroid =
self.screen_height

                self.set_pos_cam( self.sbCentroid, xCentroid, yCentroid )
```

```
                '''Sunny Bunny MNK'''
                xMNK, yMNK = self.ctr.out_filterMNK(int(pos_norm[0] /
pos_norm[2]), int(pos_norm[1] / pos_norm[2]))

                if xMNK < 0: xMNK = 0
                if xMNK > self.screen_width: xMNK = self.screen_width

                if yMNK < 0: yMNK = 0
                if yMNK > self.screen_height: yMNK = self.screen_height

                self.set_pos_cam( self.sbMNK, xMNK, yMNK)
                self.dataset.add_data((self.sb.get_coords()[0],
self.sb.get_coords()[1]),
                                      (self.sbCentroid.get_coords()[0],
self.sbCentroid.get_coords()[1]),
                                      (self.sbMNK.get_coords()[0],
self.sbMNK.get_coords()[1]),
                                      (self.sbBase.get_coords()[0],
self.sbBase.get_coords()[1]))

        if self.bunniesQ == 0 :
            self.cam.release()
            self.canvas.destroy()
            self.bunny_label.destroy()
            self._logger.warning("Game Sunny Bunny finished")
            self.plot_data() # графічний аналіз даних

            '''
            Аналітичний аналіз даних:
            отримання статичних характеристик
            всіх алгоритмів задля точнішого висновку
            '''
            print('##########################')
            print( 'Визначення найкращого алгоритму калібрування' )
            self.print_stat( self.dataset.get_dist_eye(), 'роботи
нефільтрованого значення' )
            self.print_stat(self.dataset.get_dist_centroid(), 'роботи
центроїда')
            self.print_stat(self.dataset.get_distMNK(), 'роботи МНК')

            self.save_txt()

    def plot_data(self):
        """
        Графічний аналіз розроблених алгоритмів
        керування курсором миші
        :return:
        """

        min_var, min_algorithm = self.show_min()
        log_message = ''
        if min_algorithm == "Нефільтроване значення": log_message = 'Non-
filtered'
        if min_algorithm == "Центроїд" : log_message = 'Centroid'
        if min_algorithm == "МНК": log_message = 'MNK'

        self.plotter = MatplotlibFrame(self, f'Результати роботи алгоритмів
калібрування\n'
                                             f'для {self.plot_bunnies} сонячних
зайчиків. '
                                             f'Довжина черги - {self.history_len}.
'
                                             f'Найкращий алгоритм клібрування -
```

```
{min_algorithm} (СКВ = {min_var:.3f})', ('Consolas 18'))
        self.plotter.pack()
        self._logger.warning(f'The best filter is {log_message}')

        plt_hist = self.plotter.f.subplots(1, 1)
        plt_hist.hist(self.dataset.get_dist_eye(), bins=100, alpha=0.5,
label='Нефільтроване значення')
        plt_hist.hist( self.dataset.get_dist_centroid(), bins=100, alpha=0.5,
label='Центроїд' )
        plt_hist.hist(self.dataset.get_distMNK(), bins=100, alpha=0.5,
label='МНК')
        self.plotter.f.legend(loc=2)

    def __get_statistcs(self, dataset):
        """
        Отримання статистичних даних кожного датасету
        :param dataset: датасет кожного алгоритму
        :return: статистичні характеристики одного з алгоритмів
        """

        statistics = []

        median = np.median( dataset ) # медіана датасету
        statistics.append( median )

        variance = np.var( dataset ) # дисперсія датасету
        statistics.append( variance )

        std = mt.sqrt( variance ) # середньоквадратичне відхилення датасету
        statistics.append( std )

        return statistics

    def print_stat(self, dataset, name, in_file=True):
        """
        Аналітичний аналіз даних
        шляхом порівняння статистичних характеристик
        :param dataset: датасет кожного алгоритму
        :param name: назва кожного датасету
        :return:
        """

        dataStat = self.__get_statistcs( dataset ) # статистичні дані кожного
датасету

        print('**************************')
        print( f'Статистичні хар-ки {name}' )
        print('**************************')
        print( f'Математичне сподівання = {dataStat[0]:.3f}' )
        print( f'Дисперсія = {dataStat[1]:.3f}' )
        print( f'Сігма (СКВ) = {dataStat[2]:.3f}' )
        print('---------------------------')

    def save_txt(self) :
        stat_non_filter = self.__get_statistcs(self.dataset.get_dist_eye())
        stat_centroid = self.__get_statistcs(self.dataset.get_dist_centroid())
        stat_MNK = self.__get_statistcs(self.dataset.get_distMNK())

        time_session = datetime.now()
        session = f'{time_session.day}/{time_session.month}/{time_session.year}
{time_session.hour}:{time_session.minute}'

        stat_src = 'external\\stats'
        if not os.path.exists(stat_src): os.makedirs(stat_src)
```

```
        time_marker = datetime.now().strftime('%d-%m-%Y-%H-%M')
        with open(f'{stat_src}\\stat_file_{time_marker}.txt', "w",
encoding='utf-8') as stat_file:
            stat_file.write(f'Визначення найкращого алгоритму калібрування для
сеансу {session}')
            stat_file.write(f'\nСтатистичні хар-ки роботи нефільтрованого
значення')
            stat_file.write('\n************************')
            stat_file.write(f'\nМатематичне сподівання =
{stat_non_filter[0]:.3f}')
            stat_file.write(f'\nДисперсія = {stat_non_filter[1]:.3f}')
            stat_file.write(f'\nСігма (СКВ) = {stat_non_filter[2]:.3f}')
            stat_file.write('\n---------------------------')

            stat_file.write(f'\nСтатистичні хар-ки роботи центроїда')
            stat_file.write('\n************************')
            stat_file.write(f'\nМатематичне сподівання =
{stat_centroid[0]:.3f}')
            stat_file.write(f'\nДисперсія = {stat_centroid[1]:.3f}')
            stat_file.write(f'\nСігма (СКВ) = {stat_centroid[2]:.3f}')
            stat_file.write('\n---------------------------')

            stat_file.write(f'\nСтатистичні хар-ки роботи МНК')
            stat_file.write('\n************************')
            stat_file.write(f'\nМатематичне сподівання = {stat_MNK[0]:.3f}')
            stat_file.write(f'\nДисперсія = {stat_MNK[1]:.3f}')
            stat_file.write(f'\nСігма (СКВ) = {stat_MNK[2]:.3f}')
            stat_file.write('\n---------------------------\n\n')

            stat_file.close()

    def show_min(self) :
        algorithm_set = ['Нефільтроване значення', 'Центроїд', 'МНК']
        stat_non_filter = self.__get_statistcs(self.dataset.get_dist_eye())
        stat_centroid = self.__get_statistcs(self.dataset.get_dist_centroid())
        stat_MNK = self.__get_statistcs(self.dataset.get_distMNK())

        compare_var = [stat_non_filter[2], stat_centroid[2], stat_MNK[2]]
        comparator = np.array(compare_var)
        min_algorithm = np.argmin(comparator)
        min_var = np.min(comparator)

        return min_var, algorithm_set[min_algorithm]

    def __call__(self): tk.mainloop()
```

eye_track_controller_app. py

```
import logging

import tkinter as tk
from tkinter import messagebox

import numpy as np
import cv2
import mediapipe as mp
import pyautogui

from modules.data.queue_setup import FilterCenter
from modules.calibration import CalibrationApp
from modules.controller_app import Controller
from modules.controller import ControlFrame

class EyeTrackController(tk.Tk) :
    """
    Користувацькі налаштування
    для процесу Eye Tracking
    """

    algorithm: str = ""

    def __init__( self, *args, **kwargs ) :
        tk.Tk.__init__( self, *args, **kwargs )
        tk.Tk.wm_title( self, "Controller App" )
        self._logger: logging.Logger = logging.getLogger(type(self).__name__)

        self.screen_width, self.screen_height = self.winfo_screenwidth()-15,
self.winfo_screenheight()-65
        self.wm_geometry('%dx%d+0+0' % (self.screen_width, self.screen_height))
        self.cX, self.cY = int(self.screen_width/2), int(self.screen_height/2)
        self.bind('<Escape>', lambda e: self.destroy())

        self.algorithms = ("Центроїд", "МНК")
        self.bg = Controller.get_config()[0]
        self._flag = False

        self.homo_matrix = CalibrationApp.get_homography()
        self.face_mesh = mp.solutions.face_mesh.FaceMesh(refine_landmarks=True)

        '''Mouse Controll Settings'''
        self.history_len = 21  # довжина черги (історія попередніх координат від
веб-камери)
        self.ctr = FilterCenter(self.history_len)

        self.canvas = tk.Canvas(self, width=self.screen_width,
height=self.screen_height, bg=self.bg)
        self.canvas.pack()

        self.bg_title = self.canvas.create_text(900, 100, text='Алгоритм
калібрування', fill='red', font=('Helvetica 25 bold'))

        self.banners_color = [
            ControlFrame(self.canvas, 300, 275, f'Алгоритм -
{self.algorithms[0]}'),
            ControlFrame(self.canvas, 1029, 275, f'Алгоритм -
{self.algorithms[1]}')
        ]
```

```
        self.button_start = tk.Button(self, text="START Program", width=20,
height=3, font=('Consolas', 25), command=self._start_game)
        self.button_start.place(relx=0.36, rely=0.52)

        self._color_change()

    @staticmethod
    def get_config() : return EyeTrackController.algorithm

    def _start_game(self) :
        if EyeTrackController.algorithm == "" :
            messagebox.showerror("Error",
                                 "You must choose calibration algorithm "
                                 "to start Eye Track Program\n"
                                 "RETRY AGAIN!!!")
            self._logger.warning('You must choose filter algorithm '
                                 'to start Eye Track program')
        else:
            start = messagebox.askyesno("Start Eye Track Program",
                                        f"You choose:\ncalibration algorithm -
{EyeTrackController.algorithm}\n"
                                        f"START Eye Track Program?", icon='info')
            log_message = ''
            if EyeTrackController.algorithm == 'Центроїд' : log_message =
'Centroid'
            if EyeTrackController.algorithm == 'МНК': log_message = 'MNK'
            self._logger.warning(f'You have chose filter algorithm -
{log_message}')

            if start == True : self._flag = True

    def _color_change(self):
        face_mesh = mp.solutions.face_mesh.FaceMesh(refine_landmarks=True)  #
виявлення 3-ох координат в просторі 468 точок
        cam = cv2.VideoCapture(0)

        while True and not self._flag:
            _, frame = cam.read()  # ігнорування першого аргументу
            frame = cv2.flip(frame, 1)  # вертикальний поворот
            rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            face_detector = face_mesh.process(rgb_frame)  # розпізнавання
обличчя людини
            landmarks = face_detector.multi_face_landmarks  # виявлення
орієнтирів обличчя

            if landmarks:
                single_face = face_detector.multi_face_landmarks[0].landmark

                cX = single_face[476].x + (single_face[474].x -
single_face[476].x) / 2
                cY = single_face[475].y + (single_face[477].y -
single_face[475].y) / 2
                self.update()

                '''Calculate eye position'''
                eye_pos = (cX, cY)
                eye_pos_homo = np.append(eye_pos, 1)
                pos_norm = self.homo_matrix.dot(eye_pos_homo)  # tx, ty, tz -
гомогенні координати
                '''
                Для переведення з однорідних координат в світові
                дві перших координати діляться на w != 0
                '''
```

```
                '''Sunny Bunny Centroid'''
                xCentroid, yCentroid = self.ctr.out_filter(int(pos_norm[0] /
pos_norm[2]), int(pos_norm[1] / pos_norm[2]))

                if xCentroid < 0: xCentroid = 0
                if xCentroid > self.screen_width: xCentroid = self.screen_width

                if yCentroid < 0: yCentroid = 0
                if yCentroid > self.screen_height: yCentroid =
self.screen_height

                pyautogui.moveTo(xCentroid, yCentroid)

                '''Кліпання лівого ока - вибір нової операції'''
                left_eye = [single_face[145], single_face[159]]
                blink = left_eye[0].y - left_eye[1].y  # різниця між краями
лівого ока - признак кліпання ока
                blink_threshold = 0.01  # все, що менше порогу - ознака кліпання
ока
                if blink < blink_threshold:
                    for i, ban in enumerate (self.banners_color):
                        if ban.check_pos(xCentroid,yCentroid):
                            ban.make_active()
                            EyeTrackController.algorithm = self.algorithms[i]
                        else : ban.make_disabled()

                    pyautogui.click()

                    self.config(cursor='hand2')
                    self.update()

                self.config(cursor='arrow')
                self.update()

        cam.release()
        cv2.destroyAllWindows()

    def __call__(self): tk.mainloop()
```

eye_track. py

```
import logging

import webbrowser
import tkinter as tk

import cv2
import numpy as np
import mediapipe as mp

import pyautogui
pyautogui.FAILSAFE = False

from modules.data.queue_setup import FilterCenter
from modules.calibration import CalibrationApp
from modules.eye_track_controller_app import EyeTrackController
from modules.controller_app import Controller

from objects.activity_downloader import ActivityDownloader
from objects.shop_downloader import ShopDownloader

class EyeTracker(tk.Tk) :
    """
    GUI, що відображає практичне застосування
    технології Eye Tracking на прикладі
    керування Інтернет-ресурсами однієї
    з чотирьох категорій
    """

    def __init__(self, *args, **kwargs) :
        tk.Tk.__init__(self, *args, **kwargs)
        tk.Tk.wm_title(self, "Eye Tracker")
        self._logger: logging.Logger = logging.getLogger(type(self).__name__)

        self._screen_width, self._screen_height = self.winfo_screenwidth(),
self.winfo_screenheight()
        tk.Tk.wm_state( self, 'zoomed' )
        self.bind('<Escape>', lambda e: self.destroy())

        self._activity_images = ActivityDownloader().get_image
        self._logos = ShopDownloader().get_image
        self.active, self.non_active = 'green', '#F08080'

        self.homoMatrix = CalibrationApp.get_homography()

        '''Mouse Controll Settings'''
        self.history_len = Controller.get_config()[1]  # довжина черги (історія
попередніх координат від веб-камери)
        self.algorithm = EyeTrackController.get_config()
        if self.algorithm == "Центроїд":
            self._logger.warning('You choose Centroid filter algorithm')
        if self.algorithm == "МНК":
            self._logger.warning('You choose MNK filter algorithm')

        self.ctr = FilterCenter(self.history_len)

        self._shop_frame = tk.Frame(self, width=int(self._screen_width / 2),
height=int(self._screen_height / 2),
                                    bg='yellow', relief=tk.RIDGE, borderwidth=2)

        self._eat_frame = tk.Frame(self._shop_frame, width=400, height=400,
```

```
bg=self.non_active, relief=tk.RIDGE, borderwidth=15)
        self._eat_landmark = tk.Label(self._eat_frame, bg=self.non_active,
image=self._activity_images[0])
        self._eat_landmark.place(relx=0.5, rely=0.5, anchor=tk.CENTER)
        self._eat_frame.place(relx=0.5, rely=0.5, anchor=tk.CENTER)

        self._education_frame = tk.Frame(self, width=int(self._screen_width /
2), height=int(self._screen_height / 2),
                                         bg='orange', relief=tk.RIDGE, borderwidth=2)
        self._edu_frame = tk.Frame(self._education_frame, width=400, height=400,
                                   bg=self.non_active, relief=tk.RIDGE,
borderwidth=15)
        self._edu_img = [edu_logo for edu_logo in
self._activity_images[1]['edu'].keys()]
        self._edu_landmark = tk.Label(self._edu_frame, bg=self.non_active,
image=self._edu_img[0])
        self._edu_landmark.place(relx=0.5, rely=0.5, anchor=tk.CENTER)
        self._edu_frame.place(relx=0.5, rely=0.5, anchor=tk.CENTER)

        self._house_frame = tk.Frame(self, width=int(self._screen_width / 2),
height=int(self._screen_height / 2),
                                     bg='lightgreen', relief=tk.RIDGE, borderwidth=2)
        self._electro_frame = tk.Frame(self._house_frame, width=400, height=400,
                                       bg=self.non_active, relief=tk.RIDGE,
borderwidth=15)
        self._house_img = [house_logo for house_logo in
self._activity_images[2]['house'].keys()]
        self._electro_landmark = tk.Label(self._electro_frame,
bg=self.non_active, image=self._house_img[0])
        self._electro_landmark.place(relx=0.5, rely=0.5, anchor=tk.CENTER)
        self._electro_frame.place(relx=0.5, rely=0.5, anchor=tk.CENTER)

        self._rest_frame = tk.Frame(self, width=int(self._screen_width / 2),
height=int(self._screen_height / 2),
                                    bg='lightblue', relief=tk.RIDGE, borderwidth=2)
        self._concert_frame = tk.Frame(self._rest_frame, width=400, height=400,
                                       bg=self.non_active, relief=tk.RIDGE,
borderwidth=15)
        self._rest_img = [rest_logo for rest_logo in
self._activity_images[3]['rest'].keys()]
        self._rest_landmark = tk.Label(self._concert_frame, bg=self.non_active,
image=self._rest_img[0])
        self._rest_landmark.place(relx=0.5, rely=0.5, anchor=tk.CENTER)
        self._concert_frame.place(relx=0.5, rely=0.5, anchor=tk.CENTER)

        '''
        Точка перетину всіх Frame -
        орієнтир переходу до нової операції
        '''
        self.landmarkMain = tk.Label(self, bg='red', width=1, height=1)
        self.landmarkMain.place(relx=0.5, rely=0.52, anchor=tk.CENTER)
        tk.Tk.update(self)
        self.center = (self.landmarkMain.winfo_rootx(),
self.landmarkMain.winfo_rooty())

        self.eyeLabel = tk.Label(self._shop_frame, text='Eye Pos: ')
        self.eyeLabel.place(relx=0, rely=0)

        self._shop_frame.grid(row=0, column=0, sticky=tk.NW)
        self._education_frame.grid(row=0, column=1, sticky=tk.NE)
        self._house_frame.grid(row=1, column=0, sticky=tk.SW)
        self._rest_frame.grid(row=1, column=1, sticky=tk.SE)

        self._logger.warning('Eye Track Program starts')
```

```
        self._main()


    def _open_url(self, url):
        self._logger.warning(f'You opened {url}')
        webbrowser.open(url)

    def _flipper(self, landmark, frame_flipped, logos, name:str=None,
img_key=None):
        landmark.place_forget()

        if isinstance(logos, list) :
            self._logger.warning('You choose Eat Category')

            shop_silpo = tk.Frame(frame_flipped, width=400, height=400,
relief=tk.RIDGE, borderwidth=2)
            silpo_btn = tk.Button(shop_silpo, bg='#FFA07A', image=logos[0],
command=lambda: self._open_url(

'https://shop.silpo.ua/?gclid=CjwKCAiAxreqBhAxEiwAfGfndGv0w0T1m4C1Ke0ZeLMkRZ8YTC
Fii2nckcPH9kj0P0HlKIyuKEA6GxoCkrIQAvD_BwE'))
            silpo_btn.pack(anchor=tk.CENTER)

            shop_fora = tk.Frame(frame_flipped, width=400, height=400,
relief=tk.RIDGE, borderwidth=2)

            fora_btn = tk.Button(shop_fora, bg='#90EE90', image=logos[1],
                                 command=lambda: self._open_url(

'https://shop.fora.ua/?utm_source=google&utm_medium=cpc&utm_campaign=Ukr-Srch-
Brand-
Fora&utm_content=gid_141971809650_g_c_21114&utm_term=%D1%84%D0%BE%D1%80%D0%B0_p_
_kwd-
43569792099&gclid=CjwKCAiAxreqBhAxEiwAfGfndCh6UDToTmBnMeRTRvq6Px8TcBZ_Ptd9eFtC1C
STyKyLwd4PXTW5VBoCW1oQAvD_BwE'))
            fora_btn.pack(anchor=tk.CENTER)

            shop_atb = tk.Frame(frame_flipped, width=400, height=400,
relief=tk.RIDGE, borderwidth=2)
            atb_btn = tk.Button(shop_atb, bg='#AFEEEE', image=logos[2],
                                command=lambda:
self._open_url('https://www.atbmarket.com/'))
            atb_btn.pack(anchor=tk.CENTER)

            shop_metro = tk.Frame(frame_flipped, width=400, height=400,
relief=tk.RIDGE, borderwidth=2)
            metro_btn = tk.Button(shop_metro, bg='#0000FF', image=logos[3],
                                  command=lambda:
self._open_url('https://metro.zakaz.ua/uk/'))
            metro_btn.pack(anchor=tk.CENTER)

            shop_silpo.grid(row=0, column=0, sticky=tk.NW)
            shop_fora.grid(row=0, column=1, sticky=tk.NE)
            shop_atb.grid(row=1, column=0, sticky=tk.SW)
            shop_metro.grid(row=1, column=1, sticky=tk.SE)
        elif isinstance(logos, dict) :
            if name == 'edu' :
                self._logger.warning('You choose Education Category')

                edu_frame = tk.Frame(frame_flipped, width=400, height=400,
relief=tk.RIDGE, borderwidth=2)
                edu_btn = tk.Button(edu_frame, image=logos[name].get(img_key),
                                    command=lambda :
self._open_url('https://osvita.ua/'))
```

```
                edu_btn.pack(anchor=tk.CENTER)
                edu_frame.grid(row=0, column=0, sticky=tk.NSEW)
            elif name == 'house' :
                self._logger.warning('You choose House Category')

                rozetka_frame = tk.Frame(frame_flipped, width=400, height=400,
relief=tk.RIDGE, borderwidth=2)

                rozetka_btn = tk.Button(rozetka_frame,
image=logos[name][img_key].get('rozetka'),
                                        command=lambda :
self._open_url('https://rozetka.com.ua/ua/'))

                rozetka_btn.pack(anchor=tk.CENTER)
                rozetka_frame.grid(row=0, column=0, sticky=tk.W)

                comfy_frame = tk.Frame(frame_flipped, width=400, height=400,
relief=tk.RIDGE, borderwidth=2)

                comfy_btn = tk.Button(comfy_frame,
image=logos[name][img_key].get('comfy'),
                                        command=lambda :
self._open_url('https://comfy.ua/ua/korostyshiv/'))

                comfy_btn.pack(anchor=tk.CENTER)
                comfy_frame.grid(row=0, column=1, sticky=tk.E)
            elif name == 'rest' :
                self._logger.warning('You choose Rest Category')

                rest_frame = tk.Frame(frame_flipped, width=400, height=400,
relief=tk.RIDGE, borderwidth=2)
                rest_btn = tk.Button(rest_frame, image=logos[name].get(img_key),
                                        command=lambda :
self._open_url('https://kontramarka.ua/uk'))

                rest_btn.pack(anchor=tk.CENTER)
                rest_frame.grid(row=0, column=0, sticky=tk.NSEW)

    def _main(self) :
        face_mesh = mp.solutions.face_mesh.FaceMesh(refine_landmarks=True)  #
виявлення 3-ох координат в просторі 468 точок
        cam = cv2.VideoCapture(0)

        while True:
            _, frame = cam.read()  # ігнорування першого аргументу
            frame = cv2.flip(frame, 1)  # вертикальний поворот
            rgbFrame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            face_detector = face_mesh.process(rgbFrame)  # розпізнавання обличчя
людини
            landmarks = face_detector.multi_face_landmarks  # виявлення
орієнтирів обличчя

            if landmarks:
                single_face = face_detector.multi_face_landmarks[0].landmark

                cX = single_face[476].x + (single_face[474].x -
single_face[476].x) / 2
                cY = single_face[475].y + (single_face[477].y -
single_face[475].y) / 2
                self.update()

                '''Calculate eye position'''
                eye_pos = (cX, cY)
```

```
                eye_pos_homo = np.append(eye_pos, 1)
                pos_norm = self.homoMatrix.dot(eye_pos_homo)  # tx, ty, tz - гомогенні координати

                xCentroid, yCentroid = None, None
                if self.algorithm == "Центроїд" :
                    xCentroid, yCentroid = self.ctr.out_filter(int(pos_norm[0] / pos_norm[2]), int(pos_norm[1] / pos_norm[2]))
                if self.algorithm == "МНК" :
                    xCentroid, yCentroid = self.ctr.out_filterMNK(int(pos_norm[0] / pos_norm[2]), int(pos_norm[1] / pos_norm[2]))

                if xCentroid < 0: xCentroid = 0
                if xCentroid > self._screen_width: xCentroid = self._screen_width

                if yCentroid < 0: yCentroid = 0
                if yCentroid > self._screen_height: yCentroid = self._screen_height

                pyautogui.moveTo(xCentroid, yCentroid)
                self.eyeLabel.configure(text=f'Eye Pos: ({int(xCentroid)}, {int(yCentroid)})')

                '''Кліпання лівого ока - вибір нової операції'''
                left_eye = [single_face[145], single_face[159]]
                blink = left_eye[0].y - left_eye[1].y  # різниця між краями лівого ока - признак кліпання ока
                blink_threshold = 0.01  # все, що менше порогу - ознака кліпання ока
                if blink < blink_threshold:
                    pyautogui.click()
                    self.config(cursor='hand2')
                    self.update()

                    if xCentroid < self.center[0] and yCentroid < self.center[1]:
                        self._eat_frame.configure(bg=self.active)
                        self._eat_landmark.configure(bg=self.active)
                        self.update()
                        self.after(2000, self._flipper(self._eat_landmark, self._eat_frame, self._logos))

                        self._edu_frame.configure(bg=self.non_active)
                        self._edu_landmark.configure(bg=self.non_active)
                        self._electro_frame.configure(bg=self.non_active)
                        self._electro_landmark.configure(bg=self.non_active)
                        self._concert_frame.configure(bg=self.non_active)
                        self._rest_landmark.configure(bg=self.non_active)
                    elif xCentroid > self.center[0] and yCentroid < self.center[1]:
                        self._edu_frame.configure(bg=self.active)
                        self._edu_landmark.configure(bg=self.active)
                        self.update()
                        self.after(2000, self._flipper(self._edu_landmark, self._edu_frame, self._activity_images[1], name='edu', img_key=self._edu_img[0]))

                        self._eat_frame.configure(bg=self.non_active)
                        self._eat_landmark.configure(bg=self.non_active)
                        self._electro_frame.configure(bg=self.non_active)
                        self._electro_landmark.configure(bg=self.non_active)
                        self._concert_frame.configure(bg=self.non_active)
```

```
                        self._rest_landmark.configure(bg=self.non_active)
                    elif xCentroid < self.center[0] and yCentroid >
self.center[1]:
                        self._electro_frame.configure(bg=self.active)
                        self._electro_landmark.configure(bg=self.active)
                        self.update()
                        self.after(2000, self._flipper(self._electro_landmark,
self._electro_frame, self._activity_images[2], name='house',
img_key=self._house_img[0]))

                        self._edu_frame.configure(bg=self.non_active)
                        self._edu_landmark.configure(bg=self.non_active)
                        self._eat_frame.configure(bg=self.non_active)
                        self._eat_landmark.configure(bg=self.non_active)
                        self._concert_frame.configure(bg=self.non_active)
                        self._rest_landmark.configure(bg=self.non_active)
                    elif xCentroid > self.center[0] and yCentroid >
self.center[1]:
                        self._concert_frame.configure(bg=self.active)
                        self._rest_landmark.configure(bg=self.active)
                        self.update()
                        self.after(2000, self._flipper(self._rest_landmark,
self._concert_frame, self._activity_images[3], name='rest',
img_key=self._rest_img[0]))

                        self._edu_frame.configure(bg=self.non_active)
                        self._edu_landmark.configure(bg=self.non_active)
                        self._eat_frame.configure(bg=self.non_active)
                        self._eat_landmark.configure(bg=self.non_active)
                        self._electro_frame.configure(bg=self.non_active)
                        self._electro_landmark.configure(bg=self.non_active)


                self.config(cursor='arrow')
                self.update()

            if cv2.waitKey(20) == 27: break

        cam.release()
        cv2.destroyAllWindows()
        self._logger.warning('Eye Track Program finishes')

    def __call__(self) : tk.mainloop()
```

queue_setup. py

```
import numpy as np

class FilterCenter :
    """
    Створення кільцевої черги
    для фільтрів системи Eye Tracking
    """

    def __init__(self, k, init_x=960, init_y=540) :
        self.k = k # кільцева черга на основі масиву фіксованої довжини, де k -
його довжина

        '''Масиви aX та aY проініціалізовані серединою екрану:'''
        self.aX = np.full((self.k), init_x) # 1) ширина екрану / 2
        self.aY = np.full((self.k), init_y) # 2) висота екрану / 2
```

```
        self.tail = 0 # хвіст черги - інкрементується внаслідок додавання нового елементу до черги

        self.historyX = []
        self.historyY = []

    def add_point(self, x, y):
        """
        Додавання нової точки до черги
        :param x: Ox точки
        :param y: Oy точки
        :return:
        """

        self.x = x
        self.y = y

        '''Нова точка в черзі'''
        self.aX[self.tail] = self.x
        self.aY[self.tail] = self.y
        self.tail = self.tail + 1 if self.tail < self.k-1 else 0 # хвіст черги вказує на наступний елемент черги

    def out_filter(self, x=None, y=None):
        """
        Пошук центроїду черги
        :param x: Ox точки
        :param y: Oy точки
        :return: центроїд черги
        """

        self.x = x
        self.y = y

        if x is None or y is None :
            return np.average(self.aX), np.average(self.aY)

        '''Нова точка в черзі'''
        self.aX[self.tail] = self.x
        self.aY[self.tail] = self.y
        self.tail = self.tail + 1 if self.tail < self.k-1 else 0

        return np.average(self.aX), np.average(self.aY)

    def __MNK(self, arr, tail) :
        """
        МНК-згладжування
        як один з алгоритмів зменшення
        аномальних рухів курсора
        :param arr: черга (історія координат очей)
        :param tail: поточний хвіст черги
        :return: прогнозоване значення МНК - останній елемент масиву МНК
        """

        '''
        Підготовка масиву черги до МНК-згладжування:
        елементи в порядку їхнього надходження до черги
        '''
        arrList = arr.tolist()
        arrMNK = []
        arrMNK.extend(arrList[tail:])
        arrMNK.extend(arrList[:tail])

        '''МНК-згладжування'''
```

```
        length = len( arrMNK ) # довжина масиву вхідної функції (створеного
масиву)
        funcIn = np.zeros( (length, 1) ) # матриця вхідних даних
        pointsF = np.ones( (length, 3) ) # матриця: ф-ція, аргумент ф-ції,
квадрат ф-ції (поліном 2-го ступеню)

        for i in range( length ) :
            funcIn[i, 0] = float( arrMNK[i] ) # значення ф-ції

            pointsF[i, 1] = float( i ) # аргумент ф-ції
            pointsF[i, 2] = float( i*i ) # квадрат ф-ції

        '''
        Критерій мінімізації суми квадратів різниць лівої і правої частини
рівнянь:
        AT*AX = AT*b => x = (AT*A)^-1*AT*b
        A+*b, A+ - псевдообернена матриця
        '''
        pointsFT = pointsF.T # транспонована матриця (апроксимуючої) ф-ції
        pointsFTP = pointsFT.dot( pointsF ) # AT*A
        pointsFTPInv = np.linalg.pinv( pointsFTP ) # мультиплікативна обернена
матриця
        pointsMNK = pointsFTPInv.dot( pointsFT ) # A+*b, A+ - псевдообернена
матриця

        c = pointsMNK.dot( funcIn ) # коефцієнти для регресійної моделі
        funcMNK = pointsF.dot( c ) # згладжування ф-ції за коефіцієнтами
регресійної моделі

        return funcMNK[-1, 0]

    def out_filterMNK(self, x=None, y=None):
        self.x = x
        self.y = y

        if x is None or y is None:
            return self.__MNK(self.aX, self.tail), self.__MNK(self.aY,
self.tail)

        self.aX[self.tail] = self.x
        self.aY[self.tail] = self.y
        self.tail = self.tail + 1 if self.tail < self.k-1 else 0

        return self.__MNK(self.aX, self.tail), self.__MNK(self.aY, self.tail)

    def outNumPoints(self): return self.aX, self.aY

    def printCenter(self):
        for i in range(self.k): print(f'({self.aX[i]}, {self.aY[i]}), ', end='')
        print()
```

data_stock. py

```
import numpy as np
import math as mt

class Data:
    """
    Створення датасету для визначення
    найкращого алгоритму керування курсором миші
    """
```

```
def __init__(self, history_len):
    self.i = 0

    self.data_len = history_len

    '''Масиви для кожного зайчика'''
    self.rand_coordsX = np.zeros(self.data_len, dtype=np.int32)
    self.rand_coordsY = np.zeros(self.data_len, dtype=np.int32)

    self.centroid_coordsX = np.zeros(self.data_len, dtype=np.int32)
    self.centroid_coordsY = np.zeros(self.data_len, dtype=np.int32)

    self.MNK_coordsX = np.zeros(self.data_len, dtype=np.int32)
    self.MNK_coordsY = np.zeros(self.data_len, dtype=np.int32)

    self.eye_coordsX = np.zeros(self.data_len, dtype=np.int32)
    self.eye_coordsY = np.zeros(self.data_len, dtype=np.int32)

def dist(self, x, y, x1, y1):
    """
    Евклідова відстань
    як фактор порівняння алгоритмів
    """

    return mt.sqrt((x1 - x) ** 2 + (y1 - y) ** 2)

def add_data(self, randCoords, centroidCoords, MNKCoords, eyeCoords):

    if self.i >= self.data_len: return

    self.rand_coordsX[self.i] = randCoords[0]
    self.rand_coordsY[self.i] = randCoords[1]

    self.centroid_coordsX[self.i] = centroidCoords[0]
    self.centroid_coordsY[self.i] = centroidCoords[1]

    self.MNK_coordsX[self.i] = MNKCoords[0]
    self.MNK_coordsY[self.i] = MNKCoords[1]

    self.eye_coordsX[self.i] = eyeCoords[0]
    self.eye_coordsY[self.i] = eyeCoords[1]

    self.i += 1

def getDataX(self, type):
    if type == 0: return self.rand_coordsX
    if type == 1: return self.centroid_coordsX
    if type == 2: return self.MNK_coordsX
    if type == 3: return self.eye_coordsX

def getDataY(self, type):
    if type == 0: return self.rand_coordsY
    if type == 1: return self.centroid_coordsY
    if type == 2: return self.MNK_coordsY
    if type == 3: return self.eye_coordsY

'''
Пошук евклідової відстані
між сонячним зайчиком,
чиї координати генеруються випадково
та зайчиками, що керуються очима
'''
```

```
    def get_dist_centroid(self):
        dst = np.zeros(self.data_len)
        for i in range(self.data_len):
            dst[i] = self.dist(self.rand_coordsX[i],
                               self.rand_coordsY[i],
                               self.centroid_coordsX[i],
                               self.centroid_coordsY[i])
        return dst

    def get_distMNK(self):
        dst = np.zeros(self.data_len)
        for i in range(self.data_len):
            dst[i] = self.dist(self.rand_coordsX[i],
                               self.rand_coordsY[i],
                               self.MNK_coordsX[i],
                               self.MNK_coordsY[i])
        return dst

    def get_dist_eye(self):
        dst = np.zeros(self.data_len)
        for i in range(self.data_len):
            dst[i] = self.dist(self.rand_coordsX[i],
                               self.rand_coordsY[i],
                               self.eye_coordsX[i],
                               self.eye_coordsY[i])
        return dst
```

controls_frame.py

```
class ControlFrame :
    """
    Формування банерів
    для вибору користувацьких налаштувань
    """

    def __init__(self, canvas, x, y, text:str, width=400, height=100,
bg_color='azure1', frame_disabled='red', thickness=5):
        self.canvas = canvas
        self.banner = self.canvas.create_rectangle(x, y, x+width, y+height,
fill=bg_color, outline=frame_disabled, width=thickness)
        self.text = self.canvas.create_text(x+width/2, y+height/2, text=text,
fill='black', font=('Helvetica 25 bold'))

    def make_active(self, bg_color:str=None, frame_active='green') :
        self.canvas.itemconfig(self.banner, outline=frame_active)
        if isinstance(bg_color, str) :
            self.canvas.configure(bg=bg_color)

    def make_disabled(self, frame_disabled='red') :
        self.canvas.itemconfig(self.banner, outline=frame_disabled)

    def get_pos(self) : return self.canvas.coords(self.banner)

    def check_pos(self, cX, cY) :
        return cX > self.get_pos()[0] and cY > self.get_pos()[1]\
               and cX < self.get_pos()[2] and cY < self.get_pos()[3]
```

camera_frame.py

```
import tkinter as tk

class CameraFrame( tk.Frame ) :
    """
    Вбудовування вікна OpenCV у вікно Tkinter
    """

    def __init__( self, root ):
        tk.Frame.__init__( self, root, borderwidth=1, relief=tk.FLAT )
        self.labelRainbow = tk.Label( self, bg='#DCDCDC' )
        self.camLabel = tk.Label( self, bg='#BC8F8F' )
        self.labelRainbow.pack( side=tk.LEFT )
        self.camLabel.pack( side=tk.RIGHT )
        self.camLabel.forget() # приховування камери
```

matplotlib_frame.py

```
import tkinter as tk
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import (FigureCanvasTkAgg,
NavigationToolbar2Tk)

class MatplotlibFrame ( tk.Frame ) :
    """
    Вбудовання плоттерів matplotlib у вікно tkinter
    за допомогою полотна canvas
    """

    def __init__( self, parent, text, font) :
        tk.Frame.__init__( self, parent )
        tk.Label(self, text=" " + text + " ", font=font).pack()
        self.f = Figure( figsize = ( 100, 100 ), frameon=False )
        self.f.tight_layout()
        self.canvas = FigureCanvasTkAgg( self.f, self )
        self.canvas.get_tk_widget().pack(expand=True)
        self.canvas.figure.tight_layout()
        self.canvas.draw()
        self.plot_widget = self.canvas.get_tk_widget()
        self.toolbar = NavigationToolbar2Tk( self.canvas, self ) # підключення
панелі інструментів matplotlib
        self.plot_widget.pack( side = tk.RIGHT, fill = tk.BOTH, expand = True )
        self.toolbar.update() # можливість оновлення панелі інструментів
matplotlib
```

sunny_bunny.py

```
class SunnyBunny:
    """
    Формування головного сонячного зайчика,
    чиї координати генеруються випадково
    """

    def __init__(self, canvas, size, color):
        self.canvas = canvas
        self.ball = canvas.create_oval(0, 0, size, size, fill=color)
        self.pos(100, 100)

    def pos(self, cX, cY): self.canvas.move(self.ball, cX, cY)

    def get_coords(self): return self.canvas.coords(self.ball)
```

bunny_catch.py

```
class BunnyCatch:
    """
    Формування сонячних зайчиків,
    що керуються очима
    """

    def __init__(self, canvas, size, color):
        self.canvas = canvas
        self.ball = canvas.create_oval(0, 0, size, size, fill=color)
        self.pos(100, 100)

    def pos(self, cX, cY): self.canvas.move(self.ball, cX, cY)

    def get_coords(self): return self.canvas.coords(self.ball)
```

activity_downloader.py

```
from PIL import Image, ImageTk
from dataclasses import dataclass
from typing import List

@dataclass
class ActivityDownloader :
    """
    Клас-завантажувач логотипів
    доступних операцій та
    відповідних Інтернет-ресурсів
    """

    def __init__(self) :
        self._eatPath = 'Activities\\EatTrans.png'
        self._eating = ImageTk.PhotoImage(Image.open(self._eatPath))

        self._readPath = 'Activities\\Read360.png'
        self._reading = ImageTk.PhotoImage(Image.open(self._readPath))
        self._osvitaPath = 'Activities\\osvita360.jpg'
        self._osvita = ImageTk.PhotoImage(Image.open(self._osvitaPath))
        self._edu_dict = {'edu' : {self._reading: self._osvita}}
```

```
        self._housePath = 'Activities\\Drink360.png'
        self._house = ImageTk.PhotoImage(Image.open(self._housePath))
        self._rozetkaPath = 'Activities\\rozetka200.png'
        self._rozetka = ImageTk.PhotoImage(Image.open(self._rozetkaPath))
        self._comfyPath = 'Activities\\comfy200.png'
        self._comfy = ImageTk.PhotoImage(Image.open(self._comfyPath))
        self._house_dict = {'house' : {self._house :
                                           {'rozetka' : self._rozetka,
                                            'comfy' : self._comfy}
                                       }}

        self._restPath = 'Activities\\RestTrans.png'
        self._rest = ImageTk.PhotoImage(Image.open(self._restPath))
        self._kontramarkaPath = 'Activities\\kontramarka360.jpg'
        self._kontramarka =
ImageTk.PhotoImage(Image.open(self._kontramarkaPath))
        self._rest_dict = {'rest' : {self._rest : self._kontramarka}}

        self._activity_images : List = []

    def set_image(self):
        self._activity_images.append(self._eating)
        self._activity_images.append(self._edu_dict)
        self._activity_images.append(self._house_dict)
        self._activity_images.append(self._rest_dict)

    @property
    def get_image(self):
        self.set_image()
        return self._activity_images
```

shop_downloader.py

```
from PIL import Image, ImageTk
from dataclasses import dataclass
from typing import List

@dataclass
class ShopDownloader :
    """
    Клас-завантажувач логотипів
    доступних магазинів
    """

    def __init__(self) :
        self._logo_silpo = 'Shops\\silpo_logo200.png'
        self._shop_silpo = ImageTk.PhotoImage(Image.open(self._logo_silpo))

        self._logo_fora= 'Shops\\fora_logo200.png'
        self._shop_fora = ImageTk.PhotoImage(Image.open(self._logo_fora))

        self._logo_atb = 'Shops\\atb_logo200.png'
        self._shop_atb = ImageTk.PhotoImage(Image.open(self._logo_atb))

        self._logo_metro = 'Shops\\metro_logo200.png'
        self._shop_metro = ImageTk.PhotoImage(Image.open(self._logo_metro))

        self._shop_logo : List = []

    def set_image(self):
        self._shop_logo.append(self._shop_silpo)
        self._shop_logo.append(self._shop_fora)
        self._shop_logo.append(self._shop_atb)
        self._shop_logo.append(self._shop_metro)

    @property
    def get_image(self):
        self.set_image()
        return self._shop_logo
```

logger_settings.py

```
from __future__ import annotations

import logging
import os
from logging.handlers import TimedRotatingFileHandler
from datetime import datetime

class LoggerSetup :
    """
    Створення логеру з метою
    кращого розуміння роботи ПЗ
    """

    def __init__(self) :
        self.__logger = logging.getLogger()
        self.__logger.handlers = []
        self.__message_formatter =
'%(asctime)s|%(name)s|%(levelname)s|%(message)s'
```

```
    def set_logs_format(self, message_format:str) -> LoggerSetup :
        self.__message_formatter = message_format

        return self

    def set_info_logs(self) -> LoggerSetup :
        self.__logger.setLevel(logging.INFO)

        return self

    def set_debug_logs(self) -> LoggerSetup :
        self.__logger.setLevel(logging.DEBUG)

        return self

    def set_warning_logs(self) -> LoggerSetup :
        self.__logger.setLevel(logging.WARNING)

        return self

    def set_console_logs(self) -> LoggerSetup :
        logging_formatter = logging.Formatter(self.__message_formatter)
        logging_handler = logging.StreamHandler()

        logging_handler.setFormatter(logging_formatter)
        self.__logger.addHandler(logging_handler)

        return self

    def set_logs_timetable(self, logger_src:str='external\\logs',
timetable:str='midnight',
                           duration:int=1, logs_copies:int=7) -> LoggerSetup :
        if not os.path.exists(logger_src) : os.makedirs(logger_src)

        logging_formatter = logging.Formatter(self.__message_formatter)
        time_marker = datetime.now().strftime('%d-%m-%Y')
        time_handler =
TimedRotatingFileHandler(f'{logger_src}/eye_tracker_{time_marker}.log',
                                                 when=timetable,
interval=duration, backupCount=logs_copies)

        time_handler.setFormatter(logging_formatter)
        time_handler.suffix = '%d-%m-%Y'
        self.__logger.addHandler(time_handler)

        return self
```