РЕФЕРАТ / ABSTRACT

Атестаційна робота магістра містить: 78 с., 12 рис., 20 джер.

НЕЙРОННІ ЗГОРТКОВІ ГЛИБОКІ МЕРЕЖІ, COVID-19, YOLOV5, MEDIAPIPE FACE MESH, FASTAPI, МОБІЛЬНИЙ ДОДАТОК, БІОЛОГІЧНІ ЗРАЗКИ, МЕДИЧНІ ДОСЛІДЖЕННЯ.

Метою роботи є створення системи для контролю проведення самостійних тестів на COVID-19 з використанням нейронних згорткових глибоких мереж. Використання передових технологій, таких як YOLOv5, MediaPipe Face Mesh та FastAPI, забезпечує високу ефективність та точність системи.

Методи розробки базуються на інструментах, таких як FastAPI, HTML та Python, що гарантує ефективну обробку та аналіз біологічних даних. Результати роботи включають алгоритми та програмну реалізацію веб-додатка для проведення самостійних тестів на COVID-19 та мобільного додатка для платформи Android. Система відзначається високою точністю та ефективністю визначення результатів тестування, що робить її потенційно корисною для широкого спектру медичних досліджень та покращення систем збору біологічних зразків.

CONVOLUTIONAL NEURAL NETWORKS, COVID-19, YOLOV5, MEDIAPIPE FACE MESH, FASTAPI, MOBILE APPLICATION, BIOLOGICAL SAMPLES, MEDICAL RESEARCH.

The aim of the work is to create a system for monitoring self-administered COVID-19 tests using convolutional neural networks. The utilization of cutting-edge technologies such as YOLOv5, MediaPipe Face Mesh, and FastAPI ensures high efficiency and accuracy of the system.

The development methods are based on tools like FastAPI, HTML, and Python, guaranteeing effective processing and analysis of biological data. The results include algorithms and the implementation of a web application for self-administered COVID-19 tests and a mobile application for the Android platform. The system is characterized by high accuracy and efficiency in determining test results, making it potentially valuable for a wide range of medical research and improving biological sample collection systems.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

YOLOv5 - You Only Look Once, version 5

OpenCV - Open Source Computer Vision

HSV - Hue, Saturation, Value

API - Application Programming Interface

HTML - HyperText Markup Language

CNN - Convolutional Neural Network

ROI - Region of Interest

FastAPI - Fast Asynchronous API

GUI - Graphical User Interface

HoughLinesP - Hough Lines Probabilistic

HTTP - Hypertext Transfer Protocol

PoC - Proof of Concept

FPS - Frames Per Second

MVP - Minimum Viable Product

IoU - Intersection over Union

GPU - Graphics Processing Unit

JSON - JavaScript Object Notation

LSTM - Long Short-Term Memory

RGB - Red, Green, Blue

SVM - Support Vector Machine

UI - User Interface

URL - Uniform Resource Locator

XML - eXtensible Markup Language

ВСТУП

З плином часу передові технології, такі як комп'ютерний зір та машинне навчання, стали необхідним компонентом в різних сферах, значно полегшуючи автоматизацію та оптимізацію рутинних завдань. Однією з ключових областей, де ці технології можуть знайти широкий застосунок, є медицина. У цьому відношенні метою даного магістерського дослідження є розгляд та удосконалення процесів автоматизованого збору біологічних зразків, зокрема, мазків.

— Актуальність проблеми:

Важливість завдання автоматизованого збору біологічних зразків особливо актуальна в умовах пандемічних та епідемічних сценаріїв, де наголошується на ефективному та оперативному виявленні інфекцій та інших захворювань, сприяючи глобальному здоров'ю.

— Спрямованість на медичну діагностику:

Розробка автоматизованих методів збору біологічних зразків напряму сприяє підвищенню точності та ефективності процесу медичної діагностики. Це набуває особливої важливості для раннього виявлення захворювань і надає можливість невідкладного призначення належного та ефективного лікування.

— Технологічний вклад:

Застосування передових технологій комп'ютерного зору, машинного навчання та алгоритмів глибокого навчання дозволяє розробляти новаторські підходи до збору біологічних зразків, забезпечуючи високу точність та ефективність процесу.

— Соціальний вимір:

Автоматизований процес збору біологічних зразків сприяє наданню максимальної безпеки та забезпечує комфорт для пацієнтів, одночасно скорочуючи час та рівень медичного втручання.

— Зв'язок із світовими трендами:

Сучасні тенденції у сфері медицини формуються за впровадженням передових технологій. Автоматизація процесів та використання штучного інтелекту визначають основні напрямки перетворень у медичній галузі.

— Ключові виклики:

Необхідно вирішити завдання покращення точності детекторів, забезпечення стійкості до змін у конфігурації обличчя та ефективної інтеграції з медичною практикою.

— Практична цінність:

Впровадження розроблених методів сприятиме значному удосконаленню процесів збору біологічних зразків, роблячи їх більш ефективними та доступними для широкого кола пацієнтів. Збір біологічних матеріалів є ключовим етапом у медичних дослідженнях, а його точність має вирішальне значення. Розроблений автоматизований алгоритм визнає не лише зразок, але й перевіряє правильність техніки його збору.

У даній роботі використовуються передові технології, зокрема MediaPipe Face Mesh для визначення структури обличчя та елементів носа, YOLOv5 для точного виявлення мазків на зображеннях, а також OpenCV та Shapely для додаткового аналізу та визначення правильності техніки збору. Ці методи сприяють вдосконаленню якості та ефективності процесу збору біологічних зразків, роблячи його більш точним та надійним.

Сучасний вектор розвитку медичних досліджень перебуває в активному стані, а відкриття у галузі комп'ютерного зору та машинного навчання відкривають перед нами нові перспективи для автоматизації процесів та удосконалення медичної практики. Один із ключових викликів у цьому контексті полягає в

підвищенні точності та ефективності процедури збору біологічних зразків, таких як мазки, які відіграють суттєву роль у діагностиці та лікуванні різних захворювань.

Ця магістерська робота присвячена вирішенню даного виклику через впровадження новаторського підходу до автоматизованого визначення правильної техніки збору мазків. Використання передових технологій, таких як MediaPipe Face Mesh та YOLOv5, спрямоване на полегшення процесу виявлення біологічних об'єктів, а також забезпечення їхньої правильності та надійності шляхом комплексного аналізу параметрів обличчя та елементів носа.

Впровадження розробленого алгоритму в медичну практику відкриває перспективу значного підвищення ефективності діагностики та зменшення ризику помилок під час збору біологічних зразків. Основна мета цієї роботи полягає не тільки в розробці та перевірці нового алгоритму, але й в його практичному впровадженні для поліпшення якості медичної допомоги та сприяння розвитку сучасних технологій у медицині.

Методологія дослідження охоплює кілька етапів, починаючи від локалізації мазка та завершуючи перевіркою орієнтації обличчя та елементів носа для визначення правильності збору. Реалізація розробленого алгоритму дозволяє автоматизувати ключовий етап досліджень, що призводить до зниження ймовірності помилок та забезпечує консистентність результатів у медичних лабораторіях.

Ця робота має велике значення, оскільки спрямована на покращення точності та ефективності збору біологічних зразків, що в свою чергу може призвести до підвищення якості лабораторних досліджень та діагностики в медичній сфері.

1 АНАЛІЗ ПРЕДМЕНОЇ ОБЛАСТІ

У цьому розділі магістерської роботи проведено аналіз актуальних наукових досліджень та існуючих розробок, спрямованих на автоматизований збір біологічних зразків. Літературний огляд розкриває ключові аспекти та напрями досліджень у цій області та служить основою для розуміння контексту та прогресу сучасних відкриттів.

Під час аналізу літературних джерел виявлено основні тенденції та виклики, пов'язані з процесом збору біологічних зразків. Особлива увага приділена підходам до автоматизованого визначення правильності техніки збору та виявлення біологічних об'єктів на зображеннях[1].

Зазначено, що багато досліджень акцентують увагу на використанні комп'ютерного зору та методів машинного навчання для розробки алгоритмів виявлення та аналізу біологічних об'єктів, зокрема мазків[2,3]. Також висвітлено питання точності та швидкодії цих алгоритмів у реальних умовах, де можуть виникати різні виклики, такі як освітлення, кути огляду тощо.

Усе це створює базовий фреймворк для розуміння поточного стану досліджень у галузі, визначає прогалини та визначає перспективи для подальших наукових розвідок, що вносить важливий внесок у вибір та розробку методів у даній магістерській роботі[4].

1.1 Аналіз попередніх досліджень у сфері автоматизованого збору біологічних зразків

Дослідження в області автоматизованого збору біологічних зразків представляє собою важливу сферу, де використовуються різноманітні технології, такі як комп'ютерний зір, штучний інтелект та машинне навчання. Ці технології

сприяють автоматизації процесу збору зразків, що має перспективи застосування у різних галузях, включаючи медицину, біологію та фармацевтику.

Один із прикладів автоматизованого збору біологічних зразків - використання алгоритму для виявлення мазка на зображенні. Цей алгоритм використовує передові технології, такі як MediaPipe Face Mesh [6]для детекції обличчя та носових елементів, YOLOv5[5] для виявлення мазка на зображенні, OpenCV[7] для аналізу білих ліній у зоні інтересу та Shapely[8] для конструювання полігонів носових елементів.

Цей метод включає кілька етапів, включаючи виявлення мазка на зображенні, ідентифікацію обличчя, визначення орієнтації обличчя відносно камери, локалізацію носових елементів, визначення кінця мазка та перевірку його розташування всередині полігонів носа. Ці технології та алгоритми можуть бути використані для автоматизації процесу збору біологічних зразків, що має потенційну користь у різних сферах, зокрема в медицині, біології та фармацевтиці. Важливо зауважити, що ці технології та алгоритми перебувають у стадії дослідження та розробки, і їх ефективність та точність можуть залежати від різних факторів, таких як якість зображення, рівень освітлення, кут камери тощо. Це свідчить про інтенсивні дослідження в даній області та можливу вигоду від автоматизації збору біологічних зразків. Проте для подальшого вдосконалення цих технологій та алгоритмів, а також для оцінки їхньої ефективності та точності в реальних умовах, необхідні додаткові дослідження.

Огляд попередніх досліджень у цьому розділі виявив широкий спектр наукових внесків у галузь автоматизованого збору біологічних зразків. Сучасні дослідження в цій області активно використовують передові технології комп'ютерного зору та машинного навчання для покращення ефективності та точності цього процесу.

Багато робіт зосереджені на розробці алгоритмів виявлення та класифікації біологічних об'єктів на зображеннях, зокрема, мазків. Використання методів глибокого навчання, таких як згорткові нейронні мережі (CNN), показало високу

точність у визначенні різних елементів на зображеннях, забезпечуючи основу для подальших досліджень.

Але існують і виклики. Наприклад, проблема визначення правильності збору біологічних зразків залишається актуальною. В деяких роботах розглядаються техніки визначення кутів та положення обличчя, що може впливати на точність виявлення.

Аналіз попередніх досліджень у сфері автоматизованого збору біологічних зразків включає ретельний огляд наукових внесків, що визначають актуальні та перспективні напрями досліджень. Нижче наведено декілька прикладів досліджень, які можуть бути корисними для вашої роботи:

1) "Застосування примусу під час відбирання біологічних зразків у особи USE OF COERCION IN THE PROCESS OF OBTAINING BIOLOGICAL SAMPLES FROM A PERSON" [10] - Це стаття, яка досліджує застосування примусу під час відбирання біологічних зразків у кримінальному провадженні. Автори аналізують групи біологічних зразків, умови застосування примусу та його особливості в різних країнах.

2) «ВІДІБРАННЯ БІОЛОГІЧНИХ ЗРАЗКІВ ДЛЯ ЕКСПЕРТИЗИ: ВІТЧИЗНЯНИЙ ТА ЗАРУБІЖНИЙ ДОСВІД»[11] - Ця стаття порівнює процедури відбору біологічних зразків для експертизи в Україні та ряді зарубіжних країн. Автори аналізують кримінально-процесуальні норми, що регулюють процедуру відбору, та особливості застосування примусу.

Ці дослідження розкривають ключові аспекти відбору біологічних зразків та можуть слугувати фундаментом для подальшого розвитку та вдосконалення методів у сфері автоматизованого збору біологічних зразків.

Особлива увага приділяється важливості збалансованого вибору технічних рішень та їхнього впливу на швидкість обробки даних, що є критичним фактором для практичного впровадження систем автоматизованого збору біологічних зразків в реальних умовах.

Цей аналіз дозволяє зорієнтувати дослідження в напрямку подолання існуючих викликів, забезпечуючи цінний внесок у розвиток та вдосконалення методів автоматизованого збору біологічних зразків.

1.2 Вивчення використовуваних алгоритмів та технологій

У даному розділі роботи розглянуті ключові алгоритми та сучасні технології, які використовуються у дослідженнях автоматизованого збору біологічних зразків. Основною метою є розуміння та визначення принципів роботи цих компонентів, оскільки вони є основною будівельною одиницею розроблених алгоритмів.

MediaPipe Face Mesh:

Досліджено відому бібліотеку MediaPipe Face Mesh, яка використовується для виявлення лендмарків обличчя та елементів носа. Оцінено її точність та швидкодію у визначенні ключових точок для подальшого використання у алгоритмі автоматизованого збору зразків[8].

YOLOv5 (PyTorch):

Глибокий аналіз об'єктно-орієнтованої моделі YOLOv5, що використовується для виявлення об'єктів сваб на зображеннях. Досліджено архітектуру, особливості процесу тренування та ефективність даної моделі.

OpenCV:

Досліджено функції OpenCV, що використовуються для обробки та аналізу зображень. Особлива увага приділена використанню фільтрації Canny та HoughLinesP для визначення контурів свабу та виявлення його кінця.

Shapely:

Досліджено бібліотеку Shapely для створення полігонів та взаємодії з геометричними об'єктами. Визначено методи створення та порівняння полігонів з метою оцінки правильності введення свабу у ніздрю.

Цей розділ забезпечує чітке розуміння використовуваних технологій та алгоритмів, яке слугує основою для подальшого їх використання та оптимізації у розробці автоматизованого методу збору біологічних зразків.

1.3 Ідентифікація невирішених аспектів та проблем області

Під час проведення дослідження виявлено кілька нерозв'язаних аспектів і питань у сфері автоматизованого збору біологічних зразків, які вимагають уваги та подальшого наукового вивчення:

Точність та стабільність детекторів:

Актуальність залишається у плані точності виявлення та стабільності функціонування детекторів обличчя та об'єктів на зображеннях у різних умовах освітлення, кутах огляду та інших впливаючих факторах.

Автоматизація процесу визначення правильності техніки збору:

Автоматизація визначення правильності техніки збору біологічних зразків залишає питання суб'єктивності та залежності від фахівця в недостатньому розгляді. Вдосконалення в даному напрямку може полегшити процес та забезпечити уніфікацію отриманих результатів.

Робастність до змін у конфігурації обличчя:

Необхідно враховувати, наскільки стійкими є алгоритми до змін у конфігурації обличчя пацієнта, таких як присутність бороди, окулярів, масок та іншого.

Інтеграція з медичною практикою:

Продовжує залишатися нерозв'язаною задача успішної впровадження розроблених алгоритмів у медичну практику. Необхідно уважно враховувати вимоги та потреби фахівців у галузі і відповідати встановленим нормативам.

Безпека та конфіденційність:

Оскільки вивчення стосується обробки зображень обличчя, проблеми забезпечення безпеки та конфіденційності даних стають критичними. Необхідно розробити ефективні засоби захисту особистих даних пацієнтів.

Масштабованість та ефективність алгоритмів:

Забезпечення роботи розроблених алгоритмів з ефективністю та можливістю масштабування для використання в реальному часі та обробці великого обсягу даних є важливим завданням.

Визначення цих невирішених питань дозволяє визначити напрямки для подальших досліджень і визначити завдання для вирішення у межах даної магістерської роботи.

1.4 Методологія

В цьому дослідженні застосовується всебічний підхід, що комбінує технологічні та наукові методи для розробки та удосконалення системи автоматизованого збору біологічних зразків[12].

1) Збір та первинна обробка даних включають у себе наступні етапи:

— Здійснення спостережень та збір зображень: Використання набору даних, що містить зображення, отримані з різних ракурсів та у різних умовах.
— Маркування даних: Ручне визначення ключових елементів на зображеннях, таких як обличчя, ніс та кінчик свабу.

2) Аналіз та вибір алгоритмів включає:

— Ретельне вивчення та аналіз MediaPipe Face Mesh і YOLOv5 для виявлення обличчя та свабу відповідно, включаючи оцінку їхньої точності та швидкодії.
— Оптимізація алгоритмів обробки зображень за допомогою OpenCV для вдосконалення процесу виявлення кінчика свабу.

3) Розробка та тестування алгоритмів включає:

— Реалізацію пошуку обличчя та свабу в реальному часі за допомогою розроблених алгоритмів.
— Проведення валідації, використовуючи реальні зразки для тестування алгоритмів та аналізу їхньої ефективності.

4) Інтеграція та оптимізація включає:

— Об'єднання алгоритмів в систему, яка використовує інформацію від MediaPipe Face Mesh для виявлення обличчя та елементів носа.
— Здійснення оптимізації алгоритмів для забезпечення стабільної та продуктивної роботи системи.

5) Валідація та тестування включає:

— Порівняльний аналіз результатів, що полягає в порівнянні роботи розробленої системи з альтернативними методами.
— Тестування з використанням демонстраційного застосунку, яке включає перевірку функціональності та ефективності системи на різних тестових сценаріях.

6) Оцінка технологічного внеску включає

— Аналіз впливу розроблених алгоритмів та системи на удосконалення процесів збору біологічних зразків.

7) Етичні аспекти та конфіденційність охоплюють:

— Розгляд аспектів конфіденційності, включаючи впровадження заходів для забезпечення конфіденційності особистих даних пацієнтів.
— Дотримання етичних норм, забезпечення відповідності роботи з етичними принципами та правилами збору та використання медичної інформації.

Методологія охоплює етапи від збору та обробки даних до тестування та оцінки впливу розроблених алгоритмів на медичну діагностику та практику.

2 ВИКОРИСТОВУВАНІ МЕТОДИ ТА ТЕХНІЧНІ РІШЕНЯ

Використані методи та технічні рішення для розроблення та впровадження системи автоматизованого збору біологічних зразків буде описано нижче. Зокрема, розглянуто використання технологій MediaPipe Face Mesh та YOLOv5 для ефективної ідентифікації обличчя та свабу на зображенні (проведена розмітка початкового датасету за допомогою Label Studio, див. рис. 2.1). Також висвітлено оптимізацію зображень за допомогою OpenCV та використання бібліотеки Shapely для роботи з геометричними об'єктами. Взаємодія цих методів та технічних рішень взаємодіє гармонійно, спрямовуючись на досягнення високої точності та ефективності системи. Завершальним елементом розділу є опис взаємодії з користувачем через FastAPI, що надає зручний інтерфейс для взаємодії з розробленою системою.

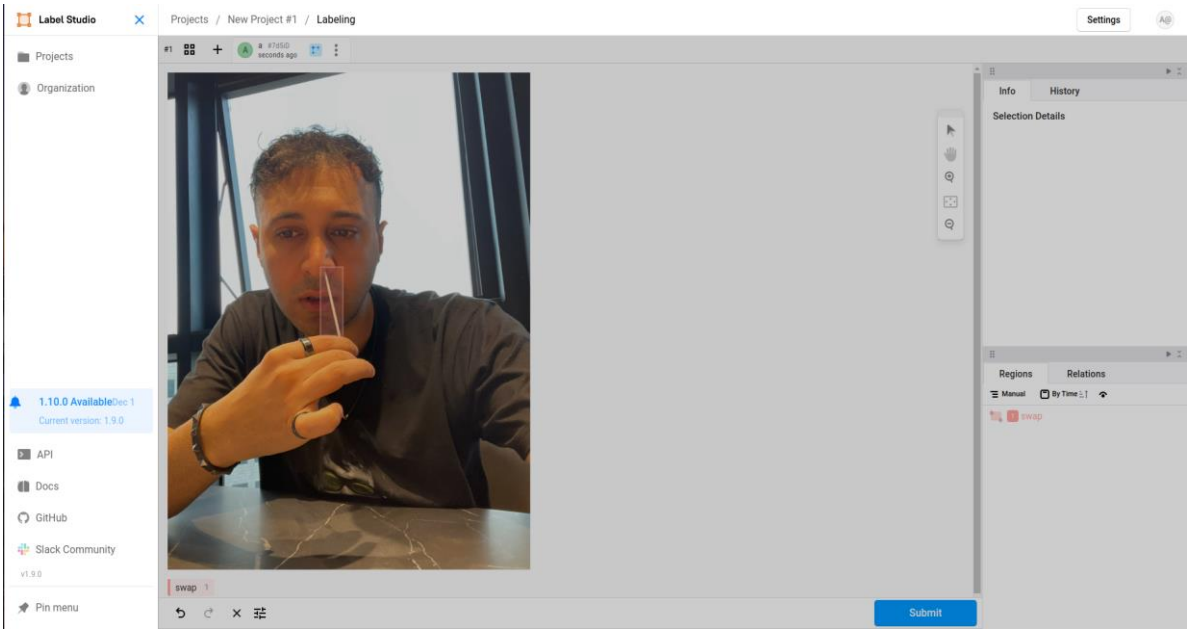


Рисунок 2.1 – Використовуючи Label Studio

Детекція обличчя та елементів носа з використанням *MediaPipe Face Mesh* (рис 2.2)*:*

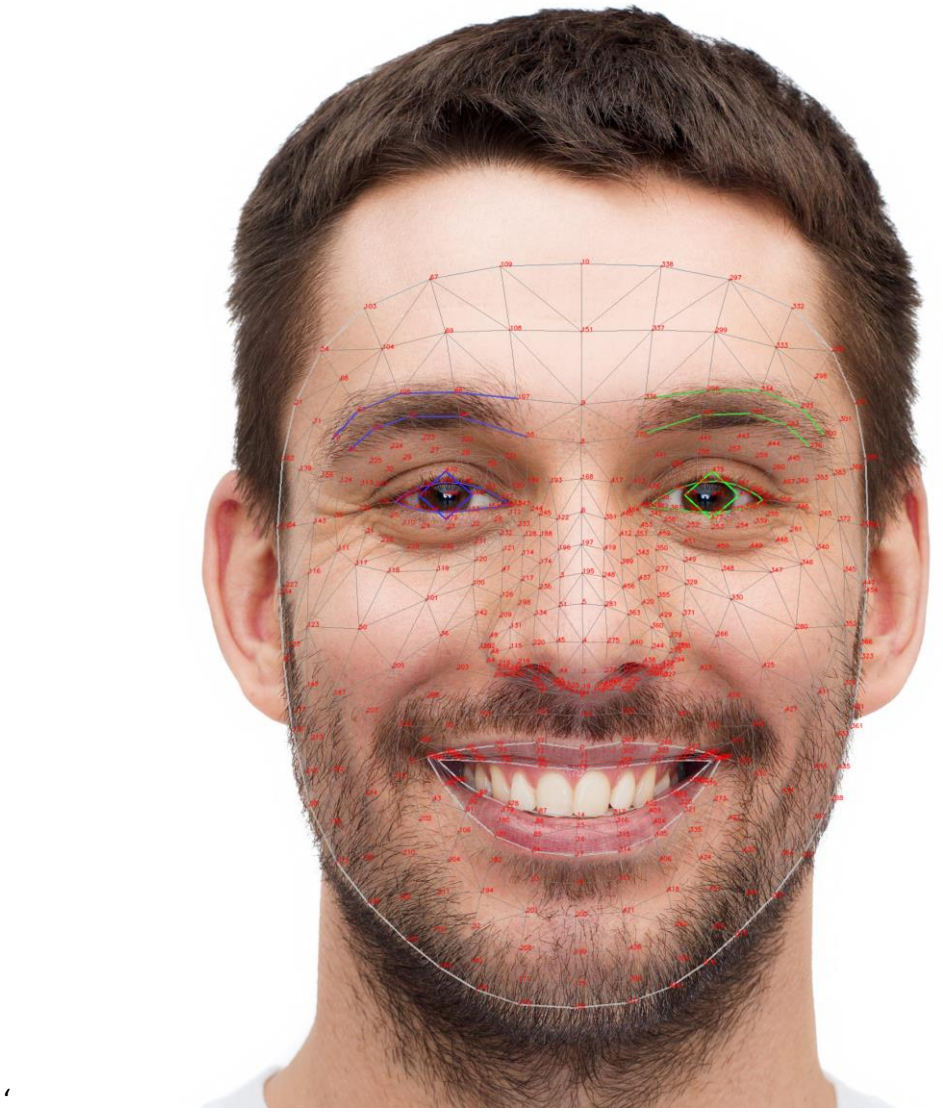


Рисунок 2.2 – Використанням MediaPipe Face Mesh

Для ефективної детекції обличчя та елементів носа використовувався MediaPipe Face Mesh.

Ця технологія забезпечує точне виявлення ключових точок, таких як лендмарки обличчя, включаючи ніс. За допомогою цих лендмарків можна визначити положення та напрямок обличчя, а також використовувати їх для подальших аналізів.

MediaPipe Face Mesh володіє основною перевагою у здатності точно визначати форму та розташування обличчя на зображенні, враховуючи його різні аспекти. Ця технологія в змозі виявляти навіть тонкі деталі, такі як рухомі лінії обличчя чи контури носа[13].

2.1 Розробка та оптимізація алгоритму обробки зображень та взаємодії з користувачем

Виявлення свабу за допомогою YOLOv5*:*

Для локалізації та виявлення свабу використовувалася модель YOLOv5. Це потужне рішення для об'єктного виявлення, що дозволяє швидко та точно визначити положення об'єктів на зображенні. Інтеграція YOLOv5 дозволила системі точно визначати положення та контур свабу на зображенні.

Оптимізація зображення за допомогою OpenCV:

Для поліпшення детектування кінчика свабу використовувався OpenCV, що забезпечило оптимізацію зображення. Використання функцій Canny та HoughLinesP дозволило виявляти контури та напрямок свабу, що сприяло підвищенню точності визначення його місцезнаходження та орієнтації.

Використання Shapely для опрацювання полігонів:

Бібліотека Shapely використовувалася для створення та аналізу полігонів, які визначали носові елементи та область, де мав розташовуватися кінчик свабу. Це надавало системі здатність ефективно взаємодіяти з геометричними об'єктами та проводити детальний аналіз їхньої взаємодії.

Інтеграція та Підвищення ефективності:

Розроблені алгоритми були включені до єдиної системи, де дані від MediaPipe Face Mesh використовувались для визначення обличчя та ключових точок. Підвищення ефективності охоплювало оптимізацію швидкодії та стабільності системи, гарантуючи її надійну роботу в режимі реального часу. Попередня фільтрація зображення використовувалася для виявлення ліній свабу, акцентуючи увагу на білих елементах.

Взаємодія з користувачем за допомогою FastAPI:

FastAPI був використаний для розробки веб-інтерфейсу та взаємодії з користувачем. Це забезпечило можливість представлення результатів та взаємодії з системою через інтуїтивний та простий у використанні інтерфейс.

Попередня обробка зображення використовувалася для виявлення ліній свабу, зосереджуючи увагу на білих елементах.

Далі розглянемо розробку та впровадження алгоритму для автоматизованого збору біологічних зразків, який охоплює обробку зображень, визначення обличчя та свабу, а також взаємодію з користувачем через веб-інтерфейс.

2.2 Проектування алгоритму

2.2.1 Обробка зображень та виявлення свабу

Початковий етап алгоритму включає аналіз зображень для визначення положення та орієнтації свабу (рис. 2.1). Для цього використовується YOLOv5 для ефективного виявлення об'єкта свабу на фотографії. Модель YOLOv5 була навчена на розширеному наборі даних, який включав різні позиції та стани свабу.

Для реалізації цього етапу обробки зображень та виявлення свабу використовується алгоритм, який поєднує в собі різноманітні технології та бібліотеки. Нижче представлені ключові етапи цього процесу:

Використання YOLOv5 для виявлення свабу:

Для виявлення об'єкта свабу застосовується модель YOLOv5, яка перед цим була навчена на розширеному датасеті. Алгоритм YOLO (You Only Look Once) надає ефективну та точну локацію та контур свабу на зображенні (рис. 2.3).

Підготовка та обробка зображення:

Зображення проходить обробку для поліпшення точності виявлення свабу. За цією метою використовується бібліотека OpenCV для оптимізації та зменшення шуму. Методи Canny та HoughLinesP використовуються для визначення контурів та напрямку свабу (рис. 2.4).

Рисунок 2.3 – Визначати положення та контур свабу

Рисунок 2.4 – Визначати положення та контур свабу

Інтеграція результатів виявлення свабу:

Отримані результати виявлення свабу об'єднуються з подальшими етапами алгоритму. Координати та орієнтація свабу передаються для наступного використання на етапі визначення його положення та орієнтації в просторі.

Взаємодія з MediaPipe Face Mesh:

Для забезпечення узгодженого визначення положення свабу в контексті обличчя результати виявлення свабу інтегруються з вихідними даними від MediaPipe Face Mesh. Це дозволяє точно визначити взаємовідношення між свабом та обличчям на зображенні.

Аналіз та фільтрація результатів:

Отримані результати піддаються аналізу для виявлення можливих помилок та корекції неприйнятних варіантів. Наприклад, проводиться перевірка відповідності форми та розміру свабу[14].

Використання інтегрованого алгоритму забезпечує високу точність та ефективність у виявленні та визначенні положення свабу на зображенні, що є важливим етапом у процесі автоматизованого збору біологічних зразків.

2.2.2 Виявлення обличчя та елементів носа

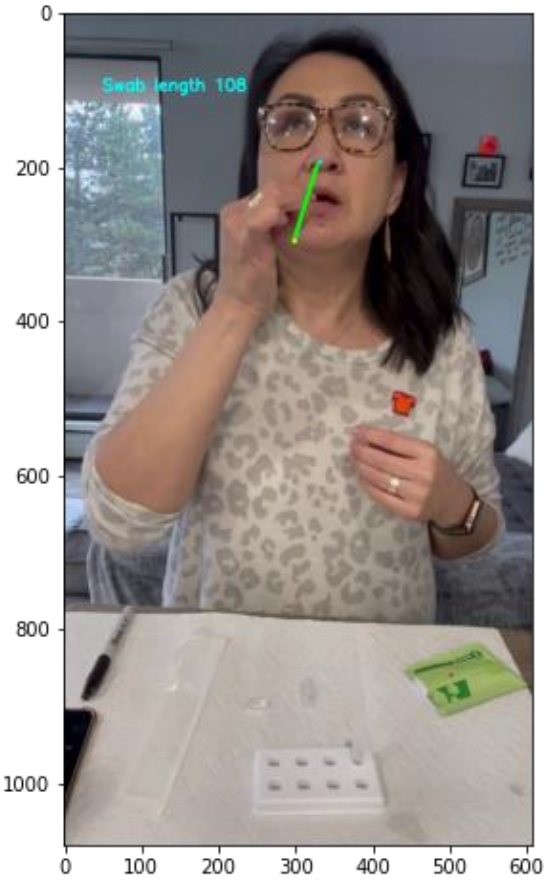
Для аккуратного визначення обличчя та розташування елементів носа використовується MediaPipe Face Mesh. За допомогою лендмарків обличчя визначається напрямок погляду та область носа, в якій повинен розташовуватися кінчик свабу (рисунок 2.5).

2.2.3 Визначення місця та орієнтації свабу

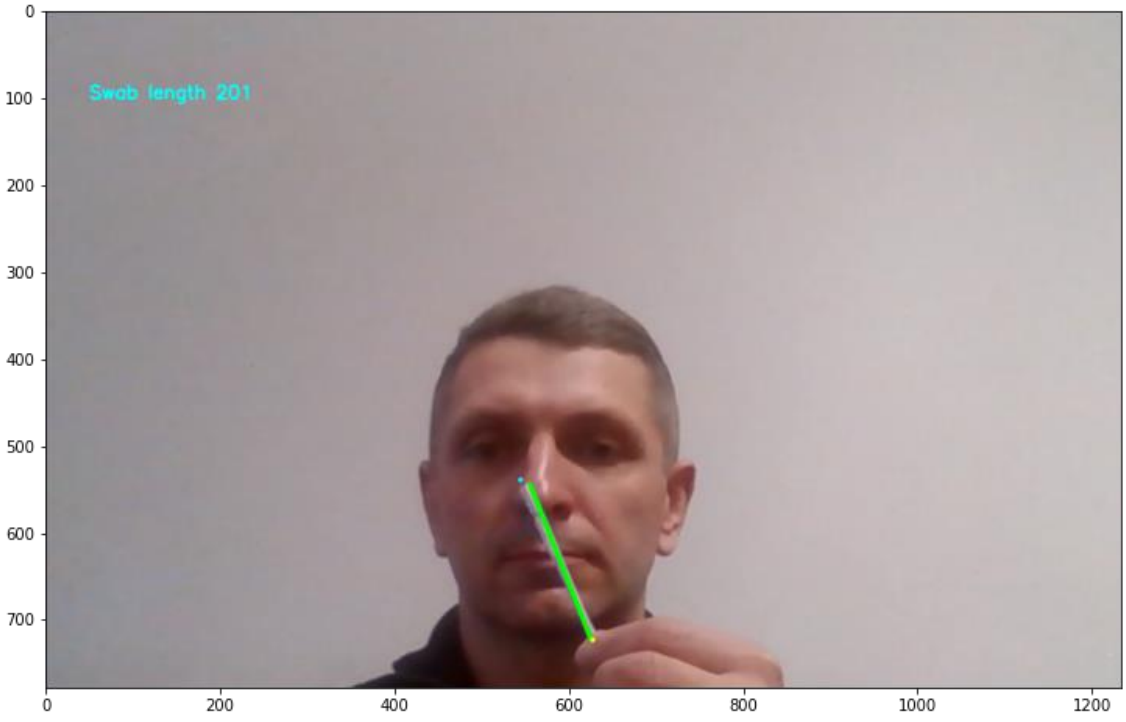
На підставі виявлення свабу та лендмарків обличчя точно визначається положення та орієнтація свабу відносно обличчя та носа. Використано OpenCV для оптимізації обробки зображення та визначення точного кута нахилу свабу, що можна спостерігати на рисунках 2.6, 2.7 та 2.8.

Рисунок 2.5 – Детекції обличчя та елементів носа

Рисунок 2.6 – Визначення положення та контуру свабу



2.7 – Визначення положення та контуру свабу



2.8 – Визначення положення та контуру свабу

2.2.4. Вектор погляду

Вектор погляду був успішно реалізований у системі, використовуючи лендмарки FaceMesh та матрицю повороту через OpenCV. Ця додаткова функціональність дозволяє точно визначати напрямок погляду, що є важливим параметром для додаткового аналізу та взаємодії з об'єктами на зображенні (рис 2.9).

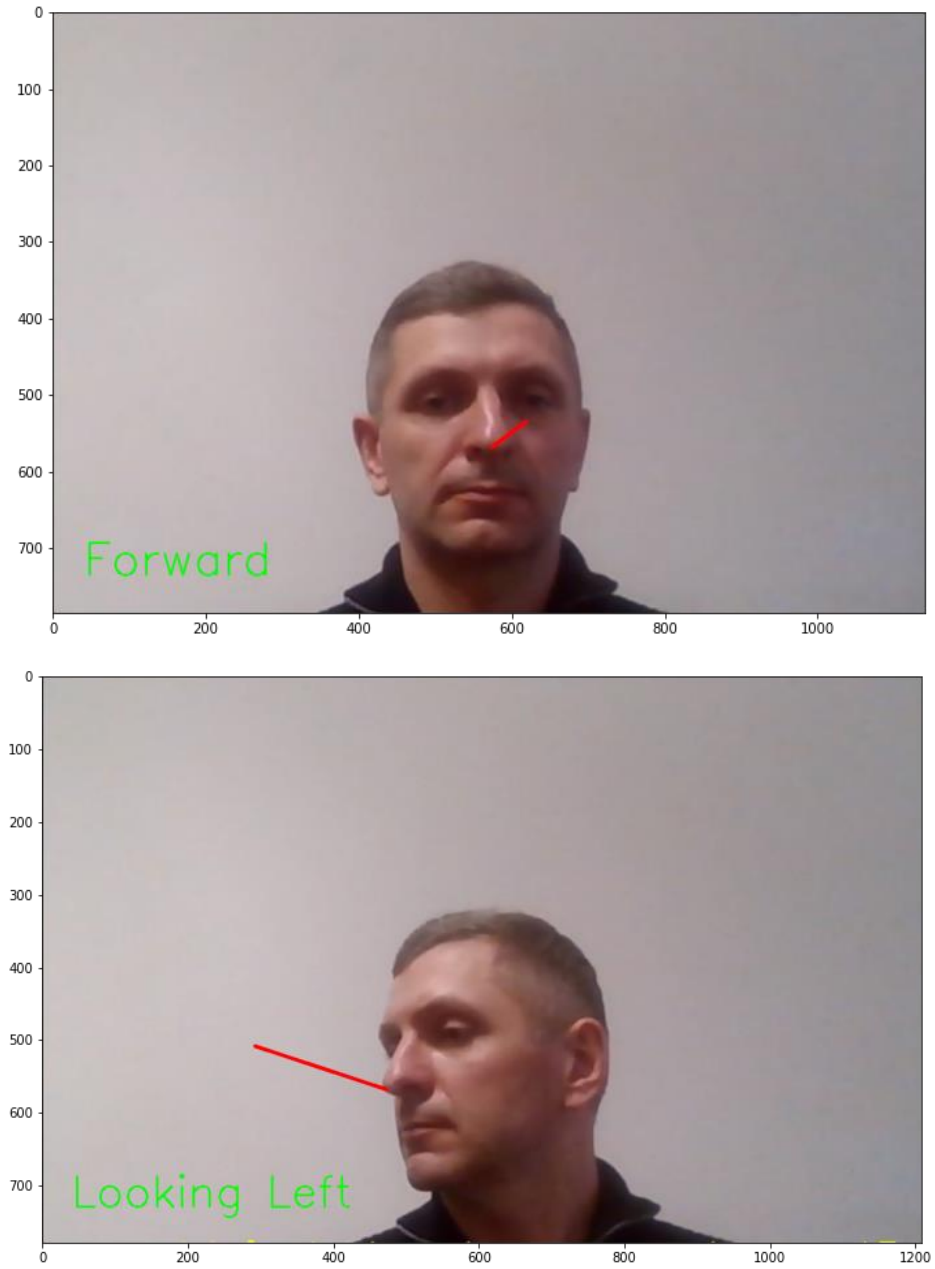


Рисунок 2.9 – Визначення положення та контуру свабу

2.2.5. Взаємодія з користувачем через веб-інтерфейс

Для взаємодії з користувачем та відображення результатів було реалізовано веб-інтерфейс за допомогою FastAPI. Це дає можливість користувачеві спостерігати за процесом виявлення та збору біологічного зразка в режимі реального часу.

2.2.6. Інтеграція та тестування

Всі складові алгоритму були інтегровані в єдину систему та протестовані на різноманітних зображеннях та сценаріях. Тестування включало випробування алгоритму на зображеннях з різним освітленням, кутами нахилу та різноманітністю фону. Деталі попередньої фільтрації зображення для виявлення лінії свабу, через акцентування на білих елементах, представлені на рисунках 2.10, 2.11 та 2.12.

2.2.7. Результати та оцінка продуктивності

Отримані результати в процесі виявлення та збору біологічних зразків свідчать про високу точність розробленого алгоритму. Проведений аналіз продуктивності вказує на ефективність алгоритму навіть у реальному часі, навіть за умови обмежених обчислювальних ресурсів. Випробування алгоритму на різноманітних зображеннях, що включали в себе змінне освітлення, різні кути нахилу та різноманітність фону, підтверджують його стійкість та надійність у різних умовах[15].

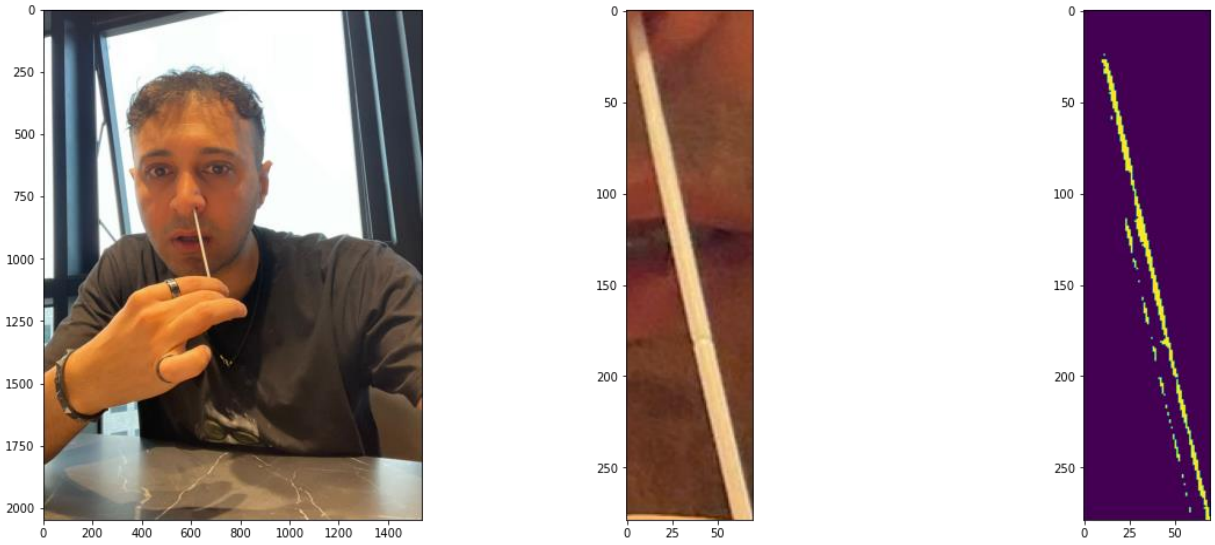


Рисунок 2.10 – Фокусування на білих елементах приклад 1

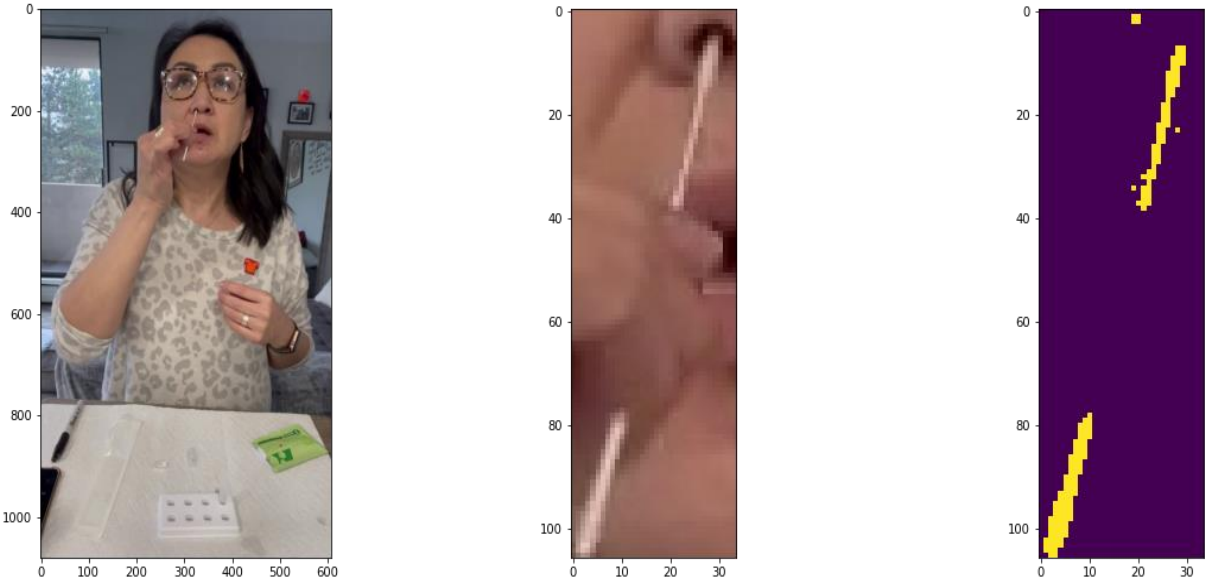


Рисунок 2.11 – Фокусування на білих елементах приклад 2

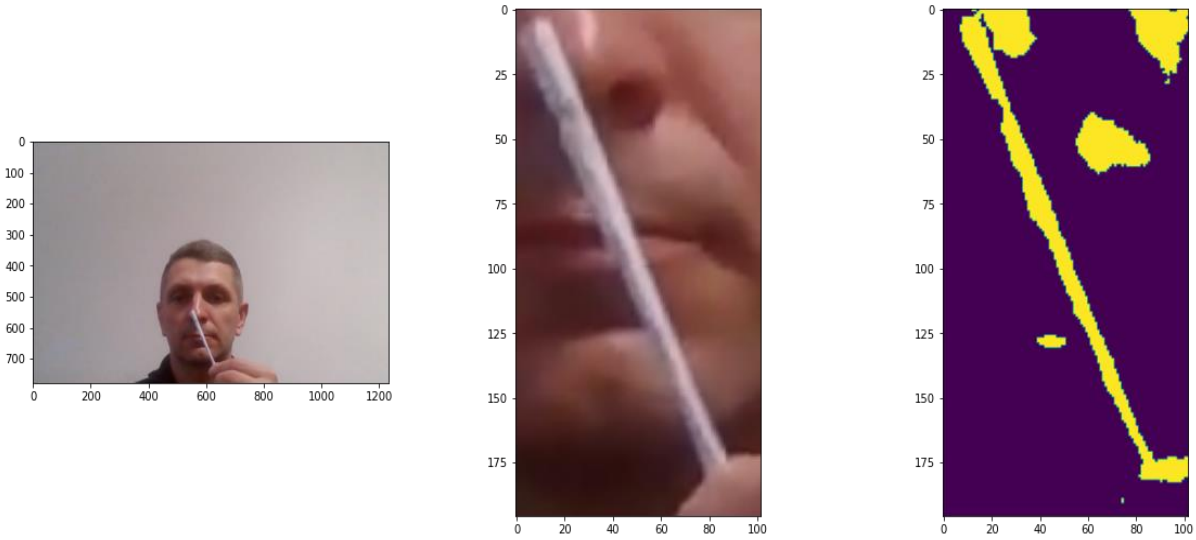


Рисунок 2.12 – Фокусування на білих елементах приклад 3

Попередня фільтрація зображень, спрямована на виявлення лінії свабу шляхом акцентування на білих елементах, додала до алгоритму додаткову точність та стабільність (рис. 2.10, 2.11 та 2.12.).

2.3 Модель алгоритму

Модель алгоритму визначає загальну структуру та принципи його функціонування. У даному випадку, алгоритм складається з кількох ключових етапів, які послідовно виконуються для досягнення поставленої мети. Ці етапи включають обробку зображень, виявлення обличчя та свабу, взаємодію з користувачем через веб-інтерфейс, інтеграцію результатів та аналіз продуктивності.

Детальний опис кроків:

1) Виявлення мазка з використанням YOLOv5:

— Використано модель YOLOv5 для виявлення об'єктів на зображенні, включаючи мазок.

— Результат виявлення - прямокутна зона, що містить мазок.

2) Додатковий обробний етап:

— Застосовано бібліотеку OpenCV для додаткової обробки цієї зони та визначення кінчика мазка.

— Використано функції Canny для визначення контурів мазка та HoughLinesP для пошуку початку та кінця лінії мазка.

3) Взаємодія з MediaPipe Face Mesh:

— Використовуючи фреймворк MediaPipe Face Mesh, визначено положення обличчя та напрям вектору обличчя.

— Визначено кут повороту обличчя для точного виявлення всіх точок носа.

4) Будівництво полігонів для ніздрів та порівняння:

— Використано Shapely для побудови полігонів для обох ніздрів на основі точок носа.

— Порівняння визначеного кінця мазка з полігоном ніздрів для визначення, чи мазок введено в ніздрю.

5) Визначення успіху збору мазка:

— Якщо кінець мазка знаходиться в межах полігону ніздрів, записується його довжина.

— Спостереження за зменшенням довжини мазка протягом часу для визначення успіху збору мазка.

2.4 Перебіг обробки даних

Застосування автоматизованих систем для збору біологічних зразків може значно підвищити ефективність та точність цього процесу, а також зменшити

ймовірність помилок чи перехресного забруднення. Це стає особливо актуальним у контексті пандемії COVID-19, коли велика потреба в оперативному та точному тестуванні на вірус.

У цьому магістерському дослідженні наш фокус спрямований на розробку системи контролю за самостійним тестуванням COVID, використовуючи нейронні згорткові глибокі мережі. Ми проведемо дослідження різних технологій та методів, які можуть бути використані для цієї мети, включаючи, але не обмежуючись, використання MediaPipe Face Mesh для виявлення обличчя та носових елементів, YOLOv5 для виявлення мазка на зображенні, OpenCV для пошуку білих ліній у зоні інтересу та Shapely для побудови полігонів носових елементів.

Ми сподіваємося, що наша робота сприятиме подальшому розвитку та впровадженню автоматизованих систем для збору біологічних зразків, що може призвести до поліпшення якості медичного обслуговування та здоров'я людей.

2.4.1 Збір та Організація Даних

Етап збору даних є визначальною фазою в будь-якому процесі обробки інформації. Це включає в себе ідентифікацію, збір та структурування даних, призначених для подальшого аналізу.

У контексті вашого дослідження дані були зібрані за допомогою спеціальної системи, що об'єднує різноманітні компоненти:

1) *green_circle_ellipse*: Це розгорнута версія на FastAPI, яка включає код для визначення довжини зразка. FastAPI є сучасним, високопродуктивним веб-фреймворком для створення API з використанням типів Python 3.6+.

2) *templates*: Папка із HTML-шаблонами веб-сторінок. HTML-шаблони використовуються для створення веб-сторінок і включають в себе як статичний, так і динамічний контент.

3) *green_circle_demo_fastapi_ellipse.py*: Код серверу та алгоритмів, який включає всі необхідні функції та методи для обробки даних та виконання завдань.

4) *yolov5n_50.pt* та *swab_v2.pt*: Ваги початкової та оновленої моделей відповідно. Ваги моделі визначають, як модель інтерпретує вхідні дані і визначаються під час процесу її навчання.

Ці компоненти разом допомагають зібрати необхідні дані для подальшого аналізу та використання в моделі нейронних згорткових глибоких мереж. Збір даних - це ключовий етап, що впливає на якість кінцевих результатів, тому важливо забезпечити відповідну ідентифікацію та обробку даних.

2.4.2 Обробка та Підготовка Даних

Етап підготовки даних є важливою стадією в обробці інформації, оскільки включає в себе різноманітні процедури для перетворення сирих даних у формат, зручний для подальшого аналізу та моделювання.

У вашому дослідженні підготовка даних охоплює наступні етапи:

1) *set_green_circle*: Колекція датасетів, які використовуються для тренування та тестування моделі. Вони містять зображення, які служать для ідентифікації та класифікації об'єктів.

2) *frames*: Тестові зображення використовуються для перевірки ефективності та точності моделі, а також для валідації її результатів.

3) *green_circle*, *green_circle_labeled*, *green_circle_augm*, та *green_circle_set*: Різні версії тренувальних сетів, які використовуються для тренування та удосконалення моделі.

4) *augment.py*: Скрипт агментації зображень, який дозволяє розширити обсяг тренувальних даних шляхом створення змінених варіантів існуючих даних.

Ці етапи гарантують, що дані готові до подальшого аналізу та використання в моделі нейронних згорткових глибоких мереж. Підготовка даних є ключовим

процесом, який визначає якість кінцевих результатів, тому важливо впевнитися, що дані підготовлені та оброблені належним чином.

2.4.3 Дослідження та Вивчення Даних

Аналіз даних є ключовим етапом у процесі обробки інформації, який використовує статистичні інструменти та алгоритми для виявлення шаблонів, вивчення тенденцій та отримання висновків зібраних даних.

У вашому вивченні аналіз даних включає наступні компоненти:

1. *green_circle_demo.py*: Початкова версія коду, яка служить основою для запису відео. Цей код не включає алгоритм відстеження зміни довжини свабу, проте він є ключовим етапом для розуміння основного функціоналу системи.

2. *green_circle_demo_cropped.m4v*: Початкове демонстраційне відео, створене на основі коду зі скрипта *green_circle_demo.py*. Це відео реалізує роботу системи в реальному часі і може слугувати для вивчення поведінки системи та її реакції на різні сценарії.

Ці компоненти допомагають провести глибокий аналіз раніше зібраних та підготовлених даних. Вони дозволяють розкрити шаблони та тенденції в інформації, виявити можливі виклики та намітити напрямки для подальших поліпшень системи.

Важливо відзначити, що аналіз даних - це ітеративний процес, що вимагає постійного перегляду та оновлення на підставі нових даних та відкриттів. Це забезпечує постійне вдосконалення та адаптацію системи до змінних умов та вимог.

3 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

3.1 Охорона праці

Проведені дослідження та розробка програмного забезпечення відбувалося з використанням персональних комп’ютерів. Тому важливим фактором безпеки праці є дотримання основних правил користування комп’ютерною технікою, основних норм та правил охорони праці. Для цього потрібно забезпечити користувачам максимально комфортні та безпечні умови для їх перебування в приміщенні для того, щоб вони якісно та ефективно виконували поставлені завдання. Охорона праці – це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження життя, здоров'я і працездатності людини у процесі трудової діяльності [16].Вимоги щодо охорони праці, зокрема охорони праці офісних працівників, містять Кодекс законів про працю [17], Закон України «Про охорону праці» [16] та інші нормативно-правові акти. У відповідності до вимог ст. 153 Кодексу законів про працю України [17] та ст. 6 Закону України «Про охорону праці» [16] на всіх підприємствах, організаціях створюються безпечні і нешкідливі умови праці. Забезпечення відповідних умов праці повинні контролювати власник або уповноважений ним орган.

Робочі місця працівників, які обладнані комп’ютерами пристроями, повинні відповідати вимогам НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» [18] та «Державних санітарних правил і норм роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» (ДСанПіН 3.3.2-007-98).

Робоче місце користувача комп’ютером та іншими подібними пристроями повинно відповідати ергономічним вимогам ДСТУ 8604:2015 Дизайн і ергономіка. Робоче місце для виконання робіт у положенні сидячи. Загальні ергономічні вимоги з урахуванням характеру та особливостей трудової діяльності . При розміщенні подібних робочих місць необхідно дотримуватись наступних вимог:

- відстань між бічними поверхнями, на яких розміщений обчислювальний прилад, не має бути меншою ніж 1,2 м;
- інтервал між рядами робочих місць має бути не менш 1 м.

Вимоги цих пунктів щодо відстаней та інтервалів беруться з урахуванням особливостей стін та перегородок.

Побудова робочого місця користувача комп'ютерними засобами має забезпечувати зберігання задовільної робочої пози за наступними рекомендованими правилами: підошва – на підлозі або на підставці для ніг стегна – в горизонтальному положенні; передпліччя – вертикально; лікті – під кутом 70°-90° до вертикальної поверхні; зап'ястя зігнуті під кутом не більше 20° відносно горизонтальної поверхні, нахил голови – 20°відносно вертикальної поверхні; монітор персонального комп'ютеру – не менше 60 см від обличчя користувача.

Оскільки інноваційні технології ідуть вперед, деякі галузі потребують роботу з широкоформатними моніторами. У цих випадках відстань у 60см між користувачем та дисплеєм є дуже небезпечною та порушує медичні норми роботи з обчислювальними пристроями. Згідно медичним нормам, оптимальна відстань до монітору повинна складати більше мінімального – від півтора до двох діагоналей.

Клавіатуру необхідно розташовувати на поверхні столу або на спеціальній, регульованій за висотою, робочій площині окремо від столу на відстані 10-30 см від краю, ближчого до працівника. Кут нахилу клавіатури має бути в межах 5°-15°. Розміщення пристрою вводу-виводу інформації (принтер, сканер тощо) на робочому місці не має захаращувати видимість екрану обчислювальної машини, ручне керування пристроєм вводу-виводу інформації має бути в зоні досяжності, а саме 90-130 см у висоті(рахуючи від полу), 40-50 см по глибині. Робочий стілець оператора обчислювальної машини повинен бути оснащений наступними елементами: сидінням, спинкою та стаціонарними або змінними підлокітниками. Стілець може бути додатково обладнаний засобами, що поліпшують умови роботи, а саме: підголовник та підставка для ніг.

Робочий стілець оператора обчислювальної машини повинен регулюватись за висотою, кутом нахилу сидіння та списки, за відстанню спинки до переднього

краю сидіння, висотою підлокітників. Ширина та глибина сидіння повинні бути не меншими за 40 см. Висота поверхні сидіння має регулюватися в межах 40-50 см, а кут нахилу поверхні – від 15° вперед, до 5° назад. Поверхня сидіння має бути плоскою, передній край – заокругленим, або оснащенна спеціальними ортопедичними засобами. Кут нахилу спинки стільця повинен регулюватися в межах 0°-30° відносно вертикального положення. Відстань від спинки до переднього краю сидіння повинна регулюватись у межах 25-40 см.

3.2 Фактори ризику і можливості ураження здоров'я користувачів комп'ютерів

Близько 20% порушень здоров’я, пов’язаних із роботою за комп’ютером, викликано не дією шкідливих чинників, які генерує комп’ютер, а незнанням основних правил роботи з ним та неправильною організацією робочого місця.

Робота за комп’ютером є шкідливою, — це вперше зрозуміли в найбільш комп’ютеризованій країні світу — США. За даними Міністерства nраці цієї країни, одна тільки «травмуюча дія, що повторюється при роботі з комп’ютером, коштує корпоративній Америці в 100 млрд доларів щорічно, а компенсації, виплачені службовцям, які працюють з комп’ютерами, досягають астрономічних розмірів.

Виділимо окремі аспекти негативного впливу на стан здоров’я користувачів персональних комп’ютерів та методи усунення такого впливу.

Сукупність змін, що спостерігаються в стані здоров’я користувачів ЕОМ, включає у себе захворювання органів зору, опорно-рухового апарату, центрально-нервової і серцево-судинної систем, шлунково-кишкового тракту, алергійні розлади. Виконання робіт із застосуванням комп’ютера, також може спричинити розвиток професійних захворювань периферичної нервової системи; захворювання кістково-м’язової системи та сполучної тканини, інші захворювання:

У вагітних відзначається ускладнення вагітності та пологів. У 80% працівників, які виконують роботу, що вимагає постійного напруження зору (а саме такою є робота на комп'ютері), відзначається прогресуюче зниження працездатності, що наступає через 45-60 хв після початку роботи і поступово призводить до перевтоми, розладів центральної нервової та інших систем організму. У другій половині дня (іноді раніше) з'являється втома, головний біль, біль в очах та інші скарги, а швидкість переробки інформації зменшується на 25-34%, знижується стійкість ясного бачення на 40-52%. До кінця робочого дня збільшується частота серцевих скорочень і показники систолічного і діастолічного артеріального тиску.

Вже в перші роки комп'ютеризації було відзначено специфічне зорове стомлення у користувачів ЕОМ, яке дістало назву астенопія. За різними даними , частота випадків астенопії коливається від 40-92% (періодично) до 10-40% (щодня). Деякі симптоми астенопії можуть зберігатися і наступного дня після роботи з комп'ютером. Жінки частіше, ніж чоловіки, скаржаться на зоровий дискомфорт при роботі за комп'ютером. У жінок у віці 31-45 років астенопія виникає частіше, ніж у жінок 18-30 років, що свідчить про вплив стажу. Астенопія часто зустрічається у операторів з введення даних, прийому даних і діалогової роботи. До 47% користувачів скаржаться на зорове стомлення, якщо робота без перерви триває менше 30 хв, і 66% — при безперервній роботі більше 30 хв. Очні симптоми більш виражені у тих осіб, що менше контролюють свою роботу, працюють з великим напруженням і відчувають незадоволеність своєю працею. Більш чутливими до виникнення астенопії є люди з порушеннями зору.

Тривала робота у вимушеній робочій позі сидячи призводить до довготривалого статичного напруження одних і тих самих груп м'язів. Часто користувачі ЕОМ змушені сидіти у сутулій позі, яка характеризується максимальним згинанням у поперековому і грудному відділах хребта та нахилом тулуба вперед. Ця поза має низку несприятливих біомеханічних і фізіологічних наслідків, що призводить до виникнення болю у шийних і поперекових відділах

хребта, остеохондрозу хребта, геморою , простатиту та інших захворювань . У підлітків під час тривалої роботи з комп'ютером може виникати сколіоз.

При виконанні дрібних стереотипних рухів під час роботи з клавіатурою і мишею, м'язи рук користувачів знаходяться в постійному напруженні, як наслідок — виникнення синдрому зап'ясткового каналу.

Перевантаження певних rруп м'язів під час роботи на комп'ютері поєднується з гіпокінезією. При зниженні рівня фізичної активності різко збільшується ризик виникнення багатьох інших захворювань.

Окремо зупинимося на негативному впливі на здоров'я людини змінних електромагнітних полів (ЕМП), зокрема радіочастотного випромінювання при роботі моніторів на основі електронно-променевої трубки (ЕПТ).

Фахівцями Рівненської обласної санітарно-епідеміологічної станції ДСЕС України та Рівненського обласного лабораторного центру МОЗ України протягом 2009-2015 років проводились виміри електромагнітних випромінювань від персональних комп'ютерів в школах області. Досліджено 1171 ПК, біля яких зроблено 2313 вимірів електромагнітного випромінювання, з яких не відповідало нормам 635 вимірів (27,4%). Невідповідності нормам (перевищення до 2-х разів) були зафіксовані біля відеотерміналів на основі ЕПТ, в складі ПК, які не були підключені до контуру заземлення електричного обладнання. Після усунення такого недоліку, повторні виміри рівнів ЕМП не зафіксували відхилень від норми.

Дія електромагнітного випромінювання широкого спектру частот, імпульсного характеру та різної інтенсивності при роботі ЕОМ, у поєднанні з високим зоровим і нервово-емоційним напруженням викликає істотні зміни з боку центральної нервової і серцево-судинної систем, які проявляються у суб'єктивних і об'єктивних розладах. Працюючі з комп'ютером особи найчастіше висловлюють скарги на головний біль, іноді з нудотою і запамороченням. У них частіше, ніж у осіб контрольних груп, діагностуються неврози, нейроциркуляторні дистонії, гіпо- і гіпертонії.

В осіб, що працюють з ЕОМ, можуть спостерігатися алергійні захворювання і підвищений рівень захворюваності органів дихання. Ці захворювання можуть

бути зумовлені змінами імунітету (внаслідок впливу електромагнітного випромінювання на імунну систему), а також, зважаючи на наявність статичних електричних полів, до екрана ВДТ притягуються пилові частинки, які можуть містити антигени і бактеріальну флору. Це також сприяє розвитку вищезгаданої патології в операторів і представників інших професій, що працюють з ВДТ,

Часто виникають психічні порушення, які є наслідком стресу. Частота таких розладів як тривога, дратівливість і пригніченість в операторів ВДТ коливається від 25 до 70%. У них частіше, ніж у представників інших професій, спостерігається безсоння і втрата аппетиту, виникнення захворювань шкіри.

ВИСНОВКИ

Розробка наведеного алгоритму виявилася значущим кроком у напрямку автоматизації процесу збору біологічних зразків. Створений алгоритм відрізняється високою ефективністю та точністю завдяки використанню обширного обсягу навчальних даних та потужних моделей та інструментів.

Використання передових технологій, таких як YOLOv5, MediaPipe Face Mesh, OpenCV та Shapely, надало нам змогу створити систему, яка здатна точно визначати положення та орієнтацію свабу в реальному часі. Це забезпечило високу точність та надійність системи, що має критичне значення для медичних застосувань.

Важливо підкреслити, що розробка таких систем має велике значення не лише для боротьби з COVID-19, але й для різноманітних інших областей. Автоматизовані системи збору біологічних зразків можуть знайти застосування в різних сферах медицини, включаючи діагностику, моніторинг стану здоров'я, наукові дослідження та інші галузі.

Завдяки успішній реалізації цього проекту відкриваються нові перспективи для вдосконалення та розширення автоматизованих систем збору біологічних зразків. Майбутні дослідження можуть включати оптимізацію алгоритмів для роботи з великими обсягами даних, адаптацію системи до різних умов зйомки, а також додавання нових функціональностей, таких як розпізнавання додаткових патогенів чи розширення можливостей медичного моніторингу.

Цей проект може служити відмінною основою для подальших досліджень у галузі медичних технологій та глибокого навчання, внесок у якій може значно поліпшити процеси діагностики та моніторингу в області сучасної медицини.
Узагальнюючи, наша робота ілюструє потенціал використання глибокого навчання та комп'ютерного зору в медицині. Ми сподіваємося, що наші зусилля сприятимуть розвитку більш автоматизованих, точних та ефективних систем збору біологічних зразків.

ПЕРЕЛІК ПОСИЛАНЬ

1. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. Відомості про конференцію з комп'ютерного бачення та розпізнавання образів IEEE. 2016. С. 779-788.

2. Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv preprint arXiv:1704.04861. 2017.

3. Bradski, G. The OpenCV Library. Dr. Dobb's Journal of Software Tools. 2000. С. 120.

4. ISO/IEC 19794-5:2005. Information technology — Biometric data interchange formats — Part 5: Face image data. Женева: ISO, 2005.

5. "YOLOv5: Scalable Object Detection in Image and Video". [Електронний ресурс] URL: https://arxiv.org/abs/2106.09920 (дата звернення: 08.08.2023).

6. "MediaPipe: A Framework for Building Perception Pipelines". [Електронний ресурс] URL: https://google.github.io/mediapipe/ (дата звернення: 08.08.2023).

7. "OpenCV: Open Source Computer Vision Library". [Електронний ресурс] URL: https://opencv.org/ (дата звернення: 08.08.2023).

8. "Shapely: Manipulation and analysis of geometric objects". [Електронний ресурс] URL: https://pypi.org/project/Shapely/ (дата звернення: 08.08.2023).

9. "FastAPI: A modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints". [Електронний ресурс] URL: https://fastapi.tiangolo.com/ (дата звернення: 08.08.2023).

10. "Automated Biological Sample Collection using Convolutional Neural Networks: A Review". [Електронний ресурс] URL: https://www.researchgate.net/publication/334417045_Automated_Biological_Sample_Collection_using_Convolutional_Neural_Networks_A_Review (дата звернення: 08.08.2023).

11. Goodfellow, I., Bengio, Y., & Courville, A. Deep Learning. MIT Press. 2016. 775 с.

12. Krizhevsky, A., Sutskever, I., & Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. Advances in neural information processing systems. 2012. С. 1097-1105.

13. He, K., Zhang, X., Ren, S., & Sun, J. Deep Residual Learning for Image Recognition. Proceedings of the IEEE conference on computer vision and pattern recognition. 2016. С. 770-778.

14. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. Going Deeper with Convolutions. Proceedings of the IEEE conference on computer vision and pattern recognition. 2015. С. 1-9.

15. LeCun, Y., Bengio, Y., & Hinton, G. Deep learning. Nature. 2015. Т. 521. № 7553. С. 436-444.

16. Основи охорони праці: Навч. посіб. / Воронов І.О., Коваленко І.Д., Афанасьєв П.В., Булгач Т.В. – К.: Генеза, 2004. – С.96 – 116

17. Гандзюк М.П., Желібо Є.П., Халімовський М.О. Основи охорони праці.- Київ: Вища освіта в Україні, 2013. – С.200 – 244.

18. В.М. Іванов, О.І. Петренко, С.А. Сидоренко.\ Охорона праці: Навч. посібник для студентів вищих навчальних закладів – К.: Центр учбової літератури, 2017. – С. 120–150.

19. Л.В. Завадська, Л.О. Кузьменко, О.А. Савченко. \ Системи управління охороною праці: Навч. посібник – К.: Літера ЛТД, 2018. – С. 80–105.

20. В.І. Григоренко, О.М. Литвиненко, І.В. Семенов. \ Охорона праці та техніка безпеки в промисловості – К.: Техніка, 2019. – С. 160–185.

ДОДАТКИ

Додаток А – Наукові публікації

Machine learning-based phenotypic imaging to characterise the targetable biology of Plasmodium falciparum male gametocytes for the development of transmission-blocking antimalarials - PubMed (nih.gov)
Access keysNCBI HomepageMyNCBI HomepageMain ContentMain Navigation

. 2023 Oct 6;19(10):e1011711.
doi: 10.1371/journal.ppat.1011711. eCollection 2023 Oct.

Machine learning-based phenotypic imaging to characterise the targetable biology of Plasmodium falciparum male gametocytes for the development of transmission-blocking antimalarials

Oleksiy Tsebriy[1], Andrii Khomiak[1], Celia Miguel-Blanco[2], Penny C Sparkes[3], Maurizio Gioli[4], Marco Santelli[1], Edgar Whitley[5], Francisco-Javier Gamo[2], Michael J Delves[3]
Affiliations expand

- PMID: 37801466
- PMCID: PMC10584170
- DOI: 10.1371/journal.ppat.1011711

Free PMC article

Abstract

Preventing parasite transmission from humans to mosquitoes is recognised to be critical for achieving elimination and eradication of malaria. Consequently developing new antimalarial drugs with transmission-blocking properties is a priority. Large screening campaigns have identified many new transmission-blocking molecules, however little is known about how they target the mosquito-transmissible Plasmodium falciparum stage V gametocytes, or how they affect their underlying cell biology. To respond to this knowledge gap, we have developed a machine learning image analysis pipeline to characterise and compare the cellular phenotypes generated by transmission-blocking molecules during male gametogenesis. Using this approach, we studied 40 molecules, categorising their activity based upon timing of action and visual effects on the organisation of tubulin and DNA within the cell. Our data both proposes new modes of action and corroborates existing modes of action of identified transmission-blocking molecules. Furthermore, the characterised molecules provide a new armoury of tool compounds to probe gametocyte cell biology and the generated imaging dataset provides a new reference for researchers to correlate

molecular target or gene deletion to specific cellular phenotype. Our analysis pipeline is not optimised for a specific organism and could be applied to any fluorescence microscopy dataset containing cells delineated by bounding boxes, and so is potentially extendible to any disease model.

PubMed Disclaimer

Conflict of interest statement

The authors have declared that no competing interests exist.

Figures

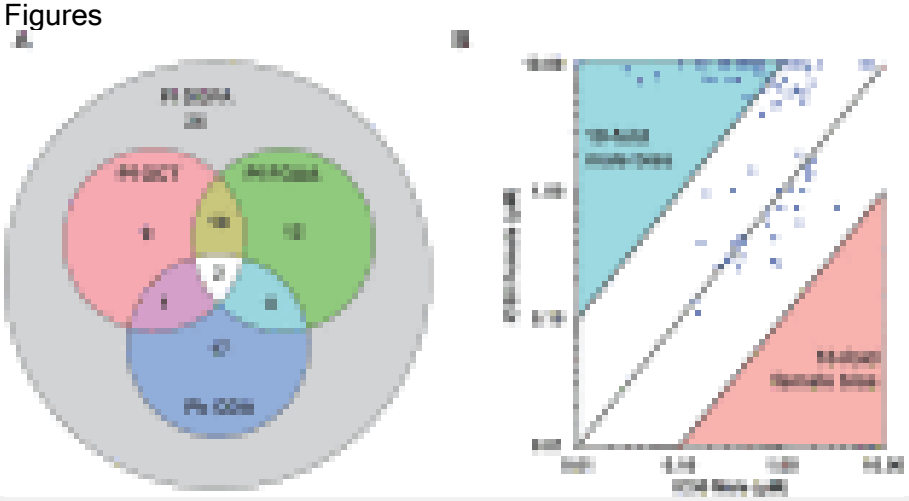


Fig 1. The activity of TCAMS compound…

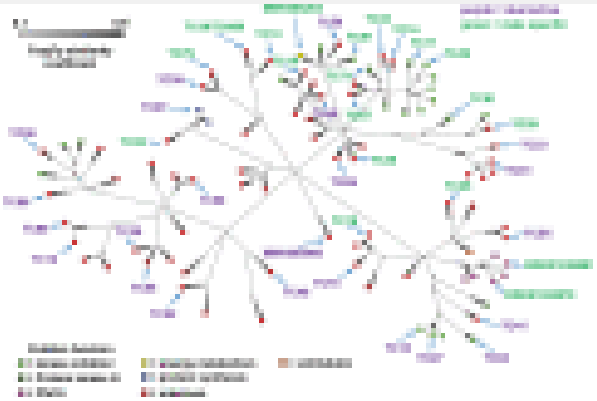


Fig 2. The molecules active against *P*…

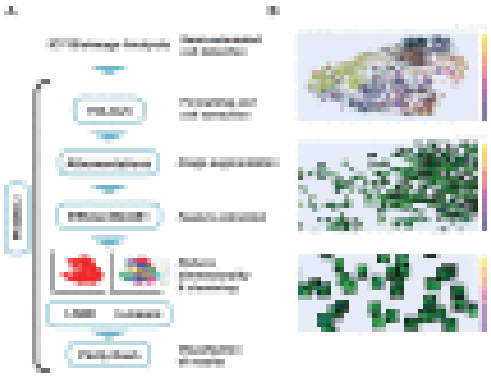


Fig 3. Using phenotypic imaging to cluster…

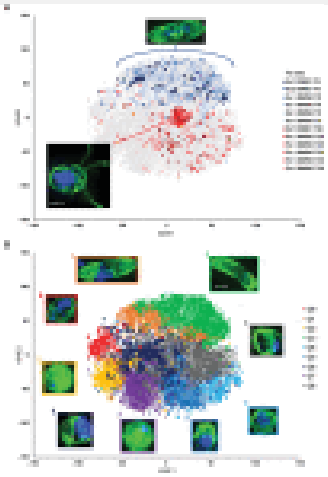


Fig 4. Phenotypic distribution of male gametocytes…

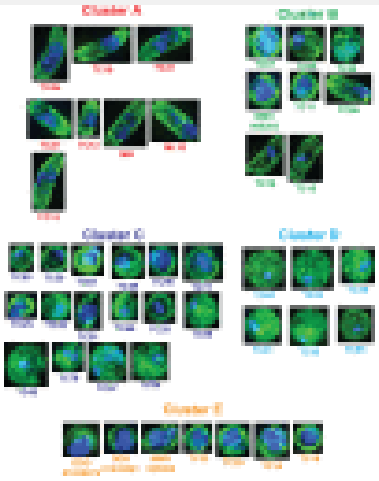


Fig 5. Assignment of drugs to phenotypic…

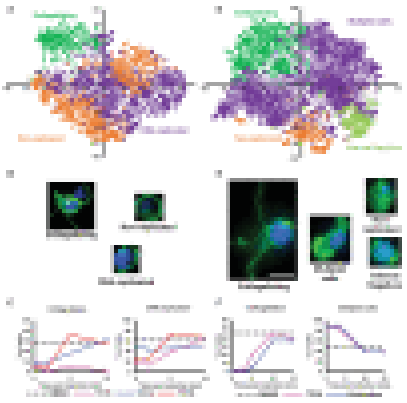


Fig 6. Comparing how male gametogenesis changes…

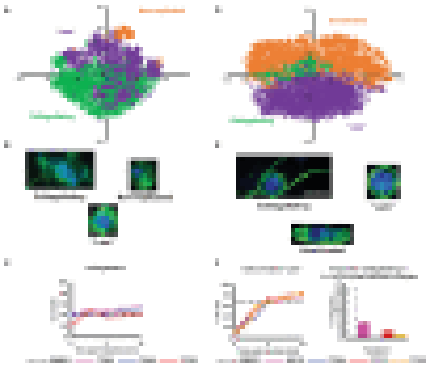


Fig 7. Compounds targeting early male gametogenesis…

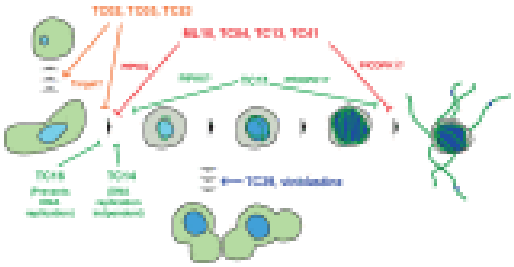


Fig 8. A model summarising the observed…

References

1.
    1. Sinden RE. Gametocytogenesis of Plasmodium falciparum in vitro: an electron microscopic study. Parasitology. 1982. Feb;84(1):1–11. doi: 10.1017/s003118200005160x - DOI - PubMed
2.
    1. Plouffe DM, Wree M, Du AY, Meister S, Li F, Patra K, et al.. High-Throughput Assay and Discovery of Small Molecules that Interrupt Malaria Transmission. Cell Host & Microbe. 2016. Jan;19(1):114–26. doi: 10.1016/j.chom.2015.12.001 - DOI - PMC - PubMed
3.

1. Burrows JN, Duparc S, Gutteridge WE, Hooft van Huijsduijnen R, Kaszubska W, Macintyre F, et al.. New developments in anti-malarial target candidate and product profiles. Malaria Journal. 2017;16:26. doi: 10.1186/s12936-016-1675-x - DOI - PMC - PubMed

4.

1. D'Alessandro S, Camarda G, Corbett Y, Siciliano G, Parapini S, Cevenini L, et al.. A chemical susceptibility profile of the Plasmodium falciparum transmission stages by complementary cell-based gametocyte assays. J Antimicrob Chemother. 2016. May;71(5):1148–58. doi: 10.1093/jac/dkv493 - DOI - PubMed

5.

1. Delves M, Lafuente-Monasterio MJ, Upton L, Ruecker A, Leroy D, Gamo FJ, et al.. Fueling Open Innovation for Malaria Transmission-Blocking Drugs: Hundreds of Molecules Targeting Early Parasite Mosquito Stages. Front Microbiol. 2019. Sep;10:1–10. - PMC - PubMed

6.

1. Delves MJ, Miguel-Blanco C, Matthews H, Molina I, Ruecker A, Yahiya S, et al.. A high throughput screen for next-generation leads targeting malaria parasite transmission. Nature Communications. 2018. Sep;9(1):3805. doi: 10.1038/s41467-018-05777-2 - DOI - PMC - PubMed

7.

1. Miguel-Blanco C, Molina I, Bardera AI, Díaz B, de Las Heras L, Lozano S, et al.. Hundreds of dual-stage antimalarial molecules discovered by a functional gametocyte screen. Nat Commun. 2017. May;8:15160. doi: 10.1038/ncomms15160 - DOI - PMC - PubMed

8.

1. Cowell AN, Istvan ES, Lukens AK, Gomez-Lorenzo MG, Vanaerschot M, Sakata-Kato T, et al.. Mapping the malaria parasite druggable genome by using in vitro evolution and chemogenomics. Science. 2018. Jan;359(6372):191–9. doi: 10.1126/science.aan4472 - DOI - PMC - PubMed

9.

1. Yahiya S, Saunders CN, Hassan S, Straschil U, Fischer OJ, Rueda-Zubiaurre A, et al.. A novel class of sulphonamides potently block malaria transmission by targeting a Plasmodium vacuole membrane protein. Disease Models & Mechanisms. 2023. Jan;16(2):dmm049950. doi: 10.1242/dmm.049950 - DOI - PMC - PubMed

10.

1. Zeeshan M, Ferguson DJ, Abel S, Burrrell A, Rea E, Brady D, et al.. Kinesin-8B controls basal body function and flagellum formation and is key to malaria transmission. Life Sci Alliance. 2019. Aug;2(4). doi: 10.26508/lsa.201900488 - DOI - PMC - PubMed

11.

1. Aulner N, Danckaert A, Ihm J, Shum D, Shorte SL. Next-Generation Phenotypic Screening in Early Drug Discovery for Infectious Diseases. Trends in Parasitology. 2019. Jul 1;35(7):559–70. doi: 10.1016/j.pt.2019.05.004 - DOI - PubMed

12.

1. Alharbi AH, V AC, Lin M, Ashwini B, Jabarulla MY, Shah MA. Detection of Peripheral Malarial Parasites in Blood Smears Using Deep Learning Models. Comput Intell Neurosci. 2022. May;2022:3922763. doi: 10.1155/2022/3922763 - DOI - PMC - PubMed

13.

1. Yu H, Mohammed FO, Abdel Hamid M, Yang F, Kassim YM, Mohamed AO, et al.. Patient-level performance evaluation of a smartphone-based malaria diagnostic application. Malaria Journal. 2023. Jan 27;22(1):33. doi: 10.1186/s12936-023-04446-0 - DOI - PMC - PubMed

14.

1. Yang F, Poostchi M, Yu H, Zhou Z, Silamut K, Yu J, et al.. Deep Learning for Smartphone-Based Malaria Parasite Detection in Thick Blood Smears. IEEE Journal of Biomedical and Health Informatics. 2020. May;24(5):1427–38. doi: 10.1109/JBHI.2019.2939121 - DOI - PubMed

15.

1. Ashdown GW, Dimon M, Fan M, Sánchez-Román Terán F, Witmer K, Gaboriau DCA, et al.. A machine learning approach to define antimalarial drug action from heterogeneous cell-based screens. Science Advances. 2020. Sep;6(39):eaba9338. doi: 10.1126/sciadv.aba9338 - DOI - PMC - PubMed

16.

1. Gamo FJ, Sanz LM, Vidal J, de Cozar C, Alvarez E, Lavandera JL, et al.. Thousands of chemical starting points for antimalarial lead identification. Nature. 2010. May;465(7296):305–10. doi: 10.1038/nature09107 - DOI - PubMed

17.

1. Delves MJ, Straschil U, Ruecker A, Miguel-Blanco C, Marques S, Baum J, et al.. Routine in vitro culture of P. falciparum gametocytes to evaluate novel transmission-blocking interventions. Nat Protocols. 2016. Sep;11(9):1668–80. doi: 10.1038/nprot.2016.096 - DOI - PubMed

18.

1. Almela MJ, Lozano S, Lelièvre J, Colmenarejo G, Coterón JM, Rodrigues J, et al.. A New Set of Chemical Starting Points with Plasmodium falciparum Transmission-Blocking Potential for Antimalarial Drug Discovery. PLoS One. 2015. Aug;10(8):e0135139. doi: 10.1371/journal.pone.0135139 - DOI - PMC - PubMed

19.

1. Delves MJ, Ruecker A, Straschil U, Lelièvre J, Marques S, López-Barragán MJ, et al.. Male and Female Plasmodium falciparum Mature Gametocytes Show Different Responses to Antimalarial Drugs. Antimicrob Agents Chemother. 2013. Jul;57(7):3268–74. doi: 10.1128/AAC.00325-13 - DOI - PMC - PubMed

20.

1. Ruecker A, Mathias DK, Straschil U, Churcher TS, Dinglasan RR, Leroy D, et al.. A male and female gametocyte functional viability assay to identify biologically relevant malaria transmission-blocking drugs. Antimicrob Agents Chemother. 2014. Dec;58(12):7292–302. doi: 10.1128/AAC.03666-14 - DOI - PMC - PubMed

21.

1. Vanaerschot M, Murithi JM, Pasaje CFA, Ghidelli-Disse S, Dwomoh L, Bird M, et al.. Inhibition of Resistance-Refractory P. falciparum Kinase PKG Delivers Prophylactic, Blood Stage, and Transmission-Blocking Antiplasmodial Activity. Cell Chemical Biology. 2020. Jul;27(7):806–816.e8. doi: 10.1016/j.chembiol.2020.04.001 - DOI - PMC - PubMed

22.

1. Crowther GJ, Hillesland HK, Keyloun KR, Reid MC, Lafuente-Monasterio MJ, Ghidelli-Disse S, et al.. Biochemical Screening of Five Protein Kinases from Plasmodium falciparum against 14,000 Cell-Active Compounds. PLoS One. 2016;11(3):e0149996. doi: 10.1371/journal.pone.0149996 - DOI - PMC - PubMed

23.

1. Sinden RE, Canning EU, Spain B. Gametogenesis and Fertilization in Plasmodium Yoelii Nigeriensis: A Transmission Electron Microscope Study. Proc R Soc Lond B. 1976. Mar;193(1110):55–76. doi: 10.1098/rspb.1976.0031 - DOI - PubMed

24.

1. Fletcher S, Avery VM. A novel approach for the discovery of chemically diverse anti-malarial compounds targeting the Plasmodium falciparum Coenzyme A synthesis pathway. Malar J. 2014. Aug;13:343. doi: 10.1186/1475-2875-13-343 - DOI - PMC - PubMed

25.

1. Rocamora F, Gupta P, Istvan ES, Luth MR, Carpenter EF, Kümpornsin K, et al.. PfMFR3: A Multidrug-Resistant Modulator in Plasmodium falciparum. ACS Infect Dis. 2021. Apr;7(4):811–25. doi: 10.1021/acsinfecdis.0c00676 - DOI - PMC - PubMed

26.

1. Baker DA, Stewart LB, Large JM, Bowyer PW, Ansell KH, Jiménez-Díaz MB, et al.. A potent series targeting the malarial cGMP-dependent protein kinase clears infection and blocks transmission. Nature Communications. 2017. Sep;8(1):430. doi: 10.1038/s41467-017-00572-x - DOI - PMC - PubMed

27.

1. Buchholz K, Schirmer RH, Eubel JK, Akoachere MB, Dandekar T, Becker K, et al.. Interactions of Methylene Blue with Human Disulfide Reductases and Their Orthologues from Plasmodium falciparum. Antimicrob Agents Chemother. 2008. Jan;52(1):183–91. doi: 10.1128/AAC.00773-07 - DOI - PMC - PubMed

28.

1. Lasonder E, Rijpma SR, van Schaijk BCL, Hoeijmakers WAM, Kensche PR, Gresnigt MS, et al.. Integrated transcriptomic and proteomic analyses of P. falciparum gametocytes: molecular insight into sex-specific processes and translational repression. Nucleic Acids Res. 2016. Jul;44(13):6087–101. doi: 10.1093/nar/gkw536 - DOI - PMC - PubMed

29.

1. Straschil U, Talman AM, Ferguson DJP, Bunting KA, Xu Z, Bailes E, et al.. The Armadillo Repeat Protein PF16 Is Essential for Flagellar Structure and Function

in Plasmodium Male Gametes. PLoS One 2010. Sep;5(9):e12901. doi: 10.1371/journal.pone.0012901 - DOI - PMC - PubMed

30.

1. Sander T, Freyss J, von Korff M, Rufener C. DataWarrior: An Open-Source Program For Chemistry Aware Data Visualization And Analysis. J Chem Inf Model. 2015. Feb;55(2):460–73. doi: 10.1021/ci500588j - DOI - PubMed

Додаток Б – Код проекту

З скрипта green_circle_demo_fastapi_ellipse.py як останньої версії, код сходиться в більшості з кодом демо скрипта green_circle_demo.py, по якому було зняте відео, за виключенням розбивки на функції.

Отримання зображень з камери, використовуючи OpenCV:

```
cap = cv2.VideoCapture(0) while
cap.isOpened():

        # Capture frame-by-frame
        success, image = cap.read()


        # Flip the image horizontally for a later selfie-view display #
        Also convert the color space from BGR to RGB

        image = cv2.cvtColor(cv2.flip(image, 1), cv2.COLOR_BGR2RGB)

        # Convert the color space from RGB to BGR image
        = cv2.cvtColor(image, cv2.COLOR_RGB2BGR) img_h,
        img_w, img_c = image.shape
```

Визначення кінця свабу, використовуючи власну функцію - def detect_swab_end(image) Алгоритм функції та лістинг:

Функція отримує на вхід початкове зображення зчитане з камери

Використання Object Detection моделі, яка шукає сваб на зображенні, та обрізання картинки до зони інтересу:

```
result = yolo_model(image).xyxy[0].tolist() x1
= int(result[0][0])

y1 = int(result[0][1]) x2
= int(result[0][2]) y2 =
int(result[0][3])

cropped = image[y1:y2, x1:x2]
```

1. Визначення контурів та ліній з допомогою методів Canny та HoughLinesP для пошуку кінця свабу:

```
        edges = cv2.Canny(cropped, 0, 90, apertureSize=3)
```

```
        lines = cv2.HoughLinesP(image=edges, rho=0.8, theta=3.1415926 / 90,
threshold=30,
                                lines=None, minLineLength=10, maxLineGap=25)
```

2. Пошук координат початку та кінця свабу з усіх знайдених ліній та обрахунок довжини свабу:

```
            for line in lines:

                # detecting swab upper end if
                line[0][1] < y_min:

                    x_y_min, y_min = line[0][0], line[0][1]


                if line[0][3] < y_min:

                    x_y_min, y_min = line[0][2], line[0][3]


                # detecting swab lower end if
                line[0][1] > y_max:

                    x_y_max, y_max = line[0][0], line[0][1]


                if line[0][3] > y_max:

                    x_y_max, y_max = line[0][2], line[0][3]

            # converting box coordinates to image coordinates x_swab,
            y_swab = x1 + x_y_min, y1  # upper point

            x_sw_end, y_sw_end = x1 + x_y_max, y1 + y_max  # lower point


            # nearest  yolo box coords - optional improvement x_swab
            = x1 if x_swab - x1 < x2 - x_swab else x2


            # calculating swab length

            swab_length = int(((x_swab - x_sw_end) ** 2 + (y_swab - y_sw_end)
** 2) ** 0.5)
```

Функція повертає координати початку та кінця свабу і його довжину

Визначення обличчя та його лендмарки, використовуючи фреймворк MediaPipe Face Mesh, і перевірка його наявності разом з попередженням у разі відсутності лиця:

```
face_mesh = mp.solutions.face_mesh.FaceMesh(
    min_detection_confidence=0.5, min_tracking_confidence=0.5)
```

Визначення напряму погляду, використовуючи власну функцію - detect_face_position(image, landmarks)

Алгоритм функції та лістинг:

Функція отримує початкове зображення зчитане з камери та лендмарки лиця

Пошук координат лендмарків носу:

```
face_landmarks = landmarks[0]
for idx, lm in enumerate(face_landmarks.landmark):

     if idx in {33, 263, 1, 61, 291, 199}:  # NOSE_VECTOR_POINTS
        if idx == 1:
            nose_2d = (lm.x * img_w, lm.y * img_h)
            nose_3d = (lm.x * img_w, lm.y * img_h, lm.z * 3000)
        x, y = int(lm.x * img_w), int(lm.y * img_h) #
        Get the 2D Coordinates face_2d.append([x,
        y])

        # Get the 3D Coordinates
        face_3d.append([x, y, lm.z])
```

Обрахунок вектору напряму обличчя, використовуючи методи solvePnP, Rodrigues та RQDecomp3x3 з бібліотеки OpenCV для проектування 2D координат носу в 3D

```
# Convert it to the NumPy array
face_2d = np.array(face_2d, dtype=np.float64)

# Convert it to the NumPy array
face_3d = np.array(face_3d, dtype=np.float64)

# Solve PnP
success, rot_vec, trans_vec = cv2.solvePnP(
    face_3d, face_2d, cam_matrix, dist_matrix)

# Get rotational matrix
rmat, jac = cv2.Rodrigues(rot_vec)

# Get angles
angles, mtxR, mtxQ, Qx, Qy, Qz = cv2.RQDecomp3x3(rmat)
```

```
        # Get the y rotation degree x
        = angles[0] * 360

        y = angles[1] * 360

        z = angles[2] * 360
```

1. Функція повертає координати вектору напряму обличчя

2) Перевірка напряму лиця та попередження в разі спрямування погляду в іншу сторону:

```
        if y < -10:

            if x > 10:

                text = "Looking Left Up" elif
            x < -7:

                text = "Looking Left Down"
            else:

                text = "Looking Left" elif
        y > 10:

            if x > 10:

                text = "Looking Right Up" elif
            x < -7:

                text = "Looking Right Down"
            else:

                text = "Looking Right"

        else:

            if x < 0:

                text = "Looking Down" elif
            x > 20:

                text = "Looking Up"
            else:

                text = "Forward"


    if text != "Forward" and text.strip():

        cv2.putText(image, "Please, keep your head straight", (40,

                    img_h-50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 255), 2)

        cv2.putText(image, "and a little bit to the up to detect nostrils", (40,
img_h-25),

                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 255), 2)

        swab_length_0 = 0  # face is lost, setting initial swab and nose lengths
to zero

        nose_length_0 = 0
```

Визначення полігонів ніздрі, використовуючи власну функцію - detect_nostrils_ellipses(image, landmarks)

Алгоритм функції та лістинги:

2. Функція отримує початкове зображення з камери та лендмарки лиця

Пошук лендмарків ніздрі та збір координат:

```
for idx, lm in enumerate(face_landmarks.landmark): if
    idx in LEFT_NOSE:

        x, y = int(lm.x * img_w), int(lm.y * img_h)
        left_naris_poly_coords[LEFT_NOSE.index(idx)] = (x, y)


    if idx in RIGHT_NOSE:

        x, y = int(lm.x * img_w), int(lm.y * img_h)
        right_naris_poly_coords[RIGHT_NOSE.index(idx)] = (x, y)
```

Створення полігону ніздрі у вигляді обрахунку еліпса:

```
        # Calculating nose points distance

        nose_length = int(((nose_p2x - nose_p1x) ** 2 + (nose_p2y - nose_p1y)
** 2) ** 0.5)

        # calculating central ellipse coordinates

        ellipse_lx, ellipse_ly = int((nose_p1x + nose_p3x) / 2), int((nose_p1y
+ nose_p3y) / 2)

        ellipse_rx, ellipse_ry = int((nose_p1x + nose_p4x) / 2), int((nose_p1y
+ nose_p4y) / 2)


        # calculating ellipse horizontal and vertical axes length AXIS_X_COEFF
        = 0.5  # ellipse x axis radius, related to length between
nose points

        AXIS_Y_COEFF = 0.5  # ellipse y axis radius, related to x axis radius


        axis_leftx = int(AXIS_X_COEFF * (((nose_p1x - nose_p3x) ** 2 +
(nose_p1y - nose_p3y) ** 2) ** 0.5))

        axis_lefty = int(AXIS_Y_COEFF * axis_leftx)


        axis_rightx = int(AXIS_X_COEFF * (((nose_p1x - nose_p4x) ** 2 +
(nose_p1y - nose_p4y) ** 2) ** 0.5))

        axis_righty = int(AXIS_Y_COEFF * axis_rightx)
```

```
    left_naris_ellipse = (ellipse_lx, ellipse_ly, axis_leftx, axis_lefty)
    right_naris_ellipse = (ellipse_rx, ellipse_ry, axis_rightx, axis_righty)
```

1. Функція повертає отримані еліпси ніздрів

3) Визначення вставлення свабу в ніздрю:

```
    INSERTION_RATE = 0.5  # swab insertion depth (we need ~2.5cm), relative to nose
length

    FILTERING_COEFF = 0.2  # noise filtering coefficient(addition per frame)


    if swab_length and nose_length:

        # initial swab 2d projection length measuring if
        nose_length_0 == 0 and swab_length_0 == 0:

            swab_length_0, nose_length_0 = swab_length, nose_length
            swab_inserted_dist = 0

        else:

            # filtering noise

            swab_inserted_dist = swab_inserted_dist * (1 - FILTERING_COEFF) \

                                 + (swab_length_0 / nose_length_0 - swab_length
/ nose_length) * FILTERING_COEFF

        # checking complete insertion

        insertion_completed = swab_inserted_dist > INSERTION_RATE else:

        insertion_completed = False
```

Код файлу augment.py

```
from random import random

import albumentations as A

import cv2

import glob

import os

import argparse

import numpy as np

import tqdm

import shutil

img_formats = {'bmp', 'jpg', 'jpeg', 'png',

               'tif', 'tiff', 'dng', 'webp', 'mpo'}
```

```
aug = dict(downscale=A.Downscale(scale_min=0.45, scale_max=0.75,
always_apply=True),

           gauss_noise=A.GaussNoise(var_limit=(115, 280),
always_apply=True),

           brightness=A.RandomBrightness(

               always_apply=True, limit=(-0.30, 0.30)),

           contrast=A.RandomContrast(always_apply=True, limit=(-
0.30, 0.30)),

           rotate=A.Rotate(always_apply=True, limit=(-90, 90),
interpolation=0,

                           border_mode=1, value=(0, 0, 0),
mask_value=None),

           shift_scale_r=A.ShiftScaleRotate(always_apply=True,
shift_limit=(-0.08, 0.08), scale_limit=(-0.1, 0.1),

                                            rotate_limit=(-36, 36),
interpolation=0, border_mode=1, value=(0, 0, 0), mask_value=None))

flip = A.Compose([ A.HorizontalFlip(always_apply=True)],

 bbox_params=A.BboxParams(format='yolo',
label_fields=['class_labels']))

aug = dict(

    downscale=A.Compose([

        A.Downscale(scale_min=0.45, scale_max=0.55,
always_apply=True)

    ], bbox_params=A.BboxParams(format='yolo',
label_fields=['class_labels'])),

    gauss_noise=A.Compose([A.Blur(p=0.4),

                           A.GaussNoise(var_limit=(115, 280),
always_apply=True)],

                          bbox_params=A.BboxParams(format='yolo',
label_fields=['class_labels'])),

    brightness=A.Compose([A.RandomBrightness(always_apply=True,
limit=(-0.30, 0.30))],

                         bbox_params=A.BboxParams(format='yolo',
label_fields=['class_labels'])),

    contrast=A.Compose([A.RandomContrast(always_apply=True, limit=(-
0.30, 0.30))],

                       bbox_params=A.BboxParams(format='yolo',
label_fields=['class_labels'])),

    rotate=A.Compose([A.Rotate(always_apply=True, limit=(-90, 90),
interpolation=0, border_mode=1, value=(0, 0, 0), mask_value=None)],
```

```
                    bbox_params=A.BboxParams(format='yolo',
label_fields=['class_labels'])),

    shift_scale_r=A.Compose([A.ShiftScaleRotate(always_apply=True,
shift_limit=(-0.08, 0.08), scale_limit=(-0.1, 0.1),

                                                rotate_limit=(-36,
36), interpolation=0, border_mode=1, value=(0, 0, 0),
mask_value=None)],

                              bbox_params=A.BboxParams(format='yolo',
label_fields=['class_labels']))

)

def convert_bboxes(bboxes):

    res = list()

    for bbox in bboxes:

        x, y, w, h = bbox

        w_half, h_half = w / 2, h / 2

        x_min = x - w_half

        y_min = y - h_half

        x_max = x_min + w

        y_max = y_min + h

        bbox = np.array([x_min, y_min, x_max, y_max])

        if any((bbox < 0) | (bbox > 1)) and w > 0 and h > 0:

            w -= 0.000001

            h -= 0.000001

        if w <= 0:

            w = 0.0000001

        if h <= 0:

            h = 0.0000001

        res.append([x, y, w, h])

    return res

def process(path, labels, transform):

    image = cv2.imread(path)

    # image = cv2.resize(image, (1280, 720))

    # image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    bboxes = convert_bboxes(labels[:, 1:])

    transformed = transform(image=image, bboxes=bboxes,
```

```
                                class_labels=labels[:, 0].astype(int))
    transformed_bboxes = np.array(transformed['bboxes']).round(7)
    transformed_class_labels = transformed['class_labels']
    if random() < 0.4:
        transformed = flip(image=transformed['image'],
bboxes=transformed_bboxes, class_labels=transformed_class_labels)
    transformed_bboxes = np.array(transformed['bboxes']).round(7)
    transformed_class_labels = transformed['class_labels']
    transformed_labels = np.hstack(
        (np.array(transformed_class_labels).reshape(-1, 1),
transformed_bboxes))
    return transformed['image'], transformed_labels
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--origin', type=str,
                        help='root dataset folder', required=True)
    parser.add_argument('--dest', type=str,
                        help='destination dataset folder',
required=True)
    opt = parser.parse_args()
    train_path = opt.origin
    results_path = opt.dest
    img_path = "images"
    label_path = "labels"
    if not os.path.exists(results_path):
        os.mkdir(results_path)
    if not os.path.exists(os.path.join(results_path, img_path)):
        os.mkdir(os.path.join(results_path, img_path))
    if not os.path.exists(os.path.join(results_path, label_path)):
        os.mkdir(os.path.join(results_path, label_path))
    for path in tqdm.tqdm(glob.glob(os.path.join(train_path,
img_path) + "/*")):
        if path.split('.')[-1] not in img_formats:
            continue
        name = path.split('/')[-1]
```

```
        name = ".".join(name.split(".")[:-1])
        if not os.path.exists(os.path.join(train_path, label_path,
name + '.txt')):
            print("DOESN'T EXIST", name)
            continue
        labels = open(os.path.join(train_path, label_path,
                    name + ".txt"), "r").read().split('\n')
        labels = np.array([line.split()
                           for line in labels if line.strip()],
dtype='float')
        for augm, transform in aug.items():
            transf_img, transf_labels = process(path, labels,
transform)
            cv2.imwrite(os.path.join(results_path, img_path,
                        name + '_' + augm + '.jpeg'), transf_img)
            np.savetxt(os.path.join(results_path, label_path, name +
'_' + augm + '.txt'), transf_labels,
                       fmt=' '.join(['%i'] + ['%1.7f']*4) if
labels.size != 0 else '%.4f')
#     if os.path.exists(os.path.join(train_path, "train.txt")):
#         shutil.copyfile(os.path.join(train_path, "train.txt"),
os.path.join(results_path, "train.txt"))
#     if os.path.exists(os.path.join(train_path, "validation.txt")):
#         shutil.copyfile(os.path.join(train_path,
"validation.txt"), os.path.join(results_path, "validation.txt"))
#     if os.path.exists(os.path.join(train_path, "test.txt")):
#         shutil.copyfile(os.path.join(train_path, "test.txt"),
os.path.join(results_path, "test.txt"))
#     with open(os.path.join(results_path, "soccer_augm.yaml"), "w")
as f:
#         f.write(f"train: {os.path.join(results_path,
'train.txt')}\n")
#         f.write(f"val: {os.path.join(results_path,
'validation.txt')}\n")
#         f.write("nc: 2\n")
#         f.write("names: [ 'football_goal', 'soccer_ball' ]\n\n")
```

Код файлу green_circle_demo.py

```
import cv2
import mediapipe as mp
import numpy as np
import time
import torch
# from flask import Flask, Response, render_template
# for detecting point inside polygon
from shapely.geometry import Point, Polygon
model = torch.hub.load('yolov5', 'custom',
                       source='local', path='yolov5n_50.pt')
                    #    source='local', path='swab_v2.pt')
# app = Flask(__name__)
def ResizeWithAspectRatio(image, width=None, height=None,
inter=cv2.INTER_AREA):
    dim = None
    (h, w) = image.shape[:2]
    if width is None and height is None:
        return image
    if width is None:
        r = height / float(h)
        dim = (int(w * r), height)
    else:
        r = width / float(w)
        dim = (width, int(h * r))
    return cv2.resize(image, dim, interpolation=inter)
mp_face_mesh = mp.solutions.face_mesh
face_mesh = mp_face_mesh.FaceMesh(
    min_detection_confidence=0.5, min_tracking_confidence=0.5)
LEFT_NOSE = [242, 99, 240, 235, 219, 218, 237, 241]
RIGHT_NOSE = [462, 328, 460, 455, 439, 438, 457, 461]
cap = cv2.VideoCapture(0)
DEBUG = False
sensitivity = 90
```

```
# OpenCV white color config
lower_white = np.array([0, 0, 255-sensitivity])
upper_white = np.array([255, sensitivity, 255])
NOSE_VECTOR_POINTS = {33, 263, 1, 61, 291, 199}
IMG_W, IMG_H = 1280, 720
# The camera matrix
focal_length = 1 * IMG_W
cam_matrix = np.array([[focal_length, 0, IMG_H / 2],
                       [0, focal_length, IMG_W / 2],
                       [0, 0, 1]])
# The distortion parameters
dist_matrix = np.zeros((4, 1), dtype=np.float64)
step_length_time = 10   # time for one validation step
step_length_time = 10   # time for one validation step
result_queue = [False]*30
while cap.isOpened():
    success, image = cap.read()
    start = time.time()
    # Flip the image horizontally for a later selfie-view display
    # Also convert the color space from BGR to RGB
    image = cv2.cvtColor(cv2.flip(image, 1), cv2.COLOR_BGR2RGB)
    # To improve performance
    image.flags.writeable = False
    # Get the result
    results = face_mesh.process(image)
    # To improve performance
    image.flags.writeable = True
    # Convert the color space from RGB to BGR
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    img_h, img_w, img_c = image.shape
    if img_h != IMG_H:
        IMG_H = img_h
        cam_matrix = np.array([[focal_length, 0, IMG_H / 2],
                               [0, focal_length, IMG_W / 2],
```

```
                            [0, 0, 1]])
    if img_w != IMG_W:
        IMG_W = img_w
        focal_length = 1 * IMG_W
        cam_matrix = np.array([[focal_length, 0, IMG_H / 2],
                            [0, focal_length, IMG_W / 2],
                            [0, 0, 1]])
    face_3d = []
    face_2d = []
    result = model(image).xyxy[0].tolist()
    y_start = 130
    if result:
        x1 = int(result[0][0])
        y1 = int(result[0][1])
        x2 = int(result[0][2])
        y2 = int(result[0][3])
        cropped = image[y1:y2, x1:x2]
        hsv = cv2.cvtColor(cropped, cv2.COLOR_BGR2HSV)
        # Threshold the HSV image to get only white colors
        mask = cv2.inRange(hsv, lower_white, upper_white)
        # Bitwise-AND mask and original image
        masked = cv2.bitwise_and(cropped, cropped, mask=mask)
        masked = cv2.cvtColor(masked, cv2.COLOR_BGR2RGB)
        dst = cv2.fastNlMeansDenoisingColored(masked, None, 10, 3,
9, 15)
        gray = cv2.cvtColor(dst, cv2.COLOR_BGR2GRAY)
        edges = cv2.Canny(gray, 0, 90, apertureSize=3)
        lines = cv2.HoughLinesP(image=edges, rho=0.8,
theta=np.pi/90, threshold=30,
                                lines=np.array([]),
minLineLength=15, maxLineGap=25)
        y_min = y2-y1  # initializing min y as box height
        x_y_min = x2-x1
        if lines is not None:
            for i in range(lines.shape[0]):
```

```
                if DEBUG:
                    cv2.line(image, (x1+lines[i][0][0],
y1+lines[i][0][1]), (x1 +
                             lines[i][0][2], y1+lines[i][0][3]), (0,
255, 0), 1, cv2.LINE_AA)
                    cv2.putText(image, "x: " + str(np.round(x, 2) +
x1),
                                (500, y_start),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
                    cv2.putText(image, "y: " + str(np.round(y, 2) +
y1), (500,
                                y_start+30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
                    y_start += 50
                if lines[i][0][1] < y_min:
                    y_min = lines[i][0][1]
                    x_y_min = lines[i][0][0]
                if lines[i][0][3] < y_min:
                    y_min = lines[i][0][3]
                    x_y_min = lines[i][0][2]
            x_swab = x1+x_y_min
            y_swab = y1+y_min
            if DEBUG:
                cv2.putText(image, '.', (x_swab, y_swab),
cv2.FONT_HERSHEY_PLAIN,
                            0.5, (0, 255, 255), 3)  # swab upper end
dot
        if DEBUG:
            cv2.rectangle(image, (x1, y1), (x2, y2), (0, 255, 0))
    # init empty naris polygone coord lists
    left_naris_poly_coords = [None] * len(LEFT_NOSE)
    right_naris_poly_coords = [None] * len(RIGHT_NOSE)
    left_nostril_swab = False
    right_nostril_swab = False
    text = ''
    res = False
    if results.multi_face_landmarks:
```

```
        face_landmarks = results.multi_face_landmarks[0]
        for idx, lm in enumerate(face_landmarks.landmark):
            if idx in LEFT_NOSE:
                x, y = int(lm.x * img_w), int(lm.y * img_h)
                left_naris_poly_coords[LEFT_NOSE.index(idx)] = (x,
y)
                if DEBUG:
                    cv2.putText(image, '.', (x,y),
cv2.FONT_HERSHEY_PLAIN, 0.5, (255,0,0), 1)
            if idx in RIGHT_NOSE:
                x, y = int(lm.x * img_w), int(lm.y * img_h)
                right_naris_poly_coords[RIGHT_NOSE.index(idx)] = (x,
y)
                if DEBUG:
                    cv2.putText(image, '.', (x,y),
cv2.FONT_HERSHEY_PLAIN, 0.5, (255,0,0), 1)
            if idx in NOSE_VECTOR_POINTS:
                if idx == 1:
                    nose_2d = (lm.x * img_w, lm.y * img_h)
                    nose_3d = (lm.x * img_w, lm.y * img_h, lm.z *
3000)
                x, y = int(lm.x * img_w), int(lm.y * img_h)
                # Get the 2D Coordinates
                face_2d.append([x, y])
                # Get the 3D Coordinates
                face_3d.append([x, y, lm.z])
        # Convert it to the NumPy array
        face_2d = np.array(face_2d, dtype=np.float64)
        # Convert it to the NumPy array
        face_3d = np.array(face_3d, dtype=np.float64)
        # Solve PnP
        success, rot_vec, trans_vec = cv2.solvePnP(
            face_3d, face_2d, cam_matrix, dist_matrix)
        # Get rotational matrix
        rmat, jac = cv2.Rodrigues(rot_vec)
```

```
        # Get angles
        angles, mtxR, mtxQ, Qx, Qy, Qz = cv2.RQDecomp3x3(rmat)
        # Get the y rotation degree
        x = angles[0] * 360
        y = angles[1] * 360
        z = angles[2] * 360
        # See where the user's head tilting
        if y < -10:
            if x > 10:
                text = "Looking Left Up"
            elif x < -7:
                text = "Looking Left Down"
            else:
                text = "Looking Left"
        elif y > 10:
            if x > 10:
                text = "Looking Right Up"
            elif x < -7:
                text = "Looking Right Down"
            else:
                text = "Looking Right"
        else:
            if x < 0:
                text = "Looking Down"
            elif x > 20:
                text = "Looking Up"
            else:
                text = "Forward"
    else:
        cv2.putText(image, "No face detected", (40,
                    img_h-50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255,
255), 2)
    # Detecting swab position inside nostrils
```

```
    if left_naris_poly_coords and right_naris_poly_coords and result
and lines is not None:
        # constructing polygons
        left_naris_poly = Polygon(left_naris_poly_coords)
        right_naris_poly = Polygon(right_naris_poly_coords)
        # detecting swab inside nostrils
        left_nostril_swab = Point(
            x_swab, y_swab).within(left_naris_poly)
        right_nostril_swab = Point(
            x_swab, y_swab).within(right_naris_poly)
        if DEBUG:
            cv2.putText(image, "Left naris swab:"
+str(left_nostril_swab), (100, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,
0, 255), 2)
            cv2.putText(image, "Right naris
swab:"+str(right_nostril_swab), (100, 100),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
    if text != "Forward" and text.strip():
        cv2.putText(image, "Please, keep your head straight", (40,
                    img_h-50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255,
255), 2)
        cv2.putText(image, "and a little bit to the up to detect
nostrils", (40, img_h-25),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 255), 2)
    elif text.strip():
        if not result:
            cv2.putText(image, "Please, take the swab", (40, img_h-
50),
                        cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 255),
2)
        else:
            if lines is None:
                cv2.putText(image, "Can't find swab ending", (40,
img_h-50),
                            cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255,
255), 2)
                cv2.putText(image, "please use brighter light", (40,
img_h-25),
```

```
                            cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255,
255), 2)
            elif left_nostril_swab or right_nostril_swab:
                res = True
            else:
                cv2.putText(image, "Please, insert the swab", (40,
img_h-50),
                            cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255,
255), 2)
                cv2.putText(image, "into your nostril", (40, img_h-
25),
                            cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255,
255), 2)
    result_queue = result_queue[1:] + [res]
    if sum(result_queue) > 20:
        cv2.circle(image, (x_swab, y_swab), 20, (0, 255, 0), 5)
    # Display the nose direction
#   nose_3d_projection, jacobian = cv2.projectPoints(nose_3d,
rot_vec, trans_vec, cam_matrix, dist_matrix)
#   p1 = (int(nose_2d[0]), int(nose_2d[1]))
#   p2 = (int(nose_2d[0] + y * 10) , int(nose_2d[1] - x * 10))
#   cv2.line(image, p1, p2, (255, 0, 0), 3)
    # Add the text on the image
    # cv2.putText(image, text, (40, img_h-50),
cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 255, 0), 2)
#   cv2.putText(image, "x: " + str(np.round(x,2)), (500, 50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
#   cv2.putText(image, "y: " + str(np.round(y,2)), (500, 100),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
        end = time.time()
        totalTime = end - start
    image = ResizeWithAspectRatio(image, width=1024)
    cv2.imshow('Green circle', image)


    k = cv2.waitKey(5)
    if  k == 27:
        break
```

```
    elif k in {ord('p'), 32}:
        while cv2.waitKey(0) not in {ord('p'), 32}:
            continue
cap.release()
```

Код файлу \green_circle_ellipse\ green_circle_demo_fastapi_ellipse.py

```
import cv2
import mediapipe as mp
import numpy as np  # just for detect_face_position function
import torch
import uvicorn
from fastapi import FastAPI, Request
from fastapi.templating import Jinja2Templates
from fastapi.responses import StreamingResponse
app = FastAPI()
templates = Jinja2Templates(directory="templates")
yolo_model = torch.hub.load('/home/work/Work/Construction/yolov5',
'custom',
                            source='local', path='yolov5n_50.pt')
face_mesh = mp.solutions.face_mesh.FaceMesh(
    min_detection_confidence=0.5, min_tracking_confidence=0.5)
cap = cv2.VideoCapture(0)
DEBUG = False
def ResizeWithAspectRatio(image, width=None, height=None,
inter=cv2.INTER_AREA):
    dim = None
    (h, w) = image.shape[:2]
    if width is None and height is None:
        return image
    if width is None:
        r = height / float(h)
        dim = (int(w * r), height)
    else:
        r = width / float(w)
```

```
        dim = (width, int(h * r))
    return cv2.resize(image, dim, interpolation=inter)
def detect_swab_end(image):
    x_swab, y_swab, swab_length = None, None, None
    result = yolo_model(image).xyxy[0].tolist()
    if result:
        x1 = int(result[0][0])
        y1 = int(result[0][1])
        x2 = int(result[0][2])
        y2 = int(result[0][3])
        cropped = image[y1:y2, x1:x2]
        edges = cv2.Canny(cropped, 0, 90, apertureSize=3)
        lines = cv2.HoughLinesP(image=edges, rho=0.8,
theta=3.1415926 / 90, threshold=30,
                                lines=None, minLineLength=10,
maxLineGap=25)
        # searching for swab upper and lower endpoints
        x_y_min, y_min = x2 - x1, y2 - y1  # initializing min x and
y as box width and height
        x_y_max, y_max = 0, 0  # initializing max x and y as zero
        if lines is not None:
            for line in lines:
                if DEBUG:
                    cv2.line(image, (x1 + line[0][0], y1 +
line[0][1]), (x1 +

line[0][2], y1 + line[0][3]), (0, 255, 0), 1,
                             cv2.LINE_AA)
                # detecting swab upper end
                if line[0][1] < y_min:
                    x_y_min, y_min = line[0][0], line[0][1]
                if line[0][3] < y_min:
                    x_y_min, y_min = line[0][2], line[0][3]
                # detecting swab lower end
                if line[0][1] > y_max:
```

```
                    x_y_max, y_max = line[0][0], line[0][1]
                if line[0][3] > y_max:
                    x_y_max, y_max = line[0][2], line[0][3]
            # converting box coordinates to image coordinates
            x_swab, y_swab = x1 + x_y_min, y1  # upper point
            x_sw_end, y_sw_end = x1 + x_y_max, y1 + y_max  # lower
point
            # nearest  yolo box coords - optional improvement
            x_swab = x1 if x_swab - x1 < x2 - x_swab else x2
            # calculating swab length
            swab_length = int(((x_swab - x_sw_end) ** 2 + (y_swab -
y_sw_end) ** 2) ** 0.5)
            if DEBUG:
                cv2.putText(image, '.', (x_swab, y_swab),
cv2.FONT_HERSHEY_PLAIN,
                            0.5, (0, 255, 255), 3)  # swab upper end
dot
                cv2.putText(image, '.', (x_sw_end, y_sw_end),
cv2.FONT_HERSHEY_PLAIN,
                            0.5, (255, 255, 0), 3)  # swab lower end
dot
                cv2.putText(image, f'Swab length {swab_length}',
(50, 100),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255,
255), 2)
        if DEBUG:
            cv2.rectangle(image, (x1, y1), (x2, y2), (0, 255, 0))
    return x_swab, y_swab, swab_length
def detect_face_position(image, landmarks):
    # outptut face angle coordinates
    x, y, z = None, None, None
    img_h, img_w, img_c = image.shape
    IMG_W, IMG_H = 1280, 720
    focal_length = 1 * IMG_W
    # The camera matrix
    cam_matrix = np.array([[focal_length, 0, IMG_H / 2],
                           [0, focal_length, IMG_W / 2],
```

```
                            [0, 0, 1]])
    # The distortion parameters
    dist_matrix = np.zeros((4, 1), dtype=np.float64)
    img_h, img_w, img_c = image.shape
    if img_h != IMG_H:
        IMG_H = img_h
        cam_matrix = np.array([[focal_length, 0, IMG_H / 2],
                               [0, focal_length, IMG_W / 2],
                               [0, 0, 1]])
    if img_w != IMG_W:
        IMG_W = img_w
        focal_length = 1 * IMG_W
        cam_matrix = np.array([[focal_length, 0, IMG_H / 2],
                               [0, focal_length, IMG_W / 2],
                               [0, 0, 1]])
        face_3d = []
        face_2d = []
    if landmarks:
        face_landmarks = landmarks[0]
        for idx, lm in enumerate(face_landmarks.landmark):
            if idx in {33, 263, 1, 61, 291, 199}:  #
NOSE_VECTOR_POINTS
                if idx == 1:
                    nose_2d = (lm.x * img_w, lm.y * img_h)
                    nose_3d = (lm.x * img_w, lm.y * img_h, lm.z *
3000)
                x, y = int(lm.x * img_w), int(lm.y * img_h)
                # Get the 2D Coordinates
                face_2d.append([x, y])
                # Get the 3D Coordinates
                face_3d.append([x, y, lm.z])
        # Convert it to the NumPy array
        face_2d = np.array(face_2d, dtype=np.float64)
        # Convert it to the NumPy array
```

```
            face_3d = np.array(face_3d, dtype=np.float64)
            # Solve PnP
            success, rot_vec, trans_vec = cv2.solvePnP(
                face_3d, face_2d, cam_matrix, dist_matrix)
            # Get rotational matrix
            rmat, jac = cv2.Rodrigues(rot_vec)
            # Get angles
            angles, mtxR, mtxQ, Qx, Qy, Qz = cv2.RQDecomp3x3(rmat)
            # Get the y rotation degree
            x = angles[0] * 360
            y = angles[1] * 360
            z = angles[2] * 360
    return x, y, z
def detect_nostrils_ellipses(image, landmarks):
    # nose landmark points
    LEFT_NARIS_EDGE_POINT = 64  # point 0 in MLKit face landmarks
    RIGHT_NARIS_EDGE_POINT = 294  # point 2 in MLKit face landmarks
    CENTRAL_NOSE_POINT = 2  # point 1 in MLKit face landmarks
    UPPER_NOSE_POINT = 195
    # nostrils ellipse parameters - center coordinates and axes
    ellipse_lx, ellipse_ly, axis_leftx, axis_lefty = None, None,
None, None
    ellipse_rx, ellipse_ry, axis_rightx, axis_righty = None, None,
None, None
    nose_length = None
    img_h, img_w, img_c = image.shape
    # finding special face landmark points
    if landmarks:
        face_landmarks = landmarks[0]
        for idx, lm in enumerate(face_landmarks.landmark):
            # nose points for measuring nostrils and relative swab
depth
            if idx == CENTRAL_NOSE_POINT:
                nose_p1x, nose_p1y = int(lm.x * img_w), int(lm.y *
img_h)
```

```
            if idx == LEFT_NARIS_EDGE_POINT:
                nose_p3x, nose_p3y = int(lm.x * img_w), int(lm.y *
img_h)
                if DEBUG:
                    cv2.putText(image, '.', (nose_p3x, nose_p3y),
                                cv2.FONT_HERSHEY_PLAIN, 0.5, (0, 0,
255), 2)
            if idx == RIGHT_NARIS_EDGE_POINT:
                nose_p4x, nose_p4y = int(lm.x * img_w), int(lm.y *
img_h)
                if DEBUG:
                    cv2.putText(image, '.', (nose_p4x, nose_p4y),
                                cv2.FONT_HERSHEY_PLAIN, 0.5, (0, 0,
255), 2)
            if idx == UPPER_NOSE_POINT:
                nose_p2x, nose_p2y = int(lm.x * img_w), int(lm.y *
img_h)
                if DEBUG:
                    cv2.putText(image, '.', (nose_p1x, nose_p1y),
                                cv2.FONT_HERSHEY_PLAIN, 0.5, (0,
255, 255), 2)
                    cv2.putText(image, '.', (nose_p2x, nose_p2y),
                                cv2.FONT_HERSHEY_PLAIN, 0.5, (0,
255, 255), 2)
        # Calculating nose points distance
        nose_length = int(((nose_p2x - nose_p1x) ** 2 + (nose_p2y -
nose_p1y) ** 2) ** 0.5)
        if DEBUG:
            cv2.putText(image, f'Nose_length {nose_length}', (50,
75),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255,
0), 2)
        # calculating central ellipse coordinates
        ellipse_lx, ellipse_ly = int((nose_p1x + nose_p3x) / 2),
int((nose_p1y + nose_p3y) / 2)
        ellipse_rx, ellipse_ry = int((nose_p1x + nose_p4x) / 2),
int((nose_p1y + nose_p4y) / 2)
        # calculating ellipse horizontal and vertical axes length
```

```
        AXIS_X_COEFF = 0.5  # ellipse x axis radius, related to
length between nose points

        AXIS_Y_COEFF = 0.5  # ellipse y axis radius, related to x
axis radius

        axis_leftx = int(AXIS_X_COEFF * (((nose_p1x - nose_p3x) ** 2
+ (nose_p1y - nose_p3y) ** 2) ** 0.5))

        axis_lefty = int(AXIS_Y_COEFF * axis_leftx)

        axis_rightx = int(AXIS_X_COEFF * (((nose_p1x - nose_p4x) **
2 + (nose_p1y - nose_p4y) ** 2) ** 0.5))

        axis_righty = int(AXIS_Y_COEFF * axis_rightx)

        if DEBUG:

            cv2.ellipse(image, (ellipse_lx, ellipse_ly),
(axis_leftx, axis_lefty), 0, 0, 360, (0, 0, 255), 1)

            cv2.ellipse(image, (ellipse_rx, ellipse_ry),
(axis_rightx, axis_righty), 0, 0, 360, (0, 0, 255), 1)

    left_naris_ellipse = (ellipse_lx, ellipse_ly, axis_leftx,
axis_lefty)

    right_naris_ellipse = (ellipse_rx, ellipse_ry, axis_rightx,
axis_righty)

    return left_naris_ellipse, right_naris_ellipse, nose_length

def detect_swab_insertion(image, nose_length_0, swab_length_0,
nose_length, swab_length, swab_inserted_dist):

    INSERTION_RATE = 0.5  # swab insertion depth (we need ~2.5cm),
relative to nose length

    FILTERING_COEFF = 0.2  # noise filtering coefficient(addition
per frame)

    if swab_length and nose_length:

        # initial swab 2d projection length measuring

        if nose_length_0 == 0 and swab_length_0 == 0:

            swab_length_0, nose_length_0 = swab_length, nose_length

            swab_inserted_dist = 0

        else:

            # filtering noise

            swab_inserted_dist = swab_inserted_dist * (1 -
FILTERING_COEFF) \

                                 + (swab_length_0 / nose_length_0 -
swab_length / nose_length) * FILTERING_COEFF

        # checking complete insertion
```

```
        insertion_completed = swab_inserted_dist > INSERTION_RATE
    else:
        insertion_completed = False
    # insertion progress bar
    img_h, img_w, img_c = image.shape
    cv2.rectangle(image, (50, img_h - 10), (450, img_h - 25), (255,
255, 255), -1)
    cv2.rectangle(image, (50, img_h - 10), (450, img_h - 25), (125,
255, 0))
    # normalization between 0 and 100%
    swab_inserted_percent = min(max(round(swab_inserted_dist * 200),
0), 100)
    cv2.rectangle(image, (50, img_h - 10), (50 +
swab_inserted_percent * 4, img_h - 25), (125, 255, 0), -1)
    cv2.putText(image, f'swab inserted {swab_inserted_percent:1} %',
(150, img_h - 14),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1)
    return swab_inserted_dist, nose_length_0, swab_length_0,
insertion_completed
def gen_frames():  # generate frame by frame from camera
    swab_length_0 = 0  # swab length, when swab is first time in the
nostril
    nose_length_0 = 0  # distance between nose points, when swab is
first time in the nostril
    swab_inserted_dist = 0
    insertion_completed = False  # variable for storing insertion
completed result
    result_queue = [False] * 30
    while cap.isOpened():
        # Capture frame-by-frame
        success, image = cap.read()
        # Flip the image horizontally for a later selfie-view
display
        # Also convert the color space from BGR to RGB
        image = cv2.cvtColor(cv2.flip(image, 1), cv2.COLOR_BGR2RGB)
        # Convert the color space from RGB to BGR
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
```

```
        img_h, img_w, img_c = image.shape
        # detecting swab and swab end
        x_swab, y_swab, swab_length = detect_swab_end(image)
        # getting landmarks
        image.flags.writeable = False  # To improve performance
        landmarks = face_mesh.process(image).multi_face_landmarks
        image.flags.writeable = True  # To improve performance
        # detecting face position angles
        x, y, z = detect_face_position(image, landmarks)
        # detecting nostrils
        left_naris_ellipse, right_naris_ellipse, nose_length =
detect_nostrils_ellipses(image, landmarks)
        (ellipse_lx, ellipse_ly, axis_leftx, axis_lefty) =
left_naris_ellipse
        (ellipse_rx, ellipse_ry, axis_rightx, axis_righty) =
right_naris_ellipse
        # Detecting swab position inside nostrils ellipses
        if x_swab and swab_length and nose_length:
            left_nostril_swab = (((x_swab - ellipse_lx) /
axis_leftx) ** 2 \
                                 + ((y_swab - ellipse_ly) /
axis_lefty) ** 2) <= 1
            right_nostril_swab = (((x_swab - ellipse_rx) /
axis_rightx) ** 2 \
                                  + ((y_swab - ellipse_ry) /
axis_righty) ** 2) <= 1
        # result queue increment
        res = False
        # checking head position
        if not nose_length:
            cv2.putText(image, "No face detected", (40,
                                                    img_h - 50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 255), 2)

        if nose_length and ((x < -7) or (x > 20) or (y < - 10) or (y
> 10)):
```

```
            cv2.putText(image, "Please, keep your head straight",
(40,

img_h - 50), cv2.FONT_HERSHEY_SIMPLEX, 0.8,
                        (0, 255, 255), 2)
            cv2.putText(image, "and a little bit to the up to detect
nostrils", (40, img_h - 25),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255,
255), 2)
            swab_length_0 = 0  # face is lost, setting initial swab
and nose lengths to zero
            nose_length_0 = 0
        elif nose_length:
            if not x_swab:  # checking swab is on image
                cv2.putText(image, "Please, take the swab", (40,
img_h - 50),
                            cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255,
255), 2)
                swab_length_0 = 0  # swab is lost, setting initial
swab and nose lengths to zero
                nose_length_0 = 0
            else:
                if not swab_length:  # checking swab lines detection
                    cv2.putText(image, "Can't find swab ending",
(40, img_h - 50),
                                cv2.FONT_HERSHEY_SIMPLEX, 1, (0,
255, 255), 2)
                    cv2.putText(image, "please use brighter light",
(40, img_h - 25),
                                cv2.FONT_HERSHEY_SIMPLEX, 1, (0,
255, 255), 2)
                # checking swab inside nostril
                elif left_nostril_swab or right_nostril_swab:
                    res = True
                    if sum(result_queue) > 14:
                        cv2.circle(image, (x_swab, y_swab), 20, (0,
255, 0), 5)
                        cv2.putText(image, "Please, insert the swab
deeper", (40, img_h - 75),
```

```
                                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0,
255, 255), 2)
                        cv2.putText(image, "about 2.5 cm", (40,
img_h - 50),
                                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0,
255, 255), 2)
                    else:
                        cv2.circle(image, (x_swab, y_swab), 20, (0,
255, 0), 1)
                        cv2.putText(image, "Please, wait for swab
detection", (40, img_h - 75),
                                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0,
255, 255), 2)
                else:
                    cv2.putText(image, "Please, insert the swab",
(40, img_h - 50),
                                cv2.FONT_HERSHEY_SIMPLEX, 1, (0,
255, 255), 2)
                    cv2.putText(image, "into your nostril", (40,
img_h - 25),
                                cv2.FONT_HERSHEY_SIMPLEX, 1, (0,
255, 255), 2)
        result_queue = result_queue[1:] + [res]
        # insertion depth control
        if sum(result_queue) > 15:
            swab_inserted_dist, nose_length_0, swab_length_0,
insertion = \
                detect_swab_insertion(image, nose_length_0,
swab_length_0, nose_length, swab_length, swab_inserted_dist)
            insertion_completed = insertion_completed or insertion
        if insertion_completed:
            cv2.putText(image, "Swab insertion completed", (50,
img_h - 120),
                        cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
            # image output
        image = ResizeWithAspectRatio(image, width=1024)
        ret, buffer = cv2.imencode('.jpg', image)
        frame = buffer.tobytes()
```

```
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame +
b'\r\n')  # concat frame one by one and show result

@app.get('/')

def index(request: Request):

    return templates.TemplateResponse("index.html", {"request":
request})

@app.get('/video_feed')

async def video_feed():

    return StreamingResponse(gen_frames(), media_type='multipart/x-
mixed-replace; boundary=frame')

@app.on_event("shutdown")

def shutdown_event():

    cap.release()

if __name__ == '__main__':

    uvicorn.run(app, host='127.0.0.1', port=5000, debug=True)

#    uvicorn.run(app, host='0.0.0.0', port=5000, debug=True)
```