Міністерство освіти і науки України
Одеський національний політехнічний університет

Інститут штучного інтелекту і робототехніки

Кафедра «Комп'ютерні системи»

Ярмола Сергій Валерійович

Студент групи АК-151

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

ДОСЛІДЖЕННЯ МЕТОДІВ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ СЕРВІСІВ ДЛЯ ВИЗНАЧЕННЯ ПОЗИЦІЙ ОЧЕЙ ТА ЇХНЬОГО РУХУ

Напрям підготовки: 123 – Комп`ютерна інженерія

Спеціалізація - Інститут штучного інтелекту і робототехніки

Керівник: Кудря Володимир Григорович,
доктор технічних наук, доцент

Одеса – 2020

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТУ

Ярмолі Сергію Валерійовичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Дослідження методів підвищення ефективності сервісів для визначення позиції очей та їхнього руху

керівник проекту (роботи) Кудря Володимир Григорович, доктор технічних наук, доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом ректора ОНПУ від " 16 " 11 2020 року № 467-в

2. Строк подання студентом проекту (роботи) 23.12.2020

3. Вихідні дані до проекту (роботи) у розділі "Завдання на проведення досліджень"

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити 1. Відомі рішення щодо знаходження точки погляду людини

2. Завдання на проведення дослідження

3. Розробка програмної моделі визначення позиції очей та їх руху

4. Результати дослідження

5. Перелік графічного матеріалу (зточним зазначенням обов'язкових креслень)

1) Об'єкт, предмет, мета та вирішені завдання у дослідженні знаходження позиції очей та їх руху

2) Програмна модель здійснення визначення напряму погляду.

6. Консультанти розділів проекту (роботи)

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

7. Дата видачі завдання_____15.03.2020_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Вивчення науково-технічної та патентної літератури з питань	15.03 - 26.04	Виконав
	виконання робочого айтрекінгу		
2	Формування розширених вимог та постановка задачі на	27.04 - 13.05	Виконав
	проведення дослідження		
3	Розробка алгоритмів та моделі сервісу визначення	14.05 - 30.06	Виконав
	погляду людини		
4	Верифікація моделі сервісу визначення позиції очей та їхнього	01.09 - 24.10	Виконав
	руху		
5	Дослідження моделі сервісу визначення позиції очей та їхнього	25.10 - 01.12	Виконав
	руху		
6	Обробка та оцінка результатів моделювання сервісу визначення	02.12 - 23.12	Виконав
	позиції очей та їхнього руху		

Студент_____Ярмола С.В.
(підпис) (прізвище та ініціали)

Керівник проекту (роботи)_____Кудря В.Г.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Ярмола Сергій Валерійович. Дослідження методів підвищення ефективності сервісів для визначення позицій очей та їхнього руху – Магістерська кваліфікаційна робота. Одеса, 2020: 75 с., 16 рис., 1 додаток, 14 джерел.

Об'єкт дослідження – процес визначення напряму погляду очей для перевірки його ефективності в розробленій моделі.

Предмет дослідження – зменшення часу та спрощення обчислень для визначення напряму погляду очей та їх руху за допомогою комплексної програмної моделі.

Мета роботи – моделювання та дослідження комплексної програмної моделі визначеня напряму погляду для оцінки ефектиності отриманих результатів.

Методи дослідження базуються на теорії ідентифікації та оптимізації стохастичних систем, алгоритму Хааса, теорії проектування комп'ютерних систем, математичного моделювання.

У роботі проведено аналіз практичних методів та реальних засобів для визначення позиції очей та їх руху. Дослідженння методів реалізації технології айтрекінгу на основі мобільної платформи андроїд. Збудована програмна модель для сервісу визначення напряму погляду ока.

НАПРЯМОК ПОГЛЯДУ, ВИЗНАЧЕННЯ ПОЗИЦІЇ ОЧЕЙ, АЙТРЕКІНГ, ВІДСТЕЖЕННЯ РУХУ ОЧЕЙ, ДОСТОВІРНІСТЬ ВИЗНАЧЕННЯ, ЕФЕКТИВНІСТЬ МЕТОДУ ДОСЛІДЖЕННЯ.

ABSTRACT

Yarmola Serhiy Valeriyovych. Research of methods of improving the efficiency of services to determine the positions of the eyes and their movement – Master's qualification work. Odessa, 2020: 76 p., 30 fig., 1 appendix, 12 sources.

The object of the study is the process of determining the direction of eye view to test its effectiveness in the developed model.

The subject of the study is to reduce the time and increase the calculation to determine the direction of the eye and their movement using a comprehensive software model.

The purpose of the work is to model and study a complex software model of a certain direction of review to assess the effectiveness of the results. Research methods are based on theories of identification and optimization of stochastic systems, Haas algorithm, theory of computer systems design, mathematical modeling. The analysis of practical methods and real means for determining the poetry of the eyes and their movement is carried out in the work. Research of methods of realization of technologies of aytraking on the basis of a mobile platform android. A software model has been built for the service of determining the direction of eye examination.

DIRECTION OF VIEW, DETERMINATION OF EYE POSITION, AITREKING, EYE MOVEMENT TRACKING, ACCURACY OF DETERMINATION, EFFECTIVENESS OF THE RESEARCH METHOD.

ЗМІСТ

ВСТУП

Основна частина картини, яку ми можемо побачити, падає на периферійний зір, лише на невеликій ділянці навколо точки, в яку ми дивимось, ми чітко бачимо з прийнятною контрастною структурою. Рух погляду відбувається різко і складається із зупинок (фіксацій) та швидких рухів (саккад) [1]. Процес визначення точки, на яку спрямований погляд чи рух очей щодо голови (iTracking) [2-8], рідко дозволяє знайти відповідь на проблему проблеми, яка сформульована класичним способом. Але це дуже добре допомагає знайти причини виявлених труднощів. Наприклад, розглядаючи області візуалізованого інтерфейсу, які респондент ігнорував, ми розуміємо, чому він неправильно зрозумів концепцію продукту. Або, виявивши на певному етапі заповнення форму вагання респондента, ми розуміємо причину: людина багато разів перечитує текст. Найкраще iTracking допомагає вирішити проблеми, пов'язані з видимістю елементів, точками фокусування, психічним перенапруженням та відволіканням уваги.

Електронний трекер незамінний, якщо ви хочете дізнатись, як користувачі переглядають результати пошуку, аналізують та шукають те, що їм потрібно. Особливо, коли справа стосується роботи зі ЗМІ або аналізу фотографічних зображень. Респонденти не можуть провести самоаналіз і пояснити, як саме вони вибрали правильну картинку, на які образи звернули увагу, а на які ні.

Є також багато обмежень, які в кожному випадку ставлять під сумнів необхідність iTracking. Хороший електронний трекер із програмним забезпеченням (а без нього немає аналізу статистики та візуалізації) коштує десятки тисяч доларів. Аналіз руху очей - тривалий і складний процес, тому використання відстеження очей значно збільшує час обробки результатів дослідження. Проблеми з калібруванням [3-5] також трапляються у 10-20% респондентів. Іноді вони виражають неточності у фіксаціях, і буває, що людину в принципі не можна відкалібрувати.

Щоб електронний трекер працював якомога точніше, потрібно уважно оглянути очі користувача. Ось чому вам потрібно калібрувати. Під час

калібрування трекер аналізує світло, що відбивається від очей користувача. Калібрування виконується шляхом відстеження точки, відео чи іншого графічного елемента, що рухається на моніторі. Далі калібрування поєднуються з унікальною 3D-моделлю людського ока, і вони разом створюють оптимальне зображення для відстеження очей.

Для точного відстеження напрямку зору пристрій повинен ідентифікувати зіниці користувача. Це робиться за допомогою світлого або темного методу визначення зіниць. Світловий метод визначення зіниць працює подібно до компактної камери зі спалахом. В результаті його використання у людини червоніють очі. Збільшення відстані спалаху або опромінювача від об'єкта може уникнути цього ефекту. При цьому використовується темний метод визначення координат зіниць.

Для одних краще працює темний метод визначення зіниць, для інших - світлий. Такі фактори, як розмір зіниці, вік та навколишнє освітлення впливають на те, наскільки один із двох методів працює для людини.

Айтрекер [5] може реально покращити якість досліджень, дати нове розуміння рішення проблеми та чітко підтвердити ваші висновки. Однак він ніколи не працює ізольовано від класичних методів: спостерігає за діяльністю респондента та спілкується з ним.

Як і будь-яка інша технологія, що розвивається, айтрекер може широко використовуватися для усунення бар'єрів для виходу на ринок. Однак зараз лише значні витрати на використання електронного відстеження дозволять отримати максимальну рентабельність інвестицій лише тим, хто вже використав увесь потенціал більш дешевих, низькотехнологічних методів дослідження.

Метою роботи є поширення технології айтрекінгу серед широкого кола користувачів шляхом розробки доступної технології відстеження поведінки зіниці ока. Реалізація технології на платформі андроїд

Для досягнення поставленої мети вирішуються наступні задачі:

– вивчається технологія айтрекінгу, методи отримання експериментальних даних;

– розробляються алгоритми та програмна модель мобільного айтрекінгу;

– перевіряється правильність розробленої моделі системи;

– перевіряється здатність даної моделі коректної обробки результатів дослідження.

Об'єктом дослідження є процес визначення позиції очей та їх руху.

Предметом дослідження є достовірність визначення позиції очей та їхнього руху.

Дослідження та методи вивчення руху очей стають все більш поширеними в багатьох дисциплінах - від психології та маркетингу до освіти та навчання. Це пов’язано з тим, що дослідження з відстеження очей та методології досліджень пропонують нові способи збору даних, формування дослідницьких питань та роздумів про навколишнє середовище. Дослідники також отримують нові уявлення про те, як працює зорова система та як вона взаємодіє з увагою, пізнанням та поведінкою, і як результат, дослідження, засновані на методах дослідження очей, збільшуються у кожній дисципліні.

Нові дослідження з використанням технологій відстеження очей постійно публікуються, і новими програмами цього інноваційного способу проведення досліджень діляться дослідники з усіх континентів та країн.

Незважаючи на те, що існує значний консенсус щодо точності (або її відсутності) відстеження очей, метод її вимірювання або обчислення ще не визначений. В принципі, точність відстеження очей відноситься до різниці між виміряним напрямком погляду та фактичним напрямком погляду людини, розташованого в центрі основної зони. Це може відрізнятися на екрані та може залежати від зовнішніх умов, таких як освітлення, якість калібрування та індивідуальні характеристики очей. Точність можна визначити кількісно з точки зору систематичної помилки, як пояснювалося вище. Вона вимірюється як відстань у градусах між положенням відомої цільової точки та середнім показником набору зібраних вихідних даних вибірки від учасника, який дивиться на точку. Ця помилка може бути усереднена за набором цільових точок, які розподілені по дисплею..

Робота містить чотири розділи, висновок і перелік посилань.

У першому розділі розглядаються основні теоретичні відомості технології відстеження руху очей, проводиться критичний огляд відомих рішень по дослідженню та методології визначення позиції та їх руху .

В другому розділі формулюються розширені технічні вимоги до розробки, визначаються призначення й область використання програми, вихідні дані, а також ставиться задача на відслідкування руху очей.

У третьому розділі аналізується задача дослідження руху очей та їх позиції . Визначаються вимоги для програмної моделі.

У четвертому розділі розробляються алгоритми та програмна модель для проведення дослідження. Наводяться та обробляються результати моделювання.

У висновку роботи формулюються висновки та основні результати, що отримані у проведеному дослідженні.

1 ВІДОМІ РІШЕННЯ ЩОДО ВИЗНАЧЕННЯ ПОЗИЦІЙ ОЧЕЙ ТА ЇХНЬОГО РУХУ

1.1 Основні положення методології відстеження очей

1.1.1 Вимірювальний пристрій, який найчастіше використовується для дослідування рухів очей, зазвичай буває як очний трекер. Загалом існує два типи моніторингу руху очей: ті, що вимірюють положення ока відносно голови, і ті, які вимірюють орієнтацію ока в просторі, або «точки уваги». Останнє вимірювання зазвичай використовується, коли концернується ідентифікація елементів у візуальній сцені, наприклад, у графічних інтерактивних додатках. Можливо, найбільш широко використовуваним пристроєм для дослідження є відео візеркалення рогівки ока. Існує чотири широкі категорії методології вимірювання руху очей, що передбачають використання або вимірювання: Електроокулографія, склеральна контактна лінза, Фотоокулографія або відеоокулографія, а також комбіноване відображення на основі рогівки. Електроокульографія спирається на записи електричних потенційних різів шкіри, що оточують порожнину ока. У середині 1970-х років ця техніка була найбільш широко застосовувана методом руху очей. Сьогодні, можливо, найбільш широко застосовується техніка руху очей, в першу чергу використовується для точки розгляду вимірювань, є метод, заснований на відзеркалені рогівки. Перший метод об'єктивних вимірювань очей з використанням відбиття рогівки був опублікований в 1901 році . Для підвищення точності, методи з використанням контактних лінз були розроблені в 1950-х роках. Пристрої, прикріплені до контактної лінзи, варіювалися від всіх дзеркал до котушок дроту. Вимірювальні прилади, що покладаються на фізичний контакт з очним яблуком, як правило, забезпечують дуже чіткі вимірювання. Очевидним недоліком цих пристроїв є їх інвазивна вимога носіння контактної лінзи. Так звані неінвазивні (іноді їх називають віддаленими) трекерами очей, як правило, покладаються на вимірювання видимих особливостей ока, наприклад, зіниці, межі

райдужної оболочки або відбиття рогівки від тісно розташованого спрямованого джерела світла.

1.1.2 Електроокулографія, найпоширеніший метод запису руху очей близько 40 років тому (застосовується донині), спирається на вимірювання різниці електричних потенціалів шкіри електродів, розміщених навколо ока. Зображення суб'єкта, який носить апарат EOG, показано на рис. 1.1. Зафіксовані потенціали знаходяться в діапазоні 15–200 мкВ, з номінальною чутливістю близько 20 мкВ / град руху очей. Ця методика вимірює рухи очей щодо положення голови, і тому загалом не придатна для вимірювання точок зору, якщо положення голови також не вимірюється (наприклад, використовуючи головний трекер).

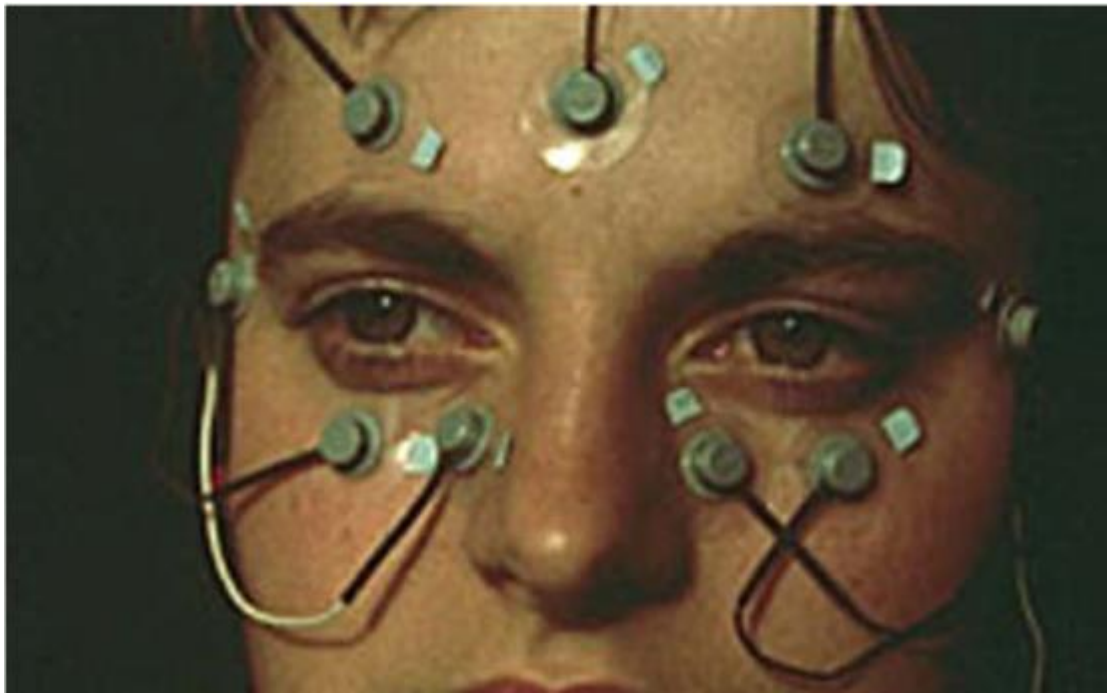

Рисунок 1.1 – Приклад вимірювання руху очей за допомогою електроокулографії

1.1.3 Одним з найбільш точних методів вимірювання руху очей є прикріплення механічного або оптичного еталонного об'єкта, встановленого на контактній лінзі, яка потім безпосередньо знаходиться на оці. Такі ранні записи (приблизно 1898) використовують пластир паризького кільця, прикріплений безпосередньо до рогівки та за допомогою механічних зв’язків з ручками для запису. Ця техніка еволюціонувала до використання сучасної контактної лінзи, до

якої кріпиться кріплення. Контактна лінза обов'язково велика, простягається над рогівкою та склерою (кришталик може зісковзувати, якщо лінза покриває лише рогівку).

На ніжці, прикріпленій до підошви, розміщені різні механічні або оптичні прилади: відбиваючий люмінофор, лінійні схеми та дротяні котушки були найбільш популярнимиреалізаціями в магнітооптичних конфігураціях. В основному методі використовується дротяна котушка, яка потім вимірюється, рухаючись через електромагнітне поле. Спосіб введення контактної лінзи показаний на рис. 1.2. Хоча пошукова котушка склери є найбільш точним методом вимірювання руху очей (з точністю до 5–10 дугових секунд в обмеженому діапазоні близько 5◦), це також найбільш нав'язливий метод. Встановлення лінзи вимагає обережності та практики. Носіння кришталика викликає дискомфорт. Цей метод також вимірює положення очей щодо голови і, як правило, не підходить для вимірювання точки зору.

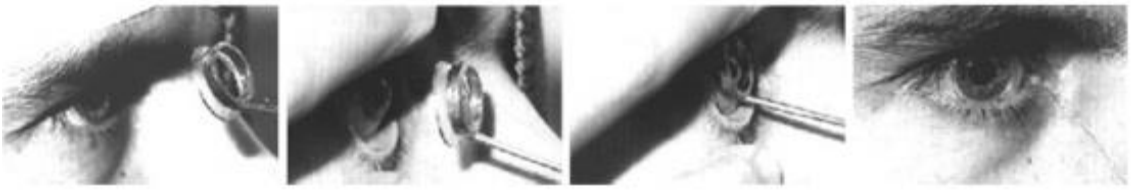

Рисунок 1.2 – Приклад вставки кільця склери для вимірювання руху ока котушки пошуку

1.1.4 Фотоокулографія або відеоокулографія об’єднують широкий спектр технік запису руху очей, що включають вимірювання відмінних особливостей заниження / перекладу очей, наприклад, очевидна форма зіниці, положення лімшини (межа райдужної оболонки склери) та відблиски рогівки близько розташованого джерела направленого світла (часто інфрачервоного). Ці методи, хоча і різні за підходом, тут згруповані, оскільки часто не забезпечують вимірювання точки зору. Приклад записаних зображень ока, що використовуються для фото- чи відеоокулографії, показані на рис. 1.3. Вимірювання окулярних характеристик, передбачених цими методами вимірювання, може проводитися автоматично, а може не передбачати візуальний

огляд зафіксованих рухів очей (як правило, записаних на відеонакопичувачі). Візуальна оцінка, виконана вручну (наприклад, проходження по кадру за кадром), може бути надзвичайно нудною, схильною до помилок і обмеженою тимчасовою частотою дискретизації відеопристрою. Деякі з цих методів вимагають фіксації голови, наприклад, за допомогою упору для голови

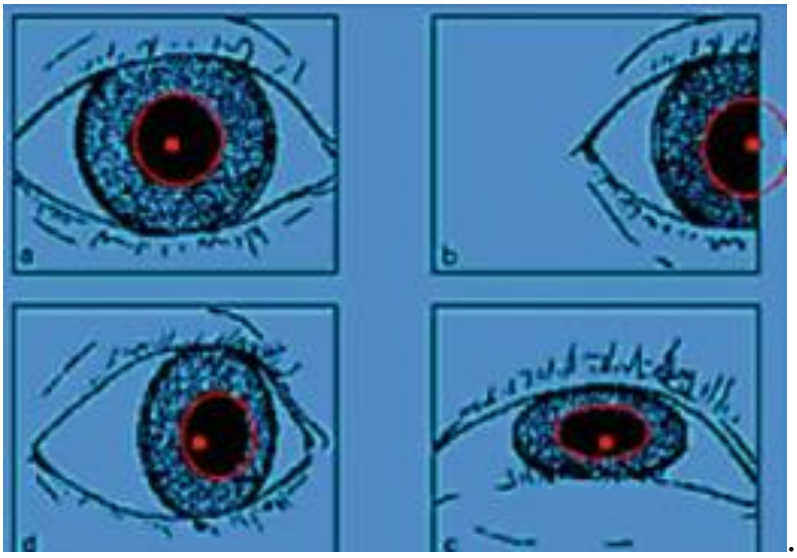

Рисунок 1.3 – Приклад знаходження явного розміру зіниці

1.1.5 Хоча вищезазначені методи в цілому підходять для вимірювання руху очей, вони не часто забезпечують вимірювання точки зору. Щоб забезпечити це вимірювання, або голова повинна бути зафіксована так, щоб положення ока щодо головної та точки зору збігалося, або потрібно виміряти численні окулярні особливості, щоб розрізнити рух голови від обертання очей. Дві такі особливості - це відбиття рогівки (джерела світла, зазвичай інфрачервоного) та центр зіниці. Відео-трекери використовують відносно недорогі камери та апарат для обробки зображень для обчислення точки зору в реальниному часі.

Оптика як настільних, так і не настільних систем по суті ідентична, за винятком розміру. Ці пристрої, які стають все більш доступними, є найбільш придатними для використання в інтерактивних системах. Відбиття рогівки від джерела світла (як правило, інфрачервоне) вимірюється відносно місця розташування центру зіниці. Відбиття рогівки відомі як відбиття Пуркіньє, або зображення Пуркіньє. Відеотранслятор очей, як правило, знаходить перше

зображення Пуркіньє. За допомогою відповідних процедур калібрування, ці відстежувачі очей можуть вимірювати точку зору глядача на відповідно розташованій (перпендикулярно площинній) поверхні, на якій відображаються точки калібрування.

Дві точки відліку на оці потрібні для відокремлення рухів очей від рухів головою. Позиційна різниця між центром зіниці та відбиттям рогівки змінюється при чистому обертанні очей, але залишається відносно постійною при незначних рухах голови. Приблизні відносні положення відбитків зіниці та першого Пуркіньє графічно показані на рис. 1.4, коли ліве око обертається, щоб зафіксувати дев'ять відповідних розміщених точок калібрування. Відбиття Пуркіньє показано у вигляді маленького білого кола в безпосередній близькості від зіниці, представленого чорним колом. Оскільки інфрачервоне джерело світла зазвичай розміщується в певному фіксованому положенні щодо ока, зображення Пуркіньє є відносно стабільним, тоді як очне яблуко, а отже і зіниця, обертається навколо орбіти. Так звані трекери очей п’ятого покоління також вимірюють четверте зображення Пуркіньє. Вимірюючи перше і четверте відбиття Пуркіньє, ці зонди очей з подвійним зображенням Пуркіньє відокремлюють поступальні та обертальні рухи. Обидва відбиття рухаються разом через абсолютно однакову відстань при перекладі очей, але зображення рухаються на різні відстані (таким чином змінюючи їх поділ) при обертанні очей. На жаль, хоча індикатор очей Пуркіньє є досить точним, може знадобитися стабілізація голови.

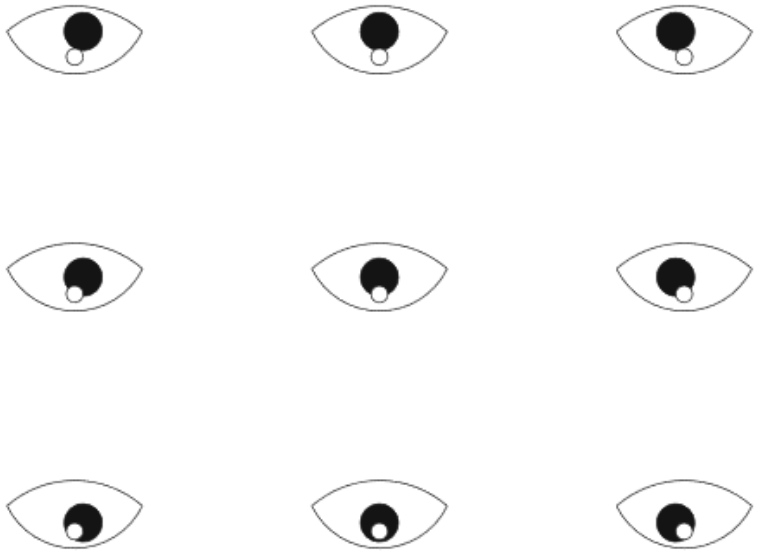

Рисунок 1.4 – Відносне розташування зображень зіниці та першого Пуркіньє, як це бачить камера очей відстежувала

1.2 Обзор необхідного апаратного обладнання

1.2.1 Доступні декілька типів трекерів, починаючи від склеральних котушок і закінчуючи відслідковуванням очей рожевого відблиску на основі відео. Незважаючи на те, що кожна з них має свої переваги та недоліки (наприклад, точність проти частоти дискретизації), для графічних або інтерактивних додатків відстежувач відбиття рогівки на основі відео є, мабуть, найбільш практичним пристроєм. Ці пристрої працюють шляхом зйомки відеозображень ока (освітлених інфрачервоним джерелом світла), обробки відеокадрів (зі швидкістю відеокадрів) та виведення координат x та y ока щодо екрана, що переглядається. Координати x-y зазвичай типово зберігаються самим пристроєм відстеження очей, або можуть бути надіслані на віасеріальний кабель графічного хоста.

Перевага відстежувача очей на основі відео в порівнянні з іншими пристроями полягає в тому, що він є відносно неінвазивним, досить точним (приблизно до 1◦візуального кута в межах 30◦ перегляду) і, здебільшого, неважким для інтеграції з графічною системою. Основним обмеженням трекера на базі відео є частота дискретизації, яка зазвичай обмежується частотою кадрів відео, 60 Гц. Отже, зазвичай можна розраховувати на отримання зразків рухів

очей принаймні кожні 16 мс (зазвичай слід очікувати більшої затримки, оскільки відстежувачу очей потрібен час для обробки кожного відеокадру, а графічному хосту потрібен час для оновлення його відображення).

1.2.2 Інтеграція айтрекера в графічну систему залежить головним чином від правильної доставки графічного відеопотоку до очей-трекера та подальшого прийому розрахованих 2D-координат ока очей відстежувача. У наступному описі налаштування системи описана повна графічна система, що включає два очей-трекери: один, встановлений на столі, монокулярний окуляр для очей, встановлений під стандартним телевізійним дисплеєм, а інший - бінокулярний очей-трекер, встановлений всередині головного дисплея (HMD) . Обидва дисплеї працюють на одному графічному хості.

Блок трекера очей (ПК) розташований між постійним струмом. одиниці відстеження та графічний дисплей із двома головками відстежує праворуч від зображення. Дисплей ПК - це невеликий плоский дисплей, що знаходиться ліворуч від подвійних графічних моніторів. Як дисплеї HMD, так і телевізори (і графічні) керуються графічним хостом, який не потрапляє в поле зору зображення (на підлозі під робочим столом перед видимим кріслом). Чотири маленьких телевізійних монітора на верхній частині ПК для відстеження очей відображає зображення лівої та правої сцени (те, що бачить користувач), та зображення лівого та правого ока (те, що бачить відстежувач очей). Наступний опис системної інтеграції базується на конкретних апаратних пристроях, встановлених у віртуальному Clemson's Лабораторії відстеження очей реальності (VRET), описана тут для довідки та представлена на рис. 1.4. Основним механізмом рендерингу є ПК Dell Dual 1,8 ГГц Pentium 4 (Xeon) (1 ГБ оперативної пам'яті) під управлінням Fedora Core 4 і оснащений графічною картою annVidia GeForce4 FX5950U. Айтрекер від ISCAN, і система включає як бінокулярні камери, встановлені в HMD Virtual Research V8 (висока роздільна здатність), так і монокулярну камеру, встановлену на віддаленому блоці нахилу. Обидва набори оптики функціонують однаково, знімаючи відеозображення ока і відправлення відеосигналу на ПК для відстеження очей для обробки. Блок нахилу у поєднанні з дистанційним блоком камери / світла, встановленого на столі, використовується для неінвазивного відстеження ока в режимі реального часу під час руху голови об'єкта. Це дозволяє

обмежити рух голови, але підборіддя зазвичай використовується для обмеження положення голови обстежуваного під час експериментів для підвищення точності. V8 HMD пропонує вікно роздільної здатності 640 × 480 з окремими каналами для лівого та правого ока. Відстеження положення та орієнтації HMD забезпечується зграєю птахів (6 FOB) Вознесення, 6 градусів свободи, постійного струму. електромагнітна система із затримкою 10 мс на датчик. Ручна мишка з відстеженням 6 DOF забезпечує користувачеві управління напрямком руху. HMD оснащений навушниками для локалізації звуку. Незважаючи на те, що наведені нижче вказівки щодо інтеграції та встановлення ґрунтуються на обладнанні, доступному в лабораторії Clemson VRET, інструкції повинні застосовуватись практично до будь-якого відстежувача очей відбиття рогівки на відео. Першочергове значення для правильної системної інтеграції мають наступні:

1. Знання відеоформату, який вимагає відстежувач очей, як вхідних даних (наприклад, NTSC або VGA)

2. Знання формату даних, який айтрекер створює як вихідний результат. Перший пункт має вирішальне значення для забезпечення належного зображення як користувачеві, так і слідопиту. Для відстеження очей потрібен вхід помірного сигналу, щоб він міг накласти розраховану точку зору для відображення на моніторі сцени, який переглядає оператор (експериментатор). Друга вимога потрібна для правильної інтерпретації точки зору з боку приймаючої системи. Зазвичай це забезпечує виробник очних шляхів. Для читання та інтерпретації цієї інформації хост-системі потрібно забезпечити драйвер пристрою. Вторинні вимоги до належної інтеграції: Здатність очей-трекера забезпечувати дрібнозернистий контроль курсора над своїм стимулюючим калібрувальним стимулом (перехрестя або інший символ) Можливість пристрою відстеження очей передавати свій робочий режим хосту разом з інформацією очної точки зору та точки зору. Ці дві точки пов’язані з правильним вирівнюванням графічного дисплея із зображенням сцени відстеження очей та точкою калібрування. По суті, обидві можливості відстеження очей потрібні для правильного відображення між графікою та координатами зору відстеження очей. Це вирівнювання здійснюється у два етапи. По-перше, оператор може використовувати власний стимулюючий пристрій калібрування для зчитування значень дисплеїв відслідковувача очей.

Роздільна здатність відстежувача очей може бути вказана в діапазоні 512 × 512, однак на практиці може бути важко сформувати графічне вікно, яке точно відповідатиме розмірам відеодисплея. Різниці в порядку одного або двох пікселів зіпсують правильне вирівнювання. Отже, гарною практикою є спочатку відобразити пусте графічне вікно, а потім скористатися курсором відстежувача очей, щоб виміряти протяжність вікна в опорній рамці відстежувача очей. Оскільки це вимірювання повинно бути якомога точнішим, необхідний точний контроль курсору. По-друге, відстежував очей працює в трьох основних режимах: скидання (неактивний), калібрування та запуску. Графічна програма повинна синхронізуватися з цими режимами, щоб у кожному режимі можна було створити належний дисплей:

• Скинути: не відображати нічого (чорний екран) або окрему точку калібрування.

• Калібрування: відображати простий невеликий стимул для калібрування [9] (наприклад, маленьку крапку orcircle) в положенні стимулюючого калібрування очей: відображає необхідний стимул, над яким слід реєструвати рухи очей. Найважливіше, щоб було досягнуто належного вирівнювання між точками стимулюючого калібрування підводки та тими, хто графічного дисплея. Без цього вирівнювання дані, що повертаються засобом відстеження очей, будуть безглуздими. Правильне вирівнювання графіки та опорних кадрів відстеження очей досягається за допомогою простого лінійного відображення між відстеженням очей та координатами графічного вікна.

1.2.3 Двома основними міркуваннями щодо встановлення є підключення відео та послідовних кабелів між графічним хостом та системами відстеження очей. Підключення цих кабелів порівняно просто. Створення драйвера програмного забезпечення для інтерпретації даних також є простим і зазвичай сприяє опису продавцем формату даних та швидкості передачі. На відміну від цього, спочатку найбільшу проблему може викликати підключення відеосигналу. Обов’язково, щоб графічний хост міг генерувати відеосигнал у форматі, який очікується як графічним дисплеєм (наприклад, телевізором або пристроєм HMD), так і засобом відстеження очей.

У найпростішому випадку, якщо хост-комп'ютер здатний генерувати відеосигнал, який підходить як для відображення стимулу, так і для відстежувача очей, все, що тоді потрібно, - це відеорозділювач, який подаватиметься як на дисплей стимулів, так і в око відстежувача. Наприклад, припустимо, що відображення стимулу управляється сигналом NTSC (наприклад, ателевізія) , а головний комп'ютер здатний генерувати сигнал відображення в цьому форматі. (Це було можливо в лабораторії Clemson VRET, тому що хост SGI може надсилати копію всього, що є на графічному дисплеї, за допомогою належного використання їх комбінованої команди.) Якщо трекер очей може також керуватися сигналом NTSC, то установка є простою.

Блок управління відео HMD перенаправляє копію лівого відеоканалу через активний прохідний спліттер через комутатор на лівий монітор графічного дисплея. Комутатор ефективно «краде» сигнал, призначений для графічних дисплеїв, і передає його в HMD. Лівий вимикач на розподільній коробці має два налаштування: монітор або HMD. Правий перемикач на розподільній коробці має три налаштування: монітор, HMD або обидва. Якщо для монітора встановлений правий перемикач, сигнал до HMD не надсилається, забезпечуючи фактично біокулярне відображення в HMD (замість бінокулярного або стереоскопічного дисплея). Якщо для правого перемикача встановлено значення HMD, графічний дисплей згасне, оскільки HMD не забезпечує аналогічного проходження правого відеоканалу.

Якщо для правого перемикача встановлено значення обох, правий відеоканал просто розподіляється між HMD і монітором, що призводить до абінокулярного відображення як у HMD, так і на моніторах. Це останнє налаштування забезпечує безпідсилення сигналу, а отже, як правий РК-дисплей в HMD, так і дисплеї монітору на правій графіці виглядають неяскравими. Це в основному використовується для тестування. Весь відеокабель між графічним хостом, комутатором та HMD - це відео VGA. Однак трекер працює на NTSC. Це є причиною двох перетворювачів VGA / NTSC, які вставляються у шлях відео. Ці перетворювачі виводять сигнал NTSC на пристрій відстеження очей, а також забезпечують активні прохідні сигнали для сигналу VGA, так що, коли працює, сигнал VGA здається не порушеним. Потім відслідковувач очей обробляє

відеосигнали сцени і виводить сигнал на монітори сцени з власним накладеним сигналом, що містить обчислену точку зору (представлену курсором перехрестя).

Ці два невеликі дисплеї показують оператору, що знаходиться в полі зору користувача, а також те, що він / вона дивиться. Зображення очей, як правило, не становлять ускладнень, оскільки цей відеосигнал обробляється ексклюзивно засобом очей. У випадку з зоровим трекером у лабораторії Clemson'sVRET і бінокулярні очні камери HMD, і настільна монокулярна камера - це NTSC, і сигнали надходять безпосередньо в очний трекер. встановленої на голові, Весь відеокабель між графічним хостом, комутатором та HMD - це відео VGA. Однак очей-трекер працює на NTSC. Це є причиною двох перетворювачів VGA / NTSC, які вставляються у шлях відео. Ці перетворювачі виводять сигнал NTSC на пристрій відстеження очей, а також забезпечують активні прохідні сигнали для сигналу VGA, так що, коли працює, сигнал VGA здається не порушеним. Потім відслідковувач очей обробляє відеосигнали сцени і виводить сигнал на монітори сцени з власним накладеним сигналом, що містить обчислену точку зору (представлену курсором перехрестя). Ці два невеликі дисплеї показують оператору, що знаходиться в полі зору користувача, а також те, що він / вона дивиться. Зображення очей, як правило, не становлять ускладнень, оскільки цей відеосигнал обробляється ексклюзивно засобом очей. У випадку з очним трекером в лабораторії Clemson's VRET, бінокулярні очні камери HMD і настільна монокулярна камера є NTSC, і сигнали надходять безпосередньо в очний трекер. Апаратне забезпечення для обробки подвійних зображень лівого та правого ока та сцени.

Його можна перемкнути для роботи в монокулярному режимі [11], для чого потрібні лише зображення лівого ока та сцени. У цьому випадку для перемикання сигналу між зображенням ока, генерованим лівою камерою в HMD, і камерою на настільному блоці використовується простий відеоперемикач. Перша установка в Клемсоні використовувала монітори дисплея від Silicon Graphics, Inc. (SGI), які керувалися або відео VGA, або відеозаписами R, G, B за замовчуванням, що передаються відеокабелями 13W3. Для керування HMD потрібно відео VGA, підключене кабелями HD15. Щоб правильно підключити відеопристрої, потрібні були спеціальні кабелі 13W3-HD15. Хоча це здається дрібницею, потрібні були

спеціально побудовані кабелі. Ці кабелі були недешевими, і на їх побудову та доставку знадобився день-два. Якщо терміни та фінансування є важливими, планування системи до належного прокладання кабелів просто необхідне! (Сьогодні це викликає менше занепокоєння, хоча правильне узгодження кабелів все ще є проблемою, цього разу не стільки щодо відеосигналу, скільки щодо монітора KeyboardVideo або перемикача KVM.)

Проблема, яку особливо важко було вирішити під час дослідження у Клемсоні була результатом формату сигналу VGA, що випромінюється комп'ютером SGIhost. Спочатку, перед тим, як було встановлено пристрій для відстеження очей, HMD тестували для належного відображення. Висновок комутатора використовувався безпосередньо для керування HMD. Все функціонувало належним чином. Однак після вставки HMD у відеосхему, відстежувач очей не буде працювати. Пізніше було виявлено, що проблема полягає у перетворювачах VGA / NTSC: ці перетворювачі очікують більш поширеного сигналу VGA, який використовує синхронізуючий сигнал, синхронізований з горизонтальним полем відео (сигнал h-синхронізації; горизонтальні та вертикальні сигнали синхронізації знаходяться на контактах 13 і 14 кабелю VGA HD15).

Хост-комп'ютер SGI за замовчуванням видає синхронізований зелений сигнал VGA, залишаючи висновки 13 і 14 позбавлені сигналу синхронізації. VR HMD містить схему, яка буде читати або h-sync, або sync-on-green VGA-відео, і тому тихо функціонує. Як виявилося, несправність полягала в неправильній початковій проводці розподільної коробки. У розподільній коробці спочатку бракували з'єднань для висновків 13 і 14, оскільки вони не потрібні для синхронізації зеленого сигналу VGA. Як тільки це було реалізовано, всю кабельну установку довелося розібрати і розподільну коробку переобладнати.

З переробленою комутаційною коробкою необхідно було перевірити користувацькі кабелі 13W3, щоб підтвердити, що ці компоненти передають сигнали через виводи 13 і 14. Нарешті, слід створити нову конфігурацію дисплея (за допомогою команди SGI'scomcombine) для керування всією ланцюгом із горизонтальним сигналом синхронізації замість синхронізація за замовчуванням

на зеленому. Висновок: необхідно впевнитись у форматі відео, необхідному для всіх компонентів, до конкретного сигналу, що передається на окремих кабельних штифтах

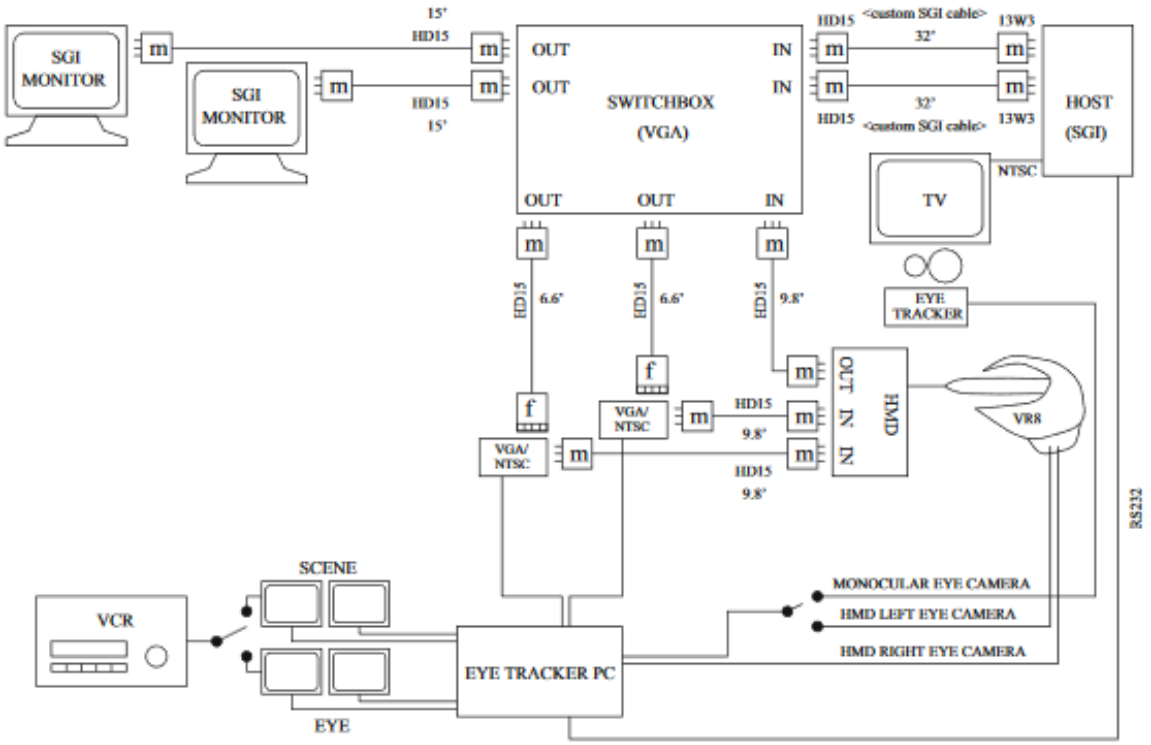

Рисунок 1.5 – Підключення відеосигналу лабораторії VRET в Університеті Клемсона

У цій главі представлені ключові моменти встановлення айтрекера та його інтеграція в переважно комп’ютерну графічну систему. Хоча, можливо, важко узагальнити з цього конкретного досвіду (тематичне дослідження), проте є два пункти, які вважаються ключовими для успішної інсталяції та використання пристрою:

- Маршрутизація сигналу
- Синхронізація.

1.3 Програмне забезпечення

1.3.1 При розробці графічного додатка для відстеження очей найважливішою вимогою є складання координат довічного трекера до відповідного кадру програми застосування. Окотрекер обчислює точку зору глядача (POR) щодо опорної кадри екрана відстежувача очей, наприклад, площину 512 × 512 пікселів, перпендикулярну оптичній осі. Тобто для кожного ока відстежувач очей повертає зразок координат координат x- та yy-координат POR у кожному циклі вибірки (наприклад, раз на кожні 16 мс для пристрою 60 Гц). Цю пару координат необхідно відобразити на межі вікна перегляду програми. Якщо використовується бінокулярний відстежувач очей, дві пари зразків координат повертаються протягом кожного циклу відбору проб, xl, yl для лівого POR та xr, yr для правого ока. Програма віртуальної реальності (VR) повинна в тому самому циклі оновлення також відображати координати положення голови та пристрою відстеження орієнтації (наприклад, зазвичай використовується 6 DOF-трекер). У наступних розділах обговорюється відображення координат екрана очей відстеження до координат програми як для монокулярного, так і для бінокулярного застосування [13]. В монокулярному випадку очікується типовий додаток для перегляду 2D-зображень, і координати відповідно відображаються у 2D (ортогональних) координатах області перегляду (очікується, що координати області огляду відповідають розмірам зображення, що відображається). У випадку бінокуляра (VR) координати очей відстежувача наносяться на розміри найближчої оглядової площини оглядової площі. Для обох обчислень описаний метод вимірювання розмірів області перегляду програми в референтному кадрі відстежувача очей, де для отримання вимірювань використовується власний курсор управління (точна роздільна здатність) відстежувача очей. Цієї інформації має бути достатньо для більшості програм перегляду 2D-зображень. Для додатків VR наступні розділи описують необхідне відображення пристрою відстеження положення голови / орієнтації. Незважаючи на те, що цей розділ повинен узагальнювати найточніші види 6 пристроїв відстеження DOF, обговорення в деяких випадках є специфічним для Вознесенського зграї птахів (FOB), д. електромагнітний 6 DOF трекер. Цей розділ обговорює, як отримати орієнтований

на голову вигляд та вектори з матриці, що повертається трекером, а також пояснює перетворення довільного вектора за допомогою отриманої матриці перетворення. Цей останній висновок використовується для перетворення вектору газу в орієнтовані на голову координати, тобто спочатку отриманий та відносно вимірювання POR бінокулярного очей відстежувача очей. Нарешті, наведено висновок обчислення вектора погляду в 3D, і запропоновано метод для обчислення тривимірної точки погляду в VR.

Під час роботи з пристроєм для відстеження очей дані, отримані з пристрою відстеження, повинні бути зіставлені в діапазоні, що відповідає даному застосуванню. Якщо ви працюєте у VR, дані 2D-візуалізатора, виражені в координатах екрана відстежувача очей, повинні бути зіставлені з 2D-розмірами найближчого оглядового площини. При роботі з зображеннями дані 2D-відслідковувачів очей повинні бути зіставлені з координатами 2D-відображуваного зображення. Загалом, якщо x′∈ [a, b] потрібно відобразити в діапазон [c, d], ми маємо:

$$x=c+(x'-a)(d-c)(b-a)$$

Це лінійне відображення двох (одновимірних) систем координат (або ліній у цьому випадку). Рівняння має пряме тлумачення: Значеннях'перекладається (зміщується) до свого початку, віднімаючиа. Потім значення (x′ − a) нормується діленням на діапазон (b − a). Потім нормоване значення (x′ − a) / (b − a) масштабується до нового діапазону (d − c). Нарешті, нове значення перекладається (зміщується) у належне відносне розташування в новому діапазоні шляхом додавання c.

1.3.2 Тривалість тривимірного перегляду, використовувана в перетвореннях перспективного перегляду, визначається параметрами ліворуч, праворуч, знизу, зверху, поблизу, далеко. Рисунок 1.5 показує розміри екрана відстежувача очей (ліворуч) та розміри оглядової площі (праворуч). Зверніть увагу, що джерело відстежувача очей знаходиться у верхньому лівому куті екрана, а початкове місце перегляду - унизу ліворуч (це загальна розбіжність між зображеннями та графіком). Для перетворення координат очей відстеження (x ′, y ′) у графічні

координати (x, y), використовуючи рівняння, маємо:

$$x = \text{ліво} + \frac{x'(\text{право}-\text{ліво})}{512},$$

$$y = \text{низ} + \frac{(512-y')(\text{верх}-\text{низ})}{512}.$$

Зауважимо, що доданок (512 − y ′) обробляє їх координацію перетворення дзеркала так, щоб лівий верхній кут екрана відстежувача очей був перетворений в нижній лівий кут оглядової площі. , з початком на (0,0), тоді рівняння зменшаться до:

$$x = \frac{x'(640)}{512} = x'(1.3),$$

$$y = \frac{(512-y')(480)}{512} = (512 - y')(0.9375).$$

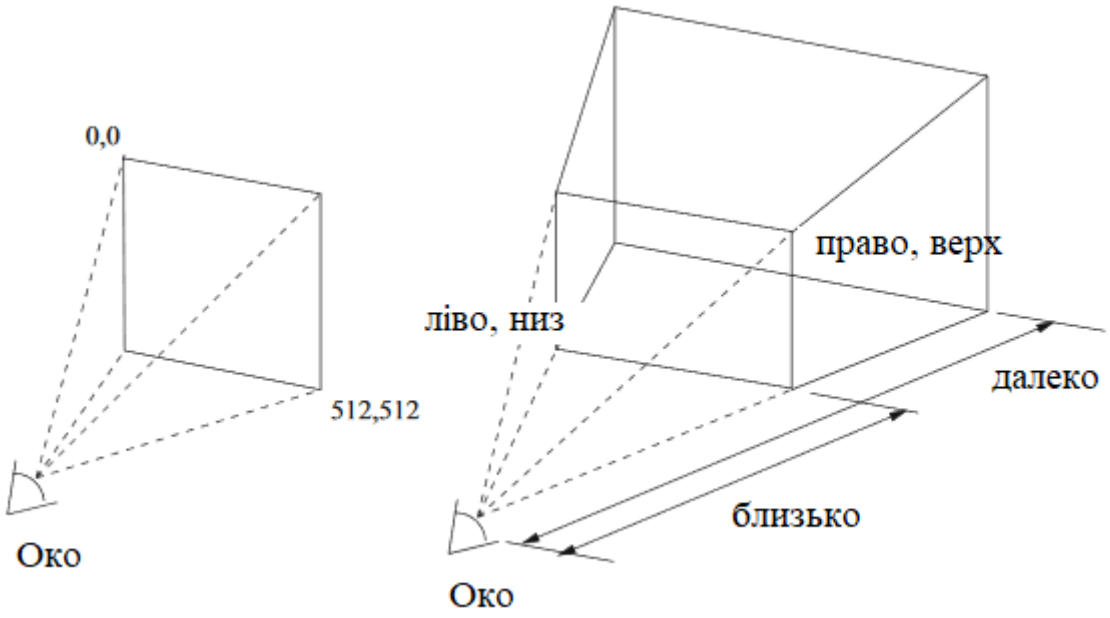

Рисунок 1.6 – Айтрекер для відображення VR

1.4 Висновки

Властивості та різновиди відеоорієнтованих відстежувачів, які видають результати, складаються з:

• часто положення зіниці та відбиття рогівки від очної камери,

• завжди необроблених зразків даних із позначками часу та (x, y) координати,

• іноді значення швидкості і рідко значення прискорення,

• іноді відео, накладене на погляд,

• у багатьох відслідковувачів очей розмір зіниці є безкоштовною додатковою. повинні бути враховані при розробці експерименту.

Роздільна здатність очної камери та частота дискретизації є прикладами важливих апаратних властивостей [12], які впливають на те, які типи рухів очей можуть і не можуть бути виміряні.

Програмне забезпечення, що супроводжує відстежувачі очей, містить алгоритми, які виконують, наприклад, аналіз зображення для пошуку відбитків зіниці та рогівки в очному відео, оцінку погляду та обчислення швидкості руху та прискорення очей за зразками необроблених даних.

Разом з учасниками та середовищем запису такі властивості визначають якість записаних даних і, отже, значною мірою обмежують запитання дослідження, на які можна звернутись, та тип аналізу, який можна провести з даними. Тому важливо мати базове розуміння того, як працює ваш відстежувач, щоб успішно розробити експеримент, записати дані та проаналізувати записані дані.

2 ЗАВДАННЯ НА ПРОВЕДЕННЯ ТА ЗАСОБИ МОДЕЛЮВАННЯ

2.1 Найменування та область застосування

2.1.1 Тема магістерської роботи: “ Дослідження методів підвищення ефективності сервісів для визначення позицій очей та їхнього руху” – присвячена дослідженню процесу виконання відстежування позиції очей та точки погляду.

Областю застосування розробки є UX-дослідження:

– Оцінка замітності елементів;

– Знаходження фокусу уваги користувача;

– Айтрекер дозволяє зрозуміти чи реагує користувач на повідомлення, увідомлення

Об’єкт дослідження полягає у процесі визначення напряму погляду очей для перевірки його ефективності в розробленій моделі.

Предметом досліджень є зменшення часу та спрощення обчислень для визначення напряму погляду очей та їх руху за допомогою комплексної програмної моделі.

2.1.2 Призначення досліджень

Дані дослідження призначені для вивчення особливостей загальної методології визначення руху очей.

До таких особливостей відносяться визначення позиції саккад та фіксацій.

В роботі аналізується, за допомогою загального методу визначення фокусу погляду користувача, порівняння позиції ближнього інфрачервоного світла (відбитого оком) з позицією зіниці. Ця інформація в поєднанні з інформацією про становище голови спостережуваного може бути екстрапольована для визначення точки, на якій сфокурісован погляд користувача, за допомогою чого визначаються відповідні координати на екрані.

2.1.3 Організація досліджень та постановка задачі

Для проведення досліджень розробляється програмна модель айтрекінгу на мобільних пристроях.

Технічне завдання полягає у розробці програмної моделі мобільного айтрекнгу з відображенням координат точки погляду на екрані.

Для виконання завдання необхідно дослідити алгоритми виконання аналізу фотографій, представлений на рисунку 2.2, та знаходження на них зіниці ока людини; також коректного визначення точки фіксації та саккад.

Вихідними даними для розробки є вибірка фотографій респондента(100-200 штук).

Ставиться задача на проведення дослідженя та програмування належного збору даних айтрекера з очей, включаючи відповідне відображення вихідних координат зору; реалізації двух важливих компонентів програмування, а саме механізм взаємодії з користувачем та відображення стимулів.

Зіставлення координат має вирішальне значення як для калібрування очей відстеження, так і для подальшої реєстрації координації руху очей в кінечній програмі.

2.2 Структура комплексної програмної моделі

Структурна схема системи [14], що реалізує запропоновану технологію, представлена на рис. 2.1. Система включає:

1. Керуючий модуль, що складається з блоків: генератора тестових сигналів (ГТС), пристрою управління (УУ).

2. Реєструючий модуль, що складається з блоків: інтелектуальної обробки даних (ИОД), відображення результатів експерименту (Дисплей), бази даних для зберігання результатів експерименту (БД).

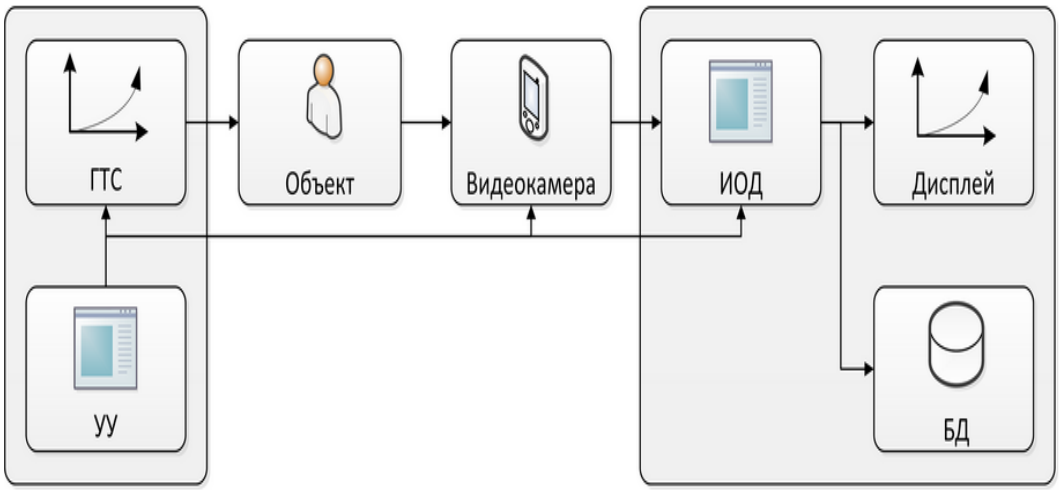

Рисунок 2.1 – Структурна схема системи відстеження руху зіниці

Експеримент, який реалізується за допомогою запропонованої системи відстеження поведінки зіниці на основі відеореєстрації проводиться в такій послідовності:

1. Голова респондента розташовується перед реєструючим пристроєм (відеокамерою) на відомій відстані.

2. У певні моменти часу на дисплеї з'являється графічний тестовий сигнал у вигляді яскравої точки. У той же час вмикається відеокамера для запису руху ока від початкового (стартового) положення до положення (фінального), що визначається світловою плямою (тестовим сигналом).

3. Після проведення серії експериментів з ОРА при різних збуреннях (не менше 2-х тестових сигналів) експеримент завершується. Файл з відео записом переміщення зіниці зберігається в пам'яті системи.

4. Після завершення експерименту «вхід-вихід» запускається додаток, що реалізує інтелектуальний алгоритм виявлення об'єкта (зіниці) в знятому відеоряді. Будується графік залежності зміни положення зіниці від часу (реакції на тестовий вплив).

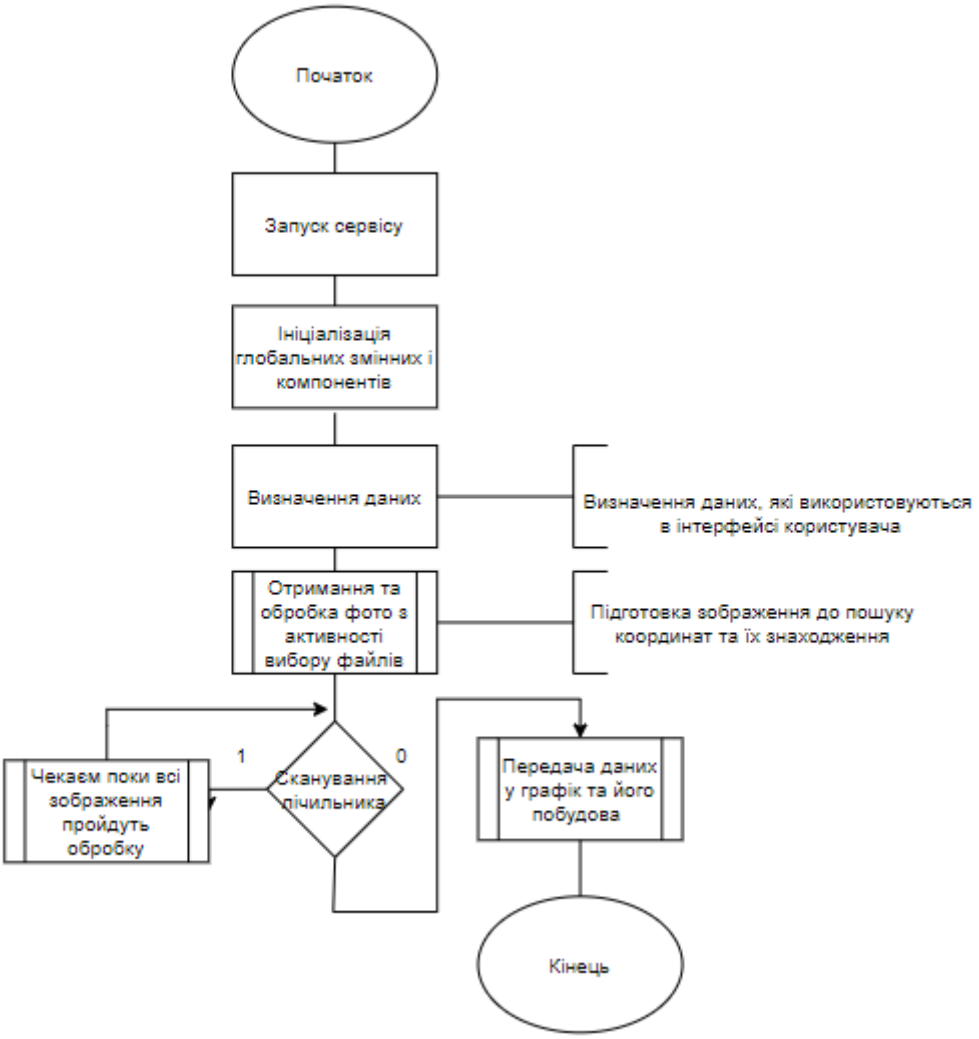

Рисунок 2.2 – Схема алгоритму комплексної моделі.

2.3 Патерни і технології програмування

2.3.1 Модель Представлення Пред'явник (англ. *Model View Presenter*, *MVP*) — шаблон проектування, що відділяє візуальне відображення та поведінку обробки подій у різні класи, а саме: Представлення (*View*) та Пред'явник (*Presenter*). Схему зв’язку між класами зображено на рис. 2.3.

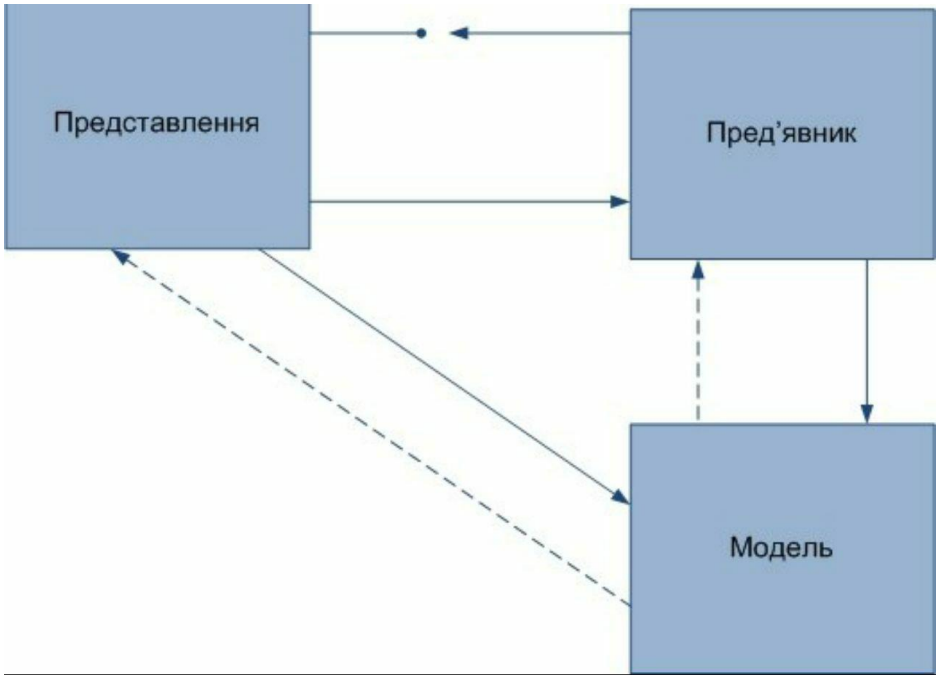

Рисунок 2.3 – Схема зв’язку між класами патерна.

2.3.2 Впровадження залежності (англ. *Dependency injection*, DI) — шаблон проектування програмного забезпечення, що передбачає надання зовнішньої залежності програмному компоненту, використовуючи «інверсію управління» (англ. *Inversion of control*, IoC) для розв'язання (отримання) залежностей.

Впровадження — це передача залежності (сервісу) залежному об'єкту (клієнту). Передавати залежності клієнту замість дозволити клієнту створити сервіс є фундаментальною вимогою до цього шаблону проектування.

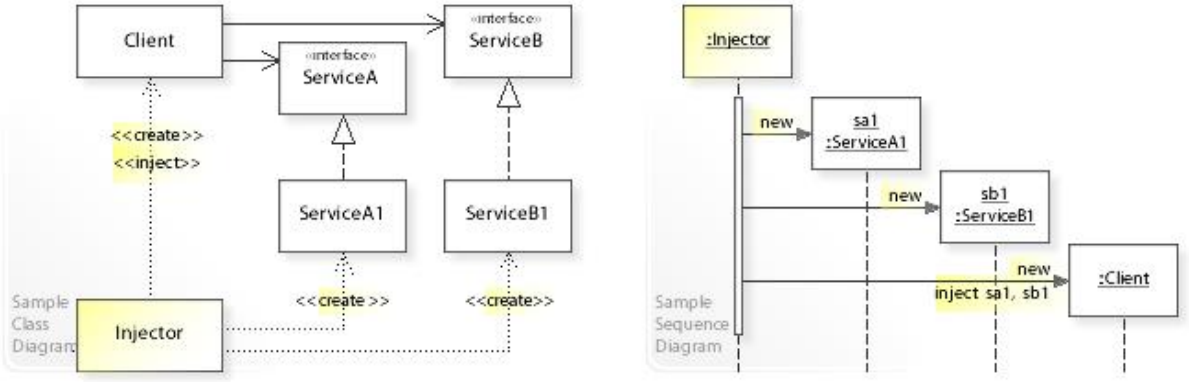

Рисунок 2.4 – Приклад UML класу та діаграми послідовності шаблону проектування Впровадження Залежностей

2.4 Середовище програмування

Для створення та тестування програми було використано середовище AndroidStudio.

Android Studio - це інтегроване середовище розробки (IDE) для роботи з платформою Android, оголошене 16 травня 2013 р. На конференції Google I / O.

IDE був у вільному доступі з версії 0.1, випущеної в травні 2013 року, а потім перейшов на бета-тестування, починаючи з версії 0.8, яка вийшла в червні 2014 року. Перша стабільна версія 1.0 була випущена в грудні 2014 року, одночасно з підтримкою для плагіна Android Development Tools (ADT) для Eclipse припинено.

Android Studio, заснована на програмному забезпеченні IntelliJ IDEA від JetBrains, є офіційним середовищем розробки додатків для Android. Це середовище розробки доступне для Windows, OS X та Linux.

З кожною новою версією Android Studio з’являються нові функції. На даний момент доступні такі функції:

- Розширений редактор макета: WYSIWYG, можливість роботи з інтерфейсом користувача

компоненти, що використовують Drag-and-Drop, функцію попереднього перегляду макета на різних конфігураціях екрана.

- Побудова додатків на основі Gradle.

- Різні типи збірки та генерації декількох файлів .apk

- рефакторинг коду

- Статичний аналізатор коду (Lint), який дозволяє знаходити проблеми продуктивність, несумісність версій тощо.

- Вбудована програма ProGuard та підпис для підписання програм.

- Шаблони основних макетів та компонентів Android.

- Підтримка розробки додатків для Android Wear та Android TV.

- Вбудована підтримка Google Cloud Platform, яка включає

інтеграція із сервісами Google Cloud Messaging та App Engine.

Android («Android») - операційна система для смартфонів, планшетів, електронних книг, цифрових програвачів, годинників та інших пристроїв.

Спочатку розроблена компанією Android Inc., яку згодом придбала компанія Google. Згодом Google ініціював створення альянсу Open Handset Alliance (OHA), який зараз підтримує та розвиває платформу. Android дозволяє створювати програми Java, які керують пристроєм за допомогою бібліотек, розроблених Google. Android Native Development Kit дозволяє передавати (але не налагоджувати) бібліотеки та компоненти програми, написані мовою C та іншими мовами.

86% проданих смартфонів у другому кварталі 2014 року мали операційну систему Android. Водночас у 2014 році було продано понад 1 мільярд пристроїв Android.

Версія 4 обрана через те, що понад 99% пристроїв, випущених з 2012 року, будуть сумісні з додатком.

Власне кажучи, додаток для Android - набір Activity.

Термін Діяльність ще не сформований українською мовою розробниками. Одні вживають слово Activity, інші - Activity.

Досвідчені розробники можуть взяти Activity як форму. Прості програми складаються з одного виду діяльності. Більш складні програми можуть мати кілька вікон, тобто вони складаються з декількох видів діяльності, якими потрібно мати можливість керувати і які можуть взаємодіяти між собою. Діяльність, яка починається першою, вважається основною. З нього можна розпочати іншу діяльність. І не тільки та, що належить до нашого додатку, але й інша програма. Користувачеві здасться, що всі заходи, які вони виконують, є частиною однієї і тієї ж програми, хоча насправді вони можуть бути визначені в різних додатках і виконуватися в різних процесах.

Заняття зазвичай займає весь екран пристрою, але це не є вимогою.

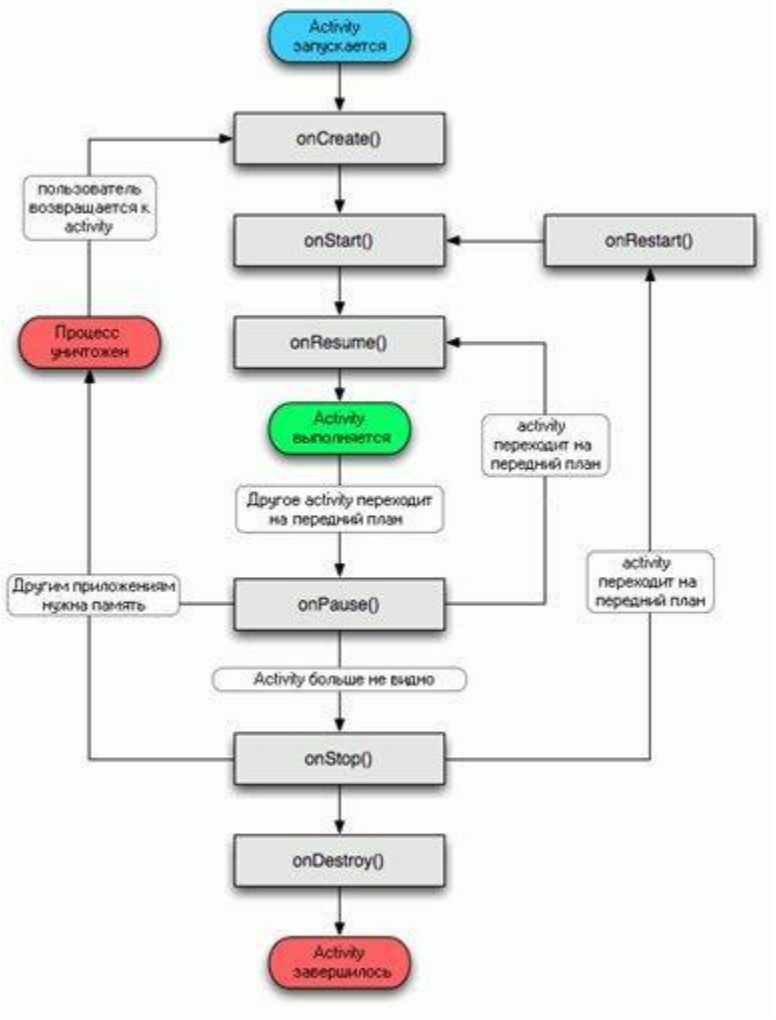

Рисунок 2.5 – Життєвий цикл активності

2.5 Використані Бібліотеки

2.5.1 OpenCV (Open Source Computer Vision Library) - це бібліотека комп'ютерного зору з відкритим кодом, обробки зображень та числових алгоритмів. Реалізовано в C / C ++, також розроблено для Python, Java, Ruby, Matlab, Lua та інших мов. Може вільно використовуватися в академічних та комерційних цілях - поширюється на умовах ліцензії BSD. До версії 1 розробляється в Центрі розробки програмного забезпечення Intel.

OpenCV написаний мовою високого рівня (C / C ++) і містить алгоритми для: інтерпретації зображень, калібрування зразка камери, усунення оптичних спотворень, визначення схожості, аналізу руху об'єкта, визначення форми та відстеження об'єкта, 3D-реконструкції, сегментації об'єкта , розпізнавання жестів тощо

Ця бібліотека дуже популярна завдяки своїй відкритості та можливості користуватися нею безкоштовно як в навчальних, так і в комерційних цілях.

Насправді OpenCV - це набір типів даних, функцій та класів для обробки зображень за допомогою алгоритмів комп'ютерного зору.

Основні модулі бібліотеки:

cxcore - ядро (містить основні структури даних та алгоритми):

- основні операції з багатовимірними числовими масивами

- матрична алгебра, математичні функції, генератори випадкових чисел

- запис / відновлення структур даних у / з XML

- основні функції 2D графіки

CV - модуль для обробки зображень та комп'ютерного зору

- основні операції із зображеннями (фільтрувальні, геометричні

трансформація, трансформація колірних просторів тощо)

- аналіз зображення (вибір відмітних ознак, морфологія, пошук контурів, гістограм)

- аналіз дорожнього руху, спостереження за об'єктами

- виявлення предметів, включаючи обличчя

- калібрування камер, елементів відновлення просторової структури

Highgui - модуль для введення / виведення зображень і відео, створення інтерфейсу користувача

- захоплення відео з камер та відеофайлів, читання / запис нерухомих зображень.

- функції для організації простого інтерфейсу (усі демонстраційні програми використовують HighGUI)

Cvaux - експериментальні та застарілі особливості

- просторове бачення: стерео калібрування, самокалібрування

- пошук стереозвуку, клацання на графіках

- знаходження та опис рис обличчя

CvCam - захоплення відео

- дозволяє знімати відео з цифрових відеокамер

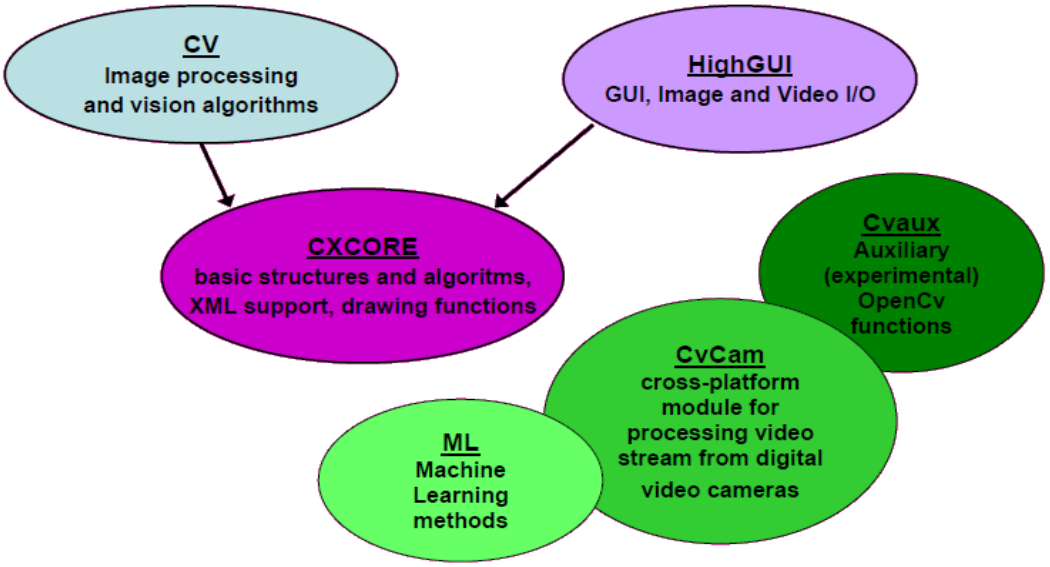

Рисунок 2.6 – Основні модулі бібліотеки

З самого початку OpenCV ставив наступні цілі:

- Дослідження машинного зору, розробка та оптимізація коду.

- Поширення інформації про машинне бачення, розвиток загального

інфраструктура, на якій могли б базуватися розробники, код повинен бути читабельним та переданим.

- Додатки повинні бути портативними, оптимізованими, код яких не є

повинні бути відкритими. Додатки також можна створювати в комерційних цілях.

2. MPAndroidChart - бібліотека для відображення графіків.

3. Realm IO - міжплатформена мобільна база даних для iOS (доступна в Swift & Objective-C) та Android. У нашому випадку ми використовуємо цю бібліотеку для зберігання результатів експериментів.

4. Dagger 2 - бібліотека для реалізації інжекції залежності шаблону.

2.5.2 Ознаки Хаара - ознаки цифрового зображення[7-8], використовувані в розпізнаванні образів. Своєю назвою вони зобов'язані похожістю з вейвлетами Хаара. Ознаки Хаара використовувалися в першому детекторі осіб, що працює в реальному часі.

Історично склалося так, що алгоритми, які працюють тільки з інтенсивністю зображення (наприклад значення RGB в кожному пікселі), мають велику обчислювальну складність. В роботі Папагеоргіу, була розглянута робота з

безліччю ознак, заснованих на вейвлетах Хаара. Віола і Джонс адаптували ідею використання вейвлетів Хаара та розробили те, що було названо ознаками Хаара. Ознака Хаара складається з суміжних прямокутних областей. Вони позиціонуються на зображенні, далі сумуються інтенсивності пікселів в областях, після чого обчислюється різниця між сумами. Ця різниця і буде значенням певної ознаки, визначеного розміру, певним чином позиційований на зображенні.

Сам по собі каскад Хаара - це набір примітивів, для яких розраховується їх згортка з зображенням. Використовуються найпростіші примітиви, що складаються з прямокутників і мають всього два рівні, +1 і -1. При цьому кожен прямокутник використовується кілька разів різного розміру. Під згорткою тут мається на увазі s = X-Y, де Y - сума елементів зображення в темній області, а X - сума елементів зображення у світлій області (можна так само брати X / Y, тоді буде стійкість при зміні масштабу).

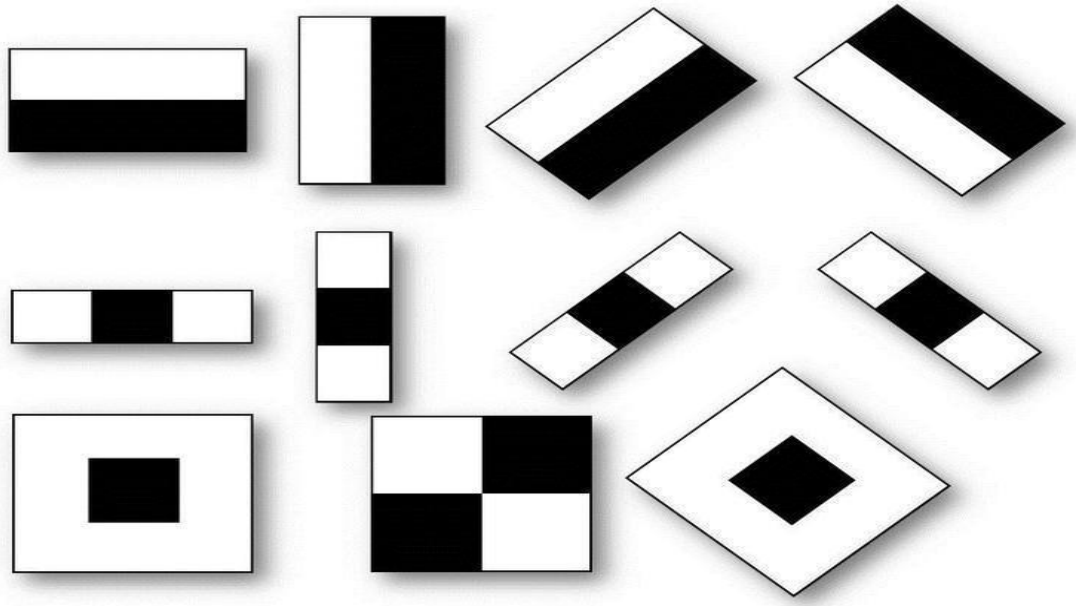

Рисунок 2.7 – Хаар-ознаки

Що дає всього лише 4 звернення до пам’яті і 3 математичних дії для підрахунку суми всіх елементів прямокутника незалежно від його розміру. При розрахунку інших згорток, відмінних від згорток примітивів Хаара, потрібна кількість дій пропорційна квадрату розміру примітиву (якщо не розраховувати

через БПФ, що можливо не для будь-яких патернів). Повернемося до нашого завдання. Нехай потрібно знайти невеликий фрагмент у великому зображенні . Для того, щоб це зробити не потрібно навчання. По одному фрагменту все одно неможливо його провести. Примітиви Хаара допоможуть отримати образ J. Досить отримати згортки J з набором Хара-ознак і порівняти з набором згорток тих же примітивів, розрахованих на I в вікнах пропорційних невеликому фрагменту.

Примітиви є найпростішим смуговим фільтром.

Основна перевага детектора Хаара: швидкість. Завдяки швидкій обробці зображення, можна з легкістю обробляти потокове відео. Детектор Хаара використовується для розпізнавання більшості класів об'єктів. До них відносяться обличчя та інші частини тіла людей, номери автомобілів, пішоходи, дорожні знаки, тварини і.т.д. Детектор Хаара реалізований в бібліотеці OpenCV. Це дає величезну перевагу. Так як готові реалізації OpenCV є під більшість існуючих операційних систем (Android, Windows, Linux, ios).

Класифікатор формується на примітивах Хаара шляхом розрахунку значень ознак. Для навчання на вхід класифікатора спочатку подається набір «правильних» зображень з попередньо виділеної областю на зображенні, далі відбувається перебір примітивів і розрахунок значення ознаки. Попередньо обчислені значення зберігаються в файлі в форматі xml.

В даний час метод Віоли-Джонса є популярним методом для пошуку об'єкта на зображенні в силу своєї високої швидкості та ефективності. В основу метода Віоли-Джонса покладені: інтегральне представлення зображення за ознаками Хаара, побудова класифікатора на основі алгоритму адаптивного бустінга і спосіб комбінування класифікаторів в каскадну структуру. Ці ідеї дозволяють здійснювати пошук об'єкта в режимі реального часу. Розглянемо їх більш детально.

Отриманий класифікатор має мінімальну помилку по відношенню до поточних значень ваг, задіяних в процедурі навчання для визначення помилки.

Для пошуку об'єкта на цифровому зображенні використовується навчений класифікатор, представлений в форматі xml. Класифікатор формується на примітивах Хаара.

2.6 Висновки

Для визначення погляду і позиції очей необхідно побудувати та дослідити його комплексну програмну модель, що відповідає вимогам інформативності, функціональності та достовірності. Основне призначення моделі складається в зменшенні часу та швидкому визначенні на фото позиції зіниці ока людини, відстежити напрям погляду та відобразити його на екрані.

Швидкість дослідження забезпечується коректністю побудови програмної моделі та постановкою експерименту.

Для забезпечення ефективності програмної моделі айтрекнгу потрібно передбачити можливості багатобічної перевірки його функціонування. Подібна перевірка вимагає введення тестування та налагодження програмної моделі перед проведенням експерименту.

Для того, щоб айтрекер працював максимально точно, потрібно ретельно вивчити очі користувача. Ось чому потрібно калібрувати. У процесі калібрування трекер аналізує світло, що відбивається від очей користувача. Дані калібрування потім поєднуються з унікальною 3D-моделлю людського ока, і разом вони створюють оптимальне зображення відстеження очей.

Часто ціль дослідження полягає у вивченні реальних сценаріїв роботи з веб-сайтом за допомогою відстеження рухомих поглядів. Предметом дослідження є компоновка інформації на веб-сайті та її постійність, тобто відповідність інформаційним потребам користувача. Даний аналіз може бути представлений у двох формах: траєкторія руху погляд та теплова карта.

Пояснюється різноманітність систем відстеження очей, починаючи від апаратного забезпечення і закінчуючи програмним забезпеченням тим, що для кожного застосування пристрою існують приватні критерії.

Дослідження розглядає критерій точності втрати в градусах як критерій успіху, де точність 0,5 градуса вважається оптимальною. В інших роботах найважливішими критеріями є стратегії калібрування, незмінність постави голови та методи оцінки погляду.

У більшості робіт із відкритим доступом використовується програмне

забезпечення. Незважаючи на бурхливий ріст інтересу до теми використання нейронних мереж в зображенні обробляючи, у обговорюваних вище роботах, як правило, автори використовують стандартну згорткову мережу для впровадження глибокого навчання або користуються інструментами бібліотеки OpenCV.

3 ПРОЕКТУВАННЯ ТА ЕКСПЕРИМЕНТ КОМПЛЕКСНОЇ ПРОГРАМНОЇ МОДЕЛІ СЕРВІСУ ВІДСЛІДКОВУВАННЯ НАПРЯМКУ ПОГЛЯДУ

3.1Комплексна програмна Модель

3.1.1 Модель представляється у вигляді POJO класу ExperimentItem для визначення даних, які відображаються і над якими проводятися інші дії в інтерфейсі користувача.

```
1.    public class ExperimentItem extends RealmObject {
2.        public static final String ID = "id";
3.        public static final String DATE = "date";
4.
5.        @PrimaryKey
6.        private long id;
7.        private String name = "";
8.        private long date;
9.
10.       private RealmList<PointRealmObject> points = new RealmList<>();
11.
12.       public ExperimentItem() {
13.
14.       }
15.
16.       public ExperimentItem(String name, long date,
17.                RealmList<PointRealmObject> points) {
18.           this.name = name;
19.           this.date = date;
20.           this.points = points;
21.       }
22.
23.       public long getId() {
24.           return id;
25.       }
26.
27.       public void setId(long id) {
28.           this.id = id;
```

```
29.         }
30.
31.         public String getName() {
32.             return name;
33.         }
34.
35.         public void setName(String name) {
36.             this.name = name;
37.         }
38.
39.         public long getDate() {
40.             return date;
41.         }
42.
43.         public void setDate(long date) {
44.             this.date = date;
45.         }
46.
47.         public RealmList<PointRealmObject> getPoints() {
48.             return points;
49.         }
50.
51.
52.         public void setPoints(RealmList<PointRealmObject> points) {
53.             this.points = points;
54.         }
55.     }
```

Клас містить поля id, name, date, points, а також конструктори для їх ініціалізації. Наслідується від класу бібліотеки RealmObject, який реалізує механізм збереження даних.

3.1.2 Початковий екран Main Activity.

Це активність, цілями якої є навігація у додатку. Щоб створити активність, потрібно успадкувати від класу AppCompatActivity.

```
1. class MainActivity : AppCompatActivity(), EyeDetectorContract.View {
```

Ініціалізація глобальних змінних і компонентів інтерфейсу відбувається в методі onCreate, він автоматично генерується для кожної активності і

запускається при активізації вікна. Щоб кнопки відповідали на кліки, реалізуємо їх метод – onClickListener.

```
2.      public override fun onCreate(savedInstanceState: Bundle?) {
3.          super.onCreate(savedInstanceState)
4.          setContentView(R.layout.activity_main)
5.          App.appComponent.inject(this)
6.          presenter.attach(this)
7.
8.          btnStart.setOnClickListener {
9.              startActivityForResult(presenter
10.                     .createFilePickerIntent(this@MainActivity),
11.                             PHOTO_PICK_SUCCESS)
12.         }
13.
14.         btnCancel.setOnClickListener {
15.             presenter.cancel()
16.         }
17.
18.         btnHistory.setOnClickListener {
19.             presenter.openHistory()
20.         }
21.         btnAbout.setOnClickListener { }
22.     }
```

Метод onActivityResult відповідає за взаємодію з іншими активностями, у нашому випадку – отримання фото з активності вибору файлів.

```
23.     override fun onActivityResult(
24.                     requestCode: Int, resultCode: Int, data: Intent?) {
25.         super.onActivityResult(requestCode, resultCode, data)
26.         if (requestCode == PHOTO_PICK_SUCCESS
27.              && resultCode == Activity.RESULT_OK) {
28.             val files =  Utils.getSelectedFilesFromResult(data!!)
29.             presenter.detectEye(files)
30.         }
31.     }
```

Далі описуються методи графічного інтерфейсу.

Для управління відображенням прогресу виористовуються методи showProgress, countProgress і hideProgress. Які показують його під час обчислень и

ховають після завершення.

```
32.     override fun showProgress() {
33.         btnStart.isEnabled = false
34.         tvStart.text = "Обработка..."
35.         progressContainer.visibility = View.VISIBLE
36.         btnHistory.visibility = View.GONE
37.         btnAbout.visibility = View.INVISIBLE
38.         btnCancel.visibility = View.VISIBLE
39.     }
40.
41.     override fun hideProgress() {
42.         btnStart.isEnabled = true
43.         progressContainer.visibility = View.GONE
44.         tvStart.text = getString(R.string.start)
45.         btnCancel.visibility = View.GONE
46.         btnHistory.visibility = View.VISIBLE
47.         btnAbout.visibility = View.VISIBLE
48.     }
49.
50.     override fun countProgress(text: String, progress: Int) {
51.         tvCount.text = text
52.         progressBar.progress = progress
53.     }
```

Завдяки методу showErrorMessage користувач буде своєчасно попередженний про помилку під час виконання программи.

```
54. override fun showErrorMessage(error: String?) {
55.         Toast.makeText(this, error, Toast.LENGTH_SHORT).show()
56.      }
```

Методи навігації showChartFragment і showHistoryFragment викликають екрани графіку та історії експериментів відповідно.

```
57. override fun showChartFragment(id: Long) {
58.         val chartFragment = ExperimentFragment.newInstance(id)
59.
60.         supportFragmentManager.beginTransaction()
61.                 .replace(R.id.content, chartFragment)
62.                 .addToBackStack(chartFragment.tag)
```

```
63.                 .commit()
64.     }
65.
66.     override fun showHistoryFragment() {
67.         val historyFragment = HistoryFragment()
68.
69.         supportFragmentManager.beginTransaction()
70.                 .replace(R.id.content, historyFragment)
71.                 .addToBackStack(historyFragment.tag)
72.                 .commit()
73.     }
```

Допоміжні методи навігації: onSupportNavigateUp, onBackPressed, onStop.

```
74.     override fun onSupportNavigateUp(): Boolean {
75.         onBackPressed()
76.         return true
77.     }
78.
79.     override fun onBackPressed() {
80.         if (supportFragmentManager.backStackEntryCount > 0) {
81.             supportFragmentManager.popBackStack()
82.         } else {
83.             super.onBackPressed()
84.         }
85.     }
86.
87.     override fun onStop() {
88.         super.onStop()
89.         presenter.detach()
90.     }
91. }
```

У класі використовуються такі змінні:

1. presenter – об’єкт бібліотеки для збереження координат.
2. btnStart, btnCancel, btnHistory – кнопки, елемент дизайну.

Екран графіку ChartFragment.

Метод newInstance ініціалізує фрагмент і пердає id – ідентифікатор збереженного експерименту.

```
1.  class ChartFragment : Fragment() {
2.      internal var id: Long = 0
3.
4.      companion object {
5.          fun newInstance(id: Long): ChartFragment {
6.              val args = Bundle()
7.              args.putLong(ID, id)
8.              val fragment = ChartFragment()
9.              fragment.arguments = args
10.             return fragment
11.         }
12.     }
```

Метод onCreateView підгружає дані про розмітку екрана з макету.

```
13. override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
14.                           savedInstanceState: Bundle?): View? {
15.         return inflater.inflate(R.layout.fragment_chart, container, false)
16.     }
17.     override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
18.         super.onViewCreated(view, savedInstanceState)
19.         id = arguments.getLong(ID)
```

Налаштування відображення графіку.

```
20. mChart.setPinchZoom(true)
21.         mChart.description.isEnabled = false
22.         mChart.axisRight.isEnabled = false
23.         mChart.legend.isEnabled = false

24.         val xAxis = mChart.xAxis
25.         xAxis.textColor = Color.WHITE
26.
27.         val leftAxis = mChart.axisLeft
28.         leftAxis.textColor = Color.WHITE
```

Завантаження даних з бд.

```
29. val entries = ArrayList<Entry>()
30.         val realm = Realm.getDefaultInstance()
31.         val experiment = realm.where(ExperimentItem::class.java)
```

```
32.                     .equalTo(ID, id).findFirst()
33.         val pointRealmObjects = experiment.points
```

Передача данних у графік.

```
34.         val dataSet = LineDataSet(entries, "График")
35.         dataSet.setDrawValues(false)
36.
37.         val lineData = LineData(dataSet)
38.         mChart.data = lineData
39.         mChart.invalidate()
40.     }
41. }
```

У класі ми використовуємо такі змінні:

1. mChart – графік;
2. id – ідентифікатор експерименту

```
1. public class HistoryAdapter extends RecyclerView
2.                           .Adapter<HistoryAdapter.ExperimentItemViewHolder> {
3.
4.     private List<ExperimentItem> items = new ArrayList<>();
5.
6.     public HistoryAdapter(List<ExperimentItem> items) {
7.         this.items = items;
8.     }
9.
10.    @Override
11.    public ExperimentItemViewHolder onCreateViewHolder(
12.                    ViewGroup parent, int viewType) {
13.        return new ExperimentItemViewHolder(LayoutInflater
14.             .from(parent.getContext())
15.                .inflate(R.layout.model_experiment_item, parent, false));
16.    }
17.
18.
19.    @Override
20.    public void onBindViewHolder(
21.                    final ExperimentItemViewHolder holder, int position) {
22.        final ExperimentItem item = getItem(position);
23.
```

```
24.         final Context context = holder.itemView.getContext();
25.
26.         holder.tvName.setText(item.getName());
27.         holder.tvDate.setText(MyUtils.parseDate(
28.                         item.getDate(), holder.itemView.getContext()));
29.
30.         holder.itemView.setOnClickListener(new View.OnClickListener() {
31.             @Override
32.             public void onClick(View view) {
33.                 ExperimentFragment chartFragment = ExperimentFragment
34.                         .Companion.newInstance(item.getId());
35.
36.                 ((AppCompatActivity)context).getSupportFragmentManager()
37.                         .beginTransaction()
38.                                 .replace(R.id.content, chartFragment)
39.                                 .addToBackStack(chartFragment.getTag())
40.                                 .commit();
41.             }
42.         });
43.
44.         holder.imbMenu.setOnClickListener(new View.OnClickListener() {
45.             @Override
46.             public void onClick(View view) {
47.                 AlertDialog.Builder builderSingle = new AlertDialog
48.                         .Builder(context);
49.
50.                 final ArrayAdapter<String> arrayAdapter = new ArrayAdapter<>(
51.                         context, R.layout.model_dialog_item);
52.                 arrayAdapter.add("Удалить");
53.
54.                 builderSingle.setAdapter(arrayAdapter, new DialogInterface
55.                     .OnClickListener() {
56.                     @Override
57.                     public void onClick(DialogInterface dialog, int which) {
58.
59.                         removeItem(item);
60.                     }
61.                 });
62.
63.                 AlertDialog dialog = builderSingle.create();
64.                 dialog.requestWindowFeature(Window.FEATURE_NO_TITLE);
65.                 builderSingle.show();
66.             }
67.         });
```

```
68.     }
69.
70.     @Override
71.     public int getItemCount() {
72.         return items.size();
73.     }
74.
75.     public ExperimentItem getItem(int position) {
76.         return items.get(position);
77.     }
78.
79.
80.     public void removeItem(ExperimentItem item) {
81.         int position = items.indexOf(item);
82.
83.         Realm realm = Realm.getDefaultInstance();
84.         final RealmResults<ExperimentItem> results = realm
85.              .where(ExperimentItem.class)
86.              .equalTo("id", item.getId()).findAll();
87.
88.         realm.executeTransaction(new Realm.Transaction() {
89.             @Override
90.             public void execute(Realm realm) {
91.                 results.deleteAllFromRealm();
92.             }
93.         });
94.
95.         items.remove(position);
96.
97.         notifyItemRemoved(position);
98.     }
99.
100.        public static class ExperimentItemViewHolder
101.                        extends RecyclerView.ViewHolder {
102.            TextView tvName;
103.            TextView tvDate;
104.
105.            ImageButton imbMenu;
106.
107.            public ExperimentItemViewHolder(View itemView) {
108.                super(itemView);
109.                tvName = (TextView) itemView.findViewById(R.id.tv_name);
110.                tvDate = (TextView) itemView.findViewById(R.id.tv_date);
111.                imbMenu = (ImageButton) itemView.findViewById(R.id.imb_menu);
```

```
112.             }
113.         }
114.
115.     }
```

3.1.3 Пред'явник

Пред'явник EyeDetectorPresenter містить логіку реагування на події, оновлює Модель (бізнес-логіки і даних з програми) і, в свою чергу, маніпулює станом представлення. Для полегшення тестування пред'явника, він має посилання на інтерфейс представлення view замість посилання на конкретну реалізацію. Як наслідок, можна легко замінити діюче представлення на макет для виконання тестів.

```
1. class EyeDetectorPresenter : EyeDetectorContract.Presenter {
2.
3.     private val subscriptions = CompositeDisposable()
4.     private lateinit var view: EyeDetectorContract.View
5.
6.     @Inject
7.     lateinit var eyeDetector: EyeDetector
8.
9.     @Inject
10.    lateinit var mRealm: Realm

11.    init {
12.        App.appComponent.inject(this)
13.    }
14.
15.    override fun subscribe(subscription: Disposable) {
16.        subscriptions.add(subscription)
17.    }
18.
19.    override fun unsubscribe() {
20.        view.hideProgress()
21.        subscriptions.clear()
22.    }
23.
24.    override fun attach(view: EyeDetectorContract.View) {
25.        this.view = view
26.    }
27.
```

```
28.     override fun detach() {
29.         unsubscribe()
30.     }
31.
32.     override fun createFilePickerIntent(context: Context): Intent {
33.         val i = Intent(context, FilePickerActivity::class.java)
34.
35.         i.putExtra(FilePickerActivity.EXTRA_ALLOW_MULTIPLE, true)
36.         i.putExtra(FilePickerActivity.EXTRA_ALLOW_CREATE_DIR, false)
37.         i.putExtra(FilePickerActivity.EXTRA_MODE,
38.                     FilePickerActivity.MODE_FILE_AND_DIR)
39.         i.putExtra(FilePickerActivity.EXTRA_START_PATH,
40.                     Environment.getExternalStorageDirectory().path)
41.
42.         return i
43.     }
44.
45.     override fun openHistory() {
46.         view.showHistoryFragment()
47.     }
48.
49.     override fun detectEye(files: List<Uri>) {
50.         view.showProgress()
51.
52.         val experiment = ExperimentItem()
53.
54.         var index = 0
55.         var size = 0
56.
57.         subscribe(Observable.fromIterable(files)
58.                 .map { Utils.getFileForUri(it) }
59.                 .flatMap { Observable.fromIterable(it.getImagesSorted()) }
60.                 .toList()
61.                 .toObservable()
62.                 .doOnNext {
63.                     size += it.size
64.                 }
65.                 .flatMap { Observable.fromIterable(it) }
66.                 .doOnNext {
67.                     val point = eyeDetector.findEye(it)
68.                     experiment.points.add(PointRealmObject(point))
69.                 }
70.                 .subscribeOn(Schedulers.io())
71.                 .observeOn(AndroidSchedulers.mainThread())
```

```
72.                 .doOnComplete {
73.                     val id = saveExperiment(experiment)
74.                     view.showChartFragment(id)
75.                     Handler().postDelayed({
76.                         view.hideProgress()
77.                     }, 500)
78.                 }
79.                 .subscribe({
80.                     index++
81.                     val text = "$index /" + size
82.                     val progress = 100 * index / size
83.                     view.countProgress(text, progress)
84.                 }, {
85.                     it.printStackTrace()
86.
87.                     view.showErrorMessage(it.message)
88.                 }))
89.     }
90.
91.     override fun cancel() {
92.         unsubscribe()
93.     }
94.
95.     private fun saveExperiment(experiment: ExperimentItem): Long {
96.         val expNumber = mRealm.where(ExperimentItem::class.java)
97.              .findAll().size + 1
98.         experiment.name = "Эксперимент №$expNumber"
99.
100.         val timeStamp = Date().time
101.
102.         experiment.date = timeStamp
103.         experiment.id = timeStamp
104.
105.         mRealm.saveEyeExperiment(experiment)
106.
107.         return timeStamp
108.      }
```

В класі EyeDetector описується логіка знаходження зіниці

```
1. class EyeDetector(private val cascade: CascadeClassifier) {
```

В даній методі відбувається підготовка зображення до пошуку координат. Сам пошук відбувається в методі getIris().

```
2. fun findEye(file: File): Point? {
3.         val rgba = Imgcodecs.imread(file.absolutePath,
4.                                 Imgcodecs.CV_LOAD_IMAGE_COLOR)
5.         val gray = Imgcodecs.imread(file.absolutePath,
6.                                 Imgcodecs.CV_LOAD_IMAGE_GRAYSCALE)
7.
8.         val eyes = MatOfRect()
9.
10.        cascade.detectMultiScale(gray, eyes)
11.
12.        var iris: Point? = null
13.        val irises = ArrayList<Point>()
14.
15.        val eyeAreas = eyes.toArray()
16.        for (i in eyeAreas.indices) {
17.            val eyeArea = eyeAreas[i]
18.
19.       Imgproc.rectangle(rgba, eyeArea.tl(),eyeArea.br(),
20.                           Scalar(255.0, 0.0, 0.0, 255.0), 2)
21.
22.            iris = getIris(gray, eyeArea)
23.
24.            if (iris != null) {
25.                Imgproc.circle(rgba, Point(iris.x, iris.y), 2,
26.                                   Scalar(255.0, 255.0, 255.0, 255.0), 2)
27.                irises.add(iris)
28.                if (irises.size == 2) {
29.                    break
30.                }
31.            }
32.        }
33.
34.        MyUtils.writeImage(file, rgba)
35.
36.        return iris
37.    }
38.
39.    fun getIris(mGray: Mat, area: Rect): Point? {
40.        var mROI = mGray.submat(area)
41.        val eyes = MatOfRect()
42.        val iris = Point()
43.
```

```
44.         cascade.detectMultiScale(mROI, eyes, 1.15, 2,
45.                 Objdetect.CASCADE_FIND_BIGGEST_OBJECT or
46.                 Objdetect.CASCADE_SCALE_IMAGE, Size(30.0, 30.0), Size())
47.
48.         val eyesArray = eyes.toArray()
49.         val i = 0
50.         while (i < eyesArray.size) {
51.             val e = eyesArray[i]
52.             e.x = area.x + e.x
53.             e.y = area.y + e.y
54.             val eye_only_rectangle = Rect(e.tl().x.toInt(),
55.                     (e.tl().y + e.height * 0.4).toInt(), e.width,
56.                     (e.height * 0.6).toInt())
57.             mROI = mGray.submat(eye_only_rectangle)
58.
59.             val mmG = Core.minMaxLoc(mROI)
60.
61.             iris.x = mmG.minLoc.x + eye_only_rectangle.x
62.             iris.y = mmG.minLoc.y + eye_only_rectangle.y
63.
64.             return iris
65.         }
66.         return null
67.     }
68. }
```

3.1.4 Клас Application відповідає за ініціалізацію OpenCV і бази даних. Також тут відбувається перевірка на існування і ініціалізація каскаду. Перевіряється установка пакета OpenCV Manager.

```
1. class Application : Application() {
2.
3.     private val TAG = "EyeTracking"
4.
5.     companion object {
6.         lateinit var appComponent: AppComponent
7.     }
8.
9.     override fun onCreate() {
10.         super.onCreate()
11.         Realm.init(this)
12.         initDagger()
13.         initOpenCV()
```

```
14.     }
15.
16.     private fun initDagger() {
17.         appComponent = DaggerAppComponent.builder()
18.                 .appModule(AppModule(this))
19.                 .eyeModule(EyeModule()).build()
20.     }



21. private fun initOpenCV() {
22.         val loaderCallback = OpenCVLoaderCallback(this)
23.
24.         if (!OpenCVLoader.initDebug()) {
25.             Log.d(TAG, "Internal OpenCV library not found.
26.                         Using OpenCV Manager for initialization.")
27.             OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_3_2_0, this,
28.                         loaderCallback)
29.         } else {
30.             Log.d(TAG, "OpenCV library found inside package. Using it!")
31.             loaderCallback.onManagerConnected(LoaderCallbackInterface.SUCCESS)
32.         }
33.     }
34. }
```

3.1.5 Компоненти:

Один з найважливіших методів програми - ініціалізація Хаар каскаду і змінних, які пов'язані з детектуванням образу, в даному випадку очі і зіниці. За це відповідає метод OpenCvLoader.initAsync().

```
1. @Component(modules = arrayOf(AppModule::class, EyeModule::class))
2. @Singleton
3. interface AppComponent {
4.
5.     fun inject(activity: MainActivity)
6.
7.     fun inject(loader: OpenCVLoaderCallback)
8.
9.     fun inject(presenter: EyeDetectorPresenter)
10. }
1. @Module
```

```
2. class AppModule(private val app: App) {
3.
4.     @Provides
5.     @Singleton
6.     fun provideContext(): Context = app
7.
8.     @Provides
9.     fun provideRealm(): Realm = Realm.getDefaultInstance()
10. }
```

```
1. @Module
2. class EyeModule {
3.     @Provides
4.     @Singleton
5.     fun provideCascade(): CascadeClassifier = CascadeClassifier()
6.
7.     @Provides
8.     @Singleton
9.     fun provideEyeDetector(cascade: CascadeClassifier):
10.                                 EyeDetector = EyeDetector(cascade)
11.
12.    @Provides
13.    @Singleton
14.    fun provideEyePresenter(): EyeDetectorPresenter = EyeDetectorPresenter()
15. }
```

3.2 Експеримент

3.2.1 Комплексна програмна модель сервісу для відслідкування руху погляду

Призначення. Комплексний програмний продукт (ПП) призначений для обробки фотографій за допомогою якої на екрані відображуватиметься графік залежності переміщення ока з плином часу . Обробка фотографій вібувається за допомоги Хаар каскад та детектування образів(очі, зіниці).

Склад. Комплексний програмний продукт написаний на платформі андроїд, відповідає патернам MVP та DI.

Вихідні дані. Вибріка фотографій респондента.

Інші дані визначаються в процесі роботи ПП. Результати моделювання

зберігаються на екрані.

3.2.2 Установка й запуск. Комплексний програмний продукт поставляється у вигляді завантажувального додатку на телефон й запускається при натисканні на кнопку «Начать експеримент».

3.2.3 Користувальницький інтерфейс. Після запуску додатку появляєься початковий екран з описом сервісу, що включає (рис. 3.1):

– розділ з додатковою інформацією про додаток;
– розділ з історією розробки айтрекінгу;
– структуру матричного поділювача заданої розрядності;
– кнопка початку експерименту.

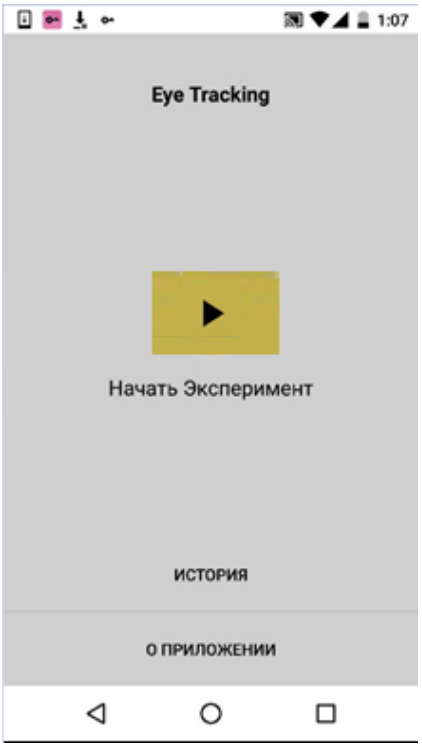

Рисунок 3.1 – Початковий екран додатку

Після додавання вибірки фотографій на телефон, при натисканні на кнопку «Начать Експеримент». Початковий екран міняється на список папок на телефоні, представлений на рис. 4.2, з них необхідно вибрати необхідну папку з загруженими фото.

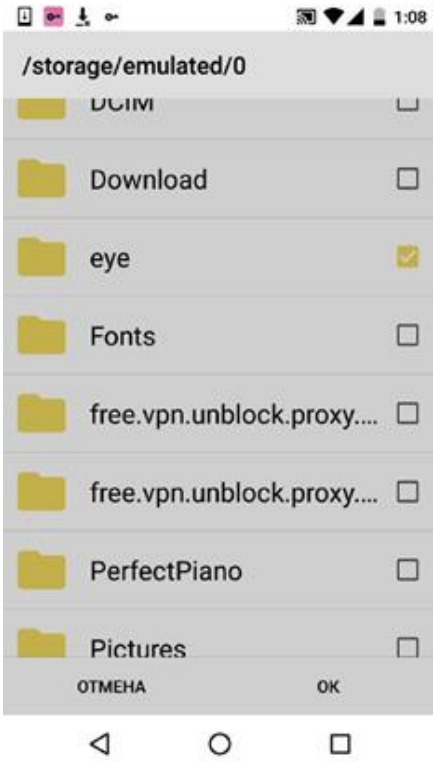

Рисунок 3.2 – Список вибору папки з набором загружених фото респондента

Після вибору папки, необхідно нажати кнопку «ок» та початку обробку фотографій.

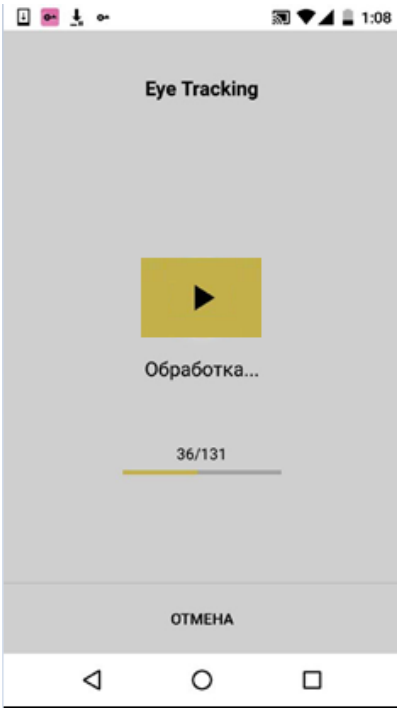

Рисунок 3.3 – Екран обробки фотографій

Моделювання процесу відстеження починається з визначення данних, над якими будуть проводитися дії в інтерфейсі користувача.

Відбувається ініціалізація глобальних змінних і компонентів інтерфейсу, яка генерується для кожного екрану при активації вікна.

Результат експерименту виглядає наступним чином:

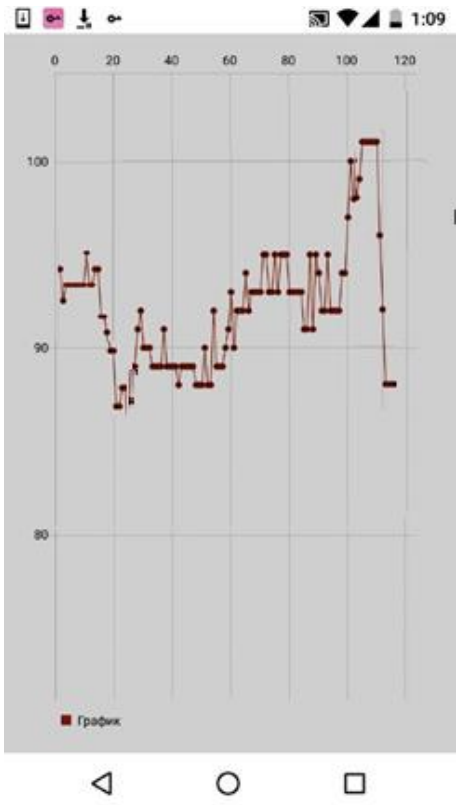

Рисунок 3.4 – Графік залежності переміщення ока від часу

3.3 Висновки

Проведене дослідження показало, що сервіс для відслідкування погляду на мобльному пристрою дійсно може коректно працювати та поліпшити якість досліджень, дати нове розуміння вирішення проблеми і наочно підтвердити відомі знахідки. Однак він ніколи не працює у відриві від класичних методів - спостереження за діяльністю респондента і спілкування з ним.

Відстеження очей є цінним інструментом для дослідження зручності використання мобільних пристроїв, але все ще існує багато проблем щодо того, як створити хорошу оцінку зручності використання, наприклад, достатньо точні дані про рух очей під малим кутом огляду на реальному мобільному пристрої.

Основною ідеєю калібрування було прийняти положення центру ока в пікселях і створити масштабований прямокутник екрана відповідно до погляду користувача. Проблеми в цьому було те, що екран малий (5,0 дюйма), що означає, що відстань до пікселів центру, що дивиться на дві сторони пристрою, дуже близько. Крім того, алгоритм Барта і Тіма [11] виявляють око з високою точністю, можна помітити, що цей алгоритм в кожному пікселі виявляє центр зіниці в різних пікселях і це тому, що алгоритм може виявити його придатним як центр одного з інших можливих центрів, пояснених в алгоритмі.

Результати показують, що чим більший екран, тим простіше розрізнити фігури але все ж із зазначених вище причин результати не завжди є достатніми. Що стосується коду, для покращення продуктивності потрібно виконати деякі оптимізації.

Також іншим методом буде побудова тривимірної моделі обличчя шляхом захоплення фотографії останнього з багатьох боків. Тоді в кожному кадрі око могло мати математично обчислити тривимірне зображення очей, яке здається схожим на інфрачервону технологію для відстеження очей. Є багато речей, які слід вдосконалити, особливо для мобільних пристроїв, де розмір екрану є невеликим.

Розглянуте програмне забезпечення може бути використане для раннього діагностування, виявлення та лікування хвороб нервної системи.

ВИСНОВОК

Результатом дослідження стала реалізація комплексної програмної моделі на платформі Android, яка відстежує око користувача. Першим кроком для досягнення цього є виявлення регіону обличчя, де знаходиться око, яке стало можливим за допомогою класифікаторів Хаар-каскадів. Для відстеження центру око два прямокутники були розміщені щодо прямокутника розпізнавання обличчя для створення область кожного ока.

Як технологія, що розвивається, iTracking може широко використовуватися для усунення бар'єрів для виходу на ринок. Однак зараз лише значні витрати на використання електронного відстеження дозволять отримати максимальну рентабельність інвестицій лише тим, хто вже використав увесь потенціал більш дешевих, низькотехнологічних методів дослідження.

Існує два різних підходи до проведення досліджень відстеження очей на пристроях смартфонів. Один - із стаціонарним / стабільним мобільним пристроєм, встановленим на столі, інший варіант - тестувати за допомогою вільного рухомого пристрою, він надає користувачеві можливість утримувати та експлуатувати пристрій природним чином і з більшою впевненістю або, скажімо, знайомством.

Версія вільного пересування також дає вам привід для тестування в більш динамічному, нелабораторному середовищі, де, наприклад будівельник використовує пристрій для просування свого завдання. Уявіть, що ви тестуєте доповнені програми тощо. Єдиним недоліком цього є коливання освітленості та непередбачувані відблиски на екрані та тіні, що закручують вихідний сигнал зіниці.

Двоступеневий процес виявлення особливостей покращує надійність методу знаходження поганої початкової позиції знаходженної точки. Це проблема, коли відбувається рух очей, оскільки око може швидко змінювати положення від кадру до кадру. Перевага мобільного відстеження очей полягає в тому, що його можна використовувати за межами лабораторії, наприклад, на торгових вулицях, в аеропорту, магазинах або навіть машині.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Саккада.//Електронний документ. URL: https://uk.wikipedia.org/wiki/%D0%A1%D0%B0%D0%BA%D0%BA%D0%B0%D0%B4%D0%B0
2. Попков Ю.С. та ін. Ідентифікація і оптимізація стохастичних систем. М .: Енергія, 1976. 440 с.
3. Пупков К.А., Єгупов Н.Д. Методи ідентіфікації СУЧАСНИХ торій математичного моделювання. Статистична динаміка і ідентифікація систем автоматичного управління: підручник для ву- Поклик. Т. 2/2-е вид. М .: МГТУ ім. Н. Е. Баумана, 2004. 638 с.
4. Дойл Ф.Джей, Пирсон Р.К., Огуннаике Б.А. Идентификация и управление с использованием моделей Вольтерра. Опубликовано SpringerTech-nology-IndustrialArts, 2001. 314 p.
5. Джаннакис Г.Б., Серпедин Е. Библиография нелинейной системы идентификации и ее применения обработки сигнала, связи и биомедицинской инженерии //SignalProcessing. 2001. Том 81, No 3. P.
6. Апарцін А.С. Про підвищення точності моделювання нелінійних динамічних систем поліномами Вольтерри // Електронне моделювання. 2001. №6.С.3-12.
7. Павленко С.В. Застосування вейвлет-фільтрації в процедурі ідентифікації нелінійних систем на основі моделей Вольтерра // Східно-європ. журн. передових технологій. 2010. №6/4 (48). С. 65-70.
8. Павленко В.Д. Ідентифікація нелінійних динамічних систем у вигляді ядер Вольтери на основі даних вимірювань імпульсних відгуків // Електрон.

моделювання. 2010. Т. 32, № 3.С. 3-18.

9. Dorr, M., Pomarjanschi, L., and Barth, B. (2009). Gaze beats mouse: A case study on a gaze-controlled Breakout, PsychNology Journal, 7 (2): pp. 197-211.
10. Ghani, U., Chaudhry, S., Sohail, M., Geelani, N. (2013). GazePointer: A real time mouse pointer control implementation based on eye gaze tracking, 16th Multi Topic Conference (INMIC), pp. 154-159.
11. Rayner, K. (1998). Eye movements in reading and information processing: 20 years of research. Psychological Bulletin, 124, 372–422.
12. Rayner, K. (2009). Eye movements and attention in reading, scene perception, and visual search. The Quarterly Journal of Experimental Psychology, 62(8), 1457–1506.
13. Матеріали конференції – Кременчук: КрНУ, 2016. – 366с. URL: http://at.kdu.edu.ua/Files/Nauka/Konf_KrNU_2016-2.pdf
14. Федорова Г.М. Метод та засоби інформаційної технології ідентифікації непараметричних динамічних моделей око-рухового апарату, 2020. 160с. URL: https://opu.ua/sites/default/files/publicFiles/dissphd/dis_fedorova.pdf

Додаток А

ЛІСТИНГ ПРОГРАМНОЇ МОДЕЛІ СЕРВІСУ ВИЗНАЧАНЕННЯ ПОЗИЦІЇ ОЧЕЙ ТА ЇХНЬОГО РУХУ

```
1.  public class ExperimentItem extends RealmObject {
2.      public static final String ID = "id";
3.      public static final String DATE = "date";
4.
5.      @PrimaryKey
6.      private long id;
7.      private String name = "";
8.      private long date;
9.
10.     private RealmList<PointRealmObject> points = new RealmList<>();
11.
12.     public ExperimentItem() {
13.
14.     }
15.
16.     public ExperimentItem(String name, long date,
17.             RealmList<PointRealmObject> points) {
18.         this.name = name;
19.         this.date = date;
20.         this.points = points;
21.     }
22.
23.     public long getId() {
24.         return id;
```

```
25.         }
26.
27.         public void setId(long id) {
28.             this.id = id;
29.         }
30.
31.         public String getName() {
32.             return name;
33.         }
34.
35.         public void setName(String name) {
36.             this.name = name;
37.         }
38.
39.         public long getDate() {
40.             return date;
41.         }
42.
43.         public void setDate(long date) {
44.             this.date = date;
45.         }
46.
47.         public RealmList<PointRealmObject> getPoints() {
48.             return points;
49.         }
50.
51.
52.         public void setPoints(RealmList<PointRealmObject> points) {
53.             this.points = points;
54.         }
55.
56.
57.     }
58.     class MainActivity : AppCompatActivity(), EyeDetectorContract.View {
59.         public override fun onCreate(savedInstanceState: Bundle?) {
60.             super.onCreate(savedInstanceState)
61.             setContentView(R.layout.activity_main)
62.             App.appComponent.inject(this)
63.             presenter.attach(this)
64.
65.             btnStart.setOnClickListener {
```

```
66.               startActivityForResult(presenter
67.                       .createFilePickerIntent(this@MainActivity),
68.                               PHOTO_PICK_SUCCESS)
69.           }
70.
71.           btnCancel.setOnClickListener {
72.               presenter.cancel()
73.           }
74.
75.           btnHistory.setOnClickListener {
76.               presenter.openHistory()
77.           }
78.           btnAbout.setOnClickListener { }
79.       }
80.   override fun onActivityResult(
81.                       requestCode: Int, resultCode: Int, data: Intent?) {
82.           super.onActivityResult(requestCode, resultCode, data)
83.           if (requestCode == PHOTO_PICK_SUCCESS
84.                && resultCode == Activity.RESULT_OK) {
85.               val files =  Utils.getSelectedFilesFromResult(data!!)
86.               presenter.detectEye(files)
87.           }
88.       }
89.   override fun showProgress() {
90.           btnStart.isEnabled = false
91.           tvStart.text = "Обработка..."
92.           progressContainer.visibility = View.VISIBLE
93.           btnHistory.visibility = View.GONE
94.           btnAbout.visibility = View.INVISIBLE
95.           btnCancel.visibility = View.VISIBLE
96.       }
97.
98.       override fun hideProgress() {
99.           btnStart.isEnabled = true
100.          progressContainer.visibility = View.GONE
101.          tvStart.text = getString(R.string.start)
102.          btnCancel.visibility = View.GONE
103.          btnHistory.visibility = View.VISIBLE
104.          btnAbout.visibility = View.VISIBLE
105.      }
106.
```

```
107.      override fun countProgress(text: String, progress: Int) {
108.          tvCount.text = text
109.          progressBar.progress = progress
110.      }
111.      override fun showErrorMessage(error: String?) {
112.          Toast.makeText(this, error, Toast.LENGTH_SHORT).show()
113.      }
114.      override fun showChartFragment(id: Long) {
115.          val chartFragment = ExperimentFragment.newInstance(id)
116.
117.          supportFragmentManager.beginTransaction()
118.                  .replace(R.id.content, chartFragment)
119.                  .addToBackStack(chartFragment.tag)
120.                  .commit()
121.      }
122.
123.      override fun showHistoryFragment() {
124.          val historyFragment = HistoryFragment()
125.
126.          supportFragmentManager.beginTransaction()
127.                  .replace(R.id.content, historyFragment)
128.                  .addToBackStack(historyFragment.tag)
129.                  .commit()
130.      }
131.      override fun onSupportNavigateUp(): Boolean {
132.          onBackPressed()
133.          return true
134.      }
135.
136.      override fun onBackPressed() {
137.          if (supportFragmentManager.backStackEntryCount > 0) {
138.              supportFragmentManager.popBackStack()
139.          } else {
140.              super.onBackPressed()
141.          }
142.      }
143.
144.      override fun onStop() {
145.          super.onStop()
146.          presenter.detach()
147.      }
```

```
148.  }
149.  class ChartFragment : Fragment() {
150.      internal var id: Long = 0
151.
152.      companion object {
153.          fun newInstance(id: Long): ChartFragment {
154.              val args = Bundle()
155.              args.putLong(ID, id)
156.              val fragment = ChartFragment()
157.              fragment.arguments = args
158.              return fragment
159.          }
160.      }
161.  override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
162.                                savedInstanceState: Bundle?): View? {
163.          return inflater.inflate(R.layout.fragment_chart, container, false)
164.      }
165.      override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
166.          super.onViewCreated(view, savedInstanceState)
167.          id = arguments.getLong(ID)


168.          mChart.setPinchZoom(true)
169.          mChart.description.isEnabled = false
170.          mChart.axisRight.isEnabled = false
171.          mChart.legend.isEnabled = false
172.          val xAxis = mChart.xAxis
173.          xAxis.textColor = Color.WHITE
174.
175.          val leftAxis = mChart.axisLeft
176.          leftAxis.textColor = Color.WHITE
177.          val entries = ArrayList<Entry>()
178.          val realm = Realm.getDefaultInstance()
179.          val experiment = realm.where(ExperimentItem::class.java)
180.                      .equalTo(ID, id).findFirst()
181.          val pointRealmObjects = experiment.points
182.  val dataSet = LineDataSet(entries, "График")
183.          dataSet.setDrawValues(false)
184.
185.          val lineData = LineData(dataSet)
186.          mChart.data = lineData
```

```
187.          mChart.invalidate()
188.      }
189.  }
190.  public class HistoryAdapter extends RecyclerView
191.                              .Adapter<HistoryAdapter.ExperimentItemViewHolder> {
192.
193.      private List<ExperimentItem> items = new ArrayList<>();
194.
195.      public HistoryAdapter(List<ExperimentItem> items) {
196.          this.items = items;
197.      }
198.
199.      @Override
200.      public ExperimentItemViewHolder onCreateViewHolder(
201.                      ViewGroup parent, int viewType) {
202.          return new ExperimentItemViewHolder(LayoutInflater
203.                .from(parent.getContext())
204.                   .inflate(R.layout.model_experiment_item, parent, false));
205.      }
206.
207.
208.      @Override
209.      public void onBindViewHolder(
210.                      final ExperimentItemViewHolder holder, int position) {
211.          final ExperimentItem item = getItem(position);
212.
213.          final Context context = holder.itemView.getContext();
214.
215.          holder.tvName.setText(item.getName());
216.          holder.tvDate.setText(MyUtils.parseDate(
217.                              item.getDate(), holder.itemView.getContext()));
218.
219.          holder.itemView.setOnClickListener(new View.OnClickListener() {
220.              @Override
221.              public void onClick(View view) {
222.                  ExperimentFragment chartFragment = ExperimentFragment
223.                              .Companion.newInstance(item.getId());
224.
225.                  ((AppCompatActivity)context).getSupportFragmentManager()
226.                              .beginTransaction()
227.                                    .replace(R.id.content, chartFragment)
```

```
                                .addToBackStack(chartFragment.getTag())
                                .commit();
            }
        });

        holder.imbMenu.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                AlertDialog.Builder builderSingle = new AlertDialog
                        .Builder(context);

                final ArrayAdapter<String> arrayAdapter = new ArrayAdapter<>(
                        context, R.layout.model_dialog_item);
                arrayAdapter.add("Удалить");

                builderSingle.setAdapter(arrayAdapter, new DialogInterface
                    .OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {

                        removeItem(item);
                    }
                });

                AlertDialog dialog = builderSingle.create();
                dialog.requestWindowFeature(Window.FEATURE_NO_TITLE);
                builderSingle.show();
            }
        });
    }

    @Override
    public int getItemCount() {
        return items.size();
    }

    public ExperimentItem getItem(int position) {
        return items.get(position);
    }


```

```
269.      public void removeItem(ExperimentItem item) {
270.          int position = items.indexOf(item);
271.
272.          Realm realm = Realm.getDefaultInstance();
273.          final RealmResults<ExperimentItem> results = realm
274.               .where(ExperimentItem.class)
275.               .equalTo("id", item.getId()).findAll();
276.
277.          realm.executeTransaction(new Realm.Transaction() {
278.              @Override
279.              public void execute(Realm realm) {
280.                  results.deleteAllFromRealm();
281.              }
282.          });
283.
284.          items.remove(position);
285.
286.          notifyItemRemoved(position);
287.      }
288.
289.      public static class ExperimentItemViewHolder
290.                      extends RecyclerView.ViewHolder {
291.          TextView tvName;
292.          TextView tvDate;
293.
294.          ImageButton imbMenu;
295.
296.          public ExperimentItemViewHolder(View itemView) {
297.              super(itemView);
298.              tvName = (TextView) itemView.findViewById(R.id.tv_name);
299.              tvDate = (TextView) itemView.findViewById(R.id.tv_date);
300.              imbMenu = (ImageButton) itemView.findViewById(R.id.imb_menu);
301.          }
302.      }
303.
304.  }
305.  class EyeDetectorPresenter : EyeDetectorContract.Presenter {
306.
307.      private val subscriptions = CompositeDisposable()
308.      private lateinit var view: EyeDetectorContract.View
309.
```

```
310.      @Inject
311.      lateinit var eyeDetector: EyeDetector
312.
313.      @Inject
314.      lateinit var mRealm: Realm

315.      init {
316.          App.appComponent.inject(this)
317.      }
318.
319.      override fun subscribe(subscription: Disposable) {
320.          subscriptions.add(subscription)
321.      }
322.
323.      override fun unsubscribe() {
324.          view.hideProgress()
325.          subscriptions.clear()
326.      }
327.
328.      override fun attach(view: EyeDetectorContract.View) {
329.          this.view = view
330.      }
331.
332.      override fun detach() {
333.          unsubscribe()
334.      }
335.
336.      override fun createFilePickerIntent(context: Context): Intent {
337.          val i = Intent(context, FilePickerActivity::class.java)
338.
339.          i.putExtra(FilePickerActivity.EXTRA_ALLOW_MULTIPLE, true)
340.          i.putExtra(FilePickerActivity.EXTRA_ALLOW_CREATE_DIR, false)
341.          i.putExtra(FilePickerActivity.EXTRA_MODE,
342.                      FilePickerActivity.MODE_FILE_AND_DIR)
343.          i.putExtra(FilePickerActivity.EXTRA_START_PATH,
344.                      Environment.getExternalStorageDirectory().path)
345.
346.          return i
347.      }
348.
349.      override fun openHistory() {
```

```
350.            view.showHistoryFragment()
351.        }
352.
353.        override fun detectEye(files: List<Uri>) {
354.            view.showProgress()
355.
356.            val experiment = ExperimentItem()
357.
358.            var index = 0
359.            var size = 0
360.
361.            subscribe(Observable.fromIterable(files)
362.                    .map { Utils.getFileForUri(it) }
363.                    .flatMap { Observable.fromIterable(it.getImagesSorted()) }
364.                    .toList()
365.                    .toObservable()
366.                    .doOnNext {
367.                        size += it.size
368.                    }
369.                    .flatMap { Observable.fromIterable(it) }
370.                    .doOnNext {
371.                        val point = eyeDetector.findEye(it)
372.                        experiment.points.add(PointRealmObject(point))
373.                    }
374.                    .subscribeOn(Schedulers.io())
375.                    .observeOn(AndroidSchedulers.mainThread())
376.                    .doOnComplete {
377.                        val id = saveExperiment(experiment)
378.                        view.showChartFragment(id)
379.                        Handler().postDelayed({
380.                            view.hideProgress()
381.                        }, 500)
382.                    }
383.                    .subscribe({
384.                        index++
385.                        val text = "$index /" + size
386.                        val progress = 100 * index / size
387.                        view.countProgress(text, progress)
388.                    }, {
389.                        it.printStackTrace()
390.
```

```
                    view.showErrorMessage(it.message)
                }))
    }

    override fun cancel() {
        unsubscribe()
    }

    private fun saveExperiment(experiment: ExperimentItem): Long {
        val expNumber = mRealm.where(ExperimentItem::class.java)
             .findAll().size + 1
        experiment.name = "Эксперимент №$expNumber"

  val timeStamp = Date().time

  experiment.date = timeStamp
  experiment.id = timeStamp

  mRealm.saveEyeExperiment(experiment)

  return timeStamp
}
class EyeDetector(private val cascade: CascadeClassifier) {
fun findEye(file: File): Point? {
        val rgba = Imgcodecs.imread(file.absolutePath,
                            Imgcodecs.CV_LOAD_IMAGE_COLOR)
        val gray = Imgcodecs.imread(file.absolutePath,
                            Imgcodecs.CV_LOAD_IMAGE_GRAYSCALE)

        val eyes = MatOfRect()

        cascade.detectMultiScale(gray, eyes)

        var iris: Point? = null
        val irises = ArrayList<Point>()

        val eyeAreas = eyes.toArray()
        for (i in eyeAreas.indices) {
            val eyeArea = eyeAreas[i]

       Imgproc.rectangle(rgba, eyeArea.tl(),eyeArea.br(),
```

```
                            Scalar(255.0, 0.0, 0.0, 255.0), 2)

            iris = getIris(gray, eyeArea)

            if (iris != null) {
                Imgproc.circle(rgba, Point(iris.x, iris.y), 2,
                                Scalar(255.0, 255.0, 255.0, 255.0), 2)
                irises.add(iris)
                if (irises.size == 2) {
                    break
                }
            }
        }

        MyUtils.writeImage(file, rgba)

        return iris
    }

    fun getIris(mGray: Mat, area: Rect): Point? {
        var mROI = mGray.submat(area)
        val eyes = MatOfRect()
        val iris = Point()

        cascade.detectMultiScale(mROI, eyes, 1.15, 2,
                Objdetect.CASCADE_FIND_BIGGEST_OBJECT or
                Objdetect.CASCADE_SCALE_IMAGE, Size(30.0, 30.0), Size())

        val eyesArray = eyes.toArray()
        val i = 0
        while (i < eyesArray.size) {
            val e = eyesArray[i]
            e.x = area.x + e.x
            e.y = area.y + e.y
            val eye_only_rectangle = Rect(e.tl().x.toInt(),
                    (e.tl().y + e.height * 0.4).toInt(), e.width,
                    (e.height * 0.6).toInt())
            mROI = mGray.submat(eye_only_rectangle)

            val mmG = Core.minMaxLoc(mROI)

```

```
473.              iris.x = mmG.minLoc.x + eye_only_rectangle.x
474.              iris.y = mmG.minLoc.y + eye_only_rectangle.y
475.
476.              return iris
477.          }
478.          return null
479.      }
480.  }
481.  class Application : Application() {
482.
483.      private val TAG = "EyeTracking"
484.
485.      companion object {
486.          lateinit var appComponent: AppComponent
487.      }
488.
489.      override fun onCreate() {
490.          super.onCreate()
491.          Realm.init(this)
492.          initDagger()
493.          initOpenCV()
494.      }
495.
496.      private fun initDagger() {
497.          appComponent = DaggerAppComponent.builder()
498.                  .appModule(AppModule(this))
499.                  .eyeModule(EyeModule()).build()
500.      }
501.  private fun initOpenCV() {
502.          val loaderCallback = OpenCVLoaderCallback(this)
503.
504.          if (!OpenCVLoader.initDebug()) {
505.              Log.d(TAG, "Internal OpenCV library not found.
506.                          Using OpenCV Manager for initialization.")
507.              OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_3_2_0, this,
508.                          loaderCallback)
509.          } else {
510.              Log.d(TAG, "OpenCV library found inside package. Using it!")
511.              loaderCallback.onManagerConnected(LoaderCallbackInterface.SUCCESS)
512.          }
513.      }
```

```
514.  }
515.  @Module
516.  class AppModule(private val app: App) {
517.
518.      @Provides
519.      @Singleton
520.      fun provideContext(): Context = app
521.
522.      @Provides
523.      fun provideRealm(): Realm = Realm.getDefaultInstance()
524.  }
525.  @Module
526.  class EyeModule {
527.      @Provides
528.      @Singleton
529.      fun provideCascade(): CascadeClassifier = CascadeClassifier()
530.
531.      @Provides
532.      @Singleton
533.      fun provideEyeDetector(cascade: CascadeClassifier):
534.                                      EyeDetector = EyeDetector(cascade)
535.
536.      @Provides
537.      @Singleton
538.      fun provideEyePresenter(): EyeDetectorPresenter = EyeDetectorPresenter()
539.  }
540.
```