Міністерство освіти і науки України
Національний аерокосмічний університет ім. М.Є. Жуковського
«Харківський авіаційний інститут»

Факультет систем управління літальними апаратами

Кафедра систем управління літальних апаратів

Пояснювальна записка

до дипломної роботи
магістра
(освітньо-кваліфікаційний рівень)

на тему Розробка нових методів і засобів для побудови мобільних систем детектування облич на базі мікрокомп'ютерів RASPBERRY PI

ХАІ.301.362.20О.151.00153040 ПЗ

Виконав: студент 6 курсу, групи 362 спеціальності

151 "Автоматизація та комп'ютерно-інтегровані технології

Освітньо-професійна програма "Інженерія мобільних додатків"

Казатинський Р.Є.
(прізвище та ініціали студента)

Керівник Краснов Л.О
(прізвище та ініціали)

Рецензент Лабазов О.О.
(прізвище та ініціали)

м. Харків – 2020 рік

Міністерство освіти і науки України

Національний аерокосмічний університет ім. М.Є. Жуковського
«Харківський авіаційний інститут»

Кафедра 301

«ЗАТВЕРДЖУЮ»
Завідуючий кафедрою
к.т.н., с.н.с., доцент
_____К.Ю. Дергачов
«__» _____2020 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на дипломне проектування
Казатинського Романа Євгеновича

1 Тема роботи: Розробка нових методів і засобів для побудови мобільних систем детектування обличь на базі мікрокомп'ютерів RASPBERRY PI
затверджена наказом по університету від «_26__»__10_____2020 р. 1775-уч.
2 Строк здачі студентом закінченої роботи «_11_»_____грудня_____2020 р.
3 Область застосування розробки: системи безпеки, покращення обслуговування, приватне використання відеоспостереження
4 Початкові дані для розроблювальної системи
4.1 Призначення і мета створення системи порівняльний аналіз показників якості відомих методів введення відеоданих в системах технічного зору і розробка нових методів і робочих алгоритмів, що забезпечують більшу швидкість при читанні відеоданих, використання систем детектування для аналізу ефективності методів.

4.2 Загальні відомості ознайомлення з рядом методів RASPBERRY PI и web-камери для порівняльної характеристики, застосування методів детектування Віоли-Джонса і Кофі.

5 Технічні вимоги до каналів системи управління

5.1 Питання, що підлягають розробці: встановити залежності і фактори, які впливають на FPS відеопотоку і ймовірність детектування.

5.2 Вимоги до структури й функціонування системи: : система написана на мові програмування Python, кросплатформенність, використання малого об'єму пам'яті, використання малого об'єму оперативної пам'яті, зручний інтерфейс.

5.3 Вимоги до показників якості системи : FPS без детектування не менше 20 кадрів в секунду, с детектуванням 15 кадрів в секунду.

5.4 Вимоги до конфігурації обчислювальної техніки Raspberry Pi 4 Model B має 4 ГБ оперативної пам'яті, швидкий 4-х ядерний процесор (1,5 ГГц), 8 мега піксельна камера з дозволом в 3280 x 2464 пікселів;

6 Умови експлуатації системи

6.1 Вимоги до програмної та інформаційної сумісності: система Windows 10, Linux, відео драйвера HS320.

6.2 Вимоги до захисту інформації від несанкціонованого доступу: немає

6.3 Вимоги до зовнішніх збурень: температура середовища (-10℃…+40℃), вологість середовища 20-80%, хімічно активні компоненти відсутні.

Характер роботи системи (безперервної, одноразового дії): безперервний

7 Додаткові функції, реалізовані системою (сигналізація про несправності, реєстрація необхідної інформації, самоконтроль самої системи і т.ін.): реєстрація інформації у реальному часі, запис статистики сеансу у файл, зручний інтерфейс, запис відео сеансу, здатність відкривати відео.

8 Обсяг виконуваних розроблювачем робіт

8.1 Етапи проведення роботи 1) вибір теми; 2) розробка ТЗ на проектування ПЗ; 3) Оцінка стану проблеми, аналіз ТЗ; 4)Опис алгоритмів методів зчитування відео

потоку 5) дослідження і аналіз якості відео потоку 6) опис алгоритмів детектування кадрів в режимі реального часу 7) проектування ПЗ; 8) експериментальна частина; 9) оформлення записки; 10) захист дипломного проекту

8.2 Обсяг розробки по кожному етапу технічне завдання (4л); реферат (1л); зміст(2л); список умовних позначень (1л); вступ (2л); аналітичний огляд літератури і існуючих методів вирішення задач(20л); Опис алгоритмів методів зчитування відео потоку (30л); дослідження і аналіз якості відео потоку (25л); реалізація системи (20л); експериментальна частина(10л); економічна частина (10л); висновок (1л); список використаних джерел (3л).

8.3 Вимоги до чисельності й кваліфікації персоналу: 1 програміст и один інженер знайомий з системами відео зору.

9 Вимоги до захисту інформації й надійності: здатність до безперервної роботи у реальному часі

10 Порядок контролю й приймання системи: система повинна досягати максимальної бистродії і мати здатність детектувати обличчя на відстані 5 метрів.

11 Дослідницька частина: опис програми, досліджування бистродії при різних методах зчитування інформації, опис модулю детектування , дослідження якості детектування , опис інтерфейсу програми

12 Експериментально-практична частина: проведення тестування системи, знаходження похибок, та знаходження оптимального результату тестування.13 Економічна частина

13.1 Розробити (розрахувати, одержати): розрахувати собівартість і ціну розробки системи детектування на базі RASPBERRY PI;

13.2 Умови і вимоги: річна програма випуску не менше 100 штук;

13.3 Очікуваний результат: виробництво даної системи повинно виходити на точку беззбитковості.

14 Перелік графічних матеріалів із зазначенням форматів: 10 плакатів формату А1

Керівник проектування
Краснов Леонід Олександрович
(П.І.Б.)
_____
« » _____ 2020 р.

Прийняв до виконання
Казатинський Р. Є._____
(П.І.Б. студента)
_____
« » _____ 2020 р.

Погоджено з питань:

проектування
Краснов Леонід Олександрович
(П.І.Б.)
_____
« » _____ 2020 р.

дослідницької частини
Краснов Леонід Олександрович
(П.І.Б.)
_____
« » _____ 2020 р.

економіки
Хлівна І. В._____
(П.І.Б.)
_____
« » _____ 2020 р

Міністерство освіти і науки України
Національний аерокосмічний університет ім. М.Є. Жуковського
«Харківський авіаційний інститут»

Факультет систем управління літальними апаратами
Кафедра систем управління літальних апаратів
Ступінь вищої освіти магістр_____
Спеціальність Автоматизація та комп'ютерно-інтегровані технології
Освітня програма Інженерія мобільних додатків_____

ЗАТВЕРДЖУЮ
Завідувач кафедри
систем управління ЛА

к.т.н., доц. _____ К.Ю. Дергачов

"____" _____20__ року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Казатинському Роману Євгеновичу

1. Тема роботи Розробка нових методів і засобів для побудови мобільних систем детектування обличь на базі мікрокомп'ютерів RASPBERRY PI
керівник роботи к. т. н., доцент Краснов Леонід Олександрович
затверджені наказом навчального закладу від 26.10. 2020 року № 1775-уч

2. Строк подання студентом роботи: 11 грудня 2020 року

3. Вихідні дані до роботи програмне забезпечення для детектування обличчя на базі мікрокомп'ютера RASPBERRY PI.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)): Аналітичний огляд літератури предметної області, аналіз шляхів та способів рішення завдання, опис та розробка алгоритму вводу відео даних розробка програмного забезпечення, аналіз результатів моделювання системи, розрахунок собівартості виготовлення програмного забезпечення.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) технічне завдання дипломного проекту, порівняльний аналіз мікрокомп'ютерів, проектування програмного продукту, інтерфейс програмного продукту, глибока

нейрона мережа в opencv, проектування методів детектування , аналіз методів детектування на основі дослідження, висновки дипломного проекту.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розробка технічного завдання	к. т. н., доцент Краснов Л.О		
Аналіз і синтез системи	к. т. н., доцент Краснов Л.О		
Проектування системи	к. т. н., доцент Краснов Л.О		
Реалізація системи	к. т. н., доцент Краснов Л.О		
Експериментальна частина	к. т. н., доцент Краснов Л.О		
Економічне обґрунтування	проф. д є. н. Хлівна І. В.		

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів дипломної роботи	Строк виконання етапів робот	Примітка
1.	Початок переддипломної практики	01.09.2020	
2.	Формулювання теми роботи. Розробка технічного завдання	18.09.2020	
3.	Математичний опис системи управління. Аналіз і синтез системи управління. Проведення експериментальних досліджень	16.10.2020	Залік з переддип-ломної практики
4.	Конструкторська частина роботи. Дослідницька частина роботи. Економічне обґрунтування розробки. Розробка питань охорони праці і безпеки в надзвичайних ситуаціях	20.11.2020	
5.	Оформлення розрахунково-пояснювальної записки і графічного матеріалу	04.12.2020	
6.	Попередній захист роботи. Рецензування роботи	11.12.2020	
7.	Захист роботи	17.12.2020	

Студент _____ Казатинський Р. Е.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Краснов Л. О.
(підпис) (прізвище та ініціали)

ЗМІСТ

СПИСОК СКОРОЧЕНЬ 12
ВСТУП 13
1 ПОСТАНОВКА ЗАДАЧІ 14
1.1 Аналіз технічного завдання 14
1.2 Критерій оцінки якості введення даних 14
1.3 Аналіз методів і засобів пошуку об'єктів при відеоспостереженні 15
1.4 Аналіз існуючих методів пошуку об’єктів 15
1.4.1 Колірна фільтрація 15
1.4.2 Виділення і аналіз контурів 16
1.4.3 Зіставлення з шаблоном 16
1.4.4 Пошук об'єктів за результатами аналізу стану особливих точок 17
1.4.5 Пошук об'єктів на основі зіставлення дескрипторів 18
1.4.6 Використання методів машинного навчання 19
1.4.7 Гістограми спрямованих градієнтів (HOG) 19
1.4.8 Мішок слів (Bag of visual Words, BoW) 20
1.4.8 Ознаки (примітиви) Хаара 20
1.4.9 Згорткова мережа 21
1.4.10 Локалізація об'єктів на зображенні 22
1.4.11 Детектування обличь на основі глибокого навчання 22
1.5 Постановка задачі проектування 23
2 ОПИС І ОСОБЛИВОСТІ СИСТЕМИ 24
2.1 Вибір апаратних і програмних засобів 24
2.1.1 Реалізація проекту на базі мікрокомп'ютера Raspberry Pi 24
2.1.2 Використані відеореєстратори 27
2.1.3 Основні ресурси мови Python і бібліотеки OpenCV 28
2.2 Підвищення якості введення і попередня обробка відео 29
2.3 Використовувані засоби програмування 31
2.4 Глибока нейрона мережа в OpenCv 32
2.4.1 Середовище Caffe 33
2.4.2 Середовище Keras 34
2.4.3 Середовище TensorFlow 34
2.4.4 Середовище Theano 34

3 ПРОЕКТУВАННЯ І РЕАЛІЗАЦІЯ СИСТЕМИ 36
3.1 Створення класів для контролю відеопотоку 36
3.1.1 Створення класу FPS. 36
3.1.2 Створення класу FR 36
3.2 Методи введення відеоданих 37
3.2.1 Класичний метод введення 37
3.2.2 Метод введення за рахунок апаратних засобів Raspberry Pi 37
3.3 Попередня обробка відеоданих 38
3.3.1 Стабілізація контрастності відеоданих 38
3.3.2 Фільтрація зашумлених зображень 39
3.3.3 Трансформація відео з RGB в Grayscale 40
3.3.4 Бінаризація кадрів відео з відсіканням по порогу яскравості 40
3.4 Алгоритм програми зчитування відеоданих 40
3.5 Виявлення та детектування облич 41
3.6 Виявлення і детектування обличь за методом Віоли-Джонса 42
3.6.1 Принципи побудови методу 42
3.6.2 Програма детектування обличь за методом Віоли-Джонса 44
3.6.3 Реалізація алгоритму Віоли-Джонса 46
3.7. Виявлення і детектування обличь за методом Кофі 48
3.7.1 Принцип побудови методу 48
3.7.2 Підготовка файлів 48
3.7.3 Алгоритм детектування за методом Кофі 49
3.7.4 Виявлення крапок (blobFromImage) 51
3.7.5 Обробка зображення за допомогою нейронної сітки (класс net) 52
3.8 Анонімізація зображень 53
3.8.1 Основний алгоритм програми 53
3.8.2 Анонімізація за допомогою методу Гауса 54
3.8.3 Методика анонімізації за допомогою укрупнення пікселів 54
3.9 Різниця між алгоритмом Віоли-Джонса и Кофі 55
3.9.1 Визначення 55
3.9.2 Тезиси різниці між алгоритмами 56
3.9.3 Використання алгоритмів 56
3.9.4 Різниця в реалізації 57
4 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ 58

4.1 Програма досліджень «Video Stream Quality Control» 58
4.1.1 Інтерфейс програми 58
4.1.2 Вікно відображення відео 60
4.1.3 Посібник користувача 63
4.1.4 Запис і збереження відеоданих в файлі 64
5 ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕНЯ ШВИДКОДІЇ 66
5.1 Вплив роздільної здатності кадру на показник швидкості введення відеоданих для web-камери 66
5.2 Вплив роздільної здатності кадру на показник швидкості введення відеоданих для рі-камери 67
5.3 Методи представлення відео та супутньої інформації 68
5.4 Негативний вплив роботи алгоритмів попередньої обробки відео на швидкість їх введення 69
6 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ДЕТЕКТУВАННЯ 73
6.1 Показники якості роботи алгоритмів та їх використання 74
6.1.1 Показник ймовірності правильного виявлення обличчя за умови його наявності в кадрі 74
6.1.2 Допоміжний показнику часу 74
6.1.3 Формат вводу даних 75
6.2 Дослідження швидкості відображення кадрів при різних методах загрузки для програмного забезпечення «Video Stream Quality Control» 75
6.2.1 Дослідження FPS відображення кадрів при зчитування з відеофайлу 75
6.2.2 Дослідження швидкості відображення кадрів при зчитування з реальної моделі 76
6.3 Дослідження детектування обличь при зміні показників 77
6.3.1. Зміна характеристики коефіцієнт масштабу методу Віоли-Джонса 77
6.3.2 Зміна коефіцієнту кількості сусідніх пікселей методу Віоли-Джонса 79
6.3.3 Зміна показнику confidence методу Кофі 80
6.3.4 Зміна показнику factor методу анонімізації 82
6.3.5 Зміна показнику block методу анонімізації 84
6.3.6 Дослідження ефективності роботи алгоритмів в залежності від відстані обличчя до відеореєстратора 85
6.3.7 Дослідження ефективності роботи алгоритмів при приховуванні частини обличчя 87
6.4 Висновок 89

7 ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ _____90
7.1 Основні статті витрат при розробці програми_____91
7.1.1 Розрахунок трудомісткості розробки програмного забезпечення ____91
7.1.2 Розрахунок витрат на розробку програмного забезпечення_____94
7.2 Додаткові статті витрат _____95
7.3 Результуюча таблиця собівартості _____97
7.4 Висновки_____97
ВИСНОВОК_____98
СПИСОК ВИКОРИСТАННИХ ДЖЕРЕЛ_____99
ПУБЛІКАЦІЇ ВИКОНАНІ В ПРОЦЕСІ ПРОВЕДЕННЯ РОБОТИ _____102
ДОДАТОК А _____103
ДОДАТОК Б _____114
ДОДАТОК В _____116

СПИСОК СКОРОЧЕНЬ

ТЗ — технічне завдання
ПК — персональний комп’ютер
ФП — функціональне програмування
ПП — програмний продукт
ООП — об’єктно-орієнтоване програмування
ФП — функціональне програмування
БД — база даних
ЗНО — зовнішнє незалежне оцінювання
КЗУ — компонент знань та умінь
СТЗ — система технічного зору
FPS — міра кількості кадрів за секунду
FR — роздільна здатність кадру
dnn — модуль глибокої нейронної мережі

ВСТУП

Бурхливо розвивається інженерія мобільних додатків активно використовує системи технічного зору (тут і далі - СТЗ) на базі малогабаритних відеореєстраторів і мікрокомп'ютерів для вирішення безлічі прикладних задач. Особливо велика потреба в таких системах при організації високоякісного відеоспостереження, в робототехніці, при оснащенні безпілотних літальних апаратів, автомобільного транспорту та ін. Ймовірно, однією з найбільш актуальних і важливих завдань СТЗ є задача розпізнавання осіб (face recognition). Практичні потреби вирішення такого завдання в найрізноманітніших ситуаціях постійно зростають. Це забезпечення безпеки і face-control в сегменті масових громадських заходів та розваг, пошук потенційно небезпечних відвідувачів, підозрюваних в терористичних намірах, верифікація банківських карт і online-платежі, а також маса інших актуальних і корисних додатків.

Завдання розпізнавання облич можливо умовно розділити на два етапи. Перший з них − достовірне виявлення і детектування облич (face detection) на цифрових зображення х і відеопослідовністях в режимі реального часу. Другий етап − ідентифікація осіб на основі виділення інформаційних ознак (виділення особливих точок особи), перетворення для порівняння його з особами в базі даних і визначення відповідності між ними.

При постановці завдання на проектування було запропоновано провести дослідження методів і засобів побудови мобільної системи детектування осіб при відеоспостереженні, визначити оптимальний склад і структуру апаратури, розробити алгоритми та методи програмування поставленого завдання. Тому спочатку розглянемо основні загальні особливості вирішення завдання виявлення і детектування об'єктів. Потім проведемо огляд, класифікацію та аналіз існуючих методів і засобів, які зазвичай використовуються при вирішенні подібних завдань. Далі визначимо особливості завдання face detection і запропонуємо конкретні технічні варіанти її вирішення.

1 ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз технічного завдання

У ТЗ наведені технічні вимоги до програмного продукту і сформовані питання, що підлягають розробці в проектувальній, експериментальній та реалізаційній частинах. Наведені вимоги до структури та функціонування системи, показників якості, конфігурації обчислювальної техніки. Також в ТЗ передбачені умови експлуатації системи. За допомогою всіх вимог наведених в ТЗ можливо визначити структуру та можливості програмного продукту, що зможе забезпечити відеоспостереження і детектування необхідних кадрів. А також оцінити ефективність методів і алгоритмів.

Також система повинна мати здатність до безперервної роботи у реальному часі і мати змогу оцінювати якість введення відео даних, тому насамперед треба ввести критерії оцінки якості введення даних.

1.2 Критерій оцінки якості введення даних

В першу чергу зупинимося на проблемі підвищення ефективності системи введення даних. З цією метою визначимо спільний критерій оцінки швидкості введення відеоданих і роздільної здатності кадрів. Для його створення введемо такі показники (і розшифруємо відповідні їм абревіатури):

- FPS − (Frame per second). Ця величина характеризує швидкість зміни/читання кадрів реєстрованого відеопотоку. Граничне значення FPS обмежена паспортними даними використовуваної відеокамери;
- FR – (Frame Resolution) описує роздільну здатність кадру відео в пікселях (320x240, 640x480, 1280x960 та ін.).

Ці найважливіші показники в основному і визначають якісні показники системи обробки відеоданих в СТЗ.

Традиційні методи захоплення відеоданих суттєво уповільнюють швидкість обробки відеоданих. Це найчастіше серйозно обмежує можливість роботи в реальному масштабі часу. Однак використовуючи багатопоточність можна помітно зменшити вплив затримки введення/виведення, залишаючи основний потік без блокування.

Зрозуміло, що при вирішенні задач візуалізації відеоданих завжди виправдано прагнення до підвищення якості за рахунок збільшення роздільної здатності кадру (FR). Однак слід пам'ятати, що велике збільшення числа пікселів в кадрі неминуче призводить до уповільнення (зменшення показника FPS). При цьому компроміс можна знайти експериментально − використовуючи пропоновані далі нові ефективні алгоритми захоплення і введення відеоданих.

Попередня обробка відеоданих це процес підвищення якості зображень відеопослідовності для отримання на основі оригіналів максимально точних і адаптованих для автоматичного аналізу зображень.

1.3 Аналіз методів і засобів пошуку об'єктів при відеоспостереженні

Загальна інформаційна концепція вирішення завдання виявлення об'єктів передбачає реалізацію двох основних сценаріїв:

- зйомка сцени з нерухомою камери при мало мінливих умовах освітлення (практично постійний фон), а інформаційні ознаки об'єкта спостереження в кадрі істотно відрізняються від ознак фону (за яскравістю, кольором, формою та ін.) Для вирішення завдання виявлення об'єкта в кадрі;
- постійно і суттєво змінюється фон при рухомому положенні камери. В цьому випадку необхідно сформувати сукупність інформаційних ознак об'єкта спостереження і шукати в кадрі області, які відповідають цим ознакам.

Такі сценарії дуже схожі на класичні задачі пасивної і активної локації, коли в першому випадку аналізується поле всіх прийнятих сигналів, а в другому випадку визначається наявність сигналів відомої форми.

Виділимо основні інформаційні ознаки і методи аналізу для пошуку потрібного об'єкту на зображенні або в кадрах відеопослідовності:

1. Якщо об'єкт суттєво виділяється на тлі за кольором, то потрібно подобрать відповідний колірний фільтр − метод колірної фільтрації.
2. Якщо відома форма об'єкта, наприклад, коло, то можна шукати окружності на зображенні − метод виділення та аналізу контурів.
3. Якщо є зображення об'єкта, то потрібно вести пошук і в інших областях, які збігаються із зображення м об'єкта − метод порівняння з шаблоном.
4. Пошук на зображенні з об'єктом відмінних рис (наприклад, кутів), які необхідно зіставити з такими ж особливостями на іншому зображенні − метод роботи з особливими точками.
5. Навчання класифікатора на зображення з об'єктом, поділ зображення на частини, перевірка класифікатором кожної частини на наявність об'єкта − методи машинного навчання.

Відзначимо, що часто використовуються і різні комбінації перерахованих методів.

1.4 Аналіз існуючих методів пошуку об’єктів

1.4.1 Колірна фільтрація

Зазвичай колірна фільтрація застосовується у випадках, коли об'єкт суттєво відрізняється від фону за кольором, а освітлення сцени рівномірно і мало змінюється. Якщо ж об'єкт на тлі за кольором істотно не виділяється і/або має складну розмальовку, то застосування методу колірних фільтрів не дає хороших результатів. У цих випадках для вирішення завдання виявлення необхідно використовувати інші інформаційні ознаки.

1.4.2 Виділення і аналіз контурів

Детектування кордонів є широко використовуваним методом для виявлення ознак об'єкта. Вона використовується для ідентифікації точок на зображенні зі значними перепадами яскравості. Основна мета виділення кордонів - зменшити складність зображення , зберігаючи його основну структуру. Детектор кордонів Канни є одним з найбільш часто використовуваних детекторів кордонів [5]. Алгоритм складається з п'яти окремих кроків, описаних далі:

- згладжування зображення фільтром Гаусса, який виявляє і усуває знайдені розриви, застосовуючи переміщувану по зображенню маску;
- розрахунок градієнтів за допомогою дискретного диференціального оператора Собеля;
- придушення неосновних максимумів, де пікселями кордонів оголошуються пікселі, в яких досягається локальний максимум в напрямку вектора градієнта. В результаті придушення локальних невизначеностей товщина лінії кордону стає рівномірною і тонкої, що збільшує точність її визначення;
- подвійна порогова фільтрація, тобто, якщо значення пікселя вище верхньої межі порогу, то він приймає максимальне значення (межа вважається достовірною), якщо нижче - піксель пригнічується, точки зі значенням, що потрапляють в діапазон між порогами, приймають фіксоване середнє значення.
- трасування області неоднозначності. Завдання зводиться до виділення груп пікселів, які отримали на попередньому етапі проміжне значення, і віднесенню їх до кордону (якщо вони пов'язані з однією з встановлених меж) або їх придушення (в іншому випадку). Піксель додається до групи, якщо він стикається з нею по одному з восьми напрямків.

Таким чином, застосування фільтра виділення кордонів до зображення може істотно зменшити кількість оброблюваних даних. Однак, одним з істотних недоліків алгоритму Канні є необхідність настройки порогових значень детектора.

Перевірку виділених ліній-кордонів на відповідність геометричним контурам об'єкта зазвичай проводять за методом перетворення Хафа (Hough Transform) [9]. Перетворення Хафа призначене для пошуку об'єктів, що належать певного класу фігур з використанням процедури голосування. Процедура голосування застосовується до простору параметрів, з якого і виходять об'єкти певного класу фігур по локального максимуму в, так званому, накопичувальному просторі (accumulator space) яке будується при обчисленні трансформації Хафа.

1.4.3 Зіставлення з шаблоном

Цей метод (Template matching) застосовується для пошуку ділянок зображень найбільш схожих з деяким заданим шаблоном. Вхідними параметрами методу є зображення , на якому необхідно шукати шаблон і зображення об'єкта,

який ми хочемо знайти на тестованому зображенні. Розмір шаблону повинен бути меншим за розмір перевіряється зображення . Кінцева мета роботи алгоритму пошуку − знайти на тестованої зображенні область, яка найкраще збігається з шаблоном.

Пошук шаблону проводиться шляхом послідовного переміщення його по тестируемому зображенню з оцінкою схожості кожної нової області з шаблоном. За результатами перевірки вибирається область, що має найвищий коефіцієнт збігу. По суті, це відсоток збігу області тестового зображення і шаблону. Template matching − хороший алгоритм для виявлення, коли необхідно швидко перевірити наявність деякого об'єкта на зображенні. Однак зазначимо, що template matching не дозволяє достовірно стверджувати, що був знайдений вихідний об'єкт, оскільки ця характеристика має імовірнісну природу. Вона залежить від масштабу, кутів огляду, поворотів зображень і наявності фізичних перешкод. Також можливі помилкові спрацьовування алгоритму, коли шуканого об'єкта насправді немає, але є якісь загальні деталі у шаблону та області на тестованому зображенні.

1.4.4 Пошук об'єктів за результатами аналізу стану особливих точок

Метод використовується в тих випадках, коли важко і неефективно зіставлення шаблонів через дії різного роду чинників, що заважають. Особливі точки (key points) це невеликі області, які виділяються на зображенні. Існують різні методи визначення таких точок і їх положення. Наприклад, визначення кутів за допомогою детектора Харріса (Harris corner detector) [11,12]. Алгоритми детектування кутових точок в кадрі визначають характерні опорні точки для нерухомих об'єктів, які згодом можна використовувати при розрахунку характеристик руху спостережуваних об’єктів. Пошук об'єктів за результатами аналізу стану особливих точок.

Для виявлених особливих точок потім обчислюються так звані дескриптори, що визначають характеристики особливих точок [13]. Їх обчислюють за заданою околиці особливих точок, як напрямки градієнтів яскравості різних частин цієї околиці.

Особливі точки застосовують для пошуку об'єктів на зображенні. Для цього необхідно мати зображення шуканого об'єкта і далі виконати наступні дії:

1. На зображенні з об'єктом визначаються особливі точки об'єкта і обчислюються їх дескриптори.
2. На уже згадуваному зображенні теж ведеться пошук особливих точок, і обчислюються відповідні їм дескриптори.
3. Потім проводиться порівняння дескрипторів особливих точок об'єкта і дескрипторів особливих точок, знайдених на зображенні.
4. Якщо знайдено достатню кількість відповідностей, то зазначається область з відповідними особливими точками.

1.4.5 Пошук об'єктів на основі зіставлення дескрипторів

Кожен дескриптор є метод, який ідентифікує деяку область зображення на основі набору ознак. Прийнято виділяти такі групи дескрипторів двовимірних зображень [14, 15]:

- локальні виконавчі дескриптори;
- дескриптори на основі спектрального подання;
- дескриптори на основі базисних функцій;
- дескриптори форми.

Відзначимо, що деякі методи за своїми характеристиками можна віднести до різних груп одночасно. Локальні виконавчі дескриптори є опис невеликої області зображення у вигляді бінарних векторів. Найбільш відомими локальними дескрипторами є:

- локальні виконавчі шаблони і їх модифікації (Local Binary Patterns - LBP) [16];
- BRIEF (Binary Robust Independent Elementary Features) [17];
- ORB (Oriented BRIEF) [18];
- BRISK (Binary Robust Invariant Scalable Key points) [19].

Дескриптори на основі спектрального уявлення використовують різні величини для ідентифікації областей: інтенсивність, колір, градієнти, статистичні характеристики. У порівнянні з двійковими, дескриптори на основі спектрального уявлення вимагають більш складних обчислень. До таких методів належать такі дескриптори і їх модифікації:

- SIFT (Scale Invariant Feature Transform) [20];
- SURF (Speeded Up Robust Features) [21];
- DAISY [22];
- HoG (Histogram of Gradients) [23];
- кореляційні шаблони (Sum of Absolute Differences (SAD);
- Sum of Squared Differences (SSD), Normalized Cross Correlation (NCC)) [24];
- Local Gradient Pattern (LGP) [25];
- код Фрімана (Chain Code Histograms (CCH)) [26];
- ознаки Хаара (HAAR Features) [27].

Дескриптори на основі базисних функцій представляють опис зображення в заданих просторах. Найбільш відомим дескриптором на основі базисних функцій є дескриптор Фур'є [28]. До цієї групи дескрипторів можна також віднести методи розрідженого кодування (Sparse Coding) [27]. Прикладом дескриптора з цієї групи є метод «мішок слів» (Bag of Words) [29]. У дескрипторах на основі розрідженого кодування замість базисних функцій використовується набір кодів для ідентифікації об'єктів.

Дескриптори форми (Polygon Shape Descriptors) дозволяють виконувати пошук об'єктів на основі таких характеристик, як площа, параметри контуру, моменти областей, центр ваги полігону, коефіцієнти прямокутності і округлості, кількість дірок. До найбільш відомих підходів відносяться:

- MSER (Maximally Stable Extremal Regions) [30],
- код Фрімана [26]
- дескріптор Фурье [28],
- контекст форми (Shape Context) [31],
- дескриптор на основі площі дискового покриття [32],
- морфологічні дескриптори форми [33],
- дескриптор на основі обмежують областей [34],
- моменти регіону [35], кривизна кордонів (curvature scale-space) [36],
- скелетон (Shock graphs) [37] та ін.

Недоліком багатьох підходів є те, що в системах комп'ютерного зору часто важко виділити кордону аналізованого об'єкта через ефектів різкої зміни освітлення, взаємних перекритій, складної текстури фону, зміни ракурсів тощо.

1.4.6 Використання методів машинного навчання

Описані вище методи виявлення об'єктів мають істотний загальний недолік − кожен з них зазвичай спирається на розпізнавання окремих (одного або декількох) локальних ознак. Природно, це помітно обмежує область їх застосування. Такі недоліки можна подолати шляхом розширення вузького класу інформаційних ознак і застосування методів машинного навчання.

Пошук об'єктів в кадрах відео потоку передбачає вирішення трьох відносно незалежних завдань:

- виявлення (detection) області на зображенні, імовірно містить об'єкти пошуку;
- розпізнавання (recognition) виявлених об'єктів;
- супроводження (tracking) − локалізація на наступних кадрах відеопослідовності розпізнаних об'єктів.

Найбільш продуктивний шлях вирішення перших двох завдань (detection і recognition) заснований на використанні методів машинного навчання, які зводяться до побудови класифікаторів зображень. Класифікатор зображень складається з двох частин − методу вилучення ознак (feature extractor) і власне класифікатора.

Розглянемо далі різні способи побудови класифікатора зображень за допомогою методів вилучення ознак.

1.4.7 Гістограми спрямованих градієнтів (HOG)

Гістограми спрямованих градієнтів (Histogram of Oriented Gradients, HOG) [45] - метод вилучення ознак із зображень, аналогічний методу обчислення дескрипторів SIFT [2] для особливих точок, але з обчисленням його не для околиці особливих точок, а для всього зображення . Загальна схема обчислення HOG виглядає так. Зображення ділиться на частини (осередки), для кожного осередку будується гістограма напрямків градієнта яскравості, далі гістограми осередків

нормуються по контрасту і об'єднуються. Приклад використання HOG ми розглянемо далі в завданні локалізації об'єктів на зображенні.

1.4.8 Мішок слів (Bag of visual Words, BoW)

При використанні методів машинного навчання для розпізнавання об'єктів на зображенні необхідно створити навчальний набір даних, що складається з двох класів:

1. набір зображень містять об'єкт (позитивні приклади),
2. набір зображень об'єкт не містять (негативні приклади).

Аналогічно методу частотного аналізу текстів (TF, term frequency), для використання методу BoW, необхідно створити словник наступним чином:

- вибрати метод визначення особливих точок і метод обчислення для них дескрипторів;
- для кожного зображення , що входить в навчальний набір, визначити особливі точки і обчислити для них дескриптори;
- об'єднати схожі дескриптори в групи, тобто провести кластеризацію безлічі отриманих дескрипторів.
- отримані кластери дескрипторів виконують роль візуальних слів і складають словник для частотного аналізу зображення . Ознаки BoW витягуються з зображення наступним чином:
- обраним на етапі складання словника методом, визначаються особливі точки на зображенні і обчислюються їх дескриптори;
- знайдені дескриптори розбиваються по кластерам словника;
- для кожного кластера словника визначається кількість знайдених дескрипторів.

Таким чином, вектор BoW-ознак визначає кількість знайдених дескрипторів в кожному кластері словника.

1.4.8 Ознаки (примітиви) Хаара

Ознаки (примітиви) Хаара (Haar-like features) визначаються шляхом вибору прямокутної області на зображенні, розбиттям її на кілька суміжних прямокутних частин, в кожній частині яких підсумовується яскравість точок і обчислюється різниця між цими сумами. Ця різниця і є значенням ознаки.

Для формування характеристики зображення зазвичай використовують кілька різних ознак Хаара, у кожного з яких виділяються свої параметри – розмір області для обчислення ознаки, кількість частин, позиція на зображенні.

На базі ознак Хаара побудований метод Віоли-Джонса [1,2]. Його часто використовують для детектування осіб на зображення х. Він об’єднує в собі примітиви Хаара, класифікатор AdaBoost [3] і метод ковзного вікна для поділу зображення на частини. Більш докладно особливості використання алгоритму Віоли-Джонса будемо розглядати далі.

1.4.9 Згорткова мережа

Це одна з моделей класифікатора зображень. Більш повна назва - згорткова штучна нейронна мережа (convolutional neural network, CNN). Особливістю цієї моделі класифікатора є вбудований механізм вилучення ознак із зображення . Він є частиною мережі і «самоналаштовується» в процесі навчання мережі.

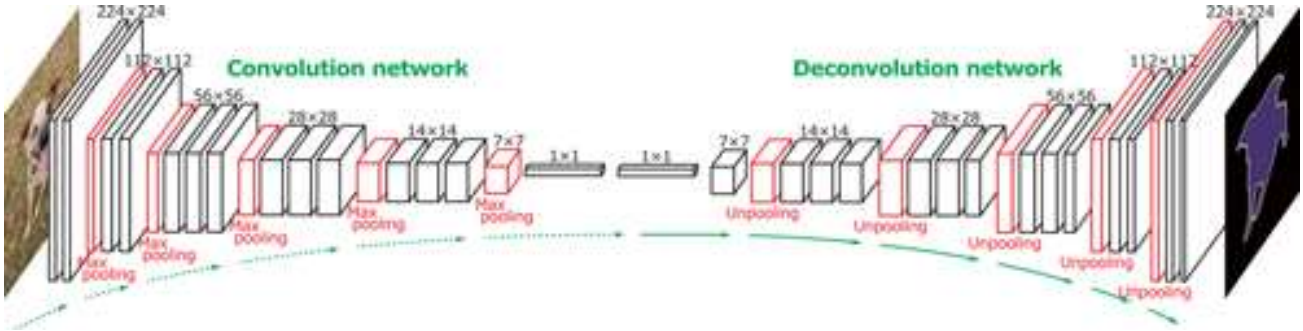


Рисунок 1.1 – Архітектура згорткової мережі

Основою повно завернутої мережі є оболонка зображення . Ключовим полем є згортковий шар. Шар згусток вказує кількість виходів з шару, ядро рулону, його крок, розмір, відступ.

Згусток операція проходить через ядро по всьому зображенню, в результаті чого відповідь на ядро рулону в кожній точці зображення . Кількість ядр кожного шару рулону дорівнює кількості виходів з шару на кількість вхідних зображень. Потім результати проходять через наступний шар рулону, отримуючи значення вже для інших ядер. Ви можете додати шари регуляризації або нормалізації для кожного шару тромбів (залежно від вибору розробника). Проходження зображень через багато шарів дозволяє отримати багату різноманітність можливих інтерпретацій зображення .

Пройшовши через необхідну кількість шарів оболонки, зображення потім потрапляє в шар групування. Цей шар зменшує розмір вхідних зображень, не зменшуючи їх кількість. Шар має ядро, яке рухається як відомість, обчислюючи єдине значення для кожної області зображення . Зменшення зображення допомагає швидше обробляти мережеві процеси. Це дозволяє додавати більше виходів у наступні шари, а також покращувати точність результатів. Справа в тому, що в меншій образі сердечника, оболонки такого ж розміру здатні захоплювати велику площу потрібного об'єкту.

Послідовності: convolution /.. /.. /convolution/pooling (де кількість звивинами прочитане шарів визначається забудовником) можна повторити кілька разів, поки не буде досягнуто мінімального розміру зображення . Цей розмір визначається експериментально.

Таким чином, ми отримуємо симетричну архітектуру щодо останнього шару групування та першого шару upsample. Шари збільшення розміру зображення також поміщаються з шарами, але кількість виходів з них поступово зменшується.

1.4.10 Локалізація об'єктів на зображенні

Завдання присвячується вирішенню питання про наявність об'єкта на зображенні і пошуку області зображення , яку він займає. Її зазвичай вирішують методом ковзного вікна (sliding window) [4], схема якого виглядає так:

- визначається розмір w вікна (досліджуваної області зображення);
- формується навчальний набір L з позитивних (на зображенні присутній об'єкт) і негативних (на зображенні відсутній об'єкт) прикладів, розмір навчальних зображень повинен відповідати розміру w вікна;
- навчається класифікатор c на отриманому наборі L;
- визначається зображення для дослідження, вибирається крок зсуву вікна d (по горизонталі і вертикалі) і коефіцієнт масштабування зображення .
- розміщується вікно w в крайньому положенні (лівий верхній кут) на досліджуваному зображенні;
- виконується класифікація поточного вікна, якщо класифікатор c визначив наявність об'єкта в поточному вікні, то треба помістити параметри вікна (поточний стан вікна, масштаб зображення і значення видане класифікатором) в список результатів R;
- якщо вікно не досягло кінцевого положення (правий нижній кут), то воно зсувається на крок d і проводиться перехід на попередній пункт. Інакше необхідно перейти до наступного пункту;
- якщо розмір зображення перевищує розмір w вікна, то зображення масштабується з коефіцієнтом s і це вікно знову розміщується в крайньому положенні (лівий верхній кут). Інакше робота припиняється.

Остаточним етапом роботи алгоритму є формування списку результатів R, який складається з позиції вікна, відповідних йому значення масштабу зображення і значення виходу класифікатора. Далі необхідно виконати обробку отриманих результатів.

1.4.11 Детектування обличь на основі глибокого навчання

Глибоке навчання - сукупність методів машинного навчання , заснованих на навчанні уявленням, а не спеціалізованим алгоритмам під конкретні завдання.

Глибинна нейронна мережа (DNN, англ . DNN - Deep neural network) - це штучна нейронна мережа (ШНМ) з декількома шарами між вхідним і вихідним шарами. DNN знаходить коректний метод математичних перетворень , щоб перетворити вихідні дані в виходять , незалежно від лінійної або нелінійної кореляції . Мережа просувається по верствам , розраховуючи ймовірність кожного

виходу . Наприклад , DNN, яка навчена розпізнавати породи собак, пройде по заданому зображенню і вирахує ймовірність того, що собака на зображенні відноситься до певної породі. Користувач може переглянути результати і вибрати ймовірності , які повинна відображати мережу (вище певного порогу, наприклад), і повернути в мережу запропоновану мітку. Кожне математичне перетворення вважається шаром , а складні DNN мають багато слоїв , звідси і назва « глибинні » або « глибокі » мережі.

DNN можуть моделювати складні нелінійні відносини . Архітектури DNN генерують композиційні моделі , в яких об'єкт виражається у вигляді багаторівневої композиції примітивів. Додаткові рівні дозволяють складати елементи з більш низьких рівнів , потенційно моделюючи складні дані з меншим кількістю одиниць , ніж дрібна мережа з аналогічними показникам.

Глибока архітектура включає в себе безліч варіантів кількох основних підходів. Кожна архітектура знайшла успіх в певних областях. Чи не завжди можливо порівняти продуктивність декількох архітектур, якщо вони НЕ були оцінені на одних і тих же наборах даних .

DNN, як правило, представляють собою мережі з прямою зв'язком , в яких дані передаються від вхідного рівня до вихідного рівня без зворотного зв'язку . Спочатку DNN створює карту віртуальних нейронів і призначає випадкові числові значення або « ваги » з'єднанням між ними. Ваги і вхідні дані множаться і повертають вихідний сигнал від 0 до 1. Якщо мережа неточно розпізнала конкретний шаблон, алгоритм буде коригувати вагові коефіцієнти. Таким чином, алгоритм може зробити певні параметри більш значущими , поки він не визначить правильні математичні маніпуляції для повної обробки даних .

1.5 Постановка задачі проектування

Для досягнення мети проекту необхідно провести порівняльний аналіз показників якості відомих методів і алгоритмів введення відеоданих для різних варіантів підключення відеодатчиків, розробити адекватні критерії оцінки якості їх роботи, запропонувати нові методи, робочі алгоритми і програмні код для їх реалізації. Провести експериментальні дослідження цих алгоритмів, оцінити ефективність їх роботи в лабораторних умовах, достовірність отриманих результатів перевірити методами статистичного аналізу.

Для цього потрібно розробити програмне забезпечення, котре може реалізовувати такі функції:

1) здатність взаємодіяти с відео файлами та камерами;
2) можливість підключення web-камери і рі-камери;
3) запис статистики сеансу у файл;
4) реєстрація інформації у реальному часі;
5) зручний інтерфейс;
6) здатність детектувати обличчя;
7) здатність до анонімізації обличчя.

2 ОПИС І ОСОБЛИВОСТІ СИСТЕМИ

При проектуванні мобільних СТЗ дизайнери апаратно-програмних комплексів повинні попередньо вирішити проблеми вибору економного автономного живлення, мікрокомп'ютера з достатньою швидкодією і об'ємом пам'яті, відповідних відеореєстраторів і сучасного програмного забезпечення.

Крім того, на етапі оформлення ескізу необхідно запровадити критерії якості відео інформації, отриманої у відповідності до прийнятих стандартів, і дозволить максимально ефективно використовувати наявні ресурси апаратного та програмного забезпечення.

Складність побудови сучасних мобільних СТЗ пов'язана з відсутністю єдиного підходу до підбору мікрокомп'ютерів і відеокамер, методів введення, попередньої обробки і запису відеоданих. Тому ми проведемо порівняльний аналіз відомих методів та інструментів, а також пропонуємо нові технічні рішення. Таке завдання досить сучасне і актуальне, оскільки дизайн сучасного СТЗ вимагає нетривіальних апаратних, алгоритмічних і програмних рішень.

Метою роботи в першій частині проекту є аналіз відомих методів введення, попередньої обробки та запису відео при проектуванні СТЗ, а також розробка і створення нових алгоритмів і програмних засобів, гарантовано забезпечують високу якість вихідних відеоданих і обробку відео в реальному масштабі часу. Оцінку якості та ефективності нових методів введення, обробки і запису відео необхідно провести в лабораторних умовах. У другій частині проекту на базі отриманих результатів необхідно розробити алгоритми детектування осіб і забезпечити їх програмну реалізацію. Крім цього в деяких випадках необхідно опціонально забезпечити режим анонімізації осіб з метою збереження Конфіденційність особистої інформації.

2.1 Вибір апаратних і програмних засобів

Основні ресурси для побудови мобільних СТЗ. Розглянемо більш уважно і детально оцінимо головні технічні аспекти проектів з побудови сучасних мобільних систем СТЗ. При цьому будемо орієнтуватися на основні положення нашої постановки завдання на проведення досліджень, і спиратися на доступні дані сучасних літературних джерел і internet-ресурсів.

2.1.1 Реалізація проекту на базі мікрокомп'ютера Raspberry Pi

Для задоволення потреб цього сегмента ринку зараз доступний досить великий ряд моделей мікрокомп'ютерів (Lego, Intel Galileo, Android IOIO OTG, Arduino та ін.). Однак в цьому ряду міцне лідерство утримує платформа Raspberry Pi [1 - 4]. Вона представлена лінійкою моделей, що мають різну апаратну реалізацію за доступними цінами. Кращі з них по продуктивності не поступаються настільним ПК. Так Raspberry Pi 4 Model B має 4 ГБ оперативної пам'яті, швидкий 4-х ядерний процесор (1,5 ГГц), підтримку двох дисплеїв з роздільною

здатністю до 4K, Gigabit Ethernet, USB 3.0, Wi-Fi, Bluetooth 5.0 і живлення через блок USB-C (5В/3А). Особливо важливо, що Raspberry Pi 4 має два порти USB 3.0. Вони в 10 разів швидше в порівнянні з USB 2.0 і добре підходять для підключення швидких периферійних блоків (флеш-накопичувачів, web-камер та ін.). На рис. 2.1 показана модель Raspberry Pi 3B

Рисунок 2.1 – Мікрокомп’ютер Raspberry Pi 3B

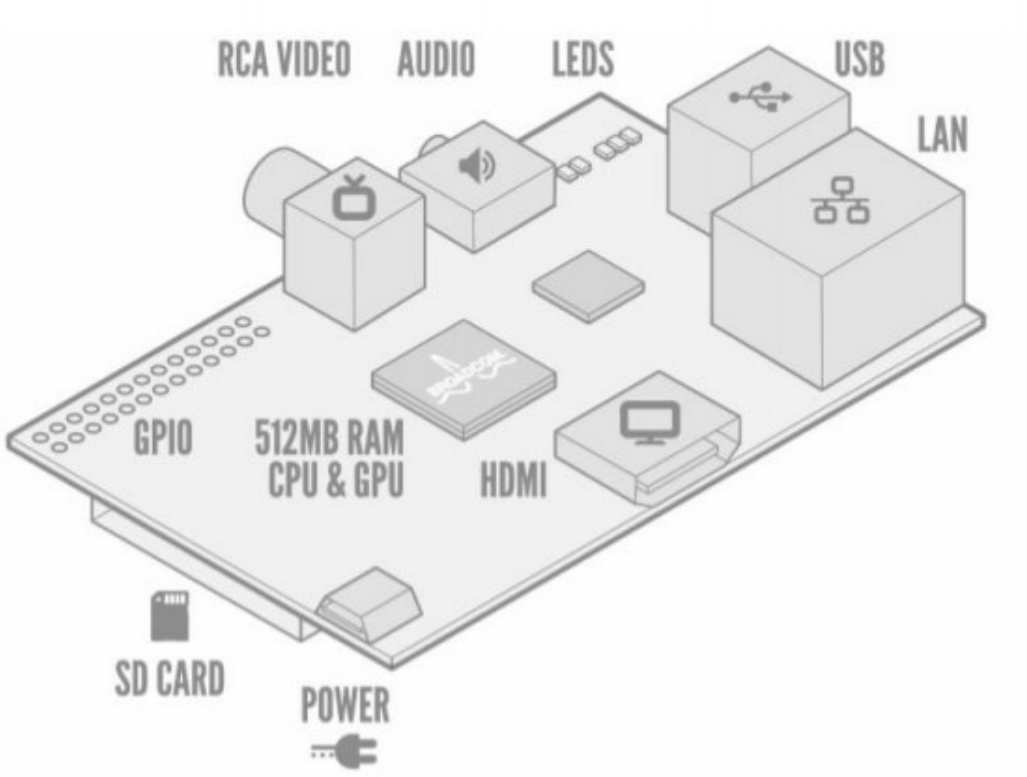


Рисунок 2.2 – Мікрокомп'ютер Raspberry Pi

Дуже корисні порти управління зовнішніми пристроями GPIO (general purpose input / output). Оскільки вони не підписані, наводимо роздруківку цоколювка цього роз'єму (рис. 2.3). Це завжди необхідно пам'ятати при створенні проектів з управління елементами автоматики та ін.

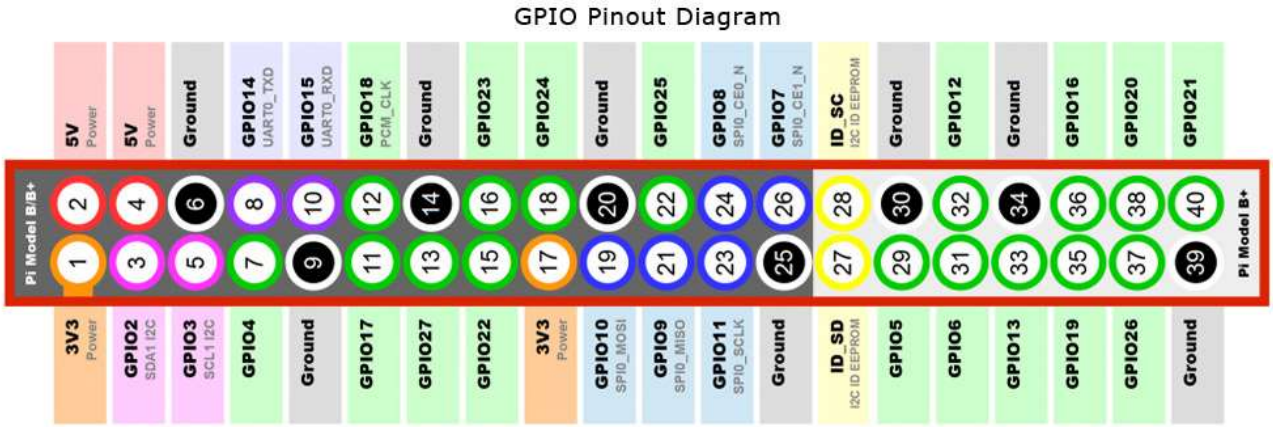


Рисунок 2.3 - Цокольовка роз'єму GPIO для Raspberry Pi

Плата підтримує велику кількість інтерфейсів(рис 2.2):
1. UART
2. Слот для карт SD, MMC, SDIO
3. Роз’єм HDMI - 3.5мм стерео
4. Композитний відеовихід
5. 2x USB 2.0
6. 10/100Mb RJ45 Ethernet
7. Wi-Fi 802.11n - Bluetooth 4.1
8. 16 портів input/output 3.3В
9. Інтерфейси I2C і SPI - Інтерфейс ARM JTAG
10. Інтерфейс DSI
11. Інтерфейс MIPI CSI-2

Процесор. Raspberry Pi B використовує 32-розрядний процесор ARM Cortex-A7 900 МГц. Він має кеш першого рівня (L1) 16 Кб, а кеш другого рівня (L2) 128 Кб. Кеш-пам'ять другого рівня використовується насамперед графічним процесором. SoC розміщується під мікросхемою оперативної пам'яті, тому його не видно.

Продуктивність. Під час роботи на частоті 700 МГц, Raspberry Pi забезпечила реальну продуктивність, приблизно еквівалентну 0,041 GFLOPS. На рівні центрального процесора продуктивність аналогічна Pentium II 300 МГц 1997-99. GPU забезпечує 1 Gpixel/s або 1.5 Gtexel/s обробки графіки або 24 GFLOPS загальної обчислювальної продуктивності.

Програмне забезпечення. Raspberry Pi працює в основному на операційних системах, заснованих на Linux ядрі. Також можлива установка Windows 10 IOT. Більш того, існують Raspberry з ліцензійною Windows 10 IOT. ARM11 заснований на 6 версії ARM, який завжди підтримує усі різновиди Linux. Для установки операційних систем існує інструмент NOOBS, або інші аналоги для запису образів ОС на носії, що дозволяють записувати на карти пам’яті операційні системи сумісні з Raspberry Pi.

2.1.2 Використані відеореєстратори

Зазвичай мобільні СТЗ комплектуються недорогими некаліброваними web-камерами з малою роздільною здатністю і можливостю їх підключення до Raspberry Pi через USB-порти. Такий метод організації відеоспостереження має ряд недоліків. Головний з них − при синтезі систем стереозору не можна використовувати одну USB-шину. Це виключає можливість синхронізації камер і помітно зменшує сумарну пропускну здатність системи. При цьому, складно врахувати розсинхронізація за рахунок багатозадачності сучасних операційних систем і функціонування планувальника завдань. Прийнятною вважається розсинхронізація камер не більше 10 мс [7].

Більш високою якістю в порівнянні з web-камерами мають спеціалізовані рі-камери для плат Raspberry Pi. Вони використовуються для монокулярних систем відеоспостереження. Наприклад, Raspberry Pi Camera Module v2 − високоякісний датчик зображення Sony IMX219 з фіксованим фокусом. Він підключається до плати мікрокомп'ютера через спеціальний CSI-інтерфейс (Camera Serial Interface) за допомогою короткого стрічкового кабелю. Розміри датчика − 25mm x 23mm x 9mm, вага − 3 г.

Основні можливості цього датчика:

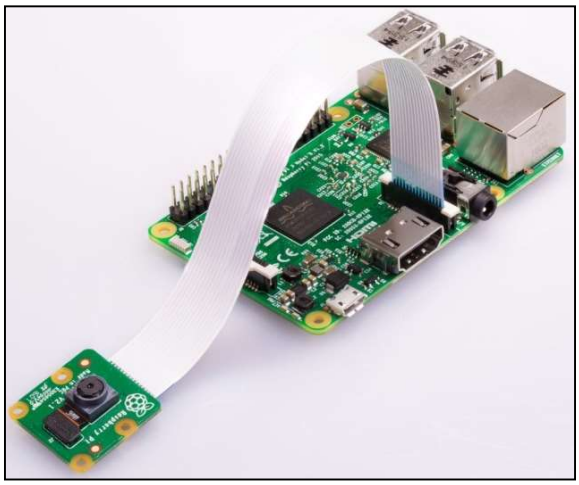
- 8 мегапіксельна камера здатна робити фотографії з дозволом в 3280 x 2464;
- Запис відео з роздільною здатністю 1080p 30FPS, 720p 60FPS, 640 × 480p 60/90FPS;
- Програмне забезпечення підтримується в останній версії Raspbian Operating System.

Тут і далі будемо використовувати скорочення p − pixel.

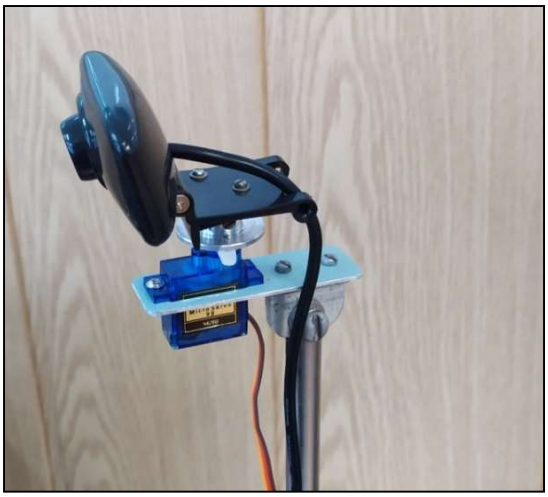
Для проведення експериментальних досліджень було створено лабораторний стенд на базі мікрокомп'ютера Raspberry Pi 3 Model B і набору з однієї web-камер і рі-камери. Зовнішній вигляд лабораторного стенду показаний на рис. 2.4. Ідентичні web-камера FC-250 (FPS = 30 при вирішенні FR = 1280x960) з можливістю їх підключення до комп'ютера через USB-порти.

Одиночна рі-камера (в нашому випадку описана раніше камера Sony IMX219 Exmor) використовується для системи монокулярного зору. Вона закріплена на окремому штативі (рис. 2.4в) і підключена до Raspberry Pi через спеціальний відеовхід CSI. Це значно знижує навантаження на центральний процесор в порівнянні з підключенням камер по USB (рис. 2.4а). Цей 8-ми мегапіксельний сенсор дозволяє захоплювати, записувати, транслювати відео і підтримувати наступні формати відео: 1080p (30FPS), 720p (60FPS), 640 × 480p (90FPS).

На рис. 1.а показаний ще один приклад - використання web-камери з USB підключенням для цілей відеоспостереження. Ця камера закріплена на спеціальній платформі, поворот якої за допомогою сервоприводу можна здійснювати через роз'єм апаратних портів введення / виводу GPIO (General Purpose Input/Output) комп'ютера Raspberry Pi.



а



б

Рисунок 2.4 – Фрагменти лабораторного стенду

2.1.3 Основні ресурси мови Python і бібліотеки OpenCV

Програмне забезпечення систем технічного зору на базі комп'ютера Raspberry Pi зазвичай використовує рекомендовану розробником операційну систему Raspbian і мову програмування Python. Вона як сучасна об'єктно-орієнтована мова, найбільш часто використовується для програмування портів введення/виводу GPIO (General Purpose Input/Output) на Raspberry Pi і входить до складу операційної системи Raspbian. До нього можна підключити велику кількість спеціалізованих бібліотек, які розширюють можливості вирішення завдань потрібних додатків. Особливо важливо, що в зв'язці з Python можна використовувати OpenCV − бібліотеку алгоритмів комп'ютерного зору, обробки зображень та чисельних алгоритмів загального призначення з відкритим кодом [6-12]. Зручність роботи з програмою Python також полягає в тому, що є можливість роботи з операційною системою Windows (на персональному комп'ютері) і на мікрокомп'ютері Raspberry Pi з операційною системою Raspbian. Однак необхідно стежити за тим, щоб версії використовуваних бібліотек в рамках даного проекту по-

вністю збігалися. Крім перерахованих вимог в СТЗ необхідно здійснювати введення даних від відеокамер в реальному масштабі часу, і при цьому вести складну обробку відеоданих. У монокулярних системах відеоспостереження це, наприклад, завдання виявлення і детектування осіб. У жорстких умовах обмежених ресурсів (апаратних і програмних) нерідко вдаються до многопотокової обробці вихідних відеоданих, що дозволяє істотно підвищити продуктивність обробки відеоданих.

Далі сформулюємо конкретні вимоги щодо побудови алгоритмів високоякісного введення, попередньої обробки та запису відео для сучасних мобільних СТЗ і наведемо приклади їх практичної реалізації у вигляді програмних кодів.

2.2 Підвищення якості введення і попередня обробка відео

Запорука успішного побудови сучасної мобільної СТЗ − комплексний системний підхід до управління якістю реєстрованих відеоданих. Для цього необхідно спільно вирішити такі завдання:

- забезпечити максимально можливу швидкість введення відеоданих при необхідній для цього додатка роздільної здатності відеокадру;
- провести один або кілька видів попередньої обробки зображень відео (preliminary image processing);
- записати результати реєстрації і обробки отриманих даних в відеофайл, параметри якого строго відповідають прийнятим мультимедійним стандартам.

Розглянемо специфіку вирішення цих та схожих завдань більш докладно.

Критерій оцінки якості введення даних. В першу чергу зупинимося на проблемі підвищення ефективності системи введення даних. З цією метою визначимо спільний критерій оцінки швидкості введення відеоданих і роздільної здатності кадрів. Для його створення введемо такі показники (і розшифруємо відповідні їм абревіатури):

- FPS − (Frame per second). Ця величина характеризує швидкість зміни/читання кадрів реєстрованого відеопотоку. Граничне значення FPS обмежено паспортними даними використовуваної відеокамери;
- FR – (Frame Resolution) описує роздільну здатність кадру відео в пікселях (320x240, 640x480, 1280x960 та ін.).

Ці найважливіші показники в основному і визначають якісні показники системи обробки відеоданих в СТЗ.

Традиційні методи захоплення відеоданих суттєво уповільнюють швидкість обробки відеоданих. Це найчастіше серйозно обмежує можливість роботи в реальному масштабі часу. Однак використовуючи багатопоточність можна помітно зменшити вплив затримки введення/виведення, залишаючи основний потік без блокування. Отже, існує можливість підвищення FPS аж до технологічної межі використовуваної відеокамери.

Зрозуміло, що при вирішенні задач візуалізації відеоданих завжди виправдано прагнення до підвищення якості за рахунок збільшення роздільної здатності кадру (FR). Однак слід пам'ятати, що велике збільшення числа пікселів в кадрі неминуче призводить до уповільнення (зменшення показника FPS). При цьому компроміс можна знайти експериментально − використовуючи пропоновані далі нові ефективні алгоритми захоплення і введення відеоданих.

Попередня обробка відеоданих це процес підвищення якості зображень відеопослідовності для отримання на основі оригіналів максимально точних і адаптованих для автоматичного аналізу зображень.

Методи попередньої обробки зображень відеопослідовності залежать від завдань дослідження і можуть включати в себе один або кілька таких видів робіт:

- корекція яскравості і контрастності кадрів;
- фільтрація зашумлених зображень;
- переклад кадрів відеопослідовності з колірного простору RGB;
- еквалізація даних;
- фільтрація даних;
- бінаризація зображень з відсіканням по порогу яскравості.

Корекція яскравості кадрів відеопотоку. Один з найбільш важливих видів попередньої обробки відеоданих - корекція яскравості і контрасту кадрів, що дозволяє істотно зменшити вплив змін освітленості сцени на якість відеоданих. Для кількісної оцінки цієї характеристики введемо FMB − (Frame medium brightness). Показник середнього рівня яскравості кадру об'єктивно відображає ступінь освітленості сцени в поточний момент часу. Знання показника середньої яскравості кадру відеопотоку (FMB) в умовах великої мінливості освітленості сцени дуже корисно. Це дозволяє за рахунок виконання порогових процедур встановлювати режим включення /скасування контрастування (еквалізациі) кадрів відеопотоку.

Крім цього, при низькому рівні яскравості кадру (порівнюючи FMB із заздалегідь визначеним порогом) в системах тривалого відеоспостереження зручно вмикати/вимикати підсвічування відеокамери.

Фільтрація зашумлених зображень. При істотному рівні шумів доцільно застосовувати різні процедури фільтрації кадрів реєструється відео-послідовності. В основному використовується три види фільтрів:

- усереднюючи (згладжуючи) НЧ фільтри;
- медіанні фільтри (ефективні для фільтрації імпульсних перешкод);
- фільтри розмиття зображень по Гаусу (gaussian blur) в основному для очищення від високочастотних шумів.

Відзначимо, що алгоритми фільтрації зображень досить добре вивчені і прості в програмній реалізації. Тому виконання таких процедур зазвичай утруднень не викликає.

Трансформація відео з RGB в Grayscale. Традиційно використовується колірний простір RGB дуже зручно для візуалізації сцени. Однак в різних технічних додатках (наприклад, траєкторних вимірюваннях) зображення відеокадрів доцільно переводити в колірній формат в відтінках сірого (grayscale), що містить

всього один канал. У ньому зазвичай використовують 30% червоного, 59% зеленого і 11% синього кольорів. Такий перехід сприяє зменшенню розмірності зображень і скорочує обсяги пам'яті для їх обробки.

Бінаризація кадрів відеопослідовності − це переклад повнокольорових або в градаціях сірого зображень в монохромні, де присутні лише два типи пікселів − 0 або 1 (чорні і білі). Такий перехід дуже продуктивний при розпізнаванні образів, визначенні координат об'єктів, проведенні траєкторних вимірювань та ін. Існують різні підходи до бінаризації, які умовно поділяються на 2 групи - порогові і адаптивні. Для порогових процедур з відсіканням по порогу яскравості використовують безліч різних критеріїв бінаризації, наприклад, Отсу, Бернса, Ейквеля, Ніблека і т.п. Найефективнішим, як по швидкодії, так і за якістю, вважається критерій Отсу. При бінаризації часто застосовують і морфологічні перетворення для розширення (дилатації) або звуження (ерозії) елементів бінарного зображення .

Запис результатів обробки в відеофайл. Найважливішим етапом при обробці відеоданих є запис отриманих результатів. При формуванні відеофайлів необхідно дотримуватися стандартних форматів відеозапису і стандартів стиснення (кодеків, що дозволяють істотно зменшувати розміри файлу). У нашій роботі будуть наведені докладні рекомендації та приклади написання програмних кодів зі створення файлів для зберігання цифрових відеоданих різних форматів: AVI (* .avi), MP4 (* .mp4), QuickTime (* .mov) і Windows Media Video (* .wmv) в комбінації з типовими кодеками: DIVX, XVID, X264 і MJPG.

2.3 Використовувані засоби програмування

При аналізі різних методів введення відеоданих в нашій роботі будемо орієнтуватися на використання мови програмування Python і доступних внутрішніх і зовнішніх ресурсів. Головна особливість написання програмних кодів на мові Python – формування необхідних можливостей проекту за рахунок підключення власних пакетів (наприклад, numpy, pip, pi-camera) і бібліотек (Pillow, Matplotlib та ін.). Так само суттєво збільшує програмні ресурси системи обробки і підключення зовнішніх бібліотек.

У нашому випадку це бібліотека OpenCV [8 – 11], імпортовані ресурси якої для вирішення поставлених завдань порівняно невеликі:

```
# import the necessary packages
from imutils.video import VideoStream
from imutils.video import FPS
import numpy as np
import argparse
import imutils
import datetime
import time
import os
import cv2
```

Підключення цих (а в деяких випадках і інших пакетів і модулів) не робить істотного навантаження на процесор і створює хороші передумови для обробки даних в реальному масштабі часу.

Зверніть увагу на використання порівняно рідко застосовуваних ресурсів.

`Imutils` – набір зручних функцій для спрощення основних операцій обробки зображень, таких як зсув, обертання, зміна розміру, відображення зображень в Matplotlib, за допомогою OpenCV і Python. Пакет `imutils` використовують і при створенні многопоточного класу Python. Він забезпечує доступ до вбудованої web-камері комп'ютера (зовнішньої камери з USB-підключенням або рі-камери) за допомогою функцій захоплення відеоданих OpenCV. Це дозволяє істотно збільшити FPS за рахунок створення нового потоку, який тільки опитує камеру на зчитування нових кадрів, а основний потік обробляє поточний кадр. Далі розглянемо цю технологію більш докладно.

`Argparse` – модуль Python для обробки опцій і аргументів командного рядка, з якої викликається скрипт. У нашій задачі модуль `argparse` раціонально використовувати для перетворення відеофайлів у звичайні зображення і їх збереження за допомогою бібліотеки OpenCV.

2.4 Глибока нейрона мережа в OpenCv

Склад модуля DNN

- API для створення нових слоїв, які по суті є будівельними одиницями нейронних мереж
- Безліч вбудованих найбільш поширених слоїв
- API для побудови і зміни нейронних мереж з слоїв
- Функціонал, необхідний для завантаження моделей нейронних мереж в форматах, які використовуються бібліотеками Caffe, Torch , TensorFlow

Рисунок 2.5 – Загальна схема тестування нейронної мережі

Основна можливість dnn полягає, звичайно ж, в завантаженні і запуску нейронних мереж (inference). При цьому модель може бути створена в будь-якому з трьох фреймворків глибокого навчання - Caffe, TensorFlow або Torch

Спосіб її завантаження і використання зберігається незалежно від того, де вона була створена.

При завантаженні відбувається конвертація моделей у внутрішнє представлення, близьке до використовуваного в Caffe. Так сталося в силу історичних причин - підтримка Caffe була додана найпершою. Однак взаємно однозначної відповідності між уявленнями немає.

Крім підтримки окремих верств, важлива також і підтримка конкретних архітектур нейронних мереж. Модуль містить приклади для класифікації (AlexNet , GoogLeNet , ResNet , SqueezeNet), сегментації (FCN , ENet), детектування об'єктів (SSD).

2.4.1 Середовище Caffe

Caffe представляє собою середу глибокого навчання , яка створювалася з урахуванням виразу , швидкості і модульного принципу. Вона була створена компанією Berkeley AI Research (BAIR) при участі членів спільноти . Янгквін Джіа (Yangqing Jia) створив проект під час своєї роботи над докторською дисертацією в Каліфорнійському університеті в Берклі . Бібліотека Caffe видана під ліцензією 2 статті про поширення програмного забезпечення університету Берклі .

Основні характеристики :

- виразна архітектура сприяє застосуванню і інноваційним розробкам ;
- розширюваний код стимулює активні розробки ;

- швидкість - Caffe пропонує один з найбільш швидких варіантів впровадження згортальних нейронних мереж (CNN).

Використовується для науково-дослідних проектів , прототипів стартапів і навіть масштабних виробничих програм в області зору , мови і мультимедіа.

2.4.2 Середовище Keras

Keras представляє собою API для нейронних мереж високого рівня , написаний на Python. Цю бібліотеку можна використовувати додатково до TensorFlow або Theano . Бібліотека Keras головним чином призначена для прискорення експериментів . Важливим умовою успішного дослідження є можливість переходу від задуму до результату з мінімальної можливої затримкою .

Основні характеристики:

- Зручність для користувача : зводить до мінімуму кількість дій користувача, необхідних в стандартних ситуаціях , а також забезпечує чіткі і практичні інструкції в разі помилки користувача .
- Модульне виконання : модель розглядається як послідовність або граф автономних повністю конфігуруються модулів , які можна з'єднати разом з мінімальним числом обмежень .
- Просте розширення : нові модулі легко додаються у вигляді нових класів і функцій , а існуючі модулі забезпечують досить прикладів .

2.4.3 Середовище TensorFlow

TensorFlow являє собою відкриту програмну бібліотеку для числових обчислень з використанням графів потоків даних. Вузли в графі представляють математичні операції, а гілки графа - багатовимірні масиви даних (тензори), якими вони обмінюються. Завдяки універсальної архітектурі обчислення можна розгортати на одному або декількох ЦП або графічних процесорах настільного комп'ютера, сервера або мобільного пристрою з одним API. TensorFlow спочатку був розроблений науковцями та інженерами для проведення машинного навчання і вивчення нейронних мереж, однак завдяки своїм загальним характером систему також можна використовувати в інших областях.

2.4.4 Середовище Theano

Theano - це бібліотека Python, призначена для визначення, оптимізації та оцінки математичних виразів, особливо виразів з багатовимірними масивами (numpy.ndarray). Theano дозволяє досягти швидкостей роботи з великими обсягами даних, які можуть конкурувати тільки із спеціально виготовленими програмами на мові CI. Theano також перевершує CI на ЦП у багато разів за рахунок використання останніх версій графічних процесорів. Theano поєднує в собі аспекти системи комп'ютерної алгебри (СКА) з аспектами оптимізуючого компілятора.

Таке поєднання СКА з оптимізуючій компіляцією особливо добре підходить для задач, в яких складні математичні вирази оцінюються багаторазово, тому для них важлива швидкість оцінки. У ситуаціях, коли оцінюється безліч різних виразів тільки один раз, Theano допомагає зменшити кількість непродуктивних компіляцій і аналізів, як і раніше забезпечуючи символьні функції, такі як автоматична диференціація.

Основні характеристики:

- Оптимізація швидкості виконання: використання g ++ або nvcc для компіляції частин графа виразів в інструкції ЦП або графічного процесора, які виконуються набагато швидше, ніж інструкції на чистому мовою Python
- Символьна диференціація: автоматичне створення символьних графів для обчислення градієнтів
- Підвищення стійкості: розпізнавання [деяких] нестійких в числовому відношенні виразів і обчислення їх з використанням більш стійких алгоритмів.

3 ПРОЕКТУВАННЯ І РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Створення класів для контролю відеопотоку

3.1.1 Створення класу FPS.

Важливим доповненням до функціональності багатопотокового видеоввода є визначення класу FPS. Цей клас використовується для візуалізації та оцінки швидкості введення даних. Він дає кількісні докази того, що многопоточність дійсно збільшує FPS.

```
fps = FPS().start()
while fvs.more():
...
...
    fps.update()
fps.stop()
printfps()
```

Результати обчислення поточних значень FPS відображаються на екрані монітора Raspberry Pi.

3.1.2 Створення класу FR

Frame Resolution є дуже важливим параметром кадру, від вибору якого залежить безпосередньо залежить FPS відеопотоку. Наприклад, найбільше можливе дозвіл рі-камери досягає FR = 1280x960. При цьому FPS досягає 30 кадрів в секунду.

Для вибору і контролю розмірів кадру був створений клас Resolution, що містить максимальний дозвіл кадру і пропорції при яких фрейм не деформується. Наведено фрагмент коду для реалізації цієї процедури:

```
fvs=VideoStream(usePiCamera(0)] >.start()
while fvs.more():
resolution.frame(fvs)
frame1=resolution.init(1,1)
frame2=resolution.init(2,1)
cv2.imshow("Frame1",frame1)
cv2.imshow("Frame2",frame2)
...
...
fvs.stop()
```

3.2 Методи введення відеоданих

3.2.1 Класичний метод введення

Класичний метод введення та обробки відеоданих з web-камери в системах монокулярного зору здійснюється за допомогою функції OpenCV для відеозахвату `cv2.VideoCapture(0)`. Наведемо фрагмент програмного коду на мові Python для практичної реалізації цього методу:

```
cv2.VideoCapture(0)
while True:
    (grabbed, frame) = stream.read()
    ...
    ...
    cv2.imshow("Frame", frame)
```

Однак функція відеозахвату `VideoCapture` і метод читання даних `read()` блокують основний потік програмного коду по обробці відеоданих до тих пір, поки кадр не буде лічений з пристрою камери, і не повернуто в основну програму.

3.2.2 Метод введення за рахунок апаратних засобів Raspberry Pi

Цей метод здійснюється за допомогою функції `pi-camera`. Однак при цьому виникає обмеження величини FSP через дії програмної навантаження. Цей метод не дозволяє використовувати дві і більше камер. Відзначимо також, що необхідна тимчасова затримка для обробки відображення кадрів. Програмний код цього методу має вигляд:

```
camera = PiCamera()
camera.framerate = 32.
Stream = camera.capture_continuous( rawCapture,format =
"bgr",use_video_port = True)
time.sleep(2.0)
for (i,f) in enumerate(stream):
frame = f.array
    ...
    ...
cv2.imshow("Frame", frame)
```

Слід зазначити, що функції `VideoCapture` і `PiCamera()` блокують основний потік програмного коду по обробці відеоданих до тих пір, поки кадр не буде лічений з пристрою камери, і не повернуто в основну програму. На жаль, цей метод, що відрізняється простотою, часто є головною перешкодою – він обмежує можливість обробки відеопотоку в реальному масштабі часу.

.3 Метод багатопотокового введення відеоданих

Цей метод передбачає створення класу `VideoStream()` для перенесення читання кадрів web-камери або USB-пристроїв в інший потік, абсолютно окремий від основного скрипту Python. Це дозволяє безперервно зчитувати кадри з потоку введення-виведення, поки основний потік обробляє поточний кадр. Як тільки основний потік завершив обробку чергового кадру, йому просто потрібно витягти поточний кадр з потоку вводу-виводу. Це досягається без очікування блокування операцій введення/виводу. Програмний код класу багатопотокового введення `VideoStream` наведено нижче:

```
fvs=VideoStream(usePiCamera= =args["picamera"] > 0).start()
time.sleep(2.0)
while True:
    frame = vs.read()
    cv2.imshow("Frame",frame)
fvs.stop()
```

3.3 Попередня обробка відеоданих

3.3.1 Стабілізація контрастності відеоданих

Реєстрація відеоданих зазвичай відбувається на тлі різного роду перешкод при постійному динамічному зміні фону спостерігається сцени. При цьому одним з домінуючих негативних факторів є мінливість освітленості. Відзначимо, що це можуть бути як швидко мінливі умови освітлення, так і повільні його зміни, зумовлені наприклад настанням сутінків. Все це призводить до погано контрольованим варіацій контрастності кадрів, і, отже, до погіршення якості обробки. Нагадаємо, що зазвичай зміна значень яскравості для відеоданих знаходиться в діапазоні чисел 0 – 255. Це відповідає формату даних `uint8`. Для подолання цих труднощів (погано контрольованих змін контрастності кадрів в залежності від освітленості) авторами запропоновано вихідну відеопослідовність з колірного простору `RGB` перетворити в простір `YUV` за допомогою функції

```
img_yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV).
```

Діапазон значень RGB становить [0 ÷ 255] для кожної компоненти, а для колірного простору YUV використовуються діапазони:

- `Y → [0 ÷ 255];`
- `U → [-112 ÷ 112];`
- `V → [-157 ÷ 157].`

Відмінною особливістю колірного простору `YUV` [5] є те, що в ньому використовується явне поділ інформації про яскравість і колір. Колір представляється у вигляді трьох компонент – яркостной (`Y`) і двох різноманітних (`U` і `V`).

Після перекладу кадру RGB відеопослідовності в колірний простір YUV в ньому здійснюється процедура еквалізації (підвищення контрастності) тільки компоненти Y за допомогою функції

```
img_iyuv[:,:,0] = cv2.equalizeHist(img_yuv[:,:,0]),
```

а потім відбувається зворотне переклад кадра з формату YUV в формат RGB:

```
img_output = cv2.cvtColor(img_yuv,cv2.COLOR_YUV2BGR).
```

При цьому баланс кольору зберігається без змін, так як різноманітних компоненти U і V перетворенням не наражалися.

Відзначимо, що перехід з колірного простору RGB в простір YUV дозволяє просто оцінити середній рівень яскравості кадру по компоненті Y. Найбільш об'єктивним і стійким показником на наш погляд є показник FMB (Frame medium brightness), який обчислюється як

$$FMB = \frac{1}{MN}\sum_{i=0}^{N}\sum_{j=0}^{M} Y(i,j),$$

де $Y(i,j)$ –двовимірний масив чисел, що визначають яскравість пікселів зображення кадру розміром MxN .

Цей показник тим більше корисний, що при малих рівнях яскравості кадру в системах відеоспостереження зручно використовувати процедуру його порівняння з попередньо встановленим порогом для автоматичного включення/вимикання системи підсвічування сцени.

Таким чином, використання процедури еквалізациі забезпечує вирівнювання гістограми розподілу яскравості і призводить показник середньої яскравості зображення до значення (FMB = 127) незалежно від того, яким цей показник був для вихідного зображення . Це дає можливість стабілізувати рівень середньої яскравості кадрів, і отже, усунути фактори нестабільності освітленості сцени, негативно впливають на подальше якість їх обробки.

3.3.2 Фільтрація зашумлених зображень

Наведемо деякі приклади використання різних фільтрів для обробки зображень відеопослідовності.

Для низькочастотної фільтрації (low pass filter) зазвичай використовують маски розмірністю розміром 3x3 і 5x5 пікселів

```
kernel_3x3=np.ones((3,3),np.float32)/9
kernel_5x5=np.ones((5,5),np.float32)/25,
```

а фільтрацію проводять за допомогою функції OpenCV, що реалізує процедуру двовимірної просторової згортки зображення з маскою фільтра:

```
output = cv2.filter2D(img, -1,kernel_3x3)
```

При збільшенні розміру маски збільшується і ступінь розмиття зображення.

Фільтрація Гауссовськіми фільтрами (Gaussian filters) проводиться за допомогою функції

```
img_gaussian = cv2.GaussianBlur(img,(n,n),0),
```

де `nxn` розмірність маски фільтра. Гауссівська фільтрація є згладжує, але на відміну від усереднювати, розмиття буде рівномірніше розподілятися від центру до країв. Фільтр Гауса є високочастотним, а значить, його вигідно застосовувати для підвищення різкості зображення .

При використанні медіанний фільтрів можна отримати хороші результати для того, щоб зберегти перепади відтінків, різних кордонів і усунути локальні піки яскравості на зображення x, які були спотворені імпульсним шумом. Процедура медіанної фільтрації забезпечується функцією

```
median = cv2.medianBlur(img,5)
```

3.3.3 Трансформація відео з RGB в Grayscale

Ця процедура, як і інші переходи з одного колірного простору в інше, проста і здійснюється за допомогою функції

```
gray = cv.cvtColor(img,cv.COLOR_RGB2GRAY).
```

3.3.4 Бінаризація кадрів відео з відсіканням по порогу яскравості

Цей метод дає можливість представити дані їх зображень в бінарному вигляді. Це дозволяє проводити з ними математичні обчислення. Алгоритм бінаризації використовує функцію бібліотеки OpenCV

```
(thresh,im_bw)    =    cv2.threshold(im_gray,0,255,cv2.THRESH_
BINARY|cv2.THRESH_OTSU)
```

з регульованим порогом відсікання яскравості `thresh`.

При бінаризації півтонового зображення із діапазоном значень пікселів (0 − 255) параметр `thresh` може змінюватися в межах від 10 до 253. Відзначимо, що при високому порозі відсікання на бінарному зображенні залишаться тільки пікселі з максимальною яскравістю (білого кольору). Всі інші пікселі будуть представлені нульовими значеннями (чорний колір).

3.4 Алгоритм програми зчитування відеоданих

Ця програма заснована на використанні описаних в нашій роботі алгоритмів введення і попередньої обробки прийнятих за допомогою web- або рі-камер вихідних відеоданих в мікрокомп'ютер Raspberry Pi. Узагальнена UML-діаграма роботи програми показана на рис. .

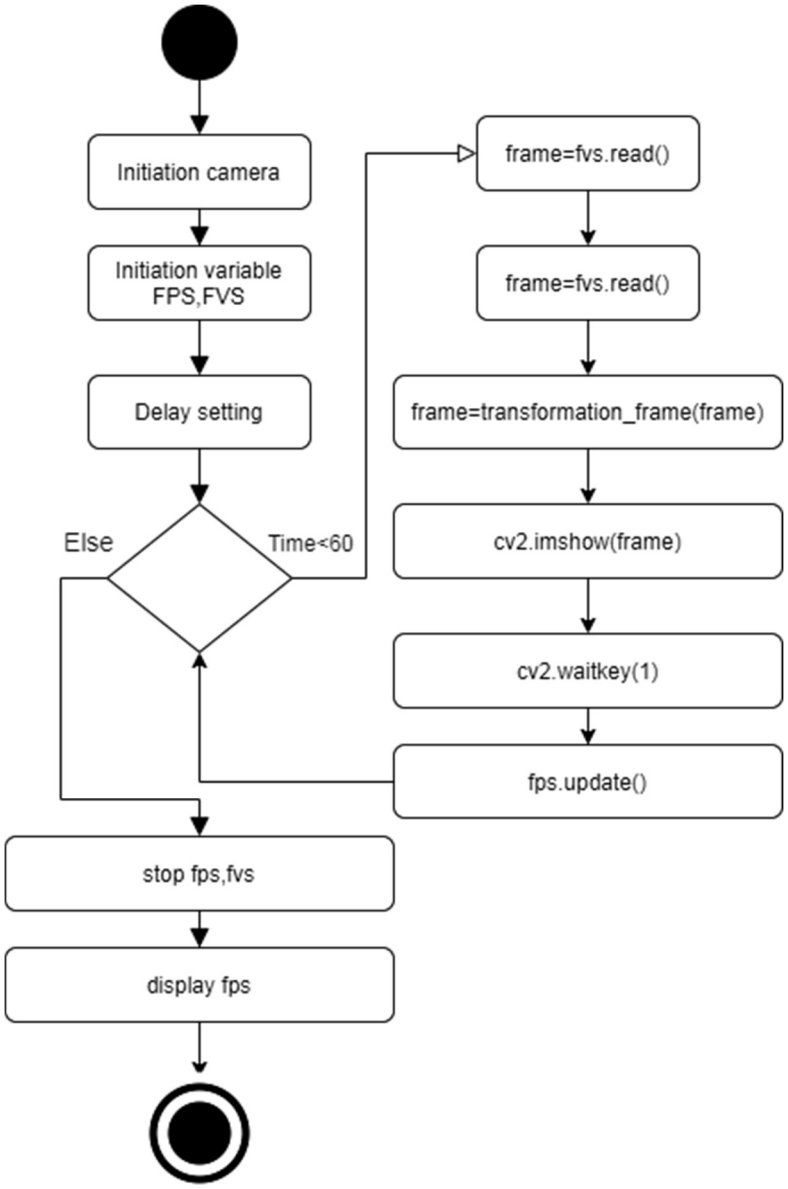


Рисунок 3.1 – UML-діаграма активності програми

3.5 Виявлення та детектування облич

У першому розділі цієї роботи було проведено огляд, детальна класифікація та аналіз методів і засобів пошуку і виявлення об’єктів при відеоспостереженні. Була сформульована загальна теоретична концепція вирішення цілого комплексу прикладних задач з виявлення та розпізнавання об’єктів, що знаходяться в зоні відео контролю. Слід зазначити, що саме завдання виявлення і розпізнавання облич є однією з перших практичних завдань, яка послужила стимулом для розвитку теорії розпізнавання об’єктів. Розпізнавання облич знайшло застосування в різних сферах людської діяльності. Цей напрямок сформувалося на початку

1980-х років, але його реальний розвиток почалося в 1990-х роках в процесі створення інформаційно-пошукових систем розпізнавання облич для ідентифікації особи.

У цьому розділі розглянемо питання, які стосуються виявлення та відстеження облич в відеопотоці з використанням найсучасніших алгоритмів і методів.

Можна використовувати різні підходи для вирішення конкретного завдання розпізнавання облич, проте слід пам’ятати, що завжди в більшій чи меншій мірі доведеться зіткнутися з проблемами:

- виявлення облич – це особливий випадок виявлення об’єкта, де завдання полягає в тому, щоб знайти місце розташування і розміри всіх облич на зображені;
- виявлення орієнтирів облич (окремий випадок виявлень орієнтирів, де завдання полягає в тому, щоб визначити основні орієнтири на обличчі);
- відстеження особи – необхідність знайти місце розташування і розміри всіх рухомих осіб в відео з урахуванням додаткової інформації, яка може бути залучена з послідовних кадрів відео;
- розпізнавання обличчя – це особливий випадок розпізнавання об’єкта, коли людина ідентифікується або перевіряється по зображенню або відео з використанням інформації, витягнутої з обличчя. Ідентифікація облич (1: N): завдання полягає в тому, щоб знайти найбільш близький збіг з невідомим чоловіком в наборі відомих облич.

Виявлення облич досягло вражаючого прогресу після роботи «Швидке виявлення об’єктів з використанням розширеного каскаду простих функцій», запропонованої Віолою і Джонсом в 2001 р., заснованої на використанні функцій Хаара [1]. В даний час цей метод став вже класичним. Після цього був запропонований ще ряд методів вирішення цього завдання на базі машинного і глибокого навчання. Ці методи отримали подальший розвиток до стадії практичного використання на базі бібліотек OpenCV і `dlib`.

OpenCV надає два підходи для виявлення облич:

- детектори облич на основі каскаду Хаара;
- детектори облич на основі глибокого навчання.

Розглянемо ці методи більш докладно.

3.6 Виявлення і детектування обличь за методом Віоли-Джонса

3.6.1 Принципи побудови методу

Основні принципи, на яких базується цей метод, такі:

- використовуються інтегральні зображення , що дозволяє швидко визначати шукані об'єкти;
- пошук потрібного об'єкта проводиться за допомогою ознак Хаара (особи і його рис);

- для вибору найбільш підходящих ознак шуканого об'єкта на даній частині зображення використовується бустінг (англ. Boost - поліпшення, посилення);
- всі ознаки надходять на вхід класифікатора, який дає результат «правда» або «брехня»;
- використовуються каскади ознак для швидкого відкидання вікон, де не знайдено обличчя.

Відзначимо, що найбільш трудомістким є процес навчання каскадів Хаара за допомогою алгоритму машинного навчання AdaBoost. Однак, в даний час для каскадних класифікаторів Хаара існує велика кількість вже навчених каскадів, в тому числі в стандартному постачанні бібліотеки OpenCV. Її установчий пакет містить цілий набір готових навчених класифікаторів, збережених у вигляді файлів з розширенням «* .xml». У цьому наборі є класифікатори, як для пошуку особи, так і його окремих частин (очей, рота, носа).

При синтезі алгоритмів розпізнавання найбільш продуктивно використання стандартних (заздалегідь навчених) класифікаторів. У зв'язку з цим доречно навести для довідки список основних попередньо навчених класифікаторів:

- `haarcascade_eye_tree_eyeglasses.xml;`
- `haarcascade_profileface.xml;`
- `haarcascade_eye.xml.`

Не зупиняючись детально на призначення і властивості окремих класифікаторів, відзначимо, що особливий інтерес серед них представляють ті, які дозволяють визначити положення очей. Це дуже корисна процедура, оскільки вона дає можливість оцінки відстаней між зіницями. Це при подальшому аналізі дозволяє усувати чинники, пов'язані з нахилом голови. Класифікатори, навчені пошуку очей, дуже чутливі до наявності очок. У більшості випадків вони дають збій, особливо для окулярів повністю приховують очі. У таких випадках слід використовувати більш ефективний класифікатор для виявлення очей

`haarcascade_eye_tree_eyeglasses.xml»`,

навчений пошуку на зображенні очей в окулярах. Його можна використовувати як запасний варіант у випадках збою в роботі звичайного класифікатора.

Найбільш ефективними є методи пошуку основних елементів в уже виділеної області обличчя, так як це локалізує регіон пошуку і скорочує час аналізу, а також, значно знижує ймовірність помилкових спрацьовувань. Вибір областей інтересу для виявлення очей, носа і рота вимагає оптимізації положення і розмірів області інтересу для кожного елемента. Приклад такого розбиття виділеної області на зони пошуку окремих елементів обличчя в нашому алгоритмі показаний на рис.3.6. Розміри областей пошуку алгоритму зазвичай визначаються експериментально. Зверніть увагу, що існує дві можливості виявлення очей − виділення загальної для двох очей області інтересу (або області пошуку) і пошук і

виявлення лівого і правого ока окремо. У OpenCV для цього передбачені відповідні класифікатори. Який з цих методів краще, визначається експериментальним шляхом.

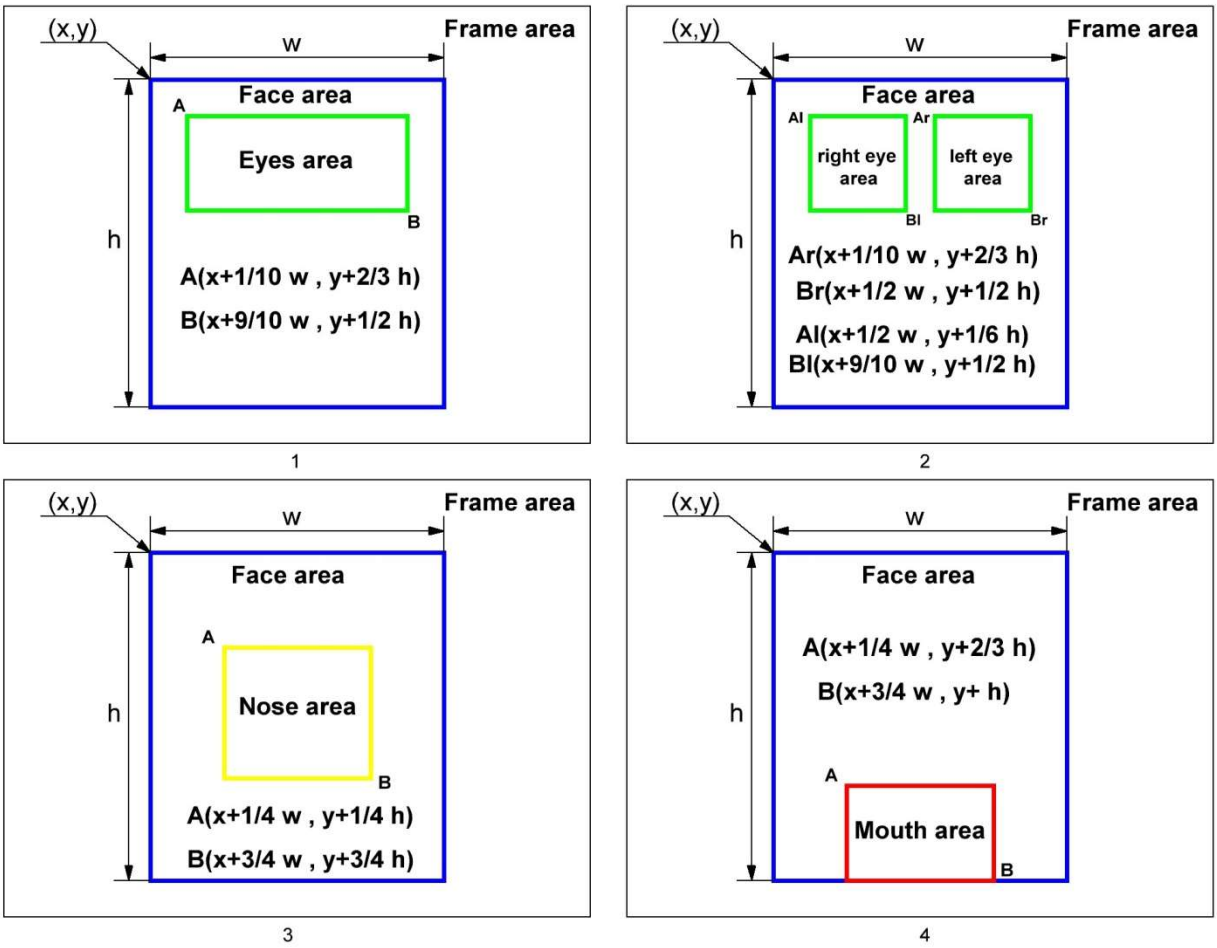


Рисунок 3.2 – Зони пошуку облич та їх елементів в кадрі
(координати окремих зон пошуку)

3.6.2 Програма детектування обличь за методом Віоли-Джонса

Робочі версії алгоритмів виявлення обличь створювалися з урахуванням описаних вище підходів і ресурсів. Було поставлено завдання виявлення обличь, очей, носа і рота. При цьому виявлення очей здійснювалося двома способами – з використанням класифікатора для детектування очей в загальній зоні пошуку і класифікаторів роздільного виявлення (правого і лівого ока окремо). Далі порівняємо ці два способи по показнику ймовірності виявлення очей.

Для економії ресурсів і простоти реалізації алгоритмів використовувався набір попередньо навчених каскадних класифікаторів Хаара для відповідних елементів обличчя, імпортований з бібліотеки OpenCV. Доречно нагадати про необхідність розмістити ці класифікатори в кореневій папці Python. Це допоможе

уникнути помилок при пошуку шляхів звернення до них і прискорити обробку вихідних даних.

Перелік використаних класифікаторів наступний:

- `face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')`
- `eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')`

Для адаптації пропонованих алгоритмів детектування облич до умов зміни зовнішнього освітлення сцени запропоновано організувати два канали обробки відеоданих. Схема такого алгоритму показана на рис. 4.2.

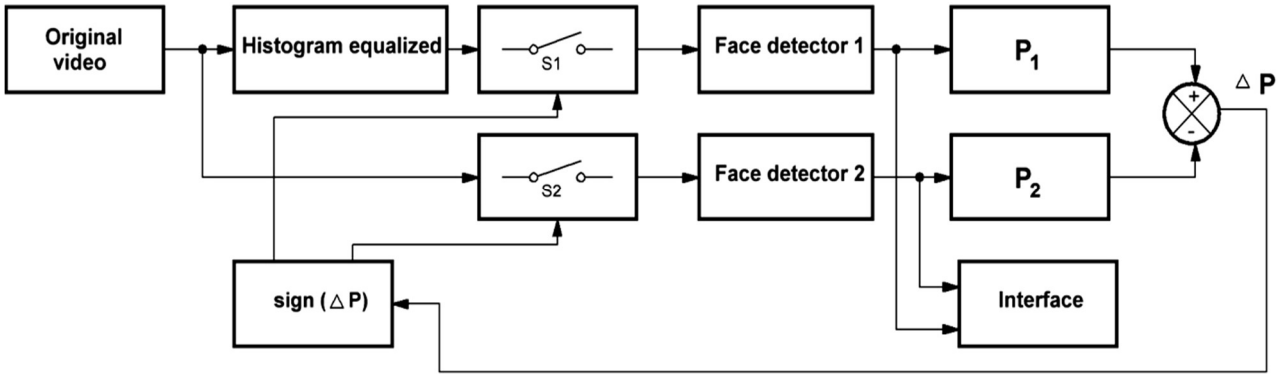


Рисунок 3.3 – Структура адаптивного алгоритму детектування облич

В одному з них вихідний відеопотік використовується безпосередньо для виявлення облич, а в другому каналі проводиться його попередня еквалізація для приведення яскравості кадру до середнього значення. На виходах обох каналів обробки в ковзних вікнах розміром в 100 кадрів обчислюються ймовірності правильного виявлення осіб і їх елементів, а потім їх зіставляють в блоці порівняння. Залежно від знаку різниці ймовірностей *ΔPn* за допомогою петлі зворотного зв'язку на вихід системи передаються результати виявлення облич з того каналу, в якому умови освітлення більш комфортні для роботи алгоритму виявлення облич.

Оскільки методи і засоби створення алгоритмів вже обговорювалися досить докладно, далі зупинимося тільки на структурні особливості побудови нових алгоритмів.

Розглянемо два різних варіанти побудови адаптивних алгоритмів виявлення облич і їх фрагментів на відеопослідовність і оцінимо переваги і недоліки кожного з них. Структурні схеми цих алгоритмів наведені на рис. 4.3 і рис. 4.4. У першому варіанті повністю реалізований принцип адаптації до змін рівня освітленості сцени, заснований на визначенні значень поточної різниці ймовірностей правильного виявлення осіб і їх елементів при використанні процедури еквалізациі кадру і без неї (див. рис. 7.2). Основним недоліком такого методу є помітне зменшення швидкодії, оскільки в цій ситуації обсяг обчислювальних операцій фактично подвоюється.

Такий підхід суттєво економніше − кількість обчислень практично вдвічі менше ніж в першому варіанті побудови алгоритму. Це дозволяє впевнено здійснювати обробку відеоданих в реальному масштабі часу.

Результати виявлення одного або декількох облич за допомогою запропонованих алгоритмів показані на рис. 7.2. Слід зазначити, що найбільш стійкі результати виявлення особи та його елементів спостерігаються за умови фронтально розташованого по відношенню до особи джерела освітлення сцени, Сприятливий геометричний фактор процедури виявлення передбачає нахили голови не більше 30º.

3.6.3 Реалізація алгоритму Віоли-Джонса

Саме детектування відбувається дуже просто. Код розглянуто нижче. Функція детектування за алгоритмом Віоли-Джонса обличь приймає як вхідний параметр фрейм, повертає змінений фрейм `current_frame` і параметр відповіщаючий за вдатність детектування обличчя `face_detected`.

```
face_detected = False
gray = cv2.cvtColor(current_frame, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray, 1.3, minNeigbors)
for (x, y, w, h) in faces:
    face_detected = True
```

Додання рамки біля відображеного обличчя

```
    if anonymaze==0:
cv2.rectangle(curent_frame, (x, y),(x + w, y + h),(0, 0, 255), 2)
        current_frame[startY:endY, startX:endX] = face
```

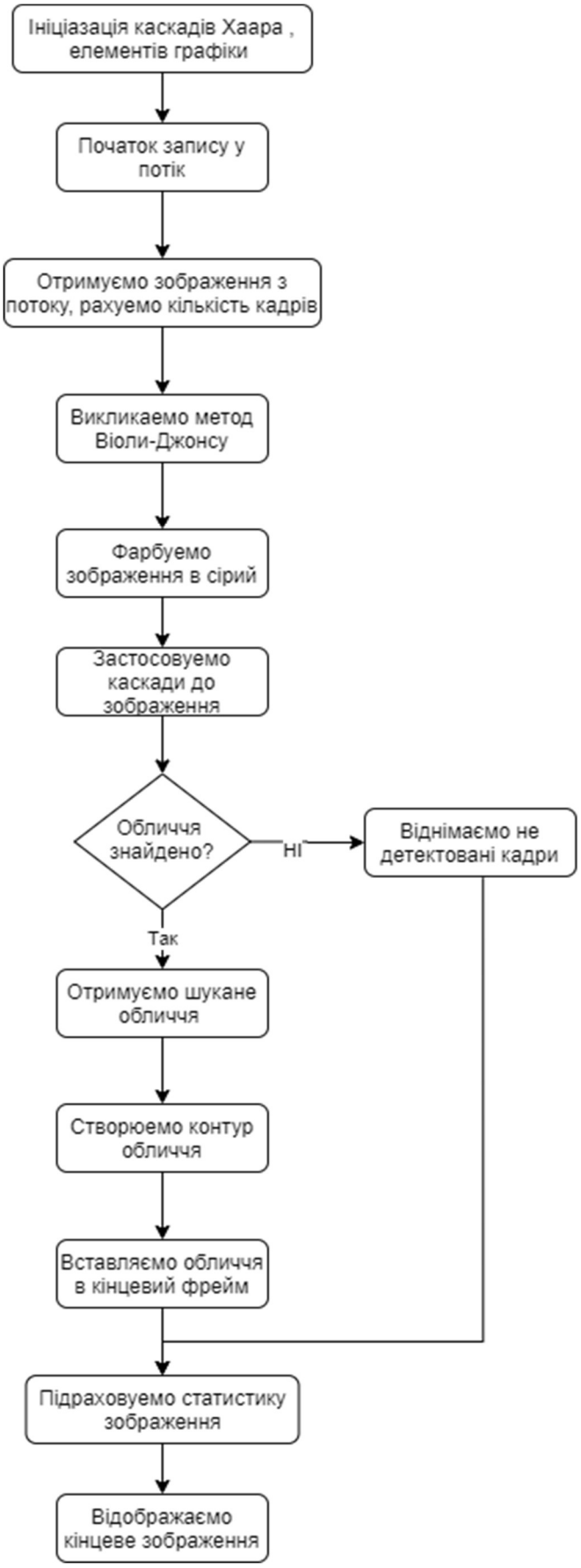


Рисунок 3.6 – Алгоритм Віоли-Джонсу

3.7. Виявлення і детектування обличь за методом Кофі

3.7.1 Принцип побудови методу

Метод Кофі базується на методі глибокої нейронної мережи

Глибинна нейронна мережа - це штучна нейронна мережа (ШНМ) з декількома шарами між вхідним і вихідним шарами. ДПС знаходить коректний метод математичних перетворень, щоб перетворити вихідні дані в виходять, незалежно від лінійної або нелінійної кореляції. Мережа просувається по вагам, розраховуючи ймовірність кожного виходу. Користувач може переглянути результати і вибрати ймовірності, які повинна відображати мережу (вище певного порогу, наприклад), і повернути в мережу запропоновану мітку. Кожне математичне перетворення вважається шаром, а складні ДПС мають багато шарів, звідси і назва «глибинні» або «глибокі» мережі.

Переваги методу є:

1) високий шанс детектування – на великій вибірці обличь шанс детектування може сягати вище 90 відсотків, так як це зв’язано з більшою кількістю параметрів приємної моделі.

2) детектування обличь неправильної форми – так як модель не залежить від строгих геометричних фігур і примітивів, скручені обличчя мають більший шанс на детектування.

3) детектування повної області обличчя при замуленні або приховуванні частини особи (окуляри, шарф , пов’язка)

4) малий шанс помилкового спрацьовування, так як багато факторів для детектування і не використовується сусідні пікселі, то алгоритм не спрацьовує повторно.

3.7.2 Підготовка файлів

Ще в серпні 2017 року було офіційно випущений OpenCV 3.3, в який був доданий модуль «глибоких нейронних мереж» (dnn).Цей модуль підтримує ряд основ глибокого навчання, включаючи Caffe, TensorFlow і Torch / PyTorch.

Детектор особи на основі Caffe можна знайти в підкаталозі face_detector прикладів dnn .

При використанні модуля глибокої нейронної мережі OpenCV з моделями Caffe були підключені два набори файлів:

1) файл. prototxt , який визначає архітектуру моделі (тобто самі шари)
2) файл. caffemodel , який містить ваги для фактичних слоїв

Обидва файли потрібні при використанні моделей, навчених використання Caffe для глибокого вивчення.

a. Стійкість до зміни освітлення, навіть якщо це локальна зміна освітлення, стійкість до шумів (примітиви є найпростіший смуговий фільтр).

b. Якщо примітиви були не дуже маленькі, то сильно стійкіше кореляції при зміні масштабу (розмір примітивів при цьому не буде впливати на точність, якщо обхід з маленьким кроком).
c. Якщо ознаки на великому зображенні розрахувати заздалегідь і при зсуві вікна пошуку брати вже пораховані і актуальні для нього - пошук значно швидше кореляції (потрібно порівняти меншу кількість елементів).

Підключення бібліотек

```
from pyimagesearch.face_blurring import anonymize_face_pixela
from pyimagesearch.face_blurring import anonymize_face_simple
from imutils.video import VideoStream
import numpy as np
import argparse
import imutils
import time
import cv2
import os
```

Загрузка файлів

```
proto="all/detecting/deep-learning-face-
detection/deploy.prototxt.txt"
caffe="all/detecting/deep-learning-face-
detection/res10_300x300_ssd_iter_140000.caffemodel"
net = cv2.dnn.readNetFromCaffe(proto,caffe)
```

Підключення web-камери

```
vs = VideoStream(0).start()
time.sleep(1)
```

3.7.3 Алгоритм детектування за методом Кофі

Саме детектування відбувається дуже просто. Основні кроки розглянуті нижче:

1) Зображення ріжеться на площини.
2) Створюється краплю (blob), яка буде передаватися і порівнюватися з вагами з файлу caffe пізніше, ми задаємо її розміри і координати.
3) Крапля передається в модуль net.
4) Вихідне зображення виходить з модуля net.

Код виконуючий основні дії:

```
(h, w) = frame.shape[:2]
   blob = cv2.dnn.blobFromImage(frame, 1.0, (300, 300),(1
04.0, 177.0, 123.0))
 net.setInput(blob)
  detections = net.forward()
    for i in range(0, detections.shape[2]):
         confidence = detections[0, 0, i, 2]
if confidence < const_confidence: continue
box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
        (startX, startY, endX, endY) = box.astype("int")
```

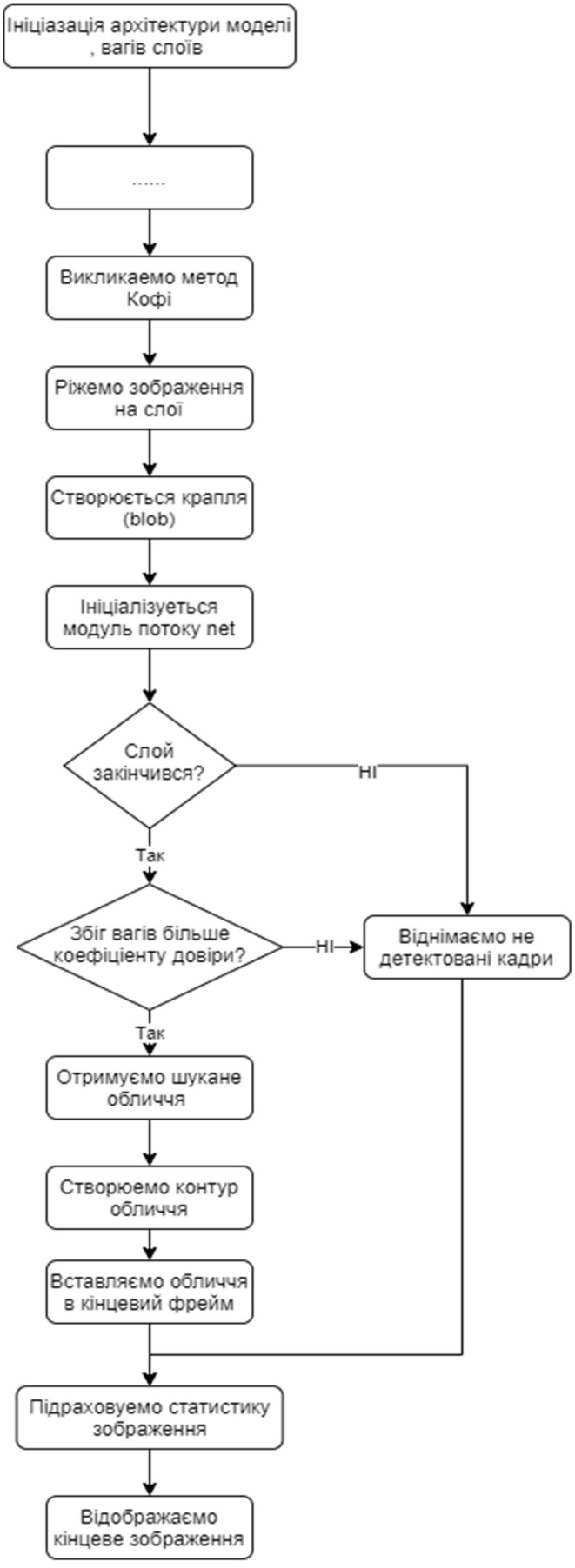


Рисунок 3.7 – Алгоритм методу Кофі

Операції котрі лежать за окремими функціями будуть розглянуті у пунктах нижче більш детально.

3.7.4 Виявлення крапок (blobFromImage)

У комп'ютерному зорі методи виявлення крапок спрямовані на виявлення областей у цифровому зображенні, які відрізняються за властивостями, такими як яскравість або колір, порівняно з навколишніми регіонами. Неофіційно крапля - це область зображення , в якій деякі властивості постійні або приблизно постійні; всі точки в краплі можна вважати в деякому сенсі схожими один на одного. Найпоширеніший метод виявлення краплі - це згортка.

Враховуючи деяку властивість інтересу, виражену функцією позиції на зображенні, є два основні класи детекторів крапок: (i) диференціальні методи, що базуються на похідних функції щодо позиції, та (ii) методи, засновані на локальні екстремуми, які ґрунтуються на знаходженні локальних максимумів і мінімумів функції. З більш пізньою термінологією, яка використовується в даній області, ці детектори можна також називати операторами точки інтересів, або альтернативно, операторами регіону інтересів (див. Також виявлення точки інтересу та виявлення кута).Існує кілька мотивацій для вивчення та розробки детекторів каплі.

Однією з головних причин є надання додаткової інформації про регіони, яка не отримується від крайових детекторів або кутових детекторів. Під час ранньої роботи в цьому районі виявлення каплі було використано для отримання регіонів, що представляють інтерес для подальшої обробки. Ці регіони можуть сигналізувати про наявність об'єктів або частин об'єктів у області зображення із застосуванням для розпізнавання об'єктів та / або відстеження об'єктів.

В інших областях, таких як аналіз гістограми, дескриптори капля можуть також використовуватися для виявлення піків із застосуванням до сегментації. Ще одне поширене використання дескрипторів крапок - як основних примітивів для аналізу текстури та розпізнавання текстур. В останніх роботах дескриптори каплі знайшли все більшу популярність як точки інтересів для широкого базового стерео збереження та сигналізації про наявність інформативних функцій зображення для розпізнавання об'єктів на основі зовнішнього вигляду на основі локальних статистичних зображень. Існує також пов'язане поняття виявлення хребта для сигналізації наявності витягнутих предметів.

```
blob = cv2.dnn.blobFromImage(image, scalefactor=1.0, size, mean, swapRB=True)
```

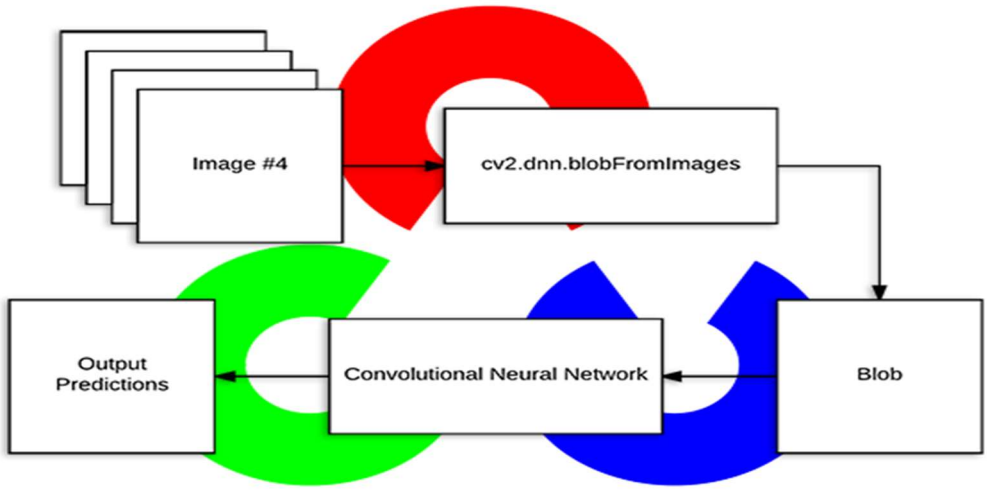


Рисунок 3.8 – Схематичний алгоритм детектування крапок

Де параметри:

1) image- це вхідне зображення , яке ми хочемо попередньо обробити, перш ніж передати його через нашу глибоку нейронну мережу для класифікації.
2) масштабний фактор після виконання середнього віднімання ми можемо необов'язково масштабувати наші зображення за деяким фактором. Це значення за замовчуванням до "1,0" (тобто без масштабування), але ми можемо також подати інше значення. Це також важливо зазначити
3) size Тут ми надаємо просторовий розмір, який очікує Конволюційна нейронна мережа. Для більшості сучасних нейронних мереж це або 224 × 224 , 227 × 227 , або 299 × 299 .
4) mean це наші середні значення віднімання. Вони можуть бути 3-кратними засобами RGB або можуть бути одиничним значенням; у цьому випадку подане значення віднімається з кожного каналу зображення . Якщо ви виконуєте середнє віднімання, переконайтеся, що ви постачаєте 3-х кордону в порядку "(R, G, B)".
5) swapRB - OpenCV передбачає, що зображення розташовані в порядку каналу BGR; однак значення "середнє" передбачає, що ми використовуємо порядок RGB. Щоб вирішити цю невідповідність, ми можемо замінити канали R і B
6) image встановивши це значення на "True". За замовчуванням OpenCV здійснює заміну цього каналу для нас.

3.7.5 Обробка зображення за допомогою нейронної сітки (класс net)

Цей клас дозволяє створювати та керувати комплексними штучними нейронними мережами.

Нейронна мережа представлена у вигляді керованого ациклічного графіка (DAG), де вершини - це екземпляри рівня, а ребра задають зв'язки між входами та виходами шарів.

Кожен мережевий шар має унікальний цілий ідентифікатор та унікальну назву рядка всередині своєї мережі. Ідентифікатор може зберігати ім’я шару або ідентифікатор шару.

Цей клас підтримує підрахунок посилань на його екземпляри, тобто його копії вказують на той самий екземпляр.

3.8 Анонімізація зображень

Анонімізацією зображення – це замулення початкового зображення в області обличчя для надання людині на зображення права конфіденційності і безпеки приватної особи.

В основі анонімізації лежить виділення області обличчя і замулення пікселей області одним із методів. В обробці застосована лише виділена область.

3.8.1 Основний алгоритм програми

Був створений цикл, що проходить по шару вихідного зображення .

Всередині циклу ми визначаємо процентне відношення кількості розпізнаного зображення до заданого числа, при низькому значенні confidence зображення буде фільтруватися навіть якщо особа майже повністю закрито, а при високому закриття кожного з основного масиву, дестабілізує фільтрацію.

Після було отримано рамка і розміри фільтра.

І застосований один з методів анонімізації.

Остання дія закриває вихідне зображення отриманим відфільтрованим ділянкою.

Код виконуючий анонімізацію зображення :

```
for i in range(0, detections.shape[2]):
    confidence = detections[0, 0, i, 2]
    if confidence > 0.5:
box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
        (startX, startY, endX, endY) = box.astype("int")
        face = frame[startY:endY, startX:endX]
        if method== "simple":
            face = anonymize_face_simple()
        else:
          face = anonymize_face_pixelate(face,blocks=100)
        frame[startY:endY, startX:endX] = face
```

3.8.2 Анонімізація за допомогою методу Гауса

Алгоритм Гауса застосовуючись у двох вимірах, ця формула утворює гауссову поверхню, яка має максимум у початку координат, контури якого - це концентричні кола з центром початку координат. Двовимірна матриця згортки попередньо обчислюється з формули та складається із двовимірних даних. Кожному елементу в результуючій матриці нове значення встановлюється середньозваженим для сусідства цих елементів. Фокальний елемент отримує найбільшу вагу (має найвище значення Гауса), а сусідні елементи отримують менші ваги, оскільки їх відстань до фокального елемента збільшується. При обробці зображень кожен елемент у матриці представляє атрибут пікселя, такий як яскравість або інтенсивність кольору, а загальний ефект називається розмиттям Гауса.

Алгоритм функції :

1) отримано матрицю зображення;
2) визначено розмір розмиття;
3) так як функція розмиття передбачає, що об'єкт розмиття не має парних розмірів, то ми їх робимо непарними;
4) перевіряємо на позитивність розмірів краплі;
5) за допомогою гаусового розмиття отримуємо необхідний фрагмент.

```
def anonymize_face_simple(image,3.0):
    (h, w) = image.shape[:2]
    kW = int(w / 5 - factor)
    kH = int(h / 5 - factor)

    if kW % 2 == 0:
        kW -= 1
    if kH % 2 == 0:
        kH -= 1
    if( kW<0 or  kH<0):
        return image
    image2=cv2.GaussianBlur(image, (kW, kH), 0)
    if (image2 is None):
        image2=image
    return image2
```

3.8.3 Методика анонімізації за допомогою укрупнення пікселів

Ідея алгоритма полягає в звичайному усередненні пікселів на певній області, котра вибирається по принципу кількості пікселів на обличчя.

Алгоритм функції :

1. Отримано матрицю зображення.
2. Отримано розміри пікселів.
3. Створюється цикл, що проходить по матриці зображення з кроком в розмір пікселя.

4. У циклі визначаються координати пікселя.
5. У циклі піксель отримує усереднений колір рамки.
6. Метод повертає вихідне зображення

```
def anonymize_face_pixelate(image, blocks=3):
    (h, w) = image.shape[:2]
    xSteps = np.linspace(0, w, blocks + 1, dtype="int")
    ySteps = np.linspace(0, h, blocks + 1, dtype="int")
    for i in range(1, len(ySteps)):
        for j in range(1, len(xSteps)):
            startX = xSteps[j - 1]
            startY = ySteps[i - 1]
            endX = xSteps[j]
            endY = ySteps[i]
            roi = image[startY:endY, startX:endX]
            (B, G, R) = [int(x) for x in cv2.mean(roi)[:3]]
            cv2.rectangle(image, (startX, startY), (endX, end
Y),(B, G, R), -1)

    return image
```

3.9 Різниця між алгоритмом Віоли-Джонса и Кофі

3.9.1 Визначення

В алгоритмі Віоли-Джонсу використовується варіація алгоритму AdaBoost, як для навчання , так і для налаштування класифікаторів. Котрий в своє чергу є алгоритмом машинного навчання.

В свою чергу алгоритм Кофі відноситься до сімейства алгоритмів глибоких нейронних мереж. Кофі забезпечує нейронні мережі, RCNN, довгу короткострокову пам'ять і повно зв’язні нейронні мережі. При цьому для прискорення навчання застосовується система графічних процесорів підтримувана архітектурою CUDA і бібліотеку CuDNN від фірми Nvidia.

Якщо відкинути деталі і зосередитися на суті можна виразити як.

Машинне навчання – це підмножина штучного інтелекту, пов'язане зі створенням алгоритмів, які можуть змінювати себе без втручання людини для отримання бажаного результату - шляхом подачі себе через структуровані дані.

Глибоке навчання – це підмножина машинного навчання, де алгоритми створюються і функціонують аналогічно машинного навчання, але існує безліч рівнів цих алгоритмів, кожен з яких забезпечує різну інтерпретацію даних, які він передає. Така мережа алгоритмів називається штучними нейронними мережами. Простими словами, це нагадує нейронні зв'язки, які є в людському мозку.

Основна відмінність між глибоким навчанням і машинним навчанням обумовлено тим, як дані представляються в систему. Алгоритми машинного навчання майже завжди вимагають структурованих даних, в той час як мережі глибокого навчання покладаються на шари ANN (штучні нейронні мережі).

3.9.2 Тезиси різниці між алгоритмами

Алгоритми машинного навчання майже завжди вимагають структурованих даних, в той час як мережі глибокого навчання покладаються на шари ANN (штучні нейронні мережі).

Алгоритми машинного навчання створені для того, щоб «вчитися» діяти, розуміючи помічені дані, а потім використовувати їх для отримання нових результатів з великою кількістю наборів даних. Однак, коли результат виходить невірним, виникає необхідність їх «доучувати».

Мережі глибокого навчання не вимагають втручання людини, так як багаторівневі шари в нейронних мережах поміщають дані в ієрархії різних концепцій, які в кінцевому підсумку вчаться на власних помилках. Проте, навіть вони можуть бути помилковими, якщо якість даних недостатньо гарне.

Оскільки алгоритми машинного навчання вимагають маркованих даних, вони не підходять для вирішення складних запитів, які включають в себе величезну кількість даних.

.

3.9.3 Використання алгоритмів

Машине навчання буде використовуватися коли даних буде достатньо для навчання, і потім він продовжить роботу на основі зрозумілих їм маркувань і класифікує мільйони інших зображень обох тварин за ознаками, які він вивчив раніше.

Нейронні мережі будуть використовувати інший підхід для вирішення цієї проблеми. Основною перевагою є те, що тут не обов'язково потрібні структуровані / помічені дані зображень для класифікації двох тварин. В даному випадку, вхідні дані (дані зображень) відправляються через різні рівні нейронних мереж, причому кожна мережа ієрархічно визначає специфічні особливості зображень.

Це схоже на те, як наш людський мозок працює для вирішення проблем - пропускає запити через різні ієрархії концепцій і пов'язаних питань, щоб знайти відповідь.

Після обробки даних через різні рівні нейронних мережах система знаходить відповідні ідентифікатори для класифікації зображень.

Хоч в даному випадку ми побачили застосування глибокого навчання для вирішення незначного запиту, - реальне застосування нейронних мереж глибокого навчання відбувається в набагато більшому масштабі. Фактично, з огляду на кількість шарів, ієрархій і концепцій, які обробляють ці мережі, глибоке навчання підходить тільки для виконання складних обчислень, а не простих

Однак слід знати, що глибоке навчання вимагає набагато більше даних, ніж традиційний алгоритм машинного навчання. Причиною цього є те, що мережі глибокого навчання можуть ідентифікувати різні елементи в шарах нейронних мереж тільки при взаємодії понад мільйон точок даних. Алгоритми машинного навчання, з іншого боку, здатні навчатися по заздалегідь запрограмованим заданим критеріям.

3.9.4 Різниця в реалізації

В основному застосування алгоритмів має однакову основу, але параметри і деталі різняться.

Так наприклад алгоритм Кофі має лише один мінливий параметр – відсоток збігу ідентифікаторів на зображені і в базі.В той же час алгоритм Віоли-Джонса має параметри кількості сусідніх пікселей і коефіцієнт масштабу.

Також виклик каскаду повертає детектоване зображення , в різницю цьому Кофі повертає відсоток збігу.

Але самою суттєвою різницею є застосований цикл в алгоритмах. В алгоритмі Віоли-Джонса цикл потрібен лише для виділення обличчя і не застосовується для детектування , тому якщо на зображені буде два обличчя контур яких збігається, то алгоритм найде лише одне обличчя, але можливо із-за параметра сусідніх пікселей, друге обличчя теж буде найденим але його контур буде в іншому місті.

В алгоритмі Кофі всередині цикл виконується виклик каскаду і ми проходим за всім зображенням поетапно тому і буде детектоване декілька обличь без перешкод.

4 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ «VIDEO STREAM QUALITY CONTROL»

4.1 Програма досліджень «Video Stream Quality Control»

Для оцінки ефективності пропонованих методів підвищення якості відеоданих були проведені експериментальні дослідження за допомогою відеореєстраторів, показаних на рис. 2.4, мікрокомп'ютера Raspberry Pi і програми «`Video Stream Quality Control`». Вона написана на мові Python із залученням відповідних ресурсів бібліотеки OpenCV. Ця програма заснована на використанні описаних вище алгоритмів введення відеоданих, прийнятих за допомогою web- або pi-камер, попередньої обробки та запису в файл отриманих даних.

З урахуванням комплексного характеру проведених досліджень в програму були закладені можливості організації різних варіантів алгоритму відеозахвату і введення даних (класичний – з функцією `cv2.VideoCapture(0)`, прискорений – за рахунок апаратних засобів комп'ютера Raspberry Pi, а також найшвидший – багатопотокового введення за допомогою спеціально для цього створюється класу `VideoStream`. Ці варіанти введення відео передбачені для різних значень роздільної здатності екрану.

Крім цього програма «`Video Stream Quality Control`» дозволяє провести попередню обробку зареєстрованих даних (в умовах недостатньої освітленості сцени можна використовувати режим стабілізації яскравості кадрів відеопотоку, при значному рівні шумів можна провести фільтрацію зашумлених зображень за допомогою набору фільтрів). У програмі передбачена можливість запису інформації в відеофайли потрібного формату з різними кодеками стиснення і можливістю подальшого їх зберігання.

4.1.1 Інтерфейс програми

Обговоримо можливості створеного програмного забезпечення, найважливішим елементом якого є призначений для користувача інтерфейс.

На рис. 3.1 показаний вигляд стартового вікна програми. Після натискання кнопки «Start» програма переходить в вікно установок (рис. 3.2). У ньому представлений ряд сервісних можливостей. В першу чергу це можливість інтерактивно провести необхідні попередні установки програми.

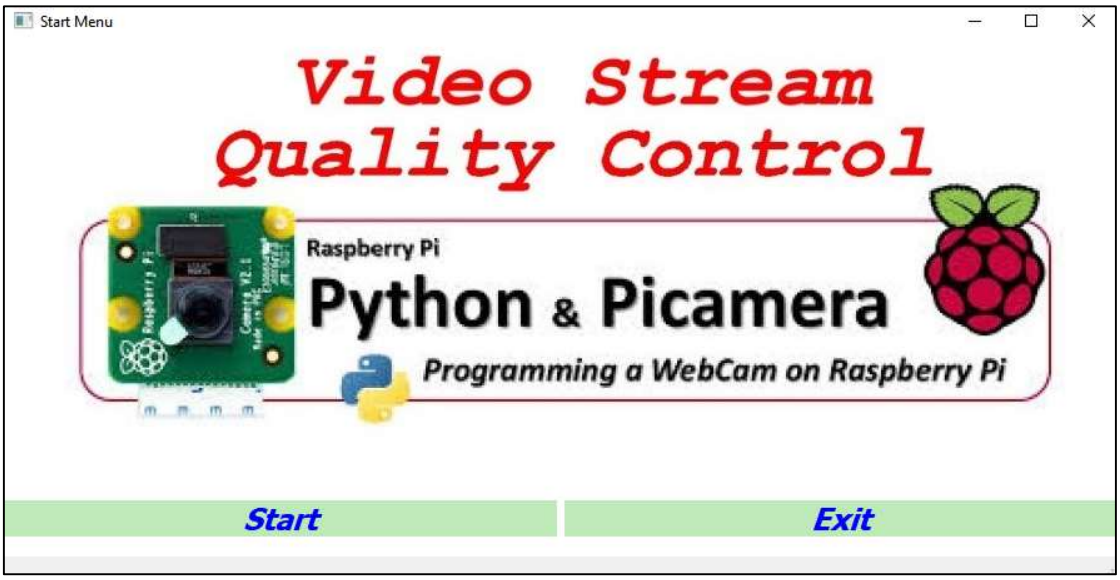
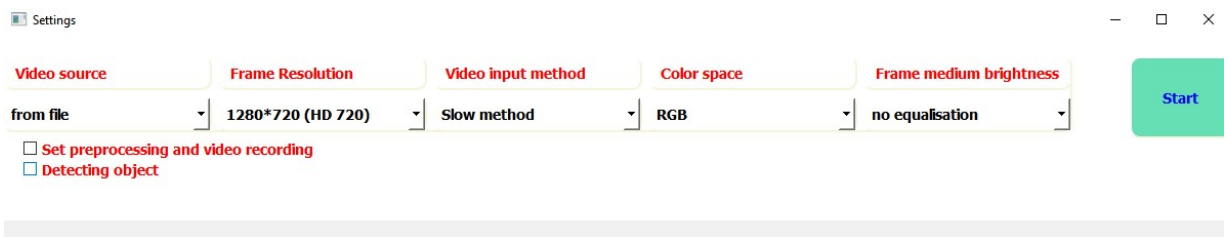
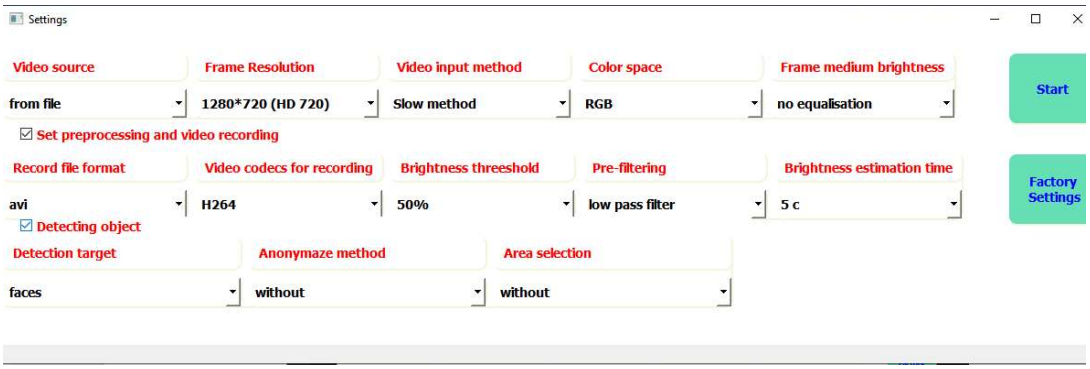


Рисунок 4.1 – Стартове вікно програми



а



б

Рисунок 4.2 – Вікно установок програми «Video Stream Quality Control»
а – установки для введення відеоданих, б – повний набір установок

На рис. 4.2а показаний зовнішній вигляд вікна установок, яке в програмі відкривається за замовчуванням і призначене тільки для настройки параметрів введення відеоданих. У тих же випадках, коли необхідно провести налаштування всіх режимів, потрібно клацнути лівою кнопкою мишки в віконці

«Set preprocessing and video recording». У ньому з'явиться пташка, і після цього відкриється вікно всіх налаштувань (рис. 3.2б).

Всі установки у вікні налаштувань організовані за принципом меню, що випадає, яке відкривається при натисканні лівою кнопкою мишки на кнопці відповідного параметра. Так, наприклад, кнопка «Video source» відповідає за вибір джерела відеоінформації відкриває меню з трьох позицій:

- from file;
- from web-camera;
- from pi-camera

Кнопка «Frame resolution» забезпечує вибір роздільної здатності кадрів відеопотоку. Наведемо можливі стандартні варіанти найменувань форматів і значення роздільної здатності кадрів, використовуваних в цьому меню, а також співвідношення сторін кадру на екрані(у додатку Б).

Для вибору методу введення відеоданих призначена кнопка меню «Video input method», що забезпечує вибір Slow (повільного) або Fast (швидкого) методів введення.

Відзначимо інтуїтивно зрозумілий характер і простоту призначеного для користувача інтерфейсу програми. Тому не має сенсу докладно зупинятися на поясненні всіх установок програми.

4.1.2 Вікно відображення відео

На рис. 3.3а показано вікно відображення на екрані відео за допомогою функції cv2.imshow(frame). Функція transformation_frame(frame) дозволяє проводити зміну формату кадрів і їх розмірів, а в разі необхідності встановлювати і режим стабілізації яскравості кадру. Також передбачено нанесення спеціальних написів в поле кадру з інформацією про поточне значення FPS відеопотоку, розмірах кадру і стабілізації яскравості зображення в тих випадках, коли це передбачено в установках. В іншому випадку ці дані відображаються на спеціальних полях внизу головного вікна перегляду.

Відзначимо, що для встановлення робочого режиму класу VideoStream (захоплення необхідної кількості кадрів вхідного відеопотоку) обраний інтервал часу 1 с., а для отримання стійких і достовірних оцінок FPS інтервал усереднення − 10 с.

На рисунку 4.5 відображається вікно алгоритму програми.

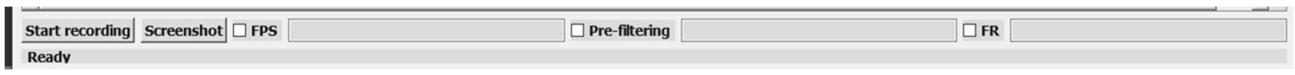


Рисунок 4.3 – Меню при відеоданих з камери

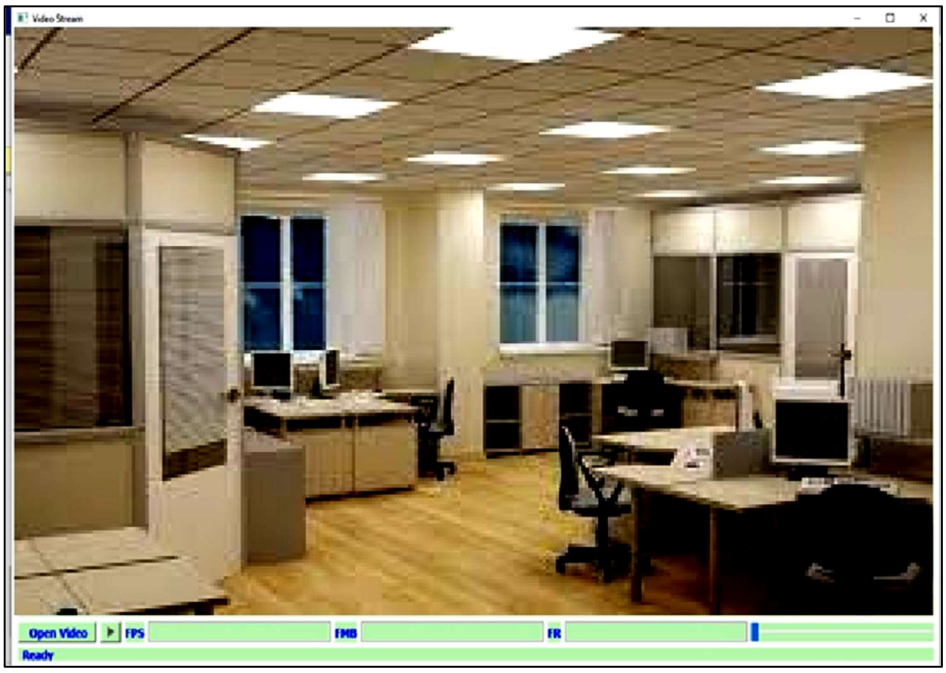


Рисунок 4.4 – Вікно відображення відеоданих для файлу

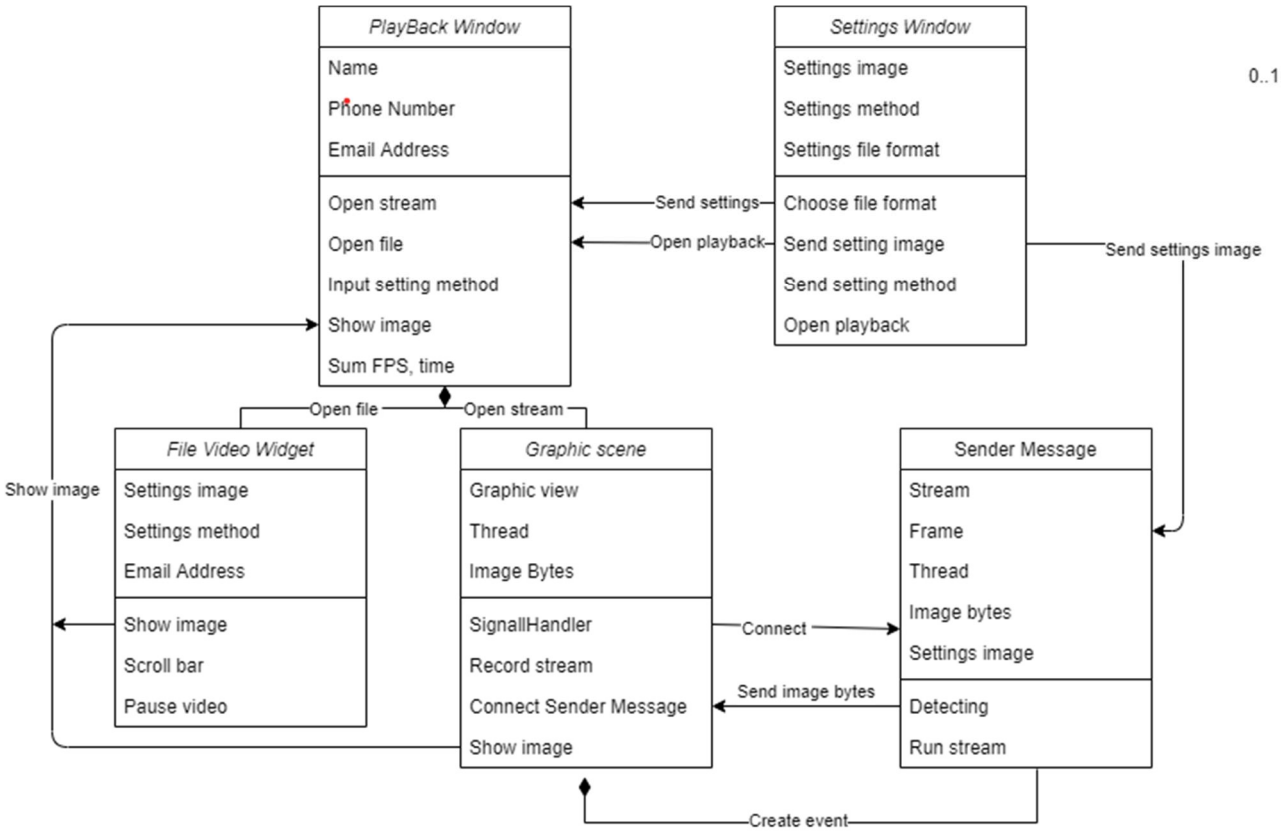


Рисунок 4.5 – Класи, які беруть участь в показі кадрів

Рисунок 4.6 – Алгоритм відображення фрейму в вікні

Це зображення спрощеного алгоритму відображення для більшої ясності , буде приведена діаграма класів відтворення з інтерфейсом.

4.1.3 Посібник користувача

Для початку програми в стартовому вікні програми натисніть кнопку “Start”. При закриті водного вікна, буде відображена вікно налаштування.

Після цього потрібно вибрати базове налаштування, параметри які можна коректувати

1) video source – файл, веб-камера, рі-камера
2) frame resolution (Додаток Б)
3) input method – повільний, швидкий
4) color space
5) brightness – можливість еквалізації

Якщо ви плануєте записати дану сесію , ви можете натиснути на чек бокс “Set preprocessing…” і тим самим налаштувати параметри запису:

1) record format
2) video codecs

Або вибрати параметри еквалізації

3) brightness threshold
4) pre-filtering
5) brightness estimation time

Якщо у вас є потреба налаштувати детектування, ви можете натиснути на чек бокс “Set preprocessing…” і побачити параметри:

1) алгоритм детектування
2) метод анонімізації
3) контур знайденого обличчя

Для повернення базових налаштування потрібно натиснути на “Factory Setting”.

Для початку запису або відображення відео натисніть кнопку “Start”.

Буде відображено 3 вікно. В залежності від джерела зображення нижнє меню буде мати різні особливості.

Для відео файлу доступні кнопка відкриття відеофайлу і шкала прокрутки відео.

Для режиму камери доступна кнопка початку запису, котра змінюється на зупинення запису при нажаті, а також кнопка знімку екрану.

Також для обох режимів доступні 3 поля , котрі можна приховувати , натиснув на чек бокс FPS, Pre-filtering,FR.

На рис. 4.7 можна побачити діаграму активності, на ній зображене запуск додатку , вибір налаштування , початок запису , закриття вікна , зміна налаштувань, початок запису , після цього не закриваючи вікно початок запису , внаслідок цього закриття вікна і відображення нового з затримкою.

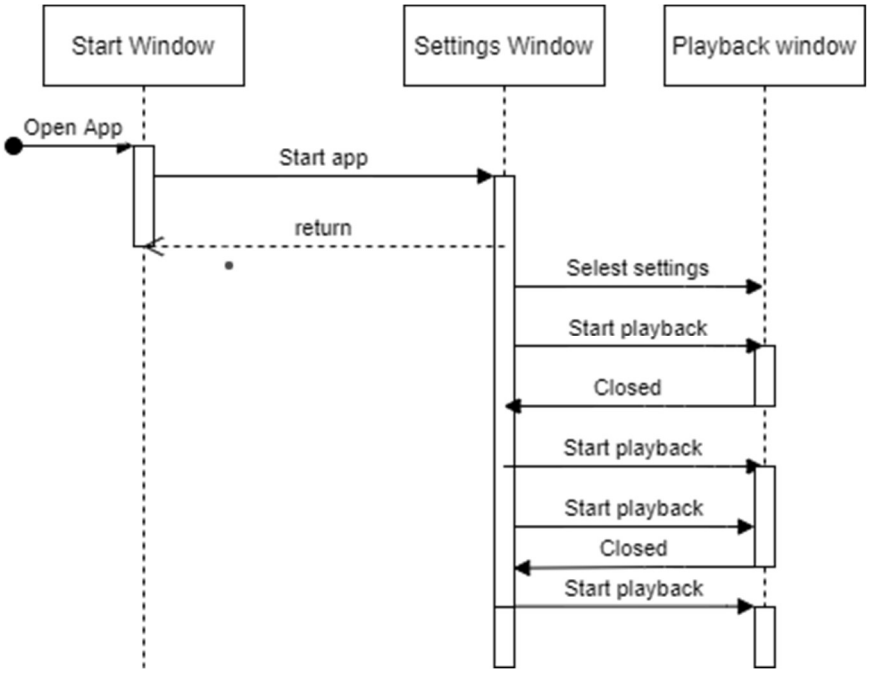


Рисунок 4.7 – Діаграма активності

4.1.4 Запис і збереження відеоданих в файлі

Для повноцінного контролю якості відеоданих, одержуваних за допомогою пропонованих алгоритмів і програмних засобів, необхідно не тільки забезпечувати можливість введення відеосигналів в реальному часі з високим FPS, але і записувати отримані дані для повторного аналізу (з метою оптимізації роботи алгоритмів і програмного забезпечення). Найбільш сприятливі умови користувачеві створюють сервісні функції програми, пов'язані із записом і архівуванням відеоданих. Для цього в інтерфейсі програми передбачено кнопки «`Start recording`» і «`Stop recording`», що примикають до нижньої частини основного вікна перегляду відеоданих. Вони показані на рис. 3.3б. Робота цих кнопок організована так - якщо в якості джерела відеоданих використується відеофайл (установка «`from file`» у вікні «`Video source`»), активною є кнопка «`Open video`». Якщо ж джерелом інформації є web- або pi-камера (установка «`from web-camera`» або «`from pi-camera`» у вікні «`Video source`»), замість кнопки «`Open video`» за замовчуванням з'являється кнопка «`Start recording`». Після її натискання починається запис відеоінформації в файл. Після натискання кнопки «`Stop recording`» реєстрація закінчується. Наведемо нескладний приклад фрагментів програмних кодів, за допомогою яких можна сформмувати функції для управління роботою кнопок «`Start recording`» і «`Stop recording`»:

```
Def StartWrite(self):
self.files = cv2.VideoWriter_fourcc(*'XVID')
filename = str(datetime.datetime.now())
```

```
filename = filename.replace(':','-')
filename = (filename[:filename.find('.')])+".avi"
self.oldname = filename
self.out= cv2.VideoWriter(filename,self.files,20.0,(640,480))
self.flag_write = "start"
self.ui.buttonStart.hide()
self.ui.buttonStop.show()

Def StopWrite(self):
       self.out.release()
       new_filename =
QFileDialog.getSaveFileName(self,"Example",".","Video
Files  (*.mp4 *.flv *.ts *.mts *.avi)")
       os.replace(self.oldname,new_filename)
```

Відзначимо, що за замовчуванням відеофайл зберігається з ім'ям, в якому відображається дата і час реєстрації, як це показано на рис. 3.4. При необхідності цей файл можна перейменувати. Приклад збереження і розміщення цих файлів в архіві показаний на рис. 5.5.

Рисунок 4.7 – Приклад збереження відеофайлу

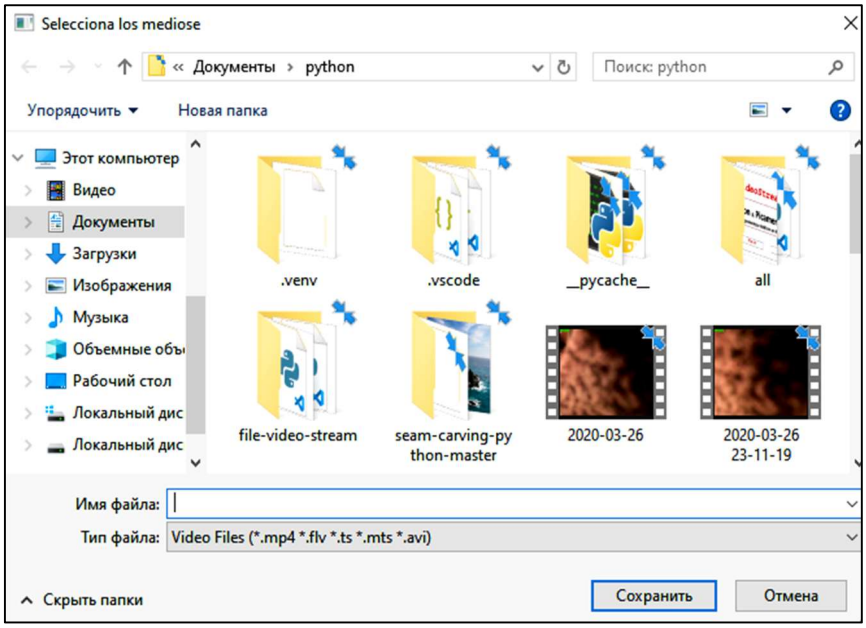


Рисунок 4.8 – Збереження і розміщення відеофайлів в архіві

5 ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕНЯ ШВИДКОДІЇ ПРОГРАМНОГО ПРОДУКТУ

При плануванні експерименту ставилося завдання виявити і вивчити взаємозв'язок між характером змін показника швидкодії FPS і роздільною здатністю FR кадрів вводиться в мікрокомп'ютер Raspberry Pi відеопотоку для різних алгоритмів введення і попередньої обробки даних. Крім цього були проведені дослідження ступеня впливу на швидкість введення додаткового навантаження на процесор комп'ютера за рахунок попередньої обробки (стабілізація яскравості кадрів відео, різні варіанти фільтрації). За цими даними сформульовані рекомендації по оптимальної конфігурації програмних засобів введення і попередньої обробки.

5.1 Вплив роздільної здатності кадру на показник швидкості введення відеоданих для web-камери

Було детально досліджено вплив роздільної здатності кадру на швидкість введення відеоданих. На рисунку показані залежності показника швидкості введення відеоданих FPS за допомогою web-камери в залежності від обраної роздільної здатності кадру. Аналіз проводився для форматів з порівняно невисоким дозволом (від QVGA − 320 × 240 до Full HD − 1920x1080). Порівняльний аналіз ефективності двох методів введення) показав безсумнівну перевагу останнього. Особливо це позначається при високій роздільній здатності кадрів. Так, наприклад, при дозволі HD 720p FPS швидкого методу одно 17.60, що перевершує такий показник повільного методу приблизно вдвічі (FPS = 9.76).

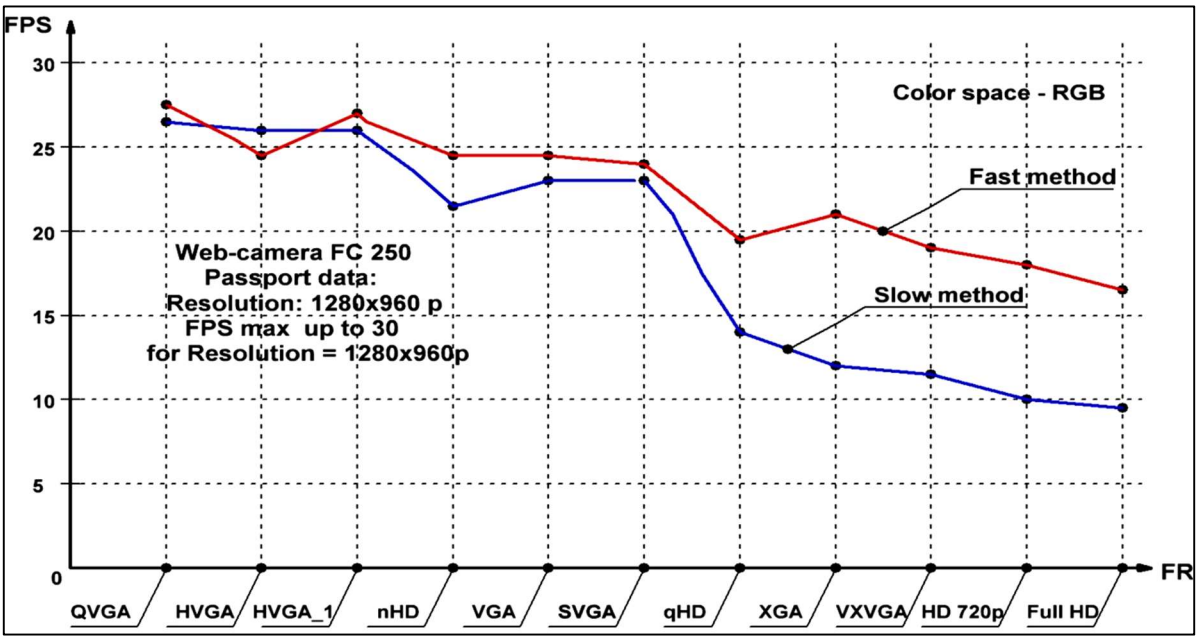


Рисунок 5.1 – Показники швидкості введення відеоданих для різної роздільної здатності кадрів відео потоку

5.2 Вплив роздільної здатності кадру на показник швидкості введення відеоданих для рі-камери

Як і в попередньому дослідженні вивчалися дані про характеристики швидкості введення відеопотоку за допомогою різних методів (повільного і швидкого). Але при цьому для відеоспостереження в комплекті з мікрокомп'ютером Raspberry Pi використовувалася спеціалізована рі-камера. Залежності значень FPS від роздільної здатності відеокадрів показані на рис. 3.7. Очевидно, що використання методу багатопотокового введення (`Fast method`) і в цьому випадку істотно перевершує можливості повільного методу (`Slow method`).

Особливо це помітно при малих значеннях роздільної здатності кадру. Наприклад, при роздільної здатності QVGA (320x240) швидкість введення даних збільшується в 3.5 рази і наближається до потенційних можливостей рі-камери за швидкодією. На підставі цих даних можна зробити висновок − в практично важливих додатках доцільно для цього завдання використовувати рі-камери і Fast method, заснований застосуванні багатопотокового класу `VideoStream`.

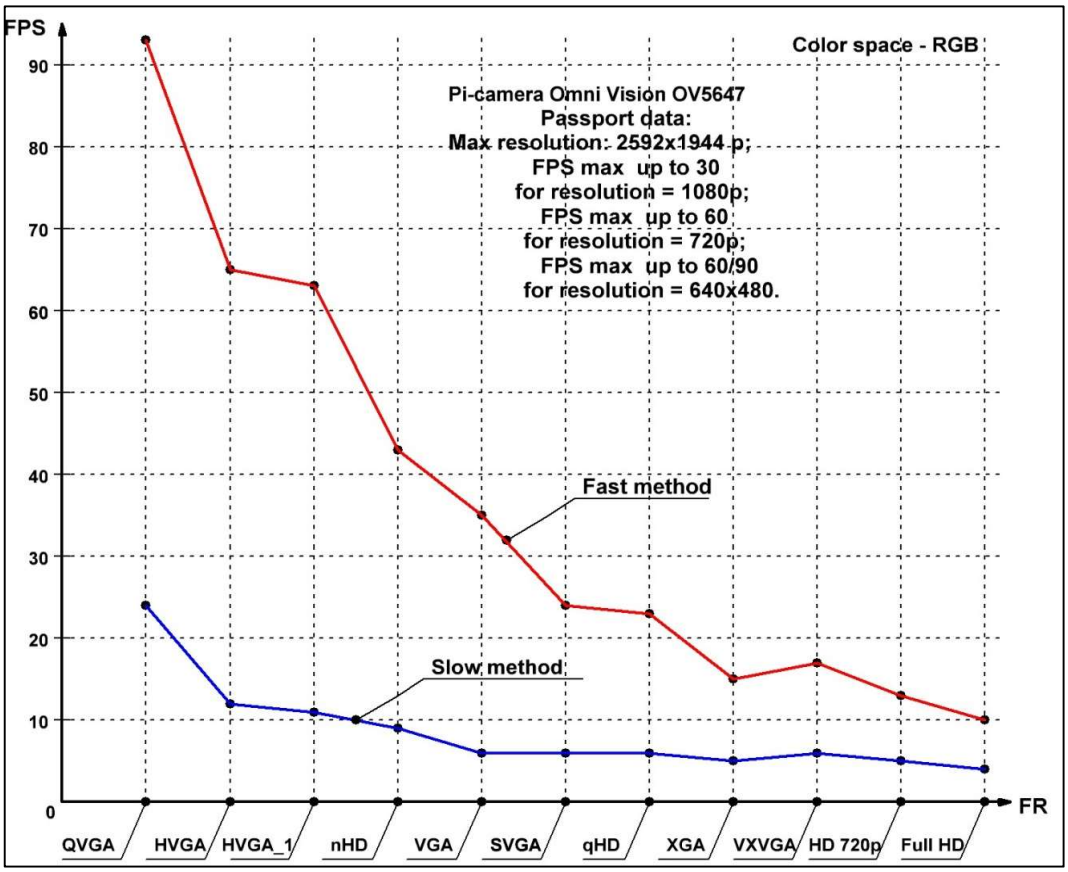


Рисунок 5.2 – Показники швидкості введення відеоданих для рі-камер

5.3 Методи представлення відео та супутньої інформації

На рис. 3.8. показаний метод формування кадрів відеопотоку. Напис, нанесена зліва внизу кадру, відображає поточне значення FPS. При перегляді відео потоку значення FPS змінюється в реальному масштабі часу. Напис в верхньому лівому кутку показує, який метод введення відеоданих використовується в даному випадку (`Slow method` – повільний метод, а `Fast method` – швидкий) і яка роздільна здатність екрана (`Frame resolution`). У наведеному прикладі на екрані комп'ютера відображається кадр відеопотоку з роздільною здатністю 1280x960. Такий рівень зазвичай використовують в системах високоякісної візуалізації сцени.

Також у лівому нижньому кутку командного вікна Python відображаються поточні розрахункові значення FPS. Наведемо фрагмент таких даних. Зрозуміло, що при такій високій роздільній здатності екрану і повільному методі введення FPS неприпустимо мало:

```
[INFO] elasped time: 10.67
[INFO] approx. FPS: 8.53
[INFO] Width 1280
[INFO] Heigh 960
```

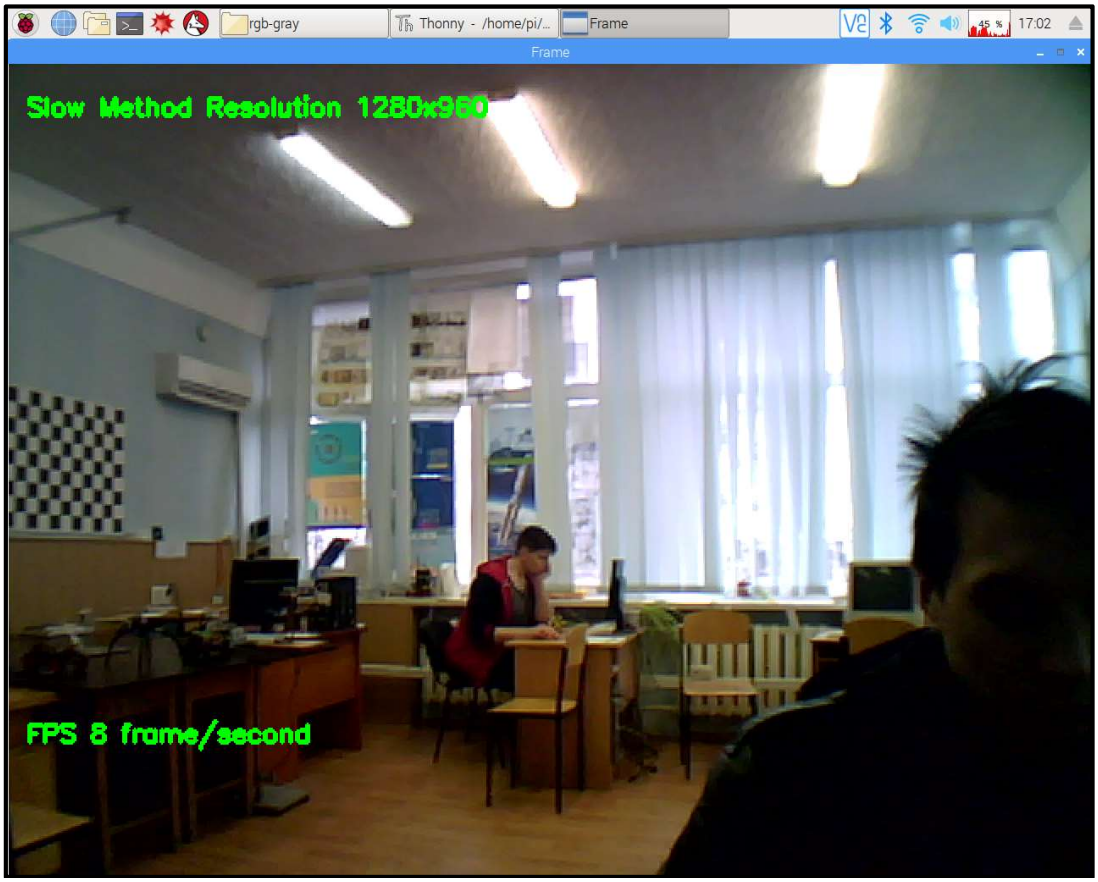
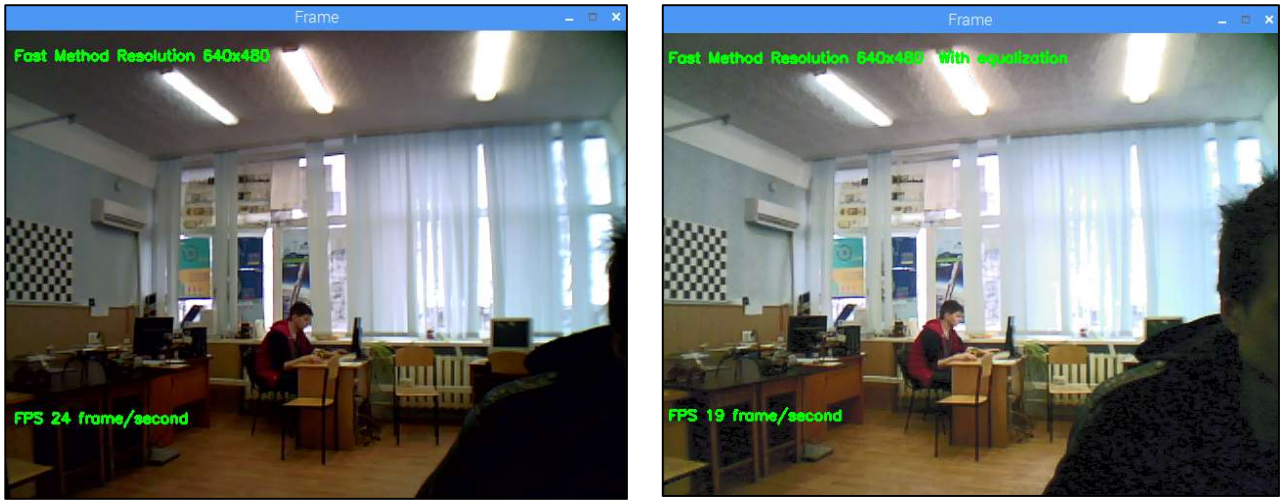


Рисунок 5.3 – Кадри відео, сформовані з роздільною здатністю 1280x960

5.4 Негативний вплив роботи алгоритмів попередньої обробки відео на швидкість їх введення

Опишемо більш докладно результати досліджень негативного впливу роботи алгоритмів попередньої обробки відеоданих на швидкість введення інформації. Зупинимося на основних чинниках − алгоритмах стабілізації яскравості кадрів відеопотоку і різних алгоритмах фільтрації зашумлених кадрів.

Наведемо приклад оцінки якості введення даних при стабілізації яскравості. На рис. 5.4 показані два видеокадра розмірністю 640x480 введені в комп'ютер Raspberry Pi за допомогою багатопотокового методу `VideoStream`.



а б

Рисунок 5.4 – Стабілізація яскравості кадрів відео потоку

Через недостатнє освітлення сцени (рис. 5.4 а) в попередню обробку відеопотоку введена процедура контрастування (еквалізациі) кадрів (рис. 5.4 б). Раніше згадувалося про те, що це вимагає перекладу кадру `RGB` відеопослідовності в колірний простір `YUV` з подальшою еквалізацією яркостной компоненти `Y` і повернення в колірний простір `RGB`. Зрозуміло, що це створює додаткове навантаження на процесор і негативно впливає на швидкість введення − FPS зменшується з 24 до 19 frame/second. Проте, якість зображення кадру помітно зростає.

Наведемо також результати докладного аналізу впливу алгоритмів попередньої обробки відеоданих (`preliminary processing of video data`) на швидкість їх введення (FPS) при фіксованому значенні дозволу екрану (FR).

Для введення даних у вікні установок «`Video source`» був обраний режим введення «`From file`» і використаний відеофайл формату `*.avi` (з дозволом екрану 640x480).

Дослідження проводилися з використанням двох методів введення «`Slow method`» (повільний) і «`Fast method`» (швидкий), заснований на багатопотоковому введенні.

Аналізувалися такі варіанти попередньої обробки відеоданих:

- без попередньої обробки відеопотоку;
- с використанням еквалізациі;
- з НЧ-фільтрацією (low pass filter);
- з медіанної фільтрацією (medianblur)
- з гаусовской фільтрацією (gaussianBlur);
- комбінація еквалізації і гаусової фільтрації.

Зовнішній вигляд вікон відображення відеоданих для кожного з цих дослідів наведено на рис. 5.6. У першому випадку при використанні методу введення «`Slow method`» для їх обробки застосовувалася медіанна фільтрація (рис. 5.6 а), а в другому випадку – «`Fast method`» і комбінація стабілізації яскравості кадрів (equalization) і гаусовской фільтрації (рис. 5.6 б).

Результати аналізу цього експерименту наочно показані на рис. 5.5. Слід зазначити, що при читанні відеоданих з файлу помітно підвищується рівень FPS через відсутність технологічних обмежень, властивих як web-, так і рі-камерам. Максимальна FPS досягає ~ 175 frame/s. Найбільш істотний негативний вплив на швидкість введення даних надає застосування медіанної фільтрації (FPS зменшується приблизно в 4 рази). І нарешті, очевидно, що застосування багатопотокового методу введення відео («`Fast method`») практично не дає виграшу за швидкодією в порівнянні з повільним методом «`Slow method`».

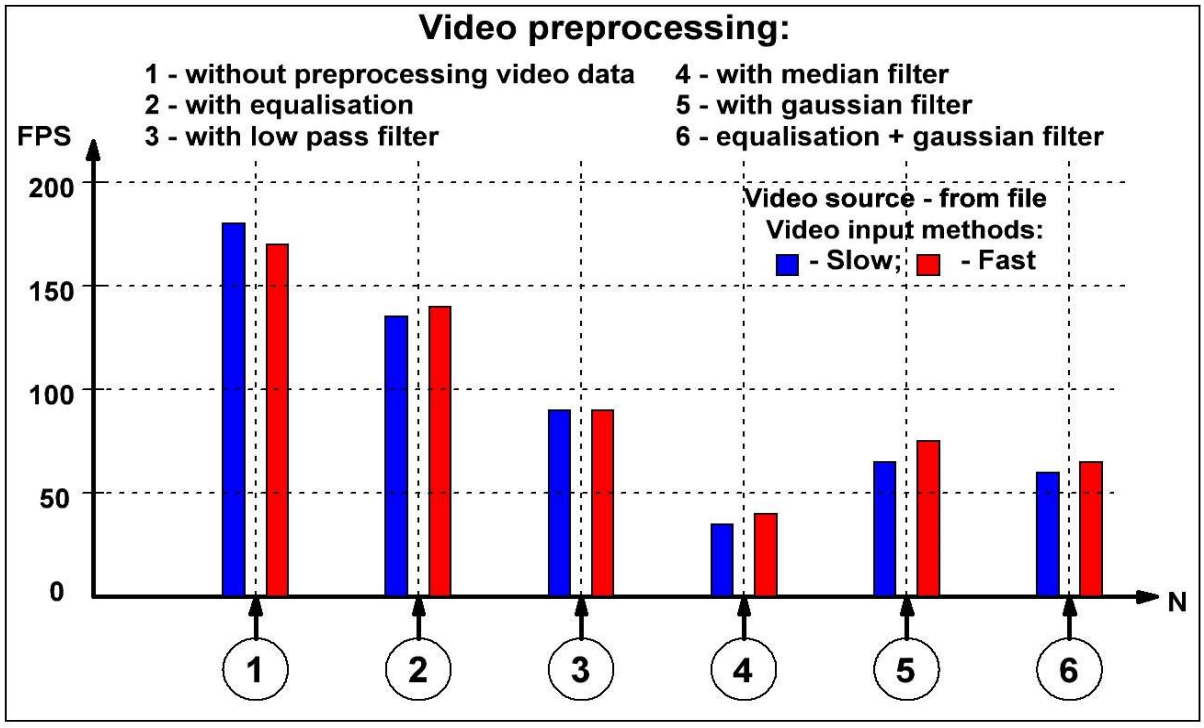
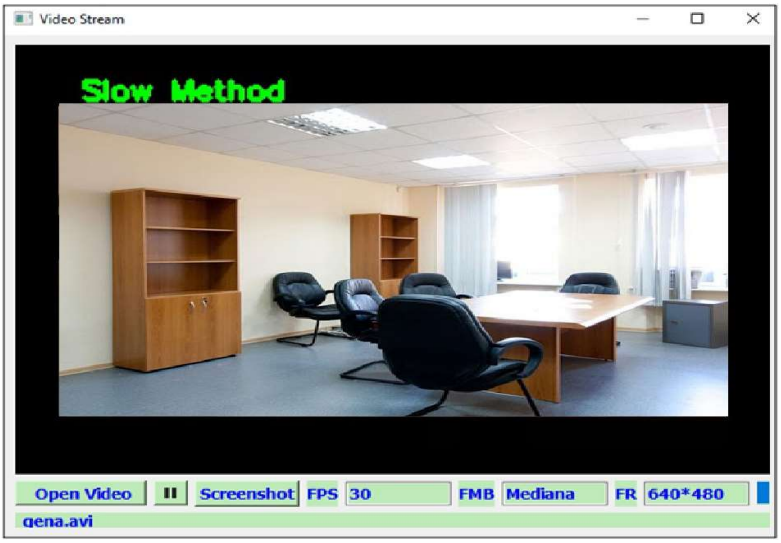
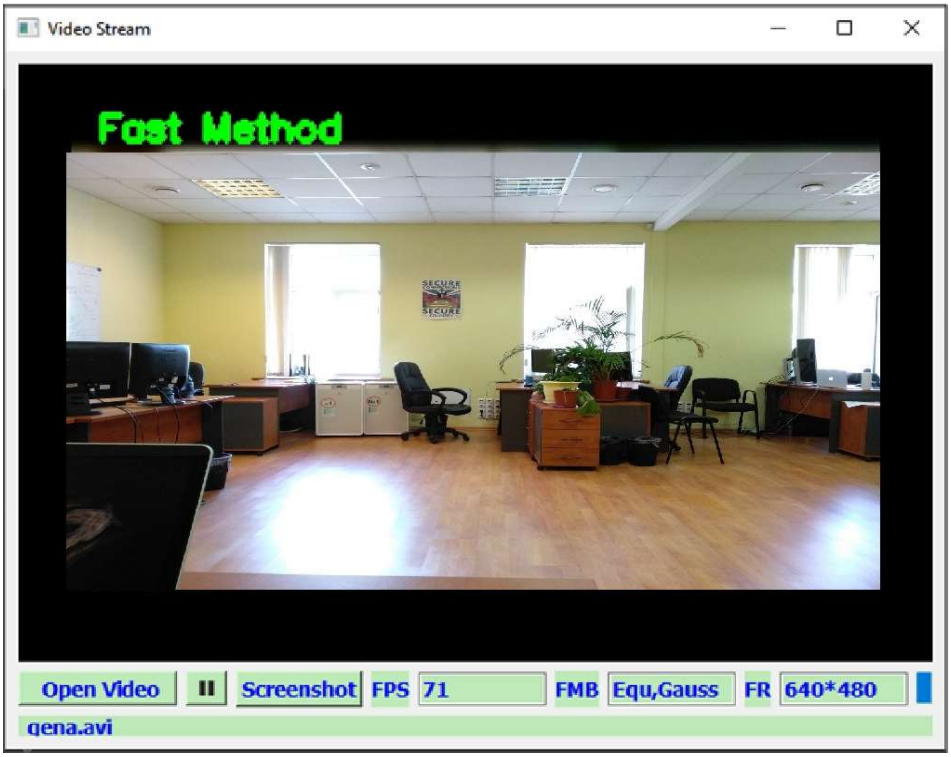


Рисунок 5.5 – Залежність швидкості введення відеоданих (FPS) при читанні відео з файлу від характеру попередньої обробки відеоданих



а



б

Рисунок 5.6 – Зображення при попередньої обробка відеоданих: а − медіанная фільтрація; б − еквалізациі + гауссова фільтрація

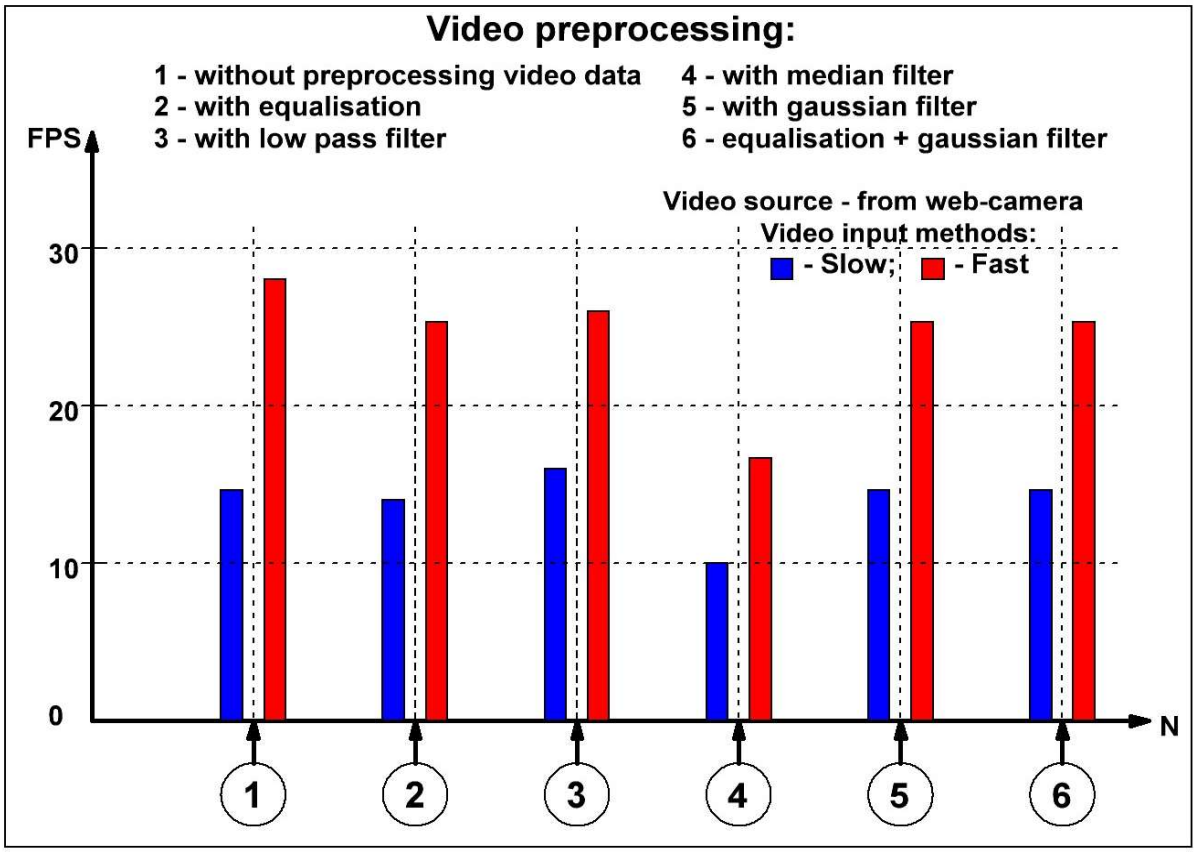


Рисунок 5.7 – Залежність швидкості введення відеоданих (FPS) при читанні відео з web-камери від характеру попередньої обробки відеоданих

Результати другої серії експериментів показані на рис. 5.7. У цьому випадку в якості джерела відео використовувалася web-камера. Всі інші параметри експерименту повністю збігаються з попередніми дослідами. Аналіз показав, що при використанні web-камери швидкість введення (FPS) при різних варіантах попередньої обробки відеоданих змінюється незначно. Однак, метод введення («`Fast method`») виявляється приблизно вдвічі швидше повільного методу «`Slow method`» і впритул наближається до потенційно можливого FPS (30 frame/s).

Запропоновано новий комплексний підхід до проблеми підвищення якості їх обробки в сучасних мобільних СТЗ на базі мікрокомп'ютеру Raspberry Pi. Спільно оптимізуються показники швидкості захоплення, обробки відеоданих, роздільної здатності видеокадра і алгоритмів попередньої обробки відео. Ефективність пропонованих методів обробки та програмних кодів досліджена експериментально.

6 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ДЕТЕКТУВАННЯ І АНОНІМІЗАЦІЇ ОБЛИЧЧЯ

Специфічні властивості алгоритмів виявлення обличь, заснованих на використанні навчених каскадних класифікаторів, визначають імовірнісний характер їх функціонування. Тому основним показником якості таких алгоритмів прийнято вважати ймовірність правильного виявлення обличчя та його елементів. Аналогічним чином була побудована графічна форма для поточної оцінки ефективності виявлення і для другого алгоритму. Вона містить чотири вікна, в яких відображаються поточні значення ймовірностей правильного виявлення обличчя, очей. Ці кошти візуалізації поточних процесів виявлення досить ефективні, але вони не дають відповіді на основне питання − яка ефективність роботи алгоритму при тривалому аналізі відеопослідовності.

Для узагальненої оцінки ефективності роботи алгоритмів можливі два методи − якісний (візуальний) і кількісний. У першому випадку проводиться візуальне спостереження за різнокольоровими прямокутниками, що обмежують виявлене обличчя і його елементів (очі, ніс і рот). Якщо процес їх відтворення сприймається як безперервний, то якість детектування можна вважати прийнятним (рівним ~ 100%). Однак, інерційність зору спостерігача при високій частоті зміни кадрів (30/с) не дозволяє візуально сприйняти пропуски виявлення окремих кадрів. Це може істотно спотворити результати тестування. Тому для кількісної оцінки ефективності в програмі для всіх кадрів відеопослідовності формується одновимірний масив (1 − за фактом виявлення особи в кадрі, і 0 − у разі пропуску особи), за яким будуються імовірнісні характеристики якості роботи алгоритму. Однак, слід мати на увазі, що для цього необхідна достовірно анотована тестова відеопослідовність. Зрозуміло, можна використовувати і більш складний тестовий відеоряд, у якого в певні інтервали часу особи в кадрі відсутні. Тоді з'являється можливість оцінити не тільки ймовірності правильного виявлення облич, але і ймовірності помилкового виявлення (виявлення обличчя за умови його відсутності в кадрі).

Була розроблена спеціальна форма узагальненої оцінки ефективності роботи алгоритму виявлення обличь і їх головних елементів для відеоряду тривалістю такого запису становить ~ 30 с. Для процедур виявлення всіх елементів побудовані періодограми, де для кожного з N кадрів у відповідність ставиться 1 (якщо виявлено) і 0 (при невиявлення) відповідного елемента обличчя. Крім цього, для кожної періодограми обчислюється ймовірність правильного виявлення на вибірці. Така форма дозволяє отримати не тільки узагальнені оцінки ймовірностей виявлення, а й наочно показує, на яких кадрах виявлення не було виконано.

6.1 Показники якості роботи алгоритмів та їх використання

6.1.1 Показник ймовірності правильного виявлення обличчя за умови його наявності в кадрі

Об'єктивним критерієм ефективності роботи будь-якого алгоритму детектування осіб і їх елементів в кадрах відеопослідовності є показник ймовірності правильного виявлення обличчя за умови його наявності в кадрі.

Процедура виявлення особи в поточному кадрі з застосуванням відповідного каскадного класифікатора в разі успіху завершується побудовою прямокутника за допомогою функції OpenCV

`cv2.rectangle(frame,(x,y)(x+w,y+h),255,0,0),2).`

Успішне завершення детектування обличчя в програмному коді алгоритму необхідно супроводжувати появою події логічної 1; в іншому випадку − появою логічного 0. У нашій роботі для створення стійкого критерію якості правильного виявлення обличь розраховувалася ймовірність $\boldsymbol{P_n}$, отримана за результатами підрахунку відносно кількості фреймів на якій вдалося детектувати обличчя до загальної кількості фреймів (Frame)

Якість роботи детектора можна вважати задовільним при $Pn \geq \mathbf{0.9}$. За показником ймовірності правильного виявлення осіб в кадрі легко аналізувати вплив різних чинників (в тому числі змін освітленості, геометричних факторів і ін.) На якість роботи алгоритмів виявлення і детектування облич. Аналогічним чином розраховуються і ймовірності правильного виявлення очей, носа, рота у відповідних областях. Ці показники обчислюються як умовна ймовірність такого події за умови правильного виявлення всього особи. З вірогідністю прийнятною для практичного використання можна вважати події виявлення окремих елементів особи незалежними, а отже, ймовірність правильного виявлення всіх елементів оцінювати як твір ймовірностей виявлення цих елементів.

6.1.2 Допоміжний показнику часу

У цьому експерименті велика увага приділяється бистродії та таким показникам як FPS та FR , котрі були розглянуті заздалегідь, тому час кожного експерименту повинен бути стиглим для того щоб можна було зняти відносні характеристики стосовно його.

Час кожного експерименту був вшитий у програму и при виконанні протягом 30 секунд програма повинна бути припинена , що реалізовано на програмному рівні. Але цей відрізок відповідає внутрішньому часу , а є зовнішній час elapsed time в котрий окрім внутрішнього додається час загрузки необхідних ресурсів, імпорту бібліотек та тимчасової паузи.

Також показник Frame , який відповідає за загальну кількісті оброблених файлів має обмеження так як відповідає лише за кадрі котрі були відображенні , а кадри котрі були взяті із потоку і декілька з них не були відображенні відповідає показник Stream Frames.

6.1.3 Формат вводу даних

Елементарно, що по-перше для більш детальної вибірки треба оцінити алгоритм не тільки на комп'ютері Raspberry Pi, але й на системі Windows , що дозволить про аналізувати відносність де яких параметрів.

По-друге деякі експерименти потребують сталі показники експерименту з великої точністю , навіть до положення обличчя в кадрі , саме тому використовується відкриття записаного файлу.

Для оцінки ефективності був зроблений ряд тестових відеозаписів, що відповідають заданим вимогам. Записи виконувалися в форматі `*avi` з розміром кадрів 480x640 пікселів з нерухомо встановленої камери. Істотні артефакти, обумовлені рухом людини в кадрі, були виключені. Відстань від обличчя до об'єктива відеокамери становило ~ 1м.

6.2 Дослідження швидкості відображення кадрів при різних методах загрузки для програмного забезпечення «Video Stream Quality Control»

6.2.1 Дослідження FPS відображення кадрів при зчитування з відеофайлу

За для отримання еталонних значень при порівнянні з котрими будуть розраховуватися показники якості і оцінка наступних методів і експериментів було промодельовано наступні ситуації. Так як дані отримані з web-камери та рі-камери вже були продемонстровані у 4 пункті, ці матеріли зосередяться на зчитування с відео-файлу, котрі були записані з других експериментів.

Для зчитування в режимі відео-файлу придатні лише 3 алгоритма звичайний , Video Stream та Pi Video Stream. Інші алгоритми не придатні або їх модифікація для придатності приведе до вподобання цим.

Аналізуючи таблицю можна побачити, що найбільш виграшні за бистродією є 3 метод , хоча без допоміжною загрузки котру додає нейромережа 2 метод має більш виграшні дані.

Виграш 2 методу у першому випадку може сказатися аномалією , так як відрізняється від попередніх експериментів. Але це відбувається тому що бібліотека Open Cv , а саме методи , вже використовуються для других функцій і ще один виклик не додає великого навантаження, а 3 метод котрий без інтерфейсу не використовує цю бібліотеку, із-за цього сповільнюється.

Але так як в наступному ми використовуємо саме детектування обличь , за еталоні значення будуть використанні данні 3 методу.

Таблиця 6.1 – Дані при різних методах загрузки і нейромережі

Час виконання	FPS	Frame	Detecting, %	Метод зчитування	Нейроме-режа
17,1	31,17	533	0	стандарт.	-
16,12	42.05	678	0	Video Stream	
15.30	39.21	600	0	Pi Video Stream	
17,21	3,2	55	98,18	стандарт.	Віола
16,15	4,83	78	98,18	Video Stream	
15,14	5,09	77	100	Pi Video Stream	
17,69	0,4	7	99,8	стандарт.	Кофі
16,5	0,61	10	99,9	Video Stream	
15,62	0,64	10	100	Pi Video Stream	

6.2.2 Дослідження швидкості відображення кадрів при зчитування з реальної моделі

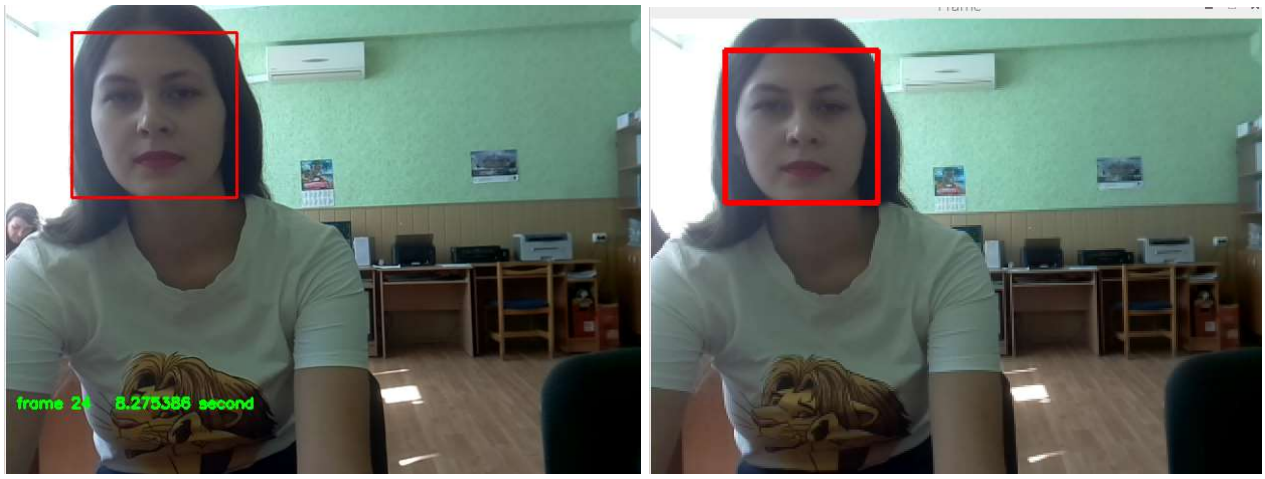
За для отримання еталонних значень при порівнянні з котрими будуть розраховуватися показники якості і оцінка наступних методів і експериментів було промодельовано наступні ситуації. Ці дані були отримані за для того щоб служити лише еталонними значеннями.

Так як це реально модель складно дивитися на відсоток детектування обличчя із-за того що не всі умови тотожні. Наглядне оцінювання досить складне так як тяжко закарбувати вдячний момент.

Але навіть по таким даним можна побачити , що алгоритм Кофі дає 100% відсоток детектування, але дуже низький FPS , котрий є незадовільним для роботи у реальному часі , але є достатнім при роботі з відеофайлами або зображеннями. І в такому випадку алгоритм Віоли-Джонса являється переможцем.

Таблиця 6.2 – Дані при різних методах загрузки і нейросітях

method	FPS	Detecting	Нейромережа
1	0,60	100	Кофі
2	0,58	100	
3	0,47	100	
4	0,57	100	
5	0,56	100	
1	5,31	94	Віола
2	6,16	83	
3	4,32	94	
4	3,73	100	
5	18,16	89	



а) б)

Рисунок 6.1 – Різні методи зчитування

а)- Video Stream, б) – Fast method Pi

6.3 Дослідження детектування обличь при зміні показників

6.3.1. Зміна характеристики коефіцієнт масштабу методу Віоли-Джонса

Перший параметр змінення котрого впливає на шанс детектування обличчя це коефіцієнт масштабу (scale factor). В основному коефіцієнт масштабу використовується для створення вашої піраміди масштабу.

Піраміда зображення - це багатомасштабне подання зображення, таким чином, що виявлення обличчя може бути інваріантним до масштабу, тобто виявлення великих і малих обличь за допомогою того самого вікна виявлення. Як варіант, ми також можемо масштабувати вікно фільтра, що в цьому випадку є більш громіздким. Тому, як і багато інших алгоритмів зору, для виявлення обличчя Віоли-Джонс побудована піраміда зображення.

Загалом модель має фіксований розмір, визначений під час навчання. Це означає, що на зображенні виявляється такий розмір обличчя, якщо він з’являється. Однак, змінивши масштаб вхідного зображення, ви можете змінити розмір більшої грані до меншої, зробивши її помітною для алгоритму. Використовуючи невеликий крок для зміни розміру, наприклад 1,05, що означає зменшення розміру на 5%, ви збільшуєте шанс збігу розміру з моделлю для виявлення.

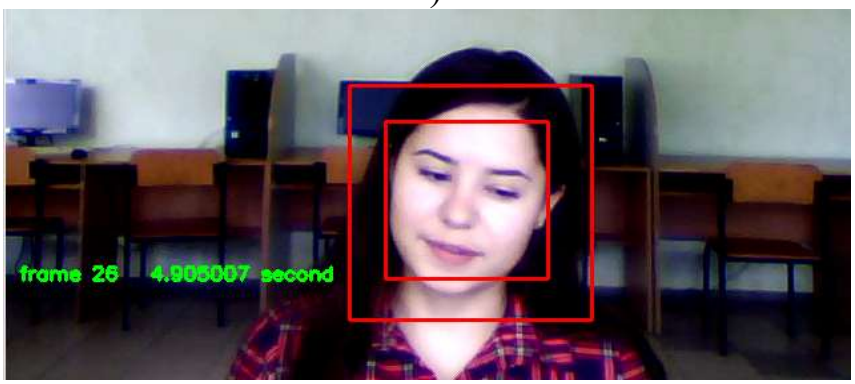
Найчастіше піраміда зображення виконується шляхом зменшення вибірки зображення за допомогою сусідніх пікселів(Neigborgs), котрі будуть розглянуті в подальшому.

Наприклад, чим нижче коефіцієнт масштабу, тим частіше виникають помилкові виклики і тим сильніше більше навантаження на процесор , наприклад при параметрі 1,05 виникають 2 помилкових виклика при чому із-за того що для побудови піраміди використовуються суміжні пікселі , детектований об’єкт може “кочувати” в зовсім інше місце. А при високому значенні помилкових викликів не має , але саме обличчя може не здетектуватися.

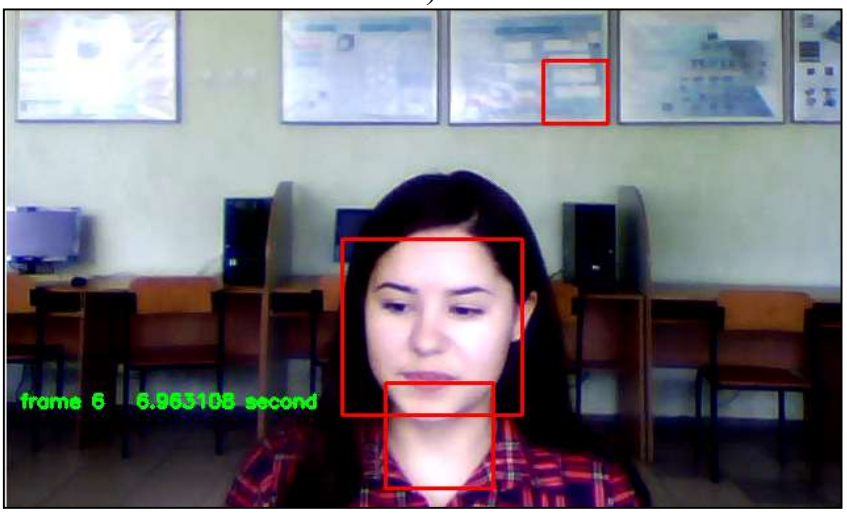
Таблиця 6.2 – Дані при зміна коефіцієнту масштабу

Scale	FPS	Frame	Detecting, %
1,05	1,12	35	100
1,1	2,16	68	100
1,3	4,98	155	84,51
1,5	6,78	211	87,67
1,7	8,78	274	70,43

а)



б)



в)

Рисунок 6.1 – Зміна фактору масштабу(а-1.05, б-1.3, в -1.7)

В подальшому використовувався параметр 1.5 , так як являється позитивним максимумом.

6.3.2 Зміна коефіцієнту кількості сусідніх пікселей методу Віоли-Джонса

Другим істотним параметр змінення котрого впливає на шанс детектування обличчя є коефіцієнту кількості сусідніх пікселей (minNeigborhs).

Ідея цього параметра полягає в тому, що детектор буде працювати в багато масштабному стилі і одночасно слідуючи стратегії ковзного вікна. Після цього кроку ви отримаєте кілька відповідей навіть для однієї області обличчя. Цей параметр, як правило, фільтрує ці відповіді, як встановивши поріг нижньої межі, тобто він буде зараховуватися як дійсне обличчя, лише якщо кількість відповідей для цього обличчя перевищує minNeighbors.

Каскадний класифікатор Хаара працює за підходом розсувних вікон. Якщо ви подивитесь на каскадні файли, ви побачите параметр розміру, який, як правило, досить невелике значення, наприклад “20 20”. Це найменше вікно, яке каскад може виявити. Отже, застосовуючи підхід до розсувного вікна, ви висуваєте вікно назовні зображення, але не змінюєте його розмір і шукаєте знову, поки не зможете змінити його розмір далі. Отже, з кожною ітерацією каскадного класифікатора Хаара зберігаються справжні результати. Отже, коли це вікно ковзає на зображенні, його розмір змінюється і знову ковзає.

Ця система також може спровокувати багато помилкових спрацьовувань. Тож для усунення помилкових спрацьовувань та отримання належного прямокутника обличчя з виявлення застосовується сусідський підхід. Якщо піксель поруч з іншими прямокутниками на відстані коефіцієнту то алгоритм пройде далі. Отже, це число визначає, скільки сусідніх пікселей потрібно для передачі його у вигляді прямокутника обличчя.

Таблиця 6.3 – Дані при зміні коефіцієнту кількості сусідніх пікселей

Neigborg	FPS	Frame	Detecting
1	4,92	154	96,1038961
3	4,99	156	89,74358974
5	4,99	156	99,84615385
7	4,95	154	99,81168831
9	5	156	78,84615385

Хоч за даними таблиці 1 варіант при найменшій кількості і виглядає найвигіднішим, але виникають деякі колізії при цьому коефіцієнті , а саме виникають дубльовані квадрати обличь і це відбувається в коротку мить , тому складно задокументувати і закарбувати. І це є причиною не приведення фотографій в цьому випадку, так як вони не наглядні.

Тому вибраним параметром Neigborgs є 7 при цьому найменше навантаження і довільний процент детектування.

6.3.3 Зміна показнику confidence методу Кофі

Так званий довірчий інтервал відноситься до діапазону значень параметрів, що містять обмеження або межі значень параметрів, інтервал, пов'язаний з рівнем довіри, що гарантує, що межі визначають інтервал, який містить, безумовно, справжнє значення параметра.

Обмеження значення параметра визначають такий спосіб, як діапазон значень, Нижня обмежена мінімальним значенням параметра (або меншим обмеженням довіри), а Верхня обмежена максимальним значенням параметра (або обмеженням верхньої довіри), таким чином, щоб можна було бути впевненим (з попередньо вказаним рівнем довіри, наприклад, наприклад 95% або 99%) що справжнє значення параметра знаходиться в межах confidence.

Довірчий інтервал може бути обчислений за допомогою параметрів припасування, що визначає рівняння, що описують геометричні особливості складання контуру виготовленої частини під оцінку. Ці довірчий інтервал ще більше поширюється на процедуру тріангуляції, досягнення детектування. Остаточні помилки потім оцінюють застосування схеми розповсюдження помилок.

Проведемо експеримент змінюючи показник confidence і спеціально закриваючи обличчя.

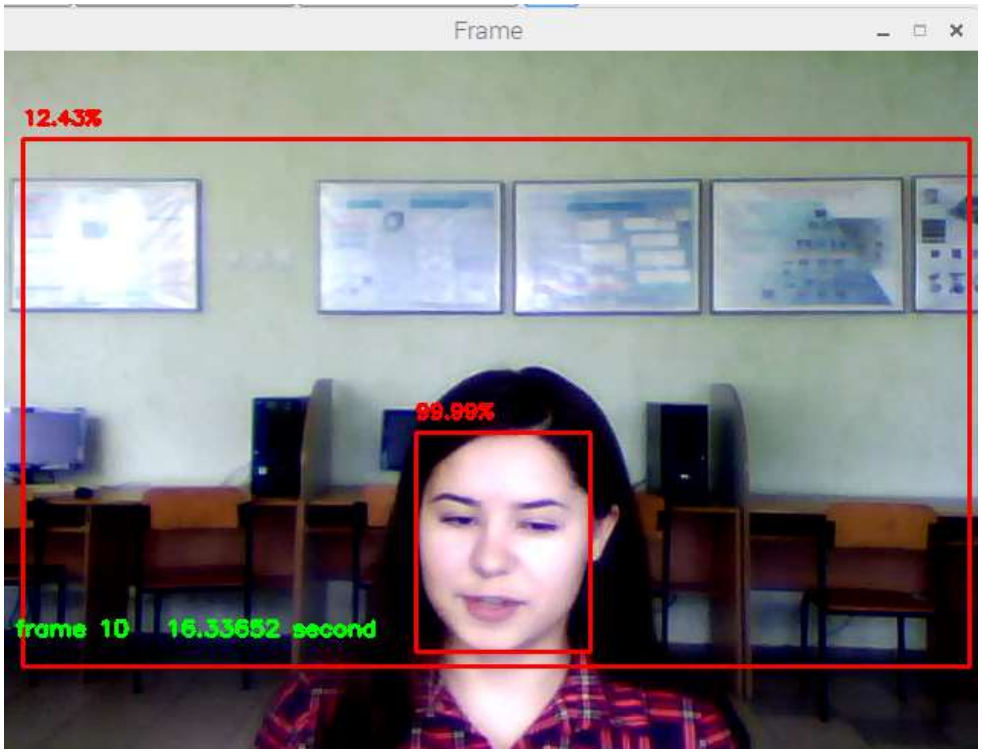
Таблиця 6.4 – Дані при зміні показнику confidence

confidence	FPS	Frame	Detecting, %
0.1	0,65	21	98
0.3	0,66	21	98
0.5	0,62	20	100
0.7	0,59	19	100
0,9	0,56	18	100

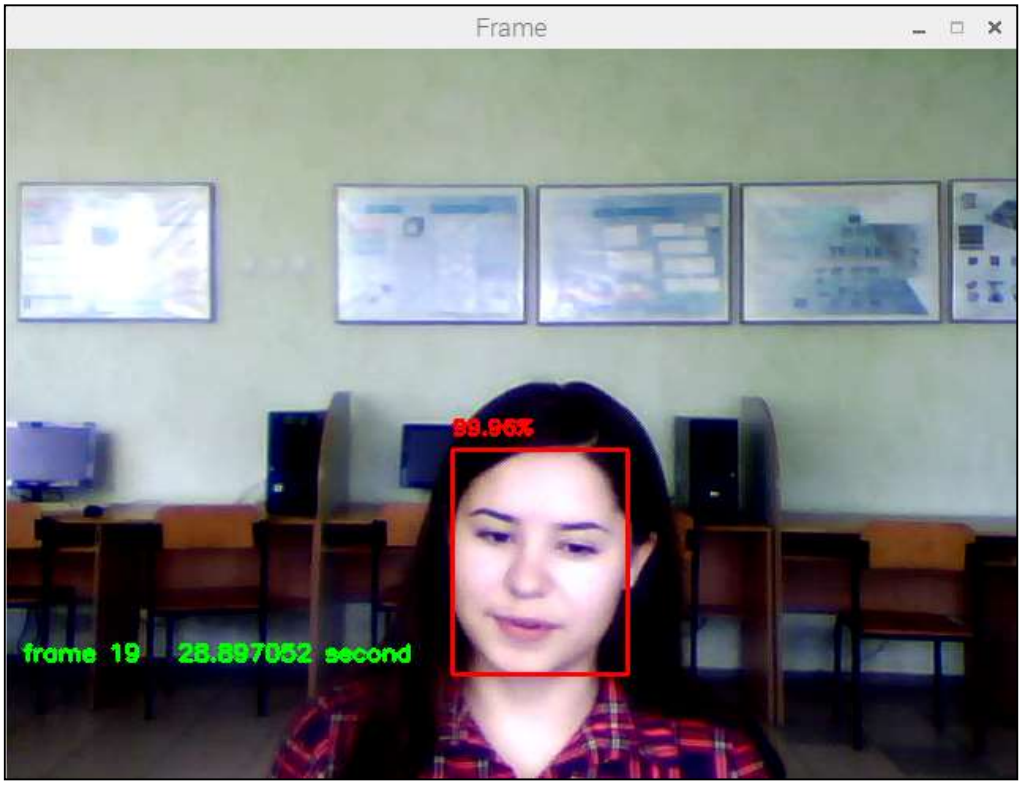
Аналізуючи дані таблиці можна звернути увагу, на те що процент детектованих в усі випадках 100 %, але цей параметр не враховує помилкові спрацьовування, котрі виникають як показано на рис.6.2-а).

При цьому саме цей параметр відповідає за вдячність детектування на приклад в статі «Video Stream Quality Control» досліджується впливання цього показника при закриті обличчя на 50%, 30% відсотків.

І оптимальнішим варіантом являється параметр 0.3 , котрий викликає маленьку кількість помилкових викликів, але має найбільші шанси детектувати обличчя.



а)



б)

Рисунок 6.2 – Зміна показнику confidence а)-0.1, б)-0.5

6.3.4 Зміна показнику factor методу анонімізації

Показник factor відповідає за кількість клітинок, що беруть участь у процесі згладжування кожного окремого пікселя.

Чому для алгоритму Кофі FPS зменшується а для Віоли-Джонса збільшується?

Насправді при збільшені параметру factor логічною є думка , що кількість операцій збільшується , отож бо збільшується навантаження на програму, що і показує залежність в алгоритмі Кофі.

Але це не являється вірним для алгоритму Віоли-Джонса так як в цьому алгоритмі використовуються показник сусідніх пікселей і із-за цього, виникають помилки у роботі цього алгоритма і не виникає додаткових перевірок і додаткових помилкових викликів, котрі раніше викликали затримку , але із-за цього шанс детектувати обличчя зменшився , хоча й не суттєво.

Але головним фактором в оцінюванні цього параметру є наглядний огляд, так як для різних ситуацій може бути потрібно різний процент анонімізації.

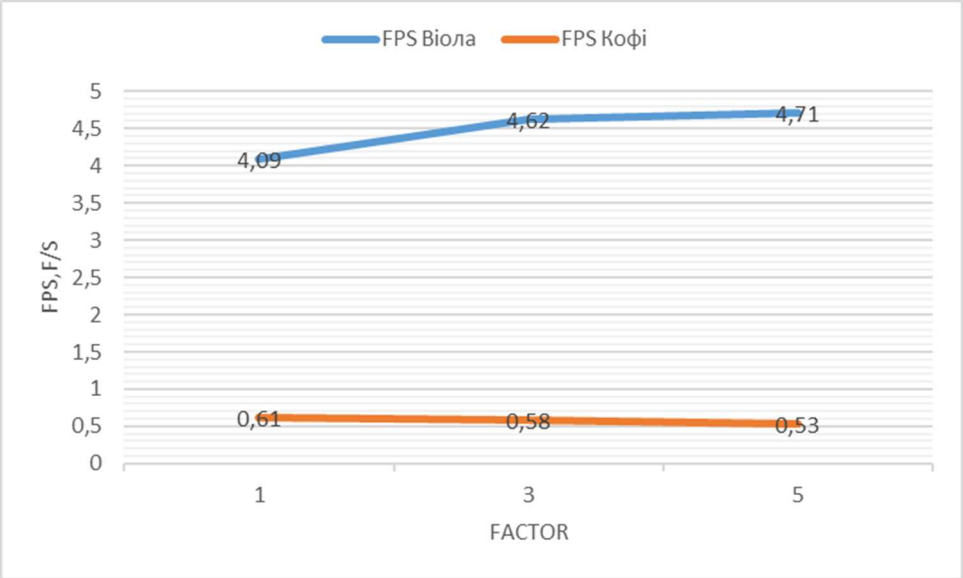


Рисунок 6.3 - Залежність FPS від зміни фактору

а)

б)

в)

Рисунок 6.4 - Зміна показнику factor

а) factor=1 б) factor=3 в) factor=5

6.3.5 Зміна показнику block методу анонімізації

Показник block, відповідає за кількість клітинок, котрі замінять зображення, до кожної з котрих буде виконане усереднювання коліру.

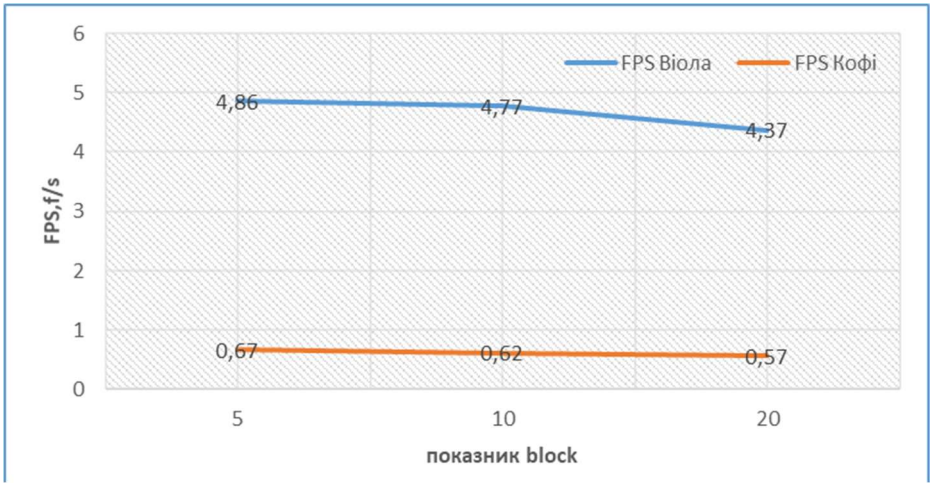


Рисунок 6.5 - Залежність FPS від зміни показнику block

При збільшені параметру block кількість операцій збільшується , отож бо збільшується навантаження на програму, що і показує залежність як при алгоритмі Кофі, так і Віоли-Джонса.

Але головним фактором в оцінюванні цього параметру є наглядний огляд, так як для різних ситуацій може бути потрібно різний процент анонімізації.

а)

б)

в)

Рисунок 6.6 - Зміна показнику block
а) block =5 б) block =10 в) block =20

6.3.6 Дослідження ефективності роботи алгоритмів в залежності від відстані обличчя до відеореєстратора

Крім цього було проведено дослідження ефективності роботи алгоритмів в залежності від відстані обличчя до відеореєстратора. Початкове відстань склало 1 м, а кінцеве відстань − 3 м. Решта умов зйомки були аналогічні попереднім дослідженням. Відзначимо, що процедура виявлення особи виявилася нечутливою до збільшення відстані між особою і відеокамерою. Але при збільшенні цієї відстані більш ніж 1,5 м це призводить до переривання процедури виявлення елементів особи (очей, носа і рота). Це обмеження пов'язане з відносним зменшенням розмірів особи в кадрі.

І при збільшені показника scale factor для алгоритму Віоли – Джонсу ефективну відстані детектувати кадри вдалося збільшити і навіть детектувати деякі кадри на відстані 3 метрів.

У алгоритму Кофі робоча дистанція складає 2 метри , але цей показник не вдалося збільшити регулюванням коефіцієнтів.

Таблиця 6.5 – Дані при різних положеннях маски алгоритму Віоли-Джонсу

Відстань, м	Scale factor	FPS	Detecting, %
1	1.5	13.62	48
1.5	1.5	13.03	20
2	1.5	13.53	2
2	1.1	12.00	55
3	1.1	12.04	10

Таблиця 6.6 – Дані при різних положеннях маски для алгоритму Кофі

Відстань, м	FPS	Detecting, %
1	4.00	91
2	4.02	30
3	3.98	0

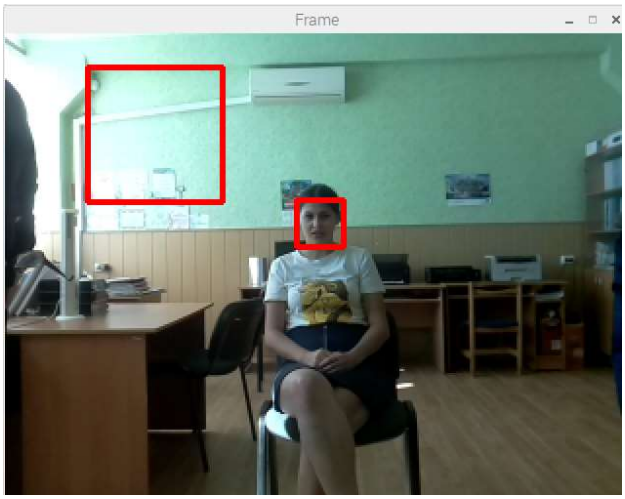


Рисунок 6.7 – Зміна показнику scale factor на відстані 2 метрів

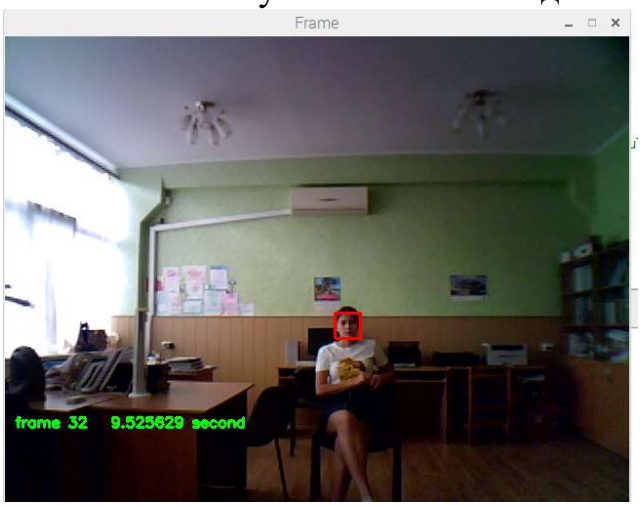


Рисунок 6.8 – Зміна показнику scale factor на відстані 3 метрів

Отже можна скласти висновок, що параметр ефективної відстані кращий у методу за алгоритмом Віоли-Джонса. Так як параметри методу можливо калібрувати це приводить до більших можливостей. Але не треба забувати , що при змінені параметру scale factor, з'являються помилкові спрацьовування(рис. 6.8) .

З іншої сторони хоч алгоритм Кофі і не має здатності збільшити діапазон ефективної дистанції, але початковий відсоток детектування більше а ніж для алгоритму- конкурента, і не має помилкових спрацьовувань.

6.3.7 Дослідження ефективності роботи алгоритмів при приховуванні частини обличчя

Було проведено експеримент за допомогою звичайної маски і покриті різному співвідношенню к відкритому обличчі. Очі не були приховувані так це один із основних маркерів алгоритму. Експеримент складався із трьох частин покриття до очей, покриття до носу і покриття лише умовне.

Для алгоритму Кофі у всіх положеннях програма могла детектувати обличчя у 100 відсотків випадків і рухи головою при цьому не запобігали цьому.

А для алгоритму Віоли – Джонсу потребували більше складності і при рухах головою , алгоритм переставав працювати і пропускав фрейми. Цьому і такі показники у таблиці.

Таблиця 6.7 – Дані при різних положеннях маски для алгоритму Віоли-Джонсу

Положення маски	FPS	Detecting, %
Під очима	12.01	75
Під носом	12.18	95
Під ротом	11.86	100

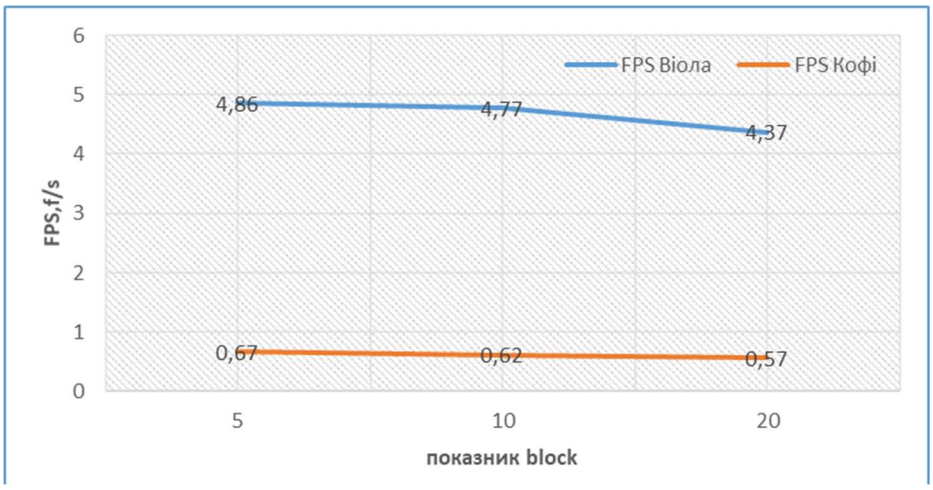


Рисунок 6.9 – Залежність FPS від зміни показнику block

Рисунок 6.10 – Детектування обличчя при різних положеннях маски(очі, ніс, рот)

6.4 Висновок

Таким чином, запропоновані нові алгоритми виявлення і детектування облич на цифрових зображення х і відеопослідовністях. У них реалізована можливість адаптивно регулювати рівень середньої яскравості кадру, що дозволяє поліпшити основні показники роботи. Можливість правильного виявлення облич для таких алгоритмів становить ~ 96%. Вони реалізовані на мові Python з використанням ресурсів бібліотеки OpenCV. Це дозволило вести обробку даних в реальному масштабі часу. Використання цих результатів дає можливість підвищити якість роботи і достовірність результатів при розпізнаванні облич в різних системах.

Аналізуючи результати експерименту можна встановити, що алгоритм Кофі не підходить для обробки інформації у реальному часі, так як відстає по параметрам швидкості від методу Віоли-Джонса. Але в той же час, предоставляє більш високий шанс детектувати обличчя у різних положеннях і не несе додаткові помилкові спрацьовування.

Отже рекомендую використовувати алгоритм Кофі тільки для відеофайлів і зображень, де швидкість не є головним фактором.

Стосовно, алгоритму Віоли-Джонса, здатність до параметризації своїх налаштувань дала змогу цьому методу збільшити максимальну границю вхідних даних, таких як ефективна відстань, приховане обличчя і тощо. А також високий параметр швидкості (FPS) дозволяє використовувати цей метод як у реальному часі, так і для окремих файлів при умові налаштування параметрів.

7 ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

Розвиток науково-технічного прогресу на сучасному етапі значною мірою залежить від впровадження обчислювальної техніки в усі сфери людської діяльності, в тому числі і в науково-дослідні проекти. Створювані на базі комп'ютерів інформаційні системи дозволяють обробляти велику кількість інформації і своєчасно приймати управлінські рішення в різних сферах, де потрібна обробка великої кількості даних. А це в свою чергу вимагає розробки засобів програмного забезпечення. За оцінками фахівців витрати на розробку програмного забезпечення ростуть швидше, ніж витрати на створення обчислювальної техніки, тому розробники програм повинні враховувати не тільки технічну, а й економічну доцільність розробки. Економічна доцільність розробки і впровадження програмного забезпечення визначається економічним ефектом, який буде отриманий виробниками при їх реалізації і споживачами при їх використанні. За величиною очікуваного економічного ефекту приймається рішення про доцільність інвестицій в розробку того чи іншого програмного продукту. За характером об'єкта вкладень інвестиції в розробку програмного забезпечення відносять до інтелектуальних інвестицій.

При створенні програмного продукту дуже важливо оцінити його вартість.

Це тим більш важливо, якщо програмний продукт є затребуваним і має всі шанси конкурувати вийти на ринок програмного забезпечення.

Існують способи і рекомендації до розрахунку собівартості розробки і написання програмного забезпечення .

Обсяг вихідних текстів програми, перш за все, відображає трудомісткість і тривалість розробки програмного забезпечення і дозволяє оцінювати відносні показники продуктивності праці фахівців-розробників. Обсяг програм в сучасних публікаціях наводиться в різних одиницях, які можна розділити на дві групи:

1) група, яка характеризує обсяг початкового програмного коду, які розробляються й аналізуються програмістом (це символи в початковому тексті програми на будь-яких мовах програмування; лексеми, які об'єднують групи символів, що мають загальне смисловий зміст в тексті програми; оператори мови програмування рівня асемблера; рядки тексту програми на мові програмування високого рівня);
2) група, яка відображає обсяг програми, що розміщується в реалізує ЕОМ (це байти, зайняті текстом програми в пам'яті ЕОМ; команди або операції реалізує , складові текст програми в об'єктних коді; слова пам'яті, обумовлені структурою даної реалізує , які використовуються для зберігання виконуваної програми і / або бази даних при функціонуванні програмних засобів).

Обсяг програми, що розміщується на , впливає на характеристики і вартість машин, яка залежить від необхідної пам'яті і продуктивності . З огляду на, що при розробці програми "Video Stream Quality" вибір не проводився і конкретні вимоги до реалізує машині не пред'являлися, будемо користуватися одиницями першої групи.

7.1 Основні статті витрат при розробці програми

Основна праця фахівця, який розробляє програмне забезпечення, вкладається в розробку тексту програми і розробку алгоритмів, за якими текст написаний. Бажано, щоб обрана одиниця виміру була б найбільшою мірою адекватна трудомісткості розробки. Крім того, одиниця виміру обсягу повинна бути наочною і просто вимірюваної [джерело?]. З цих позицій застосування числа лексем для характеристики обсягу програми поки скрутна, тим більше що відсутній досвід використання цього показника [джерело?]. Таким чином, базовим показником для визначення складових витрат праці є умовне число операторів в програмі.

Різні джерела радять вважати за число операторів в програмі наступні величини:

- число команд на мові асемблера;
- число логічних операторів в програмі, операторів переходу, арифметичних операторів і інших операторів у вихідному коді програми;
- число рядків в програмі (для мов високого рівня).

Програма "Video Stream Quality" розроблялася на мові високого рівня Python.

При його розробці були враховані такі сучасні рекомендації до структурному програмуванню як відсутність умовних і безумовних переходів, запис операторів в один рядок (за несуттєвими винятками), лінійний підхід до програмування. Отже, за число операторів в програмі можна взяти число рядків в програмі. Слід зазначити, що в це число не входять коментарі, вказівки і заголовки, так як ці конструкції не використовуються при нормальному функціонуванні програми.

7.1.1 Розрахунок трудомісткості розробки програмного забезпечення

Базовий показник для визначення складових витрат праці обчислюється за формулою:

$$\boldsymbol{Q = q \cdot c \cdot (1 + p)}, \qquad (7.1)$$

де q - число операторів (вихідних команд) в програмному продукті, так само 1086; c - коефіцієнт складності програми; p - коефіцієнт корекції програми в ході її розробки, залежить від точності і коректності поставленого завдання - приймаємо рівним 0.06.

Коефіцієнт складності програми визначається з таблиці 3.1 на перетині "групи складності" і "ступеня новизни". При цьому новизна визначається за принципом: А - розробка принципово нових завдань, Б - розробка оригінальних програм, В - розробка програм з використанням типових рішень, Г - разова звичайне завдання. А складність визначається виходячи з типу вирішуваних завдань: 1 - алгоритми оптимізації і моделювання систем, 2 - завдання обліку, звітності та статистики, 3 - стандартні алгоритми. Крім того, в таблиці вказано коефіцієнт недостатності опису програми, який буде потрібно трохи пізніше.

Таблиця 7.1 - Коефіцієнти розрахунку трудомісткості

Мова програмування	Група складності	ступінь новизни				коефіцієнт В
		А	Б	В	Г	
високого рівня	1	1,38	1,26	1,15	0,69	1,2
	2	1,30	1,19	1,08	0,65	1,35
	3	1,20	1,10	1,00	0,60	1,5
низького рівня	1	1,58	1,45	1,32	0,79	1,2
	2	1,49	1,37	1,24	0,74	1,35
	3	1,38	1,26	1,15	0,69	1,5

Програма "Video Stream Quality" написана на мові високого рівня, відноситься до моделювання систем і є принципово новою розробкою; тобто коефіцієнт складності програми в даному випадку: з = 1,38. Таким чином, знаходимо базовий показник: Q = тисяча п'ятсот вісімдесят дев'ять.

Далі, розрахуємо складові витрати праці, серед яких виділяють: витрати праці на підготовку і опис алгоритму, витрати праці на дослідження алгоритму, витрати праці на розробку алгоритму, витрати праці на програмування і налагодження та витрати праці на підготовку документації. Майже всі ці параметри будуть залежати від базового показника, що розраховується за формулою (7.1).

Витрати праці на підготовку і опис завдання може визначатися емпірично або за формулою (7.2):

$$\boldsymbol{t}_{\text{оп}} = \frac{\boldsymbol{T}_{\min}+\boldsymbol{4}\cdot\boldsymbol{T}_{\text{нв}}+\boldsymbol{T}_{\max}}{\boldsymbol{6}} = \frac{26+\boldsymbol{4}\cdot 52+78}{\boldsymbol{6}} = 52[\text{люд.год}], \quad (7.2)$$

де T_{max} - трудомісткість операції в найбільш несприятливих умовах (песимістична оцінка); Tmin - трудомісткість операції при сприятливих умовах (оптимістична оцінка); Тнв - трудомісткість операції при нормальних умовах (найбільш ймовірна оцінка). Орієнтовні величини оцінки трудомісткості операції підготовки опису завдання в залежності від числа операторів q наводяться в таблиці 7.2.

Таблиця 7.2 - Оцінка часу підготовки опису завдання

q	T_{min}	$T_{нв}$	T_{max}
100	10	15	20
500	20	35	50
1 000	25	50	75
1500	30	60	90
2000	40	70	100
2500	50	80	110
5000	70	110	150
10000	100	150	200

Витрати праці на дослідження алгоритму розв'язання задачі визначаються формулою (7.3):

$$\boldsymbol{t}_{ис} = \frac{\boldsymbol{Q \cdot B}}{(75 \div 85)\boldsymbol{k}} \text{ [люд.год]}, \qquad (7.3)$$

де Q - базовий коефіцієнт, що розраховується за формулою (5.1), В - коефіцієнт недостатності опису завдання, який береться з таблиці 3 і дорівнює 1,2; k - коефіцієнт кваліфікації програміста, залежить від стажу працівника і визначається з таблиці 9.9.

Таблиця 7.3 - Коефіцієнти кваліфікації програміста

Досвід роботи	коефіцієнт кваліфікації
До двох років	0.8
2-3 роки	1
3-5 років	1.1 - 1.2
5-7 років	1.3 - 1.4
більше 7 років	1.5 - 1.6

По таблиці визначаємо коефіцієнт k = 0.8.

Таким чином, знаходимо витрати праці на дослідження алгоритму розв'язання задачі: tic = 30 [люд.год].

Витрати праці на розробку блок-схем алгоритмів (представлених на малюнках 2.7 і 2.8) обчислюються за формулою (7.4):

$$\boldsymbol{t}_{\text{ал}} = \frac{\boldsymbol{Q}}{(20 \div 25)\boldsymbol{k}} = 88[\text{люд.год}].$$

(7.4)

Витрати праці на програмування алгоритму по блок-схемі і налагодження програми обчислюються за формулами (7.5, 7.6):

$$\boldsymbol{t}_{\text{пр}} = \frac{\boldsymbol{Q}}{(20 \div 25)\boldsymbol{k}} = 88[\text{люд.год}],$$

(9.5)

$$\boldsymbol{t}_{\text{отл}} = \frac{\boldsymbol{Q}}{(\mathbf{4} \div \mathbf{5})\boldsymbol{k}} = 397[\text{люд.год}].$$

(7.6)

Витрати праці на підготовку документів по завданню складаються з витрат праці на підготовку рукописів і часу на оформлення документів і обчислюються за формулою (7.7):

$$\boldsymbol{t}_{\partial} = \boldsymbol{t}_{\text{рук}} + \boldsymbol{t}_{\text{оф}} = \frac{\boldsymbol{Q}}{(15 \div 20)\boldsymbol{k}} + 0.75 \cdot \boldsymbol{t}_{\text{рук}} = 113 + 85 = 198[\text{люд.год}].$$

(7.7)

Сумарні витрати праці розраховуються як сума складових витрат праці за формулою (7.8):

$$\boldsymbol{t}_{\boldsymbol{\Sigma}} = \boldsymbol{t}_{\text{оп}} + \boldsymbol{t}_{\text{ис}} + \boldsymbol{t}_{\text{ал}} + \boldsymbol{t}_{\text{пр}} + \boldsymbol{t}_{\text{отл}} + \boldsymbol{t}_{\partial} = 52 + 30 + 88 + 88 + 397 + 198 = 853[\text{люд.год}].$$

(7.8)

7.1.2 Розрахунок витрат на розробку програмного забезпечення

Заробітна плата складається з двох складових: основної заробітної плати і додаткової.

Основна заробітна плата розраховується за формулою (7.9):

$$З_{\text{осн}} = \frac{\boldsymbol{t}_{\boldsymbol{\Sigma}}}{\boldsymbol{t}_{\text{ср}} \cdot \mathbf{8}} \cdot \text{ТС}[\text{грн}],$$

(7.9)

де tΣ - сумарні витрати праці, які обчислюють за формулою (7.2); tср - середня кількість днів у місяці, так само 21-го дня, множиться на кількість годин в робочому дні - 8; ТС - тарифна ставка.

Тарифна ставка являє собою МРОТ (мінімальний розмір оплати праці, станом на травень 2009 року дорівнює 4330 гривень [Федеральний закон від 17.06.2000 № 82-ФЗ "Про мінімальний розмір оплати праці"]), збільшений в залежності від тарифного коефіцієнта, яке відповідає даному виду робіт. Для 12-го розряду робіт [Постанови Уряду РФ від 06.01.93 N 14, від 27.02.95 N 189], який відповідає роботі програміста, тарифний коефіцієнт дорівнює 2.44.

Таким чином, основна заробітна плата буде становити:

$$З_{\text{осн}} = \frac{853}{21 \cdot \mathbf{8}} \cdot 4330 \cdot 2.44 = 53643[\text{грн}].$$

Додаткова заробітна плата становить 20% від основної заробітної плати, розраховується за формулою (7.10):

$$З_{доп} = 0.2 \cdot З_{осн} = 10728[грн]. \quad (7.10)$$

Сумарна заробітна плата (або фонд заробітної плати, ФЗП) обчислюється як сума основної і додаткової заробітних плат за формулою (7.11):

$$ФЗП = З_{осн} + З_{доп} = 64371[грн]. \quad (7.11)$$

7.2 Додаткові статті витрат

Серед додаткових статей витрат на розробку програмного забезпечення виділяють: витрати на матеріали і комплектуючі (вартість самого обладнання, тобто комп'ютера, до уваги не береться), відрахування на соціальне страхування, накладні витрати, амортизаційні відрахування, витрати на технічне обслуговування обладнання і вартість витраченої електроенергії при роботі на комп'ютері.

Вартість обладнання хоч і не включається в собівартість розробки програмного забезпечення, але все ж використовується при розрахунку деяких інших додаткових статей витрат. При написанні програми на в якості обладнання передбачається персональний комп'ютер, вартість якого становить:$\boldsymbol{C}_{обор} = 26400[грн]$.

Витрати на матеріали і комплектуючі, які використовуються в процесі написання програмного продукту (Смик), а також витрати на технічне обслуговування і ремонт (СТО) складають, відповідно, 1.5% і 2.5% від вартості обладнання - формули (7.12 - 7.13):

$$C_{ММК} = 0.015 \cdot С_{обор} = 396[грн], \quad (7.12)$$

$$C_{ТО} = 0.025 \cdot С_{обор} = 660[грн]. \quad (7.13)$$

Амортизаційні відрахування, процес поступового перенесення вартості засобів праці у міру їх фізичного та морального зносу на вартість вироблених з їх допомогою продукції з метою акумуляції коштів для подальшого повного відновлення. Амортизаційні відрахування провадяться за встановленими нормами амортизації, виражаються, в процентах до балансової вартості обладнання і розраховуються за формулою (7.14):

$$A_{год} = C_{обор} \cdot \frac{H_A}{100}, \quad (7.14)$$

де Собор - вартість комп'ютера; НА - норма амортизації, яка розраховується за формулою (9.15):

$$H_A = \frac{C_{обор} - C_{ликв}}{T_{норм} \cdot C_{обор}} \cdot 100\%, \quad (7.15)$$

де Слікв - ліквідаційна вартість, становить 5% від вартості обладнання: $C_{\text{ликв}} = 0.05 \cdot C_{\text{обор}} = 1320$[грн]; Тнорм - нормативний термін служби (для персонального комп'ютера приймемо Тнорм = 6 [років]). Таким чином, отримуємо: Агод = 4 180 [грн].

Варто також враховувати і витрати електроенергії при написанні програмного забезпечення. Вартість електроенергії обчислюється за формулою (7.16):

$$C_{\text{ЭЭ}} = M \cdot k_з \cdot F_{\text{эф}} \cdot C_{\text{кВт.ч}}, \quad (7.16)$$

де М - потужність (450 Вт); k з - коефіцієнт завантаження (0.8); СкВт.ч - вартість 1 кВт-год електроенергії (2 грн. 31 коп. На момент написання цієї роботи); F еф - ефективний фонд робочого часу, розраховується за формулою (7.17):

$$F_{\text{эф}} = Д_{\text{ном}} \cdot d \cdot \left(1 - \frac{f}{100}\right), \quad (7.17)$$

де Дном = 258 - номінальне число робочих днів у році; d = 8 - тривалість робочого дня [час]; f = 2% - планований відсоток часу на ремонт .

При даних значеннях параметрів і коефіцієнтів вартість електроенергії складе СЕЕ = 1 682 [грн].

Однак, отримані значення амортизаційних відрахувань і витрат на електроенергію - значення річних витрат, необхідно їх скорегувати відповідно до тимчасового коефіцієнтом, який визначається виходячи з сумарних річних експлуатаційних витрат, які розраховуються за формулою (7.18):

$$Э_з = t_\Sigma \cdot \frac{C_Э}{F_{\text{эф}}} \text{[грн]}, \quad (7.18)$$

де $C_Э = C_{\text{ЭЭ}} + C_{\text{ТО}} + A_{\text{год}}$ - сумарна річна вартість експлуатаційних витрат, F еф - ефективний фонд робочого часу, обчислений за формулою (7.17), t_Σ - загальний час використання для вирішення завдання, яке обчислюється аналогічно формулі (7.8), враховуючи лише час роботи на комп'ютері:

$t_\Sigma = t_{\text{пр}} + t_{\text{отл}} + t_д = 88 + 397 + 198 = 683$[час].

Отже, сумарні витрати на експлуатацію становитимуть: Ез = 2 203 [грн], а сам тимчасової коефіцієнт обчислюється за формулою (7.19):

$$w = \frac{Э_з}{C_Э} = 0.3377. \quad (7.19)$$

Таким чином, з огляду на тимчасової коефіцієнт, з сумарних експлуатаційних витрат скорегуємо:

- витрати на електроенергію $C_{\text{ЭЭ}} = C_{\text{ЭЭ}} \cdot w = 568$ [грн];
- амортизаційні відрахування $A_{\text{год}} = A_{\text{год}} \cdot w = 1411$ [грн].

Відрахування на соціальне страхування становлять 35.6% від усієї заробітної плати [Податковий Кодекс РФ ч. 2 від 30.05.2001, стаття 241], обчислюються за формулою (7.20):

$$\text{СС} = 0.356 \cdot \text{ФЗП} = 22916\text{[грн]}. \quad (7.20)$$

Накладні витрати, пов'язані з управлінням і обслуговуванням, утриманням та експлуатацією устаткування і іншими додатковими витратами на забезпечення процесів виробництва і обігу, становлять 50% від фонду заробітної плати, визначаються за формулою (7.21):

$$C_{\text{накл}} = 0.5 \cdot \text{ФЗП} = 32185[\text{грн}]. \quad (7.21)$$

7.3 Результуюча таблиця собівартості

Сумарні витрати на розробку програмного забезпечення вважаються як сума фонду заробітної плати, експлуатаційних витрат, витрат на соціальне страхування, накладних витрат і витрат на матеріали і комплектуючі.

Підсумкова вартість розробки програмного забезпечення представлена в таблиці 7.4.

Таблиця 7.4 - Результуюча таблиця собівартості

Стаття витрат		Сума, грн.	У відсотках від загальної суми
ФЗП	*Зосн*	53 643	43.78
	Здоп	10 728	8.76
Накладні витрати, Снакл		32 158	26.26
Соціальне страхування, СС		22 916	18.71
Експлуатаційні витрати	*СЕЕ*	568	0.46
	СТО	660	0.54
	Агод	1 411	1.15
Матеріали і комплектуючі, Смик		396	0.34
Разом:		122 480	

7.4 Висновки

В результаті економічних розрахунків було проведено економічне обґрунтування розробки програмного продукту. Розрахунок вартості проведений з урахуванням всіх трудовитрат, ПДВ, відрахувань в пенсійний фонд, у фонд зайнятості і відрахувань на соціальне страхування, накладних витрат. Розрахована собівартість розробки веб-додатку та впровадження до ринку дорівнює 122 480 грн.

ВИСНОВОК

Запропоновано новий комплексний підхід до проблеми підвищення якості їх обробки в сучасних мобільних СТЗ на базі мікрокомп'ютерів Raspberry Pi. Спільно оптимізуються показники швидкості захоплення і обробки відеоданих, роздільної здатності відео кадру і показників стабільності яскравості кадру незалежно від умов освітлення сцени. Ефективність пропонованих алгоритмів їх обробки і методів синтезу програмних кодів досліджена експериментально.

Наукова новизна. Аналіз показав істотне підвищення якості реєстрованого відеопотоку при використанні пропонованих алгоритмів. Вперше нові методи обробки відеоданих отримані на основі об'єктивних критеріїв оптимізації із застосуванням сучасних засобів програмування.

Практичне значення. У подальшому отримані результати можуть бути використані для створення різних проектів на базі мобільних СТЗ. За результатами проведених досліджень нескладно створити необхідне високоякісне програмне забезпечення.

Перспективи для подальших досліджень. В подальшому найбільш доцільним для завершення робіт по даному напрямку провести докладні дослідження характеристик введення і попередньої обробки відеоданих від двох відеокамер для формування на базі комп'ютера Raspberry Pi сучасної мобільної системи стереозору.

СПИСОК ВИКОРИСТАННИХ ДЖЕРЕЛ

1. https://www.raspberrypi.org [Електронний ресурс] - Офіційний сайт документації для роботи з комп'ютером Raspberry Pi.
2. Магда Ю. С. Raspberry Pi. Керівництво настроїти та використовувати. 2014.- 188 с.
3. http://python.org/[Електронний ресурс] - Офіційний сайт мови Python.
4. http://opencv.org. [Електронний ресурс] - Офіційний сайт розробників бібліотеки OpenCV
5. Федоров Д. Ю. Основи програмування Python: / Д. Ю. Федоров. - СПб., 2016. - 176 с.
6. C. Harris and M. Stephens. "A combined corner and edge detector"// Proceedings of the 4th Alvey Vision Conference. – 1988. - P. 147–151.
7. Krig, S. Computer vision metrics: Survey, taxonomy, and analysis / S. Krig. – Berkeley, CA: Apress Media, 2014. – 498 p. – ISBN: 978-1-4302-5929-9.
8. Jain, M. A survey on CBIR on the basis of different feature descriptor / M. Jain, D. Singh // British Journal of Mathematics & Computer Science. – 2016. – Vol. 14, Issue 6. – P. 1-13. – DOI: 10.9734/BJMCS/2016/24000.
9. Ojala, T. A comparative study of texture measures with classification based on feature distributions / T. Ojala, M. Pietikäinen, D. Hardwood // Pattern Recognition. – 1996. – Vol. 29, Issue 1. – P. 51-59. – DOI: 10.1016/0031-3203(95)00067-4.
10. Calonder, M. BRIEF-binary robust independent elementary features / M. Calonder, V. Lepetit, C. Strecha, P. Fua // European Conference on Computer Vision. – 2010. – Part IV. – P. 778-792. – DOI: 10.1007/978-3-642-15561-1_56.
11. Rublee, E. ORB: An efficient alternative to SIFT or SURF / E. Rublee, V. Rabaud, K. Konolige, G. Bradski // IEEE International Conference on Computer Vision (ICCV). – 2011. – P. 2564-2571. – DOI: 10.1109/ICCV.2011.6126544.
12. Leutenegger, S. BRISK: Binary Robust invariant scalable keypoints / S. Leutenegger, M. Chli, R. Siegwart // IEEE International Conference on Computer Vision (ICCV'11). – 2011. – P. 2548-2555. – DOI: 10.1109/ICCV.2011.6126542.
13. Lowe, D.G. Distinctive image features from scale-invariant keypoints / D.G. Lowe // International Journal of Computer Vision. – 2004. – Vol. 60, Issue 2. – P. 91-110. – DOI: 10.1023/B:VISI.0000029664.99615.94.
14. Bay, H. SURF: Speeded up robust features / H. Bay, A. Ess, T. Tuytelaars, L. Van Gool // Computer Vision and Image Understanding. – 2008. – Vol. 110, Issue 3. – P. 346-359. – DOI: 10.1016/j.cviu.2007.09.014.
15. Tola, E. DAISY: An efficient dense descriptor applied to wide-baseline stereo / E. Tola, V. Lepetit, P. Fua // IEEE Transactions on Pattern Analysis and

Machine Intelligence. – 2010. – Vol. 32, Issue 5. – P. 815-830. – DOI: 10.1109/TPAMI.2009.77.
16. Dalal, N. Histograms of oriented gradients for human detection / N. Dalal, B. Triggs // IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005). – 2005. – Vol. 1. – P. 886-893. – DOI: 10.1109/CVPR.2005.177.
17. Scharstein, D. A taxonomy and evaluation of dense two frame stereo correspondence algorithms / D. Scharstein, R. Szeliski // International Journal of Computer Vision. – 2002. – Vol. 47, Issue 1-3. – P. 7-42. – DOI: 10.1023/A:1014573219977.
18. Jun, B. Robust face detection using local gradient patterns and evidence accumulation / B. Jun, D. Kim // Pattern Recognition. – 2012. – Vol. 45, Issue 9. – P. 3304-3316. – DOI: 10.1016/j.patcog.2012.02.031.
19. Freeman, H. On the encoding of arbitrary geometric configurations / H. Freeman // IRE Transactions on Electronic Computers. – 1961. – Vol. EC-10, Issue 2. – P. 260-268. – DOI: 10.1109/TEC.1961.5219197.
20. Gonzalez, R. Digital image processing / R. Gonzalez, R. Woods. – 3rd ed. – Upper Saddle River, NJ: PrenticeHall, 2007. – 976 p. – ISBN: 978-0-13-168728-8.
21. Borg, I. Modern multidimensional scalling: Theory and applications / I. Borg, P. Groenen. – 2nd ed. – New York, NY: Springer-Verlag, 2005. – 614 p. – P. 207-212. – ISBN: 0-387- 25150-2.32. Viola, P. Rapid Object detection using a boosted cascade of simple features / P. Viola, M. Jones // Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001). – 2001. – P. 511-518. – DOI: 10.1109/CVPR.2001.990517.
22. Bracewell, R. The Fourier transform and its applications / R. Bracewell. – 3rd ed. – New York: McGraw-Hill Science, 1999. – 640 p.
23. Fei-Fei, L. Recognizing and learning object categories / L. Fei-Fei, R. Fergus, A. Torralba // Conference on Computer Vision and Pattern Recognition. – 2007.
24. Matas, J. Robust widebaseline stereo from maximally stable extremal regions / J. Matas, O. Chum, M. Urban, T. Pajdla // British Machine Vision Conference. – 2002. – P. 384-393.
25. Belongie, S. Shape matching and object recognition using shape contexts / S. Belongie, J. Malik, J. Puzicha // IEEE Transactions on Pattern Analysis and Machine Intelligence. – 2002. – Vol. 24, Issue 4. – P. 509-522. – DOI: 10.1109/34.993558.
26. Ломов, Н.А. Площадь дискового покрытия – дескриптор формы изображения / Н.А. Ломов, Л.М. Местецкий // Компьютерная оптика. – 2016. – Т. 40, № 4. – С. 516- 525. – DOI: 10.18287/2412-6179-2016-40-4-516-525.
27. Сидякин, С.В. Морфологические дескрипторы формы бинарных изображений на основе эллиптических структурирующих элементов / С.В. Сидякин, Ю.В. Визильтер // Компьютерная оптика. – 2014. – Т. 38, № 3. – С. 511-520. – DOI: 10.18287/0134-2452-2014-38-3-511-520.

28. Bauckhage, C. Bounding box splitting for robust shape classification / C. Bauckhage, J.K. Tsotsos // IEEE International Conference on Image Processing (ICIP 2005). – 2005. – P. 478-481. – DOI: 10.1109/ICIP.2005.1530096.
29. Sonka, M. Image processing, analysis and machine vision / M. Sonka, V. Hlavac, R. Boyle. – London: Chapman and Hall, 1993. – 872 p.
30. Abbasi, S. Enhancing CSS-based shape retrieval for objects with shallow concavities / S. Abbasi, F. Mokhtarian, J. Kittler // Image and Vision Computing. – 2000. – Vol. 18, Issue 3. – P. 199-211. – DOI: 10.1016/S0262-8856(99)00019- 0.
31. Siddiqi, K. A shock grammar for recognition / K. Siddiqi, B. Kimia // Proceedings of the 1996 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '96). – 1996. – P. 507-513. – DOI: 10.1109/CVPR.1996.517119.
32. Zakharov, A.A. Finding correspondences between images using descriptors and graphs / A.A. Zakharov, A.Yu. Tuzhilkin, A.L. Zhiznyakov // Procedia Engineering. – 2015. – Vol. 129. – P. 391-396. – DOI: 10.1016/j.proeng.2015.12.131.
33. Barinov, A.E. Clustering using a random walk on graph for head pose estimation / A.E. Barinov, A.A. Zakharov // 2015 International Conference on Mechanical Engineering, Automation and Control Systems (MEACS). – 2015. – DOI: 10.1109/MEACS.2015.7414876.
34. Баринов, А.Е. Алгоритм спектральной кластеризации с ограничениями для выделения лица человека на изображениях / А.Е. Баринов, А.А. Захаров, А.Л. Жизняков // Динамика систем, механизмов и машин. – 2016. – Т. 2, № 1. – С. 222-228.
35. Zakharov, A.A. Recognition of human pose from images based on graph spectra / A.A. Zakharov, A.E. Barinov, A.L. Zhyznyakov // The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. – 2015. – Vol. XL-5/W6. – P. 9-12. – DOI: 10.5194/isprsarchives-XL-5-W6-9-2015.
36. Chung, F.R.K. Spectral graph theory / F.R.K. Chung. – Provides, Rhode Island: AMS, 1997. – 207 p. – ISBN: 0- 8218-0315-8.
37. ElGhawalby, H. Measuring graph similarity using spectral geometry / H. ElGhawalby, E.R. Hancock. – In book: Image Analysis and Recognition. ICIAR 2008 / ed. by A. Campilho, M. Kamel. – Berlin, Heidelberg: Springer, 2008. – P. 517-526. – DOI: 10.1007/978-3-540-69812- 8_51.
38. K. Dergachov, L. Krasnov, O. Cheliadin, A. Zymovin. Adaptive algorithms of face detection and effectiveness assessment of their use. Advanced Information Systems. 2018. Vol. 2, № 3, National Technical University “Kharkiv Polytechnic Instityte”. ISSN 2522-9052.
39. Dergachov, L. Krasnov, O. Cheliadin, O. Plakhotnyi. Web-cameras stereo pairs color correction method and its practical implementation. Advanced Information Systems. 2018. Vol. 3, № 1, National Technical University “Kharkiv Polytechnic Instityte”, 2019, c. 29-42. ISSN 2522-9052.

ПУБЛІКАЦІЇ ВИКОНАНІ В ПРОЦЕСІ ПРОВЕДЕННЯ РОБОТИ

1. Підвищення якості введення, попередньої обробки і записи відео в мобільних системах технічного зору/ К. Ю. Дергачов, Л. О. Краснов, О. О. Челядин, Р. Є. Казатинський// Всеукраїнський конкурс студентських наукових робіт зі спеціальності “Автоматизація та комп’ютерно-інтегровані технології” 2019/2020.
2. Практична реалізація методів і засобів введення відеоданих в мікрокомп’ютерах Raspberry Pi // Міжнародна наукова інтернет-конференція "Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення" (випуск 45) 2020
3. Video data quality improvement methods and tools development for mobile vision systems K. Dergachov, L. Krasnov, O. Cheliadin, R. Kazatynskyi// ISSN 2522-9052.Advanced Information Systems. 2020. Vol. 4, № 2, National Technical University “Kharkiv Polytechnic Instityte”, 2020

ДОДАТОК А

Листинг программы face_detection.py

Імпорт бібліотеки `imutils` для методів считування данних

```
from imutils.video import FPS
from imutils.video import VideoStream
from imutils.video.pivideostream import PiVideoStream
from imutils.video import FileVideoStream
```

Імпорт бібліотек обробки зображення

```
from picamera.array import PiRGBArray
from picamera import PiCamera
import numpy as np
import argparse
import imutils
import time
import cv2
import datetime
```

Імпорт файлів детектування і анонімізації

```
from face_blurring import anonymize_face_pixelate
from face_blurring import anonymize_face_simple
from imutils import face_utils
```

Ініціалізація констант, які є параметрами виконуваних функцій

```
global fps,rwidth,method,start_streams
global nwidth,nheigh
nwidth=[1280,1280 ,1200 ,1024 ,960  ,800 ,640 ,640  ,480 ,640 ,320
]
nheigh=[960,720,600,768,540,600,480,360,320,240,240]
rwidth=6
method=5
```

```
anonymaze=0

neiro=1

video_input=False

minNeigbors=5

const_confidence=0.5

const_factor=3.0

const_block=20

name_file=0
```

Ініціалізація каскадів Хаару і Кофі

```
face_cascade=cv2.CascadeClassifier('haarcascade_frontalface_de-
fault.xml')

eye_cascade=cv2.CascadeClassifier('haarcascade_eye.xml')

net = cv2.dnn.readNetFromCaffe("deploy.pro-
totxt.txt", "res10_300x300_ssd_iter_140000.caffemodel")
```

Функція детектування за алгоритмом Віоли-Джонса обличь приймає як вхідний параметр фрейм, повертає змінений фрейм `current_frame` і параметр відповіщаючий за вдатність детектування обличчя `face_detected`.

```
def viola_face(current_frame):

    face_detected = False

    gray = cv2.cvtColor(current_frame, cv2.COLOR_BGR2GRAY)

    faces = face_cascade.detectMultiScale(gray, 1.3, minNeigbors)

    for (x, y, w, h) in faces:

        face_detected = True
```

Додання рамки біля відображеного обличчя

```
        if anonymaze==0:

            cv2.rectangle(cur-
rent_frame, (x, y), (x + w, y + h), (0, 0, 255), 2)
```

Якщо параметр анонімізації більше нуля то виклик методу анонімізації

```
        else:
            (startX, startY, endX, endY) = (x,y,x+w,y+h)
            face = current_frame[startY:endY, startX:endX]
            if anonymaze==1:
                face = anonymize_face_simple(face, fac-
tor=const_factor)
            if anonymaze==2:
                face = anonymize_face_pixe-
late(face,blocks=const_block)
            current_frame[startY:endY, startX:endX] = face


    return current_frame,face_detected
```

Функція детектування обличь за алгоритмом Кофі приймає як вхідний параметр фрейм, повертає змінений фрейм `current_frame` і параметр відповіщаючий за вдатність детектування обличчя `face_detected`.

```
def detecting_kofi(frame):
    (h, w) = frame.shape[:2]
    blob = cv2.dnn.blobFro-
mImage(cv2.resize(frame, (300, 300)), 1.0,(300, 300), (104.0, 177.
0, 123.0))
    net.setInput(blob)
    detections = net.forward()
    detect=False
    for i in range(0, detections.shape[2]):
        confidence = detections[0, 0, i, 2]
        if confidence < const_confidence:
            continue
        box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
        (startX, startY, endX, endY) = box.astype("int")
        detect=True
```

Добавление рамки біля відображеного обличчя

```
        if anonymaze==0:
            y = startY - 10 if startY - 10 > 10 else startY + 10
            cv2.rectan-
gle(frame, (startX, startY), (endX, endY),(0, 0, 255), 2)
        else:
```

Якщо параметр анонімізації більше нуля то виклик методу анонімізації

```
            face = frame[startY:endY, startX:endX]
            if anonymaze==1:
                face = anonymize_face_simple(face, fac-
tor=const_factor)
            if anonymaze==2:
                face = anonymize_face_pixe-
late(face,blocks=const_block)
            frame[startY:endY, startX:endX] = face
    return frame,detect
```

Функція основного методу викликаючего режими вводу відеоданих відносно початкового параметру

```
def body():
    if method==1:
        read_slow()
    if method==2:
        read_fast()
    if method==3:
        read_videostream()
    if method==4:
        read_picamera()
    if method==5:
        read_pivideostream()
```

Функція для здатності запускати додаток через консоль , передаючи параметри через консоль

```
def arguments():

    global args

    ap = argparse.ArgumentParser()

    ap.add_argument("-v", "--video", re-
quired=True,help="path to input video file")

    args = vars(ap.parse_args())
```

Функція відображення додаткової інформації о сесії

```
def print_information():

    print("[INFO] Name files " +str(name_file))

    print("[INFO] Number method "+str(method)+" num-
ber method neiro "+str(neiro))

    print("[INFO] elasped time: {:.2f}".format(fps.elapsed()))

    print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))

    print("[INFO] approx. Frame: "+str(int(fps._numFrames)))

    print("[INFO] Width",nwidth[rwidth],"[INFO] Heigh",nheigh[rwid
th])

    print("[INFO] frame detecting %",str(((int(fps._numFrames)-
fps._frame_detecting)*100)/fps._numFrames))
```

Функція попередньої обробки зображення, також в цьому блоку відбувається додання тексту на зображення

```
def transform_frame(frame):

    if (frame is None) :

        return

    #frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    #frame = np.dstack([frame, frame, frame])

    #frame = cv2.cvtColor(frame, cv2.COLOR_BGR2YUV)

    #frame[:,:,0] = cv2.equalizeHist(frame[:,:,0])

    #frame = cv2.cvtColor(frame, cv2.COLOR_YUV2BGR)
```

```
    if (method)<4: frame = imutils.resize(frame, width=nwidth[rwid
th],height=nheigh[rwidth])




    if(neiro!=0):

        if(neiro==1):

            frame,fps._detected=viola_face(frame)

        else:

            frame,fps._detected=detecting_kofi(frame)


    name_meth=["Slow Method","Fast Method","Videostream","Picam-
era","PiVideostream"]

    Text=name_meth[method-1]+" Resolu-
tion " +str(int(nwidth[rwidth]))+"x"+str(int(nheigh[rwidth]))

    Text_FPS="frame "+str(int(fps._num-
Frames))+" "+"  "+str(fps_1)+" second"



    cv2.putText(frame,Text_FPS, (10, 400),cv2.FONT_HERSHEY_SIM-
PLEX, 0.5, (0, 255, 0), 2)

    frame = imutils.resize(frame, width=nwidth[rwidth],height=nhei
gh[rwidth])

    return frame
```

Функція для закінчення сесії згідно з таймером

```
def out_read(start_stream):

    return ((datetime.datetime.now()-start_stream).total_sec-
onds()>20)
```

Метод зчитування відеоданих для Веб Камери, без запису в стрим.

```
def read_slow():

    if(video_input): stream = cv2.VideoCapture(0)

    else:  stream = cv2.VideoCapture(name_file)

    start_stream=datetime.datetime.now()

    while True:
```

```
        (grabbed, frame) = stream.read()
        if not grabbed:
            break
        frame=transform_frame(frame)
        cv2.imshow("Frame", frame)
        cv2.waitKey(1)
        if (out_read(start_stream)):
            break
        fps.update()
    stream.release()
```

Метод зчитування відеоданих для Веб Камери, с записом в стрим.

```
def read_videostream():
    if(video_input): fvs = VideoStream(0).start()
    else:  fvs = VideoStream(name_file).start()
    time.sleep(2.0)
    start_stream=datetime.datetime.now()
    while True:
        if (out_read(start_stream) ):
            break
        frame = fvs.read()
        frame=transform_frame(frame)
        cv2.imshow("Frame", frame)
        cv2.waitKey(1)
        fps.update()
    fvs.stop()
```

Метод зчитування відеоданих для Веб Камери через додатковий канал.

```
def read_fast():
```

```
    if(video_input): fvs = FileVideoStream(0).start()
    else:  fvs = FileVideoStream(name_file).start()
    time.sleep(1.0)
    start_stream=datetime.datetime.now()
    while fvs.more():
        frame = fvs.read()
        frame=transform_frame(frame)
        cv2.imshow("Frame", frame)
        cv2.waitKey(1)
        if (out_read(start_stream)):
            break
        fps.update()
    fvs.stop()
```

Метод зчитування відеоданих для Pi Камери без запису в стрим.

```
def read_picamera():
    camera = PiCamera()
    camera.resolution = (nwidth[rwidth],nheigh[rwidth])
    camera.framerate = 32.
    rawCapture = PiRGBArray(cam-
era, size=(nwidth[rwidth], nheigh[rwidth]))
    stream = camera.capture_continuous(rawCapture, for-
mat="bgr",use_video_port=True)
    time.sleep(2.0)
    start_stream=datetime.datetime.now()

    for (i, f) in enumerate(stream):
        frame=f.array
        frame=transform_frame(frame)

        cv2.imshow("Frame", frame)
```

```
        key = cv2.waitKey(1) & 0xFF
        if (out_read(start_stream)):
            break
        rawCapture.truncate(0)
        fps.update()
    fps.stop()
    stream.close()
    rawCapture.close()
    camera.close()
```

Метод зчитування відеоданих для Pi Камери с записjv в стрим.

```
def read_pivideostream():

    fvs = PiVideoStream().start()
    time.sleep(2.0)

    start_stream=datetime.datetime.now()

    while True:
        frame = fvs.read()
        frame=transform_frame(frame)

        cv2.imshow("Frame", frame)
        key = cv2.waitKey(1) & 0xFF
        if (out_read(start_stream)):

            break
        fps.update()
```

```
    fvs.stop()
```

Ініціація початкових даних, початок запису даних, виклик методу `body()`, зупинення запису і відображення інформації о сесії.

```
start_streams=datetime.datetime.now()
fps = FPS().start()
fps.frame_detecting=0
fps.detected=False
body()
fps.stop()
print_information()
cv2.destroyAllWindows()
```

ДОДАТОК Б

Таблиця форматів відео

№№ п/п	Название формата	Разрешение (FR) на мониторе	Пропорции изображения	Размер изображения
1	QVGA	320×240	4:3	76,8 кпикс
2	HVGA	640×240	8:3	153,6 кпикс
3	HVGA	320×480	2:3	153,6 кпикс
4	nHD	640×360	16:9	230,4 кпикс
5	VGA	640×480	4:3	307,2 кпикс
6	SVGA	800×600	4:3	480 кпикс
7	qHD	960×540	16:9	518,4 кпикс
8	XGA	1024×768	4:3	786,432 кпикс
9	WXVGA	1200×600	2:1	720 кпикс
10	HD 720p	1280×720	16:9	921,6 кпикс
11	Full HD	1920x1080	16:9	2.07 мпикс

ДОДАТОК В

Клас відповіщаючий за реалізації вікон (створений за допомогою бібліотеки QA)

```
import sys

import random


from PyQt5 import QtCore, QtGui, QtWidgets

from PyQt5.QtGui import QIcon, QFont

from PyQt5.QtCore import QDir, Qt, QUrl, QSize

from PyQt5.QtMultimedia import QMediaContent, QMediaPlayer

from PyQt5.QtMultimediaWidgets import QVideoWidget

from PyQt5.QtWidgets import (QApplication, QFileDialog, QHBoxLayout, QLabel, QPushBut-
ton, QSizePolicy, QSlider, QStyle, QVBoxLayout, QWidget, QStatusBar)

import cv2

from PIL import ImageTk,Image


class Ui_MainWindow_1(object):

    def setupUi(self, MainWindow):

        MainWindow.setObjectName("MainWindow")

        #MainWindow.resize(400, 300)

        self.centralwidget = QtWidgets.QWidget(MainWindow)

        self.centralwidget.setObjectName("centralwidget")

        self.gridLayoutWidget = QtWidgets.QWidget(self.centralwidget)

        self.gridLayoutWidget.setGeometry(QtCore.QRect(70, 150, 271, 91))

        self.gridLayoutWidget.setObjectName("gridLayoutWidget")

        self.gridLayout = QtWidgets.QGridLayout(self.gridLayoutWidget)

        self.gridLayout.setContentsMargins(0, 0, 0, 0)

        self.gridLayout.setObjectName("gridLayout")

        self.pushButton = QtWidgets.QPushButton(self.gridLayoutWidget)

        self.pushButton.setObjectName("pushButton")

        self.gridLayout.addWidget(self.pushButton, 0, 0, 1, 1)
```

```
        self.pushButton_3 = QtWidgets.QPushButton(self.gridLayoutWidget)
        self.pushButton_3.setObjectName("pushButton_3")
        self.gridLayout.addWidget(self.pushButton_3, 0, 1, 1, 1)
        self.label = QtWidgets.QLabel(self.centralwidget)
        self.label.setGeometry(QtCore.QRect(80, 40, 211, 71))
        self.label.setObjectName("label")
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 415, 21))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.pushButton.setText(_translate("MainWindow", "Start"))
        self.pushButton_3.setText(_translate("MainWindow", "Exit"))
        self.label.setText(_translate("MainWindow", " "))

        '''
        self.widget_img=QtWidgets.QWidget()
        self.widget_img.setGeometry(QtCore.QRect(0, 0, 400, 300))
```

```
        self.widget_img.setStyleSheet("background-image: url(test_image.jpg); back-
ground-attachment: fixed;border-style: outset")


        self.window_height = 600

        self.window_width = 600

        oImage =QtGui.QImage("test_image.jpg")

        sImage = oImage.scaled(QSize(self.window_height, self.window_width))

        palette = QtGui.QPalette()

        palette.setBrush(QtGui.QPalette.Window, QtGui.QBrush(sImage))

        self.centralwidget.setPalette(palette)

        '''


class Ui_MainWindow_2(object):

    def setupUi(self, MainWindow):

        MainWindow.setObjectName("MainWindow")

        MainWindow.resize(766, 319)

        self.centralwidget = QtWidgets.QWidget(MainWindow)

        self.centralwidget.setObjectName("centralwidget")

        self.pushButton = QtWidgets.QPushButton(self.centralwidget)

        self.pushButton.setGeometry(QtCore.QRect(50, 250, 75, 23))

        self.pushButton.setObjectName("pushButton")

        self.horizontalLayoutWidget = QtWidgets.QWidget(self.centralwidget)

        self.horizontalLayoutWidget.setGeometry(QtCore.QRect(40, 40, 691, 80))

        self.horizontalLayoutWidget.setObjectName("horizontalLayoutWidget")

        self.horizontalLayout = QtWidgets.QHBoxLayout(self.horizontalLayoutWidget)

        self.horizontalLayout.setContentsMargins(0, 0, 0, 0)

        self.horizontalLayout.setObjectName("horizontalLayout")

        self.verticalLayout_2 = QtWidgets.QVBoxLayout()

        self.verticalLayout_2.setObjectName("verticalLayout_2")

        self.label_5 = QtWidgets.QLabel(self.horizontalLayoutWidget)
```

```
self.label_5.setObjectName("label_5")
self.verticalLayout_2.addWidget(self.label_5)
self.comboBox = QtWidgets.QComboBox(self.horizontalLayoutWidget)
self.comboBox.setObjectName("comboBox")
self.comboBox.addItem("")
self.comboBox.addItem("")
self.comboBox.addItem("")
self.verticalLayout_2.addWidget(self.comboBox)
self.horizontalLayout.addLayout(self.verticalLayout_2)
self.verticalLayout_4 = QtWidgets.QVBoxLayout()
self.verticalLayout_4.setObjectName("verticalLayout_4")
self.label_3 = QtWidgets.QLabel(self.horizontalLayoutWidget)
self.label_3.setObjectName("label_3")
self.verticalLayout_4.addWidget(self.label_3)
self.comboBox_4 = QtWidgets.QComboBox(self.horizontalLayoutWidget)
self.comboBox_4.setObjectName("comboBox_4")
self.comboBox_4.addItem("")
self.comboBox_4.addItem("")
self.comboBox_4.addItem("")
self.comboBox_4.addItem("")
self.comboBox_4.addItem("")
self.comboBox_4.addItem("")
self.comboBox_4.addItem("")
self.comboBox_4.addItem("")
self.comboBox_4.addItem("")
self.comboBox_4.addItem("")
self.verticalLayout_4.addWidget(self.comboBox_4)
self.horizontalLayout.addLayout(self.verticalLayout_4)
self.verticalLayout_3 = QtWidgets.QVBoxLayout()
```

```
self.verticalLayout_3.setObjectName("verticalLayout_3")
self.label_2 = QtWidgets.QLabel(self.horizontalLayoutWidget)
self.label_2.setObjectName("label_2")
self.verticalLayout_3.addWidget(self.label_2)
self.comboBox_3 = QtWidgets.QComboBox(self.horizontalLayoutWidget)
self.comboBox_3.setObjectName("comboBox_3")
self.comboBox_3.addItem("")
self.comboBox_3.addItem("")
self.verticalLayout_3.addWidget(self.comboBox_3)
self.horizontalLayout.addLayout(self.verticalLayout_3)
self.verticalLayout = QtWidgets.QVBoxLayout()
self.verticalLayout.setObjectName("verticalLayout")
self.label = QtWidgets.QLabel(self.horizontalLayoutWidget)
self.label.setObjectName("label")
self.verticalLayout.addWidget(self.label)
self.comboBox_2 = QtWidgets.QComboBox(self.horizontalLayoutWidget)
self.comboBox_2.setObjectName("comboBox_2")
self.comboBox_2.addItem("")
self.comboBox_2.addItem("")
self.comboBox_2.addItem("")
self.verticalLayout.addWidget(self.comboBox_2)
self.horizontalLayout.addLayout(self.verticalLayout)



#self.horizontalLayout.addLayout(self.verticalLayout_8)
self.verticalLayoutWidget = QtWidgets.QWidget(self.centralwidget)
self.verticalLayoutWidget.setGeometry(QtCore.QRect(110, 160, 501, 81))
self.verticalLayoutWidget.setObjectName("verticalLayoutWidget")
self.verticalLayout_6 = QtWidgets.QVBoxLayout(self.verticalLayoutWidget)
```

```
self.verticalLayout_6.setContentsMargins(0, 0, 0, 0)
self.verticalLayout_6.setObjectName("verticalLayout_6")
self.checkBox = QtWidgets.QCheckBox(self.centralwidget)
self.checkBox.setObjectName("checkBox")
#self.verticalLayout_6.addWidget(self.checkBox)
self.horizontalLayout_2 = QtWidgets.QHBoxLayout()
self.horizontalLayout_2.setObjectName("horizontalLayout_2")
self.verticalLayout_5 = QtWidgets.QVBoxLayout()
self.verticalLayout_5.setObjectName("verticalLayout_5")
self.label_4 = QtWidgets.QLabel(self.verticalLayoutWidget)
self.label_4.setObjectName("label_4")
self.verticalLayout_5.addWidget(self.label_4)
self.comboBox_5 = QtWidgets.QComboBox(self.verticalLayoutWidget)
self.comboBox_5.setObjectName("comboBox_5")
self.comboBox_5.addItem("")
self.comboBox_5.addItem("")
self.comboBox_5.addItem("")
self.verticalLayout_5.addWidget(self.comboBox_5)
self.horizontalLayout.addLayout(self.verticalLayout_5)
self.verticalLayout_7 = QtWidgets.QVBoxLayout()
self.verticalLayout_7.setObjectName("verticalLayout_7")
self.label_6 = QtWidgets.QLabel(self.verticalLayoutWidget)
self.label_6.setObjectName("label_6")
self.verticalLayout_7.addWidget(self.label_6)
self.comboBox_6 = QtWidgets.QComboBox(self.verticalLayoutWidget)
self.comboBox_6.setObjectName("comboBox_6")
self.comboBox_6.addItem("")
self.comboBox_6.addItem("")
self.verticalLayout_7.addWidget(self.comboBox_6)
```

```
self.verticalLayout_10 = QtWidgets.QVBoxLayout()
self.verticalLayout_10.setObjectName("verticalLayout_10")
self.label_10 = QtWidgets.QLabel(self.verticalLayoutWidget)
self.label_10.setObjectName("label_10")
self.verticalLayout_10.addWidget(self.label_10)
self.comboBox_9 = QtWidgets.QComboBox(self.verticalLayoutWidget)
self.comboBox_9.setObjectName("comboBox_9")
self.comboBox_9.addItem("")
self.comboBox_9.addItem("")
self.comboBox_9.addItem("")

self.verticalLayout_10.addWidget(self.comboBox_9)
self.verticalLayout_11 = QtWidgets.QVBoxLayout()
self.verticalLayout_11.setObjectName("verticalLayout_11")
self.label_11 = QtWidgets.QLabel(self.verticalLayoutWidget)
self.label_11.setObjectName("label_11")
self.verticalLayout_11.addWidget(self.label_11)
self.comboBox_10 = QtWidgets.QComboBox(self.verticalLayoutWidget)
self.comboBox_10.setObjectName("comboBox_10")
self.comboBox_10.addItem("")
self.comboBox_10.addItem("")
self.verticalLayout_11.addWidget(self.comboBox_10)

self.verticalLayout_13 = QtWidgets.QVBoxLayout()
self.verticalLayout_13.setObjectName("verticalLayout_13")
self.label_13 = QtWidgets.QLabel(self.verticalLayoutWidget)
self.label_13.setObjectName("label_13")
self.verticalLayout_13.addWidget(self.label_13)
self.comboBox_12 = QtWidgets.QComboBox(self.verticalLayoutWidget)
```

```
self.comboBox_12.setObjectName("comboBox_12")
self.comboBox_12.addItem("")
self.comboBox_12.addItem("")
self.verticalLayout_13.addWidget(self.comboBox_12)
#Добавляем еще один комбобокс
self.verticalLayout_14= QtWidgets.QVBoxLayout()
self.label_14=QtWidgets.QLabel(self.horizontalLayoutWidget)
self.comboBox_14=QtWidgets.QComboBox(self.horizontalLayoutWidget)
self.verticalLayout_14.addWidget(self.label_14)
self.verticalLayout_14.addWidget(self.comboBox_14)

self.label_14.setText("Brightness estimation time")
self.comboBox_14.addItem("5 c")
self.comboBox_14.addItem("10 c")
self.comboBox_14.addItem("2 c")
self.comboBox_14.addItem("30 c")
self.comboBox_14.addItem("60 c")

#Цепляем layout
self.horizontalLayout_2.addLayout(self.verticalLayout_11)
self.horizontalLayout_2.addLayout(self.verticalLayout_13)
self.horizontalLayout_2.addLayout(self.verticalLayout_7)
self.horizontalLayout_2.addLayout(self.verticalLayout_10)
self.horizontalLayout_2.addLayout(self.verticalLayout_14)
self.verticalLayout_6.addLayout(self.horizontalLayout_2)

#Добавление 3 линейки
self.verticalLayoutWidget_2 = QtWidgets.QWidget(self.centralwidget)
self.verticalLayoutWidget_2.setGeometry(QtCore.QRect(30, 290, 501, 81))
```

```
self.verticalLayoutWidget_2.setObjectName("verticalLayoutWidget_2")
self.verticalLayout_18 = QtWidgets.QVBoxLayout(self.verticalLayoutWidget_2)
self.verticalLayout_18.setContentsMargins(0, 0, 0, 0)
self.verticalLayout_18.setObjectName("verticalLayout_18")
self.checkBox_3 = QtWidgets.QCheckBox(self.centralwidget)
self.checkBox_3.setObjectName("checkBox_3")

self.horizontalLayout_4 = QtWidgets.QHBoxLayout()
self.horizontalLayout_4.setObjectName("horizontalLayout_4")
self.verticalLayout_19 = QtWidgets.QVBoxLayout()
self.verticalLayout_19.setObjectName("verticalLayout_19")
self.label_17 = QtWidgets.QLabel(self.verticalLayoutWidget_2)
self.label_17.setObjectName("label_17")
self.verticalLayout_19.addWidget(self.label_17)
self.comboBox_16 = QtWidgets.QComboBox(self.verticalLayoutWidget_2)
self.comboBox_16.setObjectName("comboBox_16")
self.comboBox_16.addItem("")
self.comboBox_16.addItem("")
self.comboBox_16.addItem("")
self.verticalLayout_19.addWidget(self.comboBox_16)
self.horizontalLayout_4.addLayout(self.verticalLayout_19)
self.verticalLayout_20 = QtWidgets.QVBoxLayout()
self.verticalLayout_20.setObjectName("verticalLayout_20")
self.label_18 = QtWidgets.QLabel(self.verticalLayoutWidget_2)
self.label_18.setObjectName("label_18")
self.verticalLayout_20.addWidget(self.label_18)
self.comboBox_17 = QtWidgets.QComboBox(self.verticalLayoutWidget_2)
self.comboBox_17.setObjectName("comboBox_17")
self.comboBox_17.addItem("")
```

```
self.comboBox_17.addItem("")
self.comboBox_17.addItem("")
self.verticalLayout_20.addWidget(self.comboBox_17)
self.horizontalLayout_4.addLayout(self.verticalLayout_20)
self.verticalLayout_21 = QtWidgets.QVBoxLayout()
self.verticalLayout_21.setObjectName("verticalLayout_21")
self.label_19 = QtWidgets.QLabel(self.verticalLayoutWidget_2)
self.label_19.setObjectName("label_19")
self.verticalLayout_21.addWidget(self.label_19)
self.comboBox_18 = QtWidgets.QComboBox(self.verticalLayoutWidget_2)
self.comboBox_18.setObjectName("comboBox_18")
self.comboBox_18.addItem("")
self.comboBox_18.addItem("")
self.comboBox_18.addItem("")
self.verticalLayout_21.addWidget(self.comboBox_18)
self.horizontalLayout_4.addLayout(self.verticalLayout_21)
self.verticalLayout_18.addLayout(self.horizontalLayout_4)

#конец 3 линейки
MainWindow.setCentralWidget(self.centralwidget)



self.statusbar = QtWidgets.QStatusBar(MainWindow)
self.statusbar.setObjectName("statusbar")
MainWindow.setStatusBar(self.statusbar)

self.retranslateUi(MainWindow)
QtCore.QMetaObject.connectSlotsByName(MainWindow)
```

```
def retranslateUi(self, MainWindow):
    _translate = QtCore.QCoreApplication.translate
    MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
    self.pushButton.setText(_translate("MainWindow", "Start"))
    self.label_5.setText(_translate("MainWindow", "Video source"))
    self.comboBox.setItemText(0, _translate("MainWindow", "from file"))
    self.comboBox.setItemText(1, _translate("MainWindow", "from web-camera"))
    self.comboBox.setItemText(2, _translate("MainWindow", "from pi-camera"))
    self.label_3.setText(_translate("MainWindow", "Frame Resolution"))
    self.comboBox_4.setItemText(0, _translate("MainWindow", "1280*720 (HD 720)"))
    self.comboBox_4.setItemText(1, _translate("MainWindow", "1200*600 (WXVGA)"))
    self.comboBox_4.setItemText(2, _translate("MainWindow", "1024*768 (XGA)"))
    self.comboBox_4.setItemText(3, _translate("MainWindow", "960*540 (qHD)"))
    self.comboBox_4.setItemText(4, _translate("MainWindow", "800*600 (SVGA)"))
    self.comboBox_4.setItemText(5, _translate("MainWindow", "640*480 (VGA)"))
    self.comboBox_4.setItemText(6, _translate("MainWindow", "640*360 (nHD)"))
    self.comboBox_4.setItemText(7, _translate("MainWindow", "320*480 (HVGA)"))
    self.comboBox_4.setItemText(8, _translate("MainWindow", "640*240 (HVGS)"))
    self.comboBox_4.setItemText(9, _translate("MainWindow", "320*240 (QVGA)"))
    self.label_2.setText(_translate("MainWindow", "Video input method"))
    self.comboBox_3.setItemText(0, _translate("MainWindow", "Slow method"))
    self.comboBox_3.setItemText(1, _translate("MainWindow", "Fast method"))
    self.label.setText(_translate("MainWindow", "Color space"))
    self.comboBox_2.setItemText(0, _translate("MainWindow", "RGB"))
    self.comboBox_2.setItemText(1, _translate("MainWindow", "Gray"))
    self.comboBox_2.setItemText(2, _translate("MainWindow", "Binary"))
```

```
        self.checkBox.setText(_translate("MainWindow", "Set preprocessing and video re-
cording"))
        self.label_4.setText(_translate("MainWindow", "Frame medium brightness"))
        self.comboBox_5.setItemText(0, _translate("MainWindow", "no equalisation"))
        self.comboBox_5.setItemText(1, _translate("MainWindow", "30 %"))
        self.comboBox_5.setItemText(2, _translate("MainWindow", "40 %"))
        self.label_6.setText(_translate("MainWindow", "Brightness threeshold"))
        self.comboBox_6.setItemText(0, _translate("MainWindow", "50%"))
        self.comboBox_6.setItemText(1, _translate("MainWindow", "25%"))
        self.label_10.setText(_translate("MainWindow", "Pre-filtering"))
        self.comboBox_9.setItemText(0, _translate("MainWindow", "low pass filter"))
        self.comboBox_9.setItemText(1, _translate("MainWindow", "gaussian filter"))
        self.comboBox_9.setItemText(2, _translate("MainWindow", "median filter"))
        self.label_11.setText(_translate("MainWindow", "Record file format"))
        self.comboBox_10.setItemText(0, _translate("MainWindow", "avi"))
        self.comboBox_10.setItemText(1, _translate("MainWindow", "mp4"))
        self.label_13.setText(_translate("MainWindow", "Video codecs for recording"))
        self.comboBox_12.setItemText(0, _translate("MainWindow", "H264"))
        self.comboBox_12.setItemText(1, _translate("MainWindow", "XVID"))
        self.checkBox_3.setText(_translate("MainWindow", "Detecting object"))
        self.label_17.setText(_translate("MainWindow", "Detection target"))
        self.comboBox_16.setItemText(0, _translate("MainWindow", "faces"))
        self.comboBox_16.setItemText(1, _translate("MainWindow", "target"))
        self.comboBox_16.setItemText(2, _translate("MainWindow", "specials"))
        self.label_18.setText(_translate("MainWindow", "Anonymaze method"))
        self.comboBox_17.setItemText(0, _translate("MainWindow", "without"))
        self.comboBox_17.setItemText(1, _translate("MainWindow", "blur"))
        self.comboBox_17.setItemText(2, _translate("MainWindow", "pixelated"))
        self.label_19.setText(_translate("MainWindow", "Area selection"))
        self.comboBox_18.setItemText(0, _translate("MainWindow", "without"))
```

```
        self.comboBox_18.setItemText(1, _translate("MainWindow", "blur"))
        self.comboBox_18.setItemText(2, _translate("MainWindow", "pixelated"))
class Ui_MainWindow_3(QWidget):

    def setupUi(self,  parent=None):
    #def setupUi(self, MainWindow):



        super(Ui_MainWindow_3, self).__init__(parent)


        self.mediaPlayer = QMediaPlayer(None, QMediaPlayer.VideoSurface)
        btnSize = QSize(16, 16)
        self.centralwidget=QtWidgets.QWidget()
        self.videoWidget = QVideoWidget()


###Resize


        self.openButton = QPushButton("Open Video")
        self.setToolTip("Open Video File")
        self.openButton.setStatusTip("Open Video File")
        self.openButton.setFixedHeight(24)
        self.openButton.setIconSize(btnSize)
        self.openButton.setFont(QFont("Noto Sans", 8))
        self.openButton.setIcon(QIcon.fromTheme("document-
open", QIcon("D:/_Qt/img/open.png")))
        self.openButton.clicked.connect(self.abrir)


        self.playButton = QPushButton()
        self.playButton.setEnabled(False)
```

```
self.playButton.setFixedHeight(24)
self.playButton.setIconSize(btnSize)
self.playButton.setIcon(self.style().standardIcon(QStyle.SP_MediaPlay))
self.playButton.clicked.connect(self.play)

self.positionSlider = QSlider(Qt.Horizontal)
self.positionSlider.setRange(0, 0)
self.positionSlider.sliderMoved.connect(self.setPosition)

self.statusBar = QStatusBar()
self.statusBar.setFont(QFont("Noto Sans", 7))
self.statusBar.setFixedHeight(14)

self.controlLayout = QHBoxLayout()
self.controlLayout.setContentsMargins(0, 0, 0, 0)
self.controlLayout.addWidget(self.openButton)
self.controlLayout.addWidget(self.playButton)
#Add button start
self.buttonStart=QtWidgets.QPushButton()
self.buttonStart.setText("Start recording")
self.buttonStop=QtWidgets.QPushButton()
self.buttonStop.setText("Stop recording")
self.controlLayout.addWidget(self.buttonStart)
self.controlLayout.addWidget(self.buttonStop)
#end button start
#Add ScreenShot
self.buttonScreen=QtWidgets.QPushButton()
self.buttonScreen.setText("Screenshot")
self.controlLayout.addWidget(self.buttonScreen)
```

```
#Add FPS,FR,FMB

_translate = QtCore.QCoreApplication.translate

self.lineEdit_fps = QtWidgets.QLineEdit()

self.lineEdit_fr = QtWidgets.QLineEdit()

self.lineEdit_fmb = QtWidgets.QLineEdit()


self.checkBox_FPS = QtWidgets.QCheckBox("FPS")

self.checkBox_FMB= QtWidgets.QCheckBox("Pre-filtering")

self.checkBox_FR=QtWidgets.QCheckBox("FR")


self.controlLayout.addWidget(self.checkBox_FPS)

self.controlLayout.addWidget(self.lineEdit_fps)

self.controlLayout.addWidget(self.checkBox_FMB)


self.controlLayout.addWidget(self.lineEdit_fmb)

self.controlLayout.addWidget(self.checkBox_FR)

self.controlLayout.addWidget(self.lineEdit_fr)



#end


self.controlLayout.addWidget(self.positionSlider)

self.layout = QVBoxLayout()


self.layout.addWidget(self.videoWidget)


self.mediaPlayer.setVideoOutput(self.videoWidget)

self.mediaPlayer.stateChanged.connect(self.mediaStateChanged)

self.mediaPlayer.positionChanged.connect(self.positionChanged)
```

```
        self.mediaPlayer.durationChanged.connect(self.durationChanged)
        self.mediaPlayer.error.connect(self.handleError)

        self.statusBar.showMessage("Ready")

    def abrir(self):
        fileName, _ = QFileDialog.getOpenFileName(self, "Selecciona los mediose",
                "."， "Video Files (*.mp4 *.flv *.ts *.mts *.avi)")

        if fileName != '':
            self.mediaPlayer.setMedia( QMediaContent(QUrl.fromLocalFile(fileName)))
            self.playButton.setEnabled(True)
            #self.statusBar.showMessage(fileName)
            self.statusBar.showMessage("gena.avi")
            self.play()


    def play(self):
        if self.mediaPlayer.state() == QMediaPlayer.PlayingState:
            self.mediaPlayer.pause()

        else:
            self.mediaPlayer.play()

            rez=str(self.width())+"*"+str(self.height())
            self.lineEdit_fr.setText(rez)
            self.lineEdit_fmb.setText("Mediana")
            self.lineEdit_fps.setText(str(30))
```

```
def mediaStateChanged(self, state):
    if self.mediaPlayer.state() == QMediaPlayer.PlayingState:
        self.playButton.setIcon(
                self.style().standardIcon(QStyle.SP_MediaPause))
    else:
        self.playButton.setIcon(
                self.style().standardIcon(QStyle.SP_MediaPlay))

def positionChanged(self, position):
    self.positionSlider.setValue(position)

def durationChanged(self, duration):
    self.positionSlider.setRange(0, duration)

def setPosition(self, position):
    self.mediaPlayer.setPosition(position)

def handleError(self):
    self.playButton.setEnabled(False)
    self.statusBar.showMessage("Error: " + self.mediaPlayer.errorString())
```

Клас відповідаючи за дії в вікнах і евенти

```
import sys

from PyQt5 import QtCore, QtGui, QtWidgets

from window_all import *

from PyQt5.QtMultimediaWidgets import QVideoWidget

from PyQt5.QtMultimedia import QMediaContent, QMediaPlayer

from PyQt5.QtMultimediaWidgets import QVideoWidget

from PyQt5.QtGui import QIcon, QFont

from PyQt5.QtCore import QDir, Qt, QUrl, QSize

from PyQt5.QtMultimedia import QMediaContent, QMediaPlayer

from PyQt5.QtMultimediaWidgets import QVideoWidget

from queue import Queue, Empty

from PyQt5.QtWidgets import (QApplication,QGraphicsScene, QFileDialog, QHBoxLay-
out, QLabel, QPushButton, QSizePolicy, QSlider, QStyle, QVBoxLayout, QWidget, QStatus-
Bar)

import cv2

from PIL import ImageTk,Image

import wx

import os

import datetime

from imutils.video import FPS

import numpy as np

import argparse

import imutils

import cv2

import sys

from PyQt5.QtCore import *


from detecting.opencv_face_blurring.pyimagesearch.face_blurring import anony-
mize_face_pixelate
```

```
from detecting.opencv_face_blurring.pyimagesearch.face_blurring import anony-
mize_face_simple

resolu-
tion=[[1280,720],[1200,600],[1024,768],[960,540],[800,600],[640,480],[640,360],[320,480
],[640,240],[302,240]]


selfs=QtWidgets.QMainWindow

class mywindow(QtWidgets.QMainWindow):

    def __init__(self):


        super(mywindow, self).__init__()

        self.ui = Ui_MainWindow_1()









        self.ui.setupUi(self)


        self.ui.label.setStyleSheet("background-image: url(window_1_2.jpg)")

        self.ui.centralwidget.setStyleSheet("background-
color:rgb(255,255,255);color:blue;background-attachment: fixed;border-style: out-
set;font: bold italic")

        self.ui.pushButton.setStyleSheet("background-color:rgb(190,233,185)")

        self.ui.pushButton_3.setStyleSheet("background-color:rgb(190,233,185)")


        self.ui.label.setGeometry(0,0,890,274)

        self.resize(890,374)

        self.ui.pushButton.setFont(QFont("Times", 18, QtGui.QFont.Bold))

        self.ui.pushButton_3.setFont(QFont("Times", 18, QtGui.QFont.Bold))

        self.ui.gridLayoutWidget.adjustSize()

        self.ui.gridLayoutWidget.setGeometry(0,274,890,100)
```

```
        self.ui.pushButton.clicked.connect(self.StartClicked)
        self.ui.pushButton_3.clicked.connect(self.Exit)
        self.setWindowTitle("Start Menu")
    def Exit(self):
        self.close()

    def StartClicked(self):
        QtWidgets.QMainWindow.primer="s"
        self.mainWindow_2 = MainWindow_2()

        self.mainWindow_2.show()

        self.close()
class MainWindow_2(QtWidgets.QMainWindow):
    def __init__(self, parent=None):
        QtWidgets.QWidget.__init__(self, parent)
        self.ui = Ui_MainWindow_2()



        self.ui.setupUi(self)

        print(QtWidgets.QMainWindow.primer)

        self.ui.centralwidget.adjustSize()
        self.ui.centralwidget.setStyleSheet("background-
color:rgb(255,255,255);color:red;font:bold 14px")
        self.ui.checkBox.setGeometry(20,100,800,20)
        self.ui.verticalLayoutWidget.hide()
        self.ui.horizontalLayoutWidget.setGeometry(0, 20, 800, 80)
        self.ui.verticalLayoutWidget.setGeometry(0,130,800,80)
```

```
        self.ui.comboBox.setStyleSheet("color:black")
        self.ui.comboBox_2.setStyleSheet("color:black")
        self.ui.comboBox_3.setStyleSheet("color:black")
        self.ui.comboBox_4.setStyleSheet("color:black")
        self.ui.comboBox_5.setStyleSheet("color:black")
        self.ui.comboBox_6.setStyleSheet("color:black")

        self.ui.comboBox_12.setStyleSheet("color:black")
        self.ui.comboBox_9.setStyleSheet("color:black")
        self.ui.comboBox_10.setStyleSheet("color:black")



        self.ui.pushButton.setGeometry(1100,20,100,80)




        self.ui.comboBox_14.setStyleSheet("color:black")

        self.resize(1200,200)
        self.move(100,0)
        self.setWindowTitle("Settings")
        self.ui.horizontalLayoutWidget.setStyleSheet("background-
color: rgb(103,223,180)")
        StyleButton='''
    background-color: rgb(103,223,180);
    border-style: outset;
    border-width: 2px;
    border-radius: 10px;
    border-color: beige;
    font: bold 14px;
```

```
    color:blue;
    padding: 6px;
    '''
        self.ui.pushButton.setStyleSheet(StyleButton)
        self.ui.pushButton.clicked.connect(self.StartingClicked)
        self.ui.checkBox.clicked.connect(self.CheckBox_click)



        StyleLayoutWidget='''

    border-style: outset;
    border-width: 2px;
    border-radius: 10px;
    border-color: beige;

    min-width: 10em;
    padding: 6px;
'''
        #Оформление 3
        self.ui.comboBox_16.setStyleSheet("color:black")
        self.ui.comboBox_17.setStyleSheet("color:black")
        self.ui.comboBox_18.setStyleSheet("color:black")
        self.ui.checkBox_3.setGeometry(20,120,800,20)
        self.ui.verticalLayoutWidget_2.hide()
        self.ui.verticalLayoutWidget_2.setGeometry(0,220,800,80)
        self.ui.checkBox_3.clicked.connect(self.CheckBox_detect_click)
        self.ui.verticalLayoutWidget_2.setStyleSheet(StyleLayoutWidget)
        self.ui.verticalLayoutWidget_2.adjustSize()
```

```
        #end
        self.ui.horizontalLayoutWidget.setStyleSheet(StyleLayoutWidget)
        self.ui.verticalLayoutWidget.setStyleSheet(StyleLayoutWidget)

        self.ui.verticalLayoutWidget.adjustSize()
        self.ui.horizontalLayoutWidget.adjustSize()



        self.ui.pushButton_2 = QtWidgets.QPushButton(self.ui.centralwidget)
        self.ui.pushButton_2.setGeometry(QtCore.QRect(1100,130,100,80))
        self.ui.pushButton_2.setObjectName("pushButton")
        self.ui.pushButton_2.setText("Factory \n Settings")
        self.ui.pushButton_2.setStyleSheet(StyleButton)
        self.ui.pushButton_2.hide()
        # up - это поправка на размер окна
        self.up=0


def StartingClicked(self):

        selfs.resolution=self.ui.comboBox_4.currentIndex()
        selfs.source=self.ui.comboBox.currentIndex()
        selfs.target=self.ui.comboBox_16.currentIndex()
        selfs.anonymaze=self.ui.comboBox_17.currentIndex()
        selfs.area_selection=self.ui.comboBox_18.currentIndex()

        self.mainWindow_3 = MainWindow_3()
       # self.mainWindow_3.resize(888,888)
```

```
        self.mainWindow_3.show()
    def CheckBox_click(self):
        if((self.ui.checkBox.isChecked())==True):
            self.up=self.up+80
            self.ui.verticalLayoutWidget.show()
            self.ui.pushButton_2.show()
        if((self.ui.checkBox.isChecked())==False):
            self.up=self.up-80
            self.ui.verticalLayoutWidget.hide()
            self.ui.pushButton_2.hide()
        self.resize(self.width(),200+self.up)
        self.ui.checkBox_3.setGeometry(20,120+self.up,800,20)



    def CheckBox_detect_click(self):
        if((self.ui.checkBox_3.isChecked())==True):
            self.up+=80
            self.ui.verticalLayoutWidget_2.show()

        if((self.ui.checkBox_3.isChecked())==False):
            self.up-=80
            self.ui.verticalLayoutWidget_2.hide()
        self.resize(self.width(),200+self.up)
        self.ui.verticalLayoutWidget_2.setGeometry(0,60+self.up,800,80)
class SenderMessage(QtCore.QObject):
    image_bytes = QtCore.pyqtSignal(np.ndarray)

    def __init__(self):
        super().__init__()
```

```
    def detecting(self,frame):
        (h, w) = frame.shape[:2]
        blob = cv2.dnn.blobFro-
mImage(cv2.resize(frame, (300, 300)), 1.0,(300, 300), (104.0, 177.0, 123.0))
        net.setInput(blob)
        detections = net.forward()
        for i in range(0, detections.shape[2]):
            confidence = detections[0, 0, i, 2]
            if confidence < 0.5:
                continue
            box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")

            face = frame[startY:endY, startX:endX]
            if selfs.anonymaze==1:
                face = anonymize_face_simple(face, factor=3.0)
            if selfs.anonymaze==2:
                face = anonymize_face_pixelate(face,blocks=20)
            frame[startY:endY, startX:endX] = face

            #Добавление текста
            if selfs.anonymaze==0:
                text = "{:.2f}%".format(confidence * 100)
                y = startY - 10 if startY - 10 > 10 else startY + 10
                cv2.rectangle(frame, (startX, startY), (endX, endY),(0, 0, 255), 2)
                cv2.putText(frame, text, (startX, y),cv2.FONT_HERSHEY_SIM-
PLEX, 0.45, (0, 0, 255), 2)
```

```
    @pyqtSlot()
    def run(self):
        print("1")
        stream = cv2.VideoCapture(0)
        selfs.fps=1
        stream.set(cv2.CAP_PROP_FRAME_WIDTH,resolution[selfs.resolution][0])
        stream.set(cv2.CAP_PROP_FRAME_HEIGHT,resolution[selfs.resolution][1])
        print(resolution[selfs.resolution][1])
        fps = FPS().start()
        print(selfs.resolution)
        while selfs.active:
            try:


                (grabbed, frame) = stream.read()
                if not grabbed:
                    break
                self.detecting(frame)
                frame = imutils.resize(frame,resolution[selfs.resolution][0],resolu-
tion[selfs.resolution][1])
        #frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        #frame = np.dstack([frame, frame, frame])
                image_bytes = frame


                self.image_bytes.emit(image_bytes)
                fps.update()
                if(fps._numFrames%10==0):
                    fps.stop()
                    selfs.fps=fps.fps()
            except Empty:
                self.thread.quit()
```

```
                self.thread.wait()
                stream.release()
                fps.stop()
                print('stop')
                continue
            except :
                print("sss",sys.exc_info()[0])



class MainWindow_3(QtWidgets.QMainWindow):
    def __init__(self, parent=None):
        QtWidgets.QWidget.__init__(self, parent)
        self.ui = Ui_MainWindow_3()
        self.setWindowTitle("Video Stream")
        self.resize(resolution[selfs.resolution][0],resolution[selfs.resolution][1])
        self.flag_write="stop"
        self.ui.setupUi(self)
        self.ui.setStyleSheet("color:blue;background-
color:rgb(190,233,185);font:bold 14px")




        self.ui.checkBox_FPS.clicked.connect(self.CheckBox__click)
        self.ui.checkBox_FR.clicked.connect(self.CheckBox__click)
        self.ui.checkBox_FMB.clicked.connect(self.CheckBox__click)
        #self.ui.setGeometry(0,0,resolution[selfs.resolution][0],resolution[selfs.reso-
lution][1])



        if(selfs.source==0):
```

```
            self.ui.openButton.show()
            self.ui.positionSlider.show()
            self.ui.playButton.show()
            self.ui.buttonStart.hide()
            self.ui.buttonStop.hide()

        else :


            self.ui.openButton.hide()
            self.ui.positionSlider.hide()
            self.ui.playButton.hide()
            self.ui.buttonStop.hide()
            self.ui.buttonStart.show()

            self.ui.buttonStart.clicked.connect(self.StartWrite)
            self.ui.buttonStop.clicked.connect(self.StopWrite)
            self.ui.videoWidget.hide()

            self.createGraphicView()
        self.ui.layout.addLayout(self.ui.controlLayout)
        self.ui.layout.addWidget(self.ui.statusBar)
        self.ui.setLayout(self.ui.layout)

        self.ui.setGeometry(0,0,resolution[selfs.resolution][0],resolution[selfs.reso-
lution][1])

    def createGraphicView(self):
        #self.thread.destroyed()
        self.ui.scene  =QGraphicsScene(self.ui)
```

```
        self.ui.graphicView =QtWidgets.QGraphicsView(self.ui.scene, self.ui)

        #self.ui.graphicView.setGeometry(0,0,resolution[selfs.resolution][0],resolu-
tion[selfs.resolution][1]-100)


        self.ui.graphicView.setGeometry(0,0,640,380)

        self.ui.layout.addWidget(self.ui.graphicView,0)




        self.thread = QtCore.QThread()



        self.sender_message = SenderMessage()

        self.sender_message.moveToThread(self.thread)


        self.sender_message.image_bytes.connect(self.signalHandlerImageBytes)




        self.thread.started.connect(self.sender_message.run)

        self.sender_message.active=True



        self.thread.start()

        selfs.active=True


        print("start")

        #self.ui.setLayout(self.ui.layout)

    def closeEvent(self,event):
```

```
        self.close()
        selfs.active=False

        self.thread.active=False



        #self.thread.wait()
        self.thread.terminate()



        print('close')

        self.close()

    def signalHandlerImageBytes(self, frame):

        #Преобразование в Pixmap
        #self.detecting(frame)

        height, width, channel = frame.shape
        bytesPerLine = 3 * width
        qImg = QtGui.QImage(frame.data, width, height, bytesPerLine, QtGui.QImage.For-
mat_RGB888).rgbSwapped()
        self.ui.scene.addPixmap(QtGui.QPixmap(qImg))
        self.ui.lineEdit_fps.setText(str(int(selfs.fps)))
```

```
    def CheckBox__click(self):
        if(self.ui.checkBox_FMB.isChecked()==True):
            self.ui.lineEdit_fmb.show()
        if(self.ui.checkBox_FMB.isChecked()==False):
            self.ui.lineEdit_fmb.hide()
        if(self.ui.checkBox_FR.isChecked()==True):
            self.ui.lineEdit_fr.show()
        if(self.ui.checkBox_FR.isChecked()==False):
            self.ui.lineEdit_fr.hide()
        if(self.ui.checkBox_FPS.isChecked()==True):
            self.ui.lineEdit_fps.show()
        if(self.ui.checkBox_FPS.isChecked()==False):
            self.ui.lineEdit_fps.hide()
    def StartWrite(self):
        self.files= cv2.VideoWriter_fourcc(*'XVID')
        filename=str(datetime.datetime.now())
        filename=filename.replace(':','-')
        filename=(filename[:filename.find('.')])+".avi"
        self.oldname=filename
        self.out = cv2.VideoWriter(filename,self.files, 20.0, (640,480))
        self.flag_write="start"
        self.ui.buttonStart.hide()
        self.ui.buttonStop.show()
    def StopWrite(self):


        self.out.release()
        new_filename=QFileDialog.getSaveFileName(self, "Selecciona los medi-
ose",".", "Video Files (*.mp4 *.flv *.ts *.mts *.avi)")
```

```
        os.replace(self.oldname,new_filename)

        self.ui.buttonStop.hide()

        self.ui.buttonStart.show()

app = QtWidgets.QApplication([])

net = cv2.dnn.readNetFromCaffe("all/detecting/deep-learning-face-detection/deploy.pro-
totxt.txt", "all/detecting/deep-learning-face-detec-
tion/res10_300x300_ssd_iter_140000.caffemodel")

#prototxtPath = os.path.sep.join([args["face"], "deploy.prototxt"])

#weightsPath = os.path.sep.join([args["face"],"res10_300x300_ssd_iter_140000.caf-
femodel"])

#net = cv2.dnn.readNet(prototxtPath, weightsPath)

application = mywindow()

application.show()


sys.exit(app.exec())
```