РЕФЕРАТ

Дипломна робота: 72 ст. , 25 рис., 6 табл., 10 джерел та 2 додатки.

Метою даної роботи є реалізація методів по локалізації людської руки у відеопотоці. У роботі реалізовані такі методи як: віднімання фону, колірний фільтр на основі байесовскього класифікатора та обробка відеопотоку з камери глибини.

Результати роботи:

- реалізовані три підходи по локалізації людської руки на відео;
- проведений аналіз умов для надійної роботи алгоритмів;
- запропоновано і реалізовано ефективний метод навчання байесовскього класифікатора;
- проведено порівняння алгоритмів.

Результати даної роботи рекомендовано використовувати у системах взаємодії людини та комп'ютера. При подальших дослідженнях у цій області доцільно реалізувати адаптивність байесовського класифікатора під різні умови освітолення та створити систему по взаємодії з комп'ютером на основі реалізованого методу детекції руки на відео.

РОЗПІЗНАВАННЯ ОБ'ЄКТІВ, ВЗАЄМОДІЯ КОМП'ЮТЕРА ТА ЛЮДИНИ, КАМЕРА ГЛИБИНИ, БАЙЕСОВСЬКИЙ КЛАСИФІКАТОР, КОЛІРНИЙ ПРОСТІР.

ABSTRACT

The thesis: 72 p. , 25 fig., 6 tabl., 10 sources and 2 appendices.

The theme of this thesis is “Human hand detection on video”.

The purpose of this thesis is to implement and analyze 3 approaches of human hand localization in video stream. In the thesis were realized the following methods: background subtraction, color filter based on Bayes classifier and processing the video stream from depth camera.

Thesis results:

- 3 methods of human hand localization in video stream were implemented;
- for each method were analyzed the optimal working conditions;
- proposed a convenient way for Bayes classifier training;
- made a comparison of methods.

Current thesis results are proposed for using in computer-human interaction systems. In further researches it is reasonable to implement adaptive approaches for Bayes classifier to changing illumination conditions and create a computer-human interaction system based on realized approaches and gesture recognition with support vector machine or neural networks.

OBJECT RECOGNITION, COMPUTER-HUMAN INTERACTION, DEPTH CAMERA, BAYES CLASSIFIER, COLOR SPACE.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БС — Bayes classifier
ПК — портативний комп’ютер
BLOB — Binary Large Object
ПЗ - програмне забезпечення

ВСТУП

В наші часи складно уявити своє життя без комп'ютера. Він стоїть удома, на роботі, в школі чи університеті. Основний способ комунікації з цим складним девайсом є усім відомі пристрої вводу такі як: клавіатура, мишка чи в більш продвинутому випадку графічний планшет, який є просто незамінний для малювання на ПК.

Права рука людини, що працює за комп'ютером, більше 90% часу знаходиться на мишці, оскільки основна частина користувачів ПК використовує операційні системи з графічними оболонками де більша частина простих керуючих операцій здійснюється за допомоги курсора, який контролюється саме мишою. Варто подумати про те, що робить у цей час ліва рука користувача. Вона дуже часто бездіє чи нажимає прості гарячі комбінації з двухтрьох кнопок на клавіатурі, хоча це малоймовірний випадок оскільки мізерний відсоток людей знає більше 5 гарячих комбінацій. Тобто перша очевидна проблема взаємодії з ПК – мала ефективність основних способів комунікації.

Взагалі перехід операційних систем на графічну оболонку можна вважати дуже великим кроком уперед. Більшість користувачів ПК сприйняли цей крок позитивно адже для запуску програми треба лише навести на її іконку курсор та натиснути один-два рази на ліву кнопку миші. Перша комп'ютерна миша, що була відносно доступною для простих людей, мала лише одну функціональну кнопку та коштувала приблизно 25$. Вона була випущена разом із операційною системою Apple Macintosh в якій якраз з'явилась підтримка віконного інтерфейсу. Це значно пришвидшило роботу з файлами та інші рутинні операції оскільки пару кліків мишкою заміняли доволі таки складні команди в терміналі. Проте розвиток технологій та потреб користувачів призводить до того, що миша вже не може повністю покривати множину найчастіших команд користувача лише трьома кнопками та сенсором руху. Заміна середньої кнопки на колесо якраз є прикладом додавання нової функціональності миші для покриття більшої кількості команд. Саме розвиток інтернету та браузерів призвів до того, що дуже часто потрібно листати достатньо довгі сторінки і колесо прокрутки для цього підходить набагато краще. Також дуже часто до мишки додають допоміжні функціональні

кнопки, які можна запрограмувати на якусь дію чи навіть послідовність дій. Це також вимушений крок розробників мишок, проте площина поверхні мишки обмежена і місце для кнопок з часом закінчиться.

В останні пару років комп'ютерна миша еволюціонувала на багатьох пристроях у сенсорну панель. Завдяки тому, що сенсорна панель може розпізнавати до десяти пальців, з'являється можливість обробки комплексних жестів для масштабування зображення, прокрутки текста або інтернет сторінки чи навіть створення власних жестів та програмування виконання певних дій при фіксуванні цього жеста. На данний момент це один із самих ефективних мишкоподібних засобів вводу оскільки покриває дуже велику множину команд з можливістю її розширення.

Проте останній рік був дуже насиченим в плані дослідження проблеми взаємодії людини та ПК. Не так давно була випущенна камера Microsoft Kinect для приставки Xbox One. Завдяки поєднанню потоків з двох серсорів RGB та depth з'явилась реальна можливість відслудковування об'єктів у просторі. Звісно це було можливо і раніше за допомоги декількох RGB сенсорів чи навіть спеціальних рухових сенсорів, проте великим недоліком цих методів можна вважати достатньо масштабні обчислення або велику кількістю проводів. Поєднавши у камері кольоровий сенсор та сенсор відстані виходить людина не повинна взагалі нічого на себе чіпляти, не потрібно одягати спеціальний одяг з різноманітними сенсорами та передавачами. Комп'ютер може бачити людину у просторі самостійно. Також минулого року компанія Intel анонсувала технологію Realsense та лінійку камер F200, R200, SR300. Відрізняются вони лише призначенням, але головна ідея в тому, щоб дати можливість ПК та мобільним пристроям бачити у просторі. Одною із основновних частин Realsense SDK є модуль по відслідковуванню руки та аналізу статичних і динамічних жестів.

Для того щоб зрозуміти важливість цієї проблеми потрібно порівняти функціональність комп'ютерної миші та людської руки. Миша має в основному сенсор положення у 2d просторі, 3 кнопки та колесо прокрутки. Людська рука може знаходитись у 3d просторі та приймати достатньо складні форми (статичні жести) чи робити деякий комплекс рухів у 3d просторі (динамічні жести). Очевидно, що множина станів людської руки більш різнома-

нітна ніж множина станів миші. Саме тому багато дослідників нових способів взаємодії людини та комп'ютера прийшли до того, що можна керувати ПК без тримання в руках яких-небудь пристроїв лише за допомоги жестів. На данний момент спілкування людини та комп'ютера дуже схоже на спілкування людини і сліпої та глухої людини, що виступає в ролі комп'ютера. Дійсно, ПК людину не бачить та не чує взагалі.

Система по локалізації людських рук на відео має бути достатньо простою в плані обчислень так як вона повинна працювати в режимі реального часу та обробляти від 30 до 60 фреймів на секунду. Це зразу відсіює достатньо велику частину підходів які використовуються для локалізації людських рук на фото і обробляють одне зображення довше ніж за одну секунду.

Таким чином, метою цієї роботи є досліждення методів локалізації людської руки на відео.

Для досягнення цих цілей вирішені наступні задачі:

- Проведений аналіз існуючих підходів по локалізації руки на відео та обробки цифрових зображень.
- Реалізовані три методи та проведене дослідження їх роботи.
- Реалізований зручний спосіб навчання Байесовскього класифікатора

Об'єктом дослідження є системи по локалізації людської руки у відеопотоці.

Предметом дослідження є методи та алгоритми формування системи детекції людської руки за допомоги веб камери чи камери глибини.

Наукова новизна отриманих результатів полягає у тому, що запропоновані методи обробки матриці ймовірностей Байесовського класифікатора, які покращують значення обох критеріїв.

Практичними результатами роботи є реалізація трьох алгоритмів, що працюють з точністю не меншою за 80%. При дотриманні оптимальних умов точність досягає в середньому 96% за першим критерієм.

Робота складається з чотирьох розділів. У першому розділі розглядається постановка задачі дослідження та актуальність проблеми. Другий розділ присвячений критеріям якості рішення задачі та опису алгоритмів локалізації людської руки на відео. У третьому розділі здійснено огляд технологій та алгоритмів, що використовуються в роботі, проведений порівняльний аналіз та

наведено схеми програм. У четвертому розділі розглядається фунціонально-вартісний аналіз програмного продукту.

РОЗДІЛ 1 АНАЛІЗ ЗАДАЧІ РОЗПІЗНАВАННЯ ЛЮДСЬКОЇ РУКИ НА ВІДЕО

1.1 Актуальність поставленої задачі

Розпізнавання людської руки на відео - задача актуальна саме зараз оскільки це основа для більш складного процесу розрізнавання статичних та динамічних жестів.

Розпізнавання жестів насамперед це лише ідея по роботі з жестами і базується на абсолютно різних технологіях: переносимі пристрої (рукавиці чи інші контролери), зафіксовані камери (RGB камери, depth камери чи комбінація RGB камер для отримання стереоскопічного зображення) чи навіть радари (працють по принципу сканування магнітного фону та його зміни у часі).

Застосування можуть мати найрізноманітніші форми:

- Керування ПК - у 2016 році вийшли перші ноутбуки з вбудованою камерою Intel Realsense F200 (замість звичної RGB камери). На данний момент на її базі створюються програмні продукти по керуванню ПК без миші.
- Навчання мові німих - аналізуючи жести можна повністю оцифрувати мові німих та створити програму-вчителя, що буде показувати жести і перевіряти наскільки правильно користувач їх повторює
- Керування віртуальними середовищами - в кінці 2015 року компанія Microsoft презентувала прототип Hololens, що створений саме для реалізації доповненої реальності та за допомоги камер Kinect аналізувати жести користувача для більш звичної взаємодії з віртуальним середовищем оскільки майже 90% фізичної взаємодії з середовищем людина виконує за допомоги рук, а тому слід вважати, що це основний спосіб взаємодії.

Причини складності задачі:

- Нестабільні умови освітленності - освітленність дуже сильно впливає на алгоритми, що базуються на обробці RGB відеопотоку;

- Поява сторонніх об’єктів у полі зору камери - дуже сильно впливає оскільки збільшує ймовірність розпізнати цей об’єкт як той, що потребує подальшого аналізу;
- Недостатня якість камер призводить до того, що в зображенні присутні шуми;
- Прості RGB камери мають неякісні матриці і через це реєструють кольори спотворено ускладнює роботу з колірними ознаками шкіри людини;
- Колір шкіри людей різниться і тому навчання класифікаторів потрібно проводити на достатньо великих вибірках.

1.2 Аналіз існуючих підходів до вирішення задачі

1.2.1 Виділення об’єктів за допомоги віднімання фону

Віднімання фону - процедура, що за умов нерухомості камери, оцінює зміну кожного пікселя зображення відносно фону, зображення якого було отримане раніше, та отримати в результаті бінарне зображення відмінності поточного зображення від зображення фону.

Алгоритм дійсний як для кольорових зображень, так і для зображень у відтінках сірого.

Переваги цього алгоритму у тому, що його достатньо просто реалізувати та він не потребує складних обчислень (більшість операцій можна векторизувати та паралелізувати що значно прискорює швидкість обробки). Проте його найбільшим недоліком можна вважати те, що за його допомоги неможливо виділити об’єкт на динамічному фоні, він зпродукує велику кількість областей для подальшого аналізу, який може бути вже не таким тривіальним і система на його базі не зможе працювати в режимі on-line на достатньому рівні.

Незважаючи на його недоліки у 2006 році була представлена система по розпізнаванню японської мови жестів [1].

1.2.2 Відслідковування руки за допомоги контролерів

У минулих століттях саме цей підхід для роботи з відслідковуванням руки та аналізу жестів був популярним через достатньо прозору та просту програмну реалізацію. За допомоги приблизно 16 сенсорів та вбудованого в рукавицю гіроскопа можна достатньо точно відслідковувати навіть мікрорухи руки.

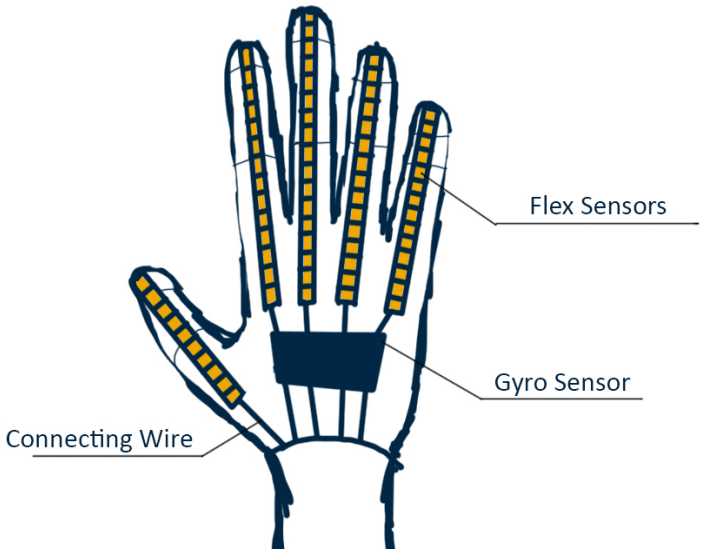


Рисунок 1.1 – Схема рукавиці з сенсорами

З точки зору розробників ПЗ простота була у тому, що на вхід вони вже мали позиції усіх ключових точок руки відносно центру (найчастіше це була лодоня), та положення центру у просторі. Для аналізу статичних жестів достатньо мати лише відносне положення усіх ключових точок відносно центру. Для аналізу динамічних жестів вже потрібні дані зміни положення руки у просторі.

Також були прототипи контролерів, що містили лише гіроскоп, проте відслідковування позиції у просторі руки як одного об’єкта без данних про позиції ключових точок на пальцях виявився нікому непотрібним і контролели швидко вийшли з виробництва.

Переваги цього підходу до відслідковування руки у просторі в тому, що він дає дуже точні результати та з мінімальною затримкою, проте він має

дуже серйозний недоліки - потреба в носінні на собі додаткових незручних приладів та складність підтримки і ремонту таких девайсів.

1.2.3 Колірні фільтри для детекції шкіри на зображенні

Оскільки будь-яке цифрове зображення можливо представити у вигляді:

$$M = matrix\, W \times H \times D\ ,\ d = dim D \tag{1.1}$$

де M - цифрове зображення, W - ширина зображення, D - розмір колірного простору. Наприклад для типових колірних просторів RGB, HSV, YCrCb $d = 3$. Для колірного простору градацій сірого $d = 1$.

Колірний простір можна представити так:

$$(x = (x_1, ..., x_d) \in D) \Leftrightarrow (0 <= x_1 <= d_1, ..., 0 <= x_d <= d_d) \tag{1.2}$$

де D - колірний простір, $d_1, ..., d_d$ - обмеження координат колірного простору.

Найчастіше зустрічається випадок коли $x_i \in \mathbb{N} \cup \{0\}, d_i = 2^8, i = 1..d$ оскільки це списується в принцип збереження данних у пам'яті ПК.

1.2.3.1 Прості колірні фільтри

Основний принцип роботи простих колірних фільтрів полягає у тому, що емпірічними методами виділяється гіперкуб, що повністю містить у собі всі можливі значення пікселей, що характеризували певний об'єкт.

Формальний вигляд простого фільтра:

$$H = x \in D \mid a_i <= x_i <= b_i, i = 1..d \tag{1.3}$$

де a_i, b_i - грані гіперкуба.

Прості колірні фільтри хороші тим, що дуже прості в реалізації, швидко працюють та є можливість паралелізації обчислень оскільки фільтр обробряє кожен піксель незалежно.

Результатом роботи колірного фільтра H зображення M є бінарна матриця Q :

$$Q = [q_{ij}],\ q_{ij} = (M[i,j] \in H), i = 1..dimW, j = 1..dimH \tag{1.4}$$

Тобто матриця $q_{ij} \in \{0,1\} \forall i,j$ і тому можна достатньо просто записати композицію простих фільттрів як поелементні бінарні операції над матрицями. Звісно за допомоги кубів можливо апроксимувати достатньо точно найскладніші фігури, проте це має свої слабкі сторони:

а) апроксимація певного тіла кубами в просторі розмірності d достатньо нетривіальна задача з точки зору математики

б) зі збільшенням точності буде рости кількість простих колірних фільтрів у ланцюгу композицій і це призведе до падіння швидкості роботи

У роботі [2] було проведене дослідження колірних характеристик шкіри людини та єкспериментальним шляхом отримані певні фільтри, за допомоги яких можна провести первинну обробку зображення щоб вилучити із зображення деякі області, що майже напевно не містять кольори близькі до колькору шкіри людини. Також у цій роботі запропоновані фільтри для найбільш поширених колірних просторів - RGB, HSV та YCbCr.

1.2.3.2 Гаусівська модель

Одним з більш складних підходів є припущення, що ймовірність належності кольору пікселя до множини кольорів певного об'єкта є деяка випадкова величина, що розподілена за гаусівським законом розподілу [3].

Також цей підхід враховує змінні умови освітлення шляхом поєднання двух гаусівських моделей. Перша гаусівська модель будується у спеціальних умовах звичайного освітлення, друга - при черезмірному освітленні.

Алгоритм навчання такої моделі - метод вибору математичного сподівання та кореляційної матриці.

Класифікація проводиться згідно з вибраною довірчою ймовірністю. У конкретному випадку колірного простору розмінності 2 це буде еліпс.

Основним недоліком цього підходу є те, що на практиці достатньо важко правильно підібрати математичне сподівання та кореляційну матрицю і тому точність методу недостатня.

1.2.3.3 Байесовська модель

Байесовський класифікатор дуже поширений у задачі локалізації людської руки на відео. Його використовують для пошуку об’єктів з певними колірними характеристиками.

За основу береться теорема Байеса про апостеріорні ймовірності. Байесовська модель дозволяє обрахувати ймовірності того, що піксель з певним кольором належить шуканому об’єкту.

Навчання такої моделі можна розбити на 2 етапи:

а) Обробка зображення для навчання - на зображеннях призначених для навчання замальовуються усі сторонні елементи окрім шкіри і таким чином формується маска зображення.

б) Навчання - підраховується кількість кожного пікселя що потрапляє у область шкіри на зображенні та в кінці навчання ділиться на загальну кількість пікселів, що належали шкірі.

Проте для хорошої точності класифікатора потрібна чимала вибірка. Виділення руки на зображенні за допомоги графічних редакторів є нескладною задачею, проте дуже повільною. Тому окрім реалізації Байесовського класифікатора слід ще подумати над можливістю обробки ймовірностей після навчання та розробки більш зручного та швидкого методу навчання класифікатора.

1.2.4 Ознаки Хаара та метод Віоли-Джонса

Метод Віоли-Джонса дуже популярний та використовується в багатьох сферах. Добре себе показує в задачі розпізнавання облич та навіть пішоходів на вулиці.

Цей алгоритм можливо використати і для задачі локалізації людської руки на відео [4] [5], проте в цьому випадку потрібно враховувати деякі особливості.

Використовуючи ознаки Хаара потрібно розуміти, що конкрентний класифікатор буде розпізнавати лише певний, а головне один, жест, причому допускаються повороти відносно центра кратні 45 градусів, що не дуже підходить для роботи з відео на якому рука може приймати різноманітні форми та повертатися.

1.3 Формалізація постановки задачі дослідження

1.3.1 Формальний опис задачі

Вхідними данними є відеопотік з кольорової камери чи камери глибини. Потрібно реалізувати алгоритми обробки кожного фрейма відеопотоку та алгоритми локалізації людської руки.

Основними алгоритмами локалізації руки на відео вибрані такі:

а) Віднімання фону;
б) Байесовський класифікатор;
в) Обробка відеопотоку камери глибини.

Також потрібно ввести критерії для оцінки роботи алгоритмів та можливості подальшого їх порівняння.

Останнім етапом роботи є порівняння алгоритмів з урахуванням таких факторів: складність обчислень та швидкість обробки, оптимальні умови роботи і вартість обладнання.

Основні цілі:

а) Реалізувати класичний та найпоширеніший метод, що використовується для вирішення поставленої задачі - віднімання фону;
б) Реалізувати та удосконалити байесовський класифікатор для класифікації кольору людської шкіри на відео і локалізації руки. Удосконалити метод навчання класифікатора;
в) Представити метод обробки відеопотоку камери глибини як найбільш сучасний підхід до вирішення поставленої задачі.

1.4 Висновки за розділом

У даному розділі був проведений аналіз актуальності поставленої задачі, проведений обширний аналіз існуючих підходів до її розв’язання та визначені основні цілі роботи.

Задача актуальна саме в останні 5 років через стрімкий розвиток ПК та можливість застосовувати складні алгоритми у системах реального часу. Проблема взаємодії людини та комп’ютера на данний момент захопила майже усіх компанійгігантів по розробці ПЗ. Для роботи з жестами руки розроблені новітні камери, що потребують масштабних обчислень і тому в роботі також розглядаються алгоритми, що можуть використовуватись і на бюджетних ПК.

РОЗДІЛ 2 ТЕОРЕТИЧНА ЧАСТИНА

2.1 Загальний підхід до задачі розпізнавання людської руки на відео

Загальний підхід до розв'язання поставленої задачі можна розбити на окремі етапи:

а) Отримання поточного зображення з камери - у випадку простої RGB камери це лише отримання поточного зображення, проте у випадку Intel Realsense камери цей етап може потребувати деяких простих обчислень. Оскільки камера від компанії Intel має окрім звичайного RGB сенсору ще й інфрачервоний для отримання карти глибини може виникнути проблема синхронізації двох відеопотоків оскільки фізично ці два сенсори знаходяться на деякій відстані один від одного. Методи для синхронізації це звичайні алгоритми синхронізації відеопотоків 2 чи більше камер зі стереоскопії. Саме такий і реалізований у драйвері librealsense.

б) Попередня обробка зображення - використовується для стабілізації будь-якого цифрового зображеня припускаючи що сенсори не ідеальні та можуть вносити деяку похибку, яку можливо зменшити використавши фільтри зглажування: box фильтр, гаусівський фільтр чи фильтр медіаною.

в) Застосування вибраного алгоритму до зображення - цей етап також можна узагальнити для трьох вибраних підходів оскільки в усіх випадках вхідні дані алгоритму складаються лише з зображення, а вихідні з бінарного зображення з виділеними областями в яких рнймовірно локалізована людська рука.

г) Заключна обробка вихідного результату вибраного алгоритму - отримавши бінарне зображення з виділеними областями потрібно обробити усі області та відфільтрувати ті, що по певним параметрам немає сенсу розглядати. Наприклад, вважаючи, що рука на зображенні повинна мати розмір більший ніж деяка фіксована величина можливо відсіяти велику кількість малих за площею областей. Те саме стосується і занад-

то великих областей, які за деяких причин були помічені алгоритмом як рука, проте вони завеликі для обробки і їх також немає сенсу розглядати.

На виході буде отримано бінарне зображення з областями, в яких ймовірно знаходиться людська. На цьому задача локалізації людської руки завершується і цей результат може подаватися на вхід алгоритмів по аналізу жестів. Які працюють з кожною областю окремо та можуть проводити свою спеціальну перевірку на те, чи знаходиться у цій області людська рука.

2.2 Колірні моделі та простори

Оскільки цифрова електроніка оперує лише дискретною математикою, то над вченими 20 століття постала проблема представлення кольорів у ЕОМ. Основна ідея представлення кольорів прийшла з науки біології та експериментальним шляхом було доведено, що людина є трихроматом [6]. Сітчатка ока трихроматів має 3 види рецепторів, що називаються ковбочками, які відповідають за колірний зір. Кожна з цих ковбочок має як параметр деяку довжину хвилі, на яку вона дає максимальний зворотній сигнал.

За історичних обставин склалося так, що еталонним колірним простором є XYZ. Ця колірна модель була запропонована та прийнята організацією CIE (International Commission on Illumination) у 1931 році. Саме ця модель є базовою для практично усіх інших колірних моделей.

Експерименти, що були проведені Девідом Райтом та Джоном Гілдом у 1930х роках послужили основою для визначення функції колірної відповідності.

Колір у моделі XYZ задається таким чином:

$$\begin{aligned} X &= \int_{380}^{780} I(\lambda) * x(\lambda)\mathrm{d}\lambda \\ Y &= \int_{380}^{780} I(\lambda) * y(\lambda)\mathrm{d}\lambda \\ Z &= \int_{380}^{780} I(\lambda) * z(\lambda)\mathrm{d}\lambda \end{aligned} \tag{2.1}$$

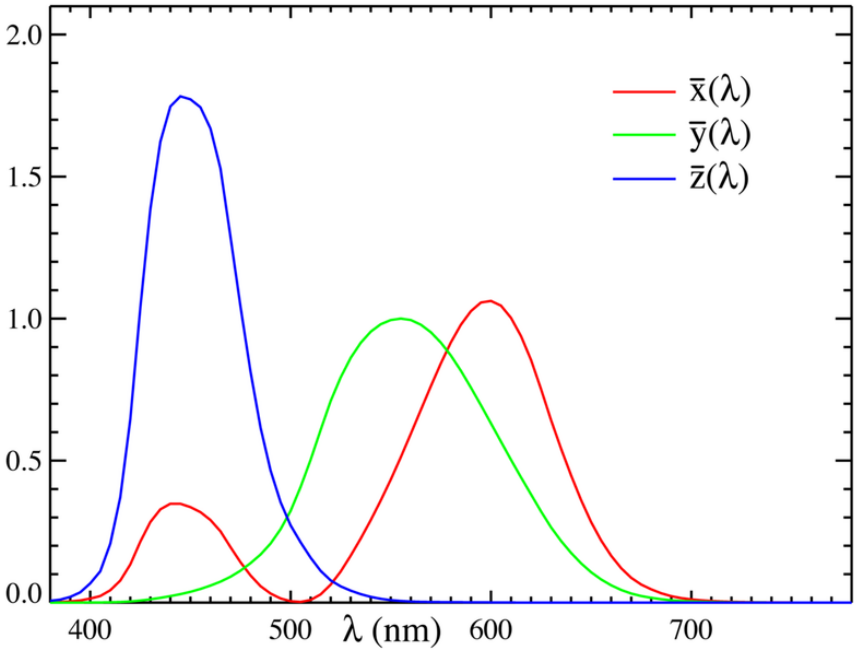


Рисунок 2.1 – Функції колірної відповідності

Саме ця модель задає правила змішування кольорів та задає обмеження що накладаються на спектральні складові, які мають один колір.

Якщо формально побудувати переріз простору XYZ площиною $X + Y + Z = const$, то 2 з 3 координат будуть лінійно незалежні і їх можна записати так:

$$\begin{aligned} x &= \frac{X}{X+Y+Z} \\ y &= \frac{Y}{X+Y+Z} \\ z &= \frac{Z}{X+Y+Z} \end{aligned} \tag{2.2}$$

Такий переріз називається хроматичною діаграмою.

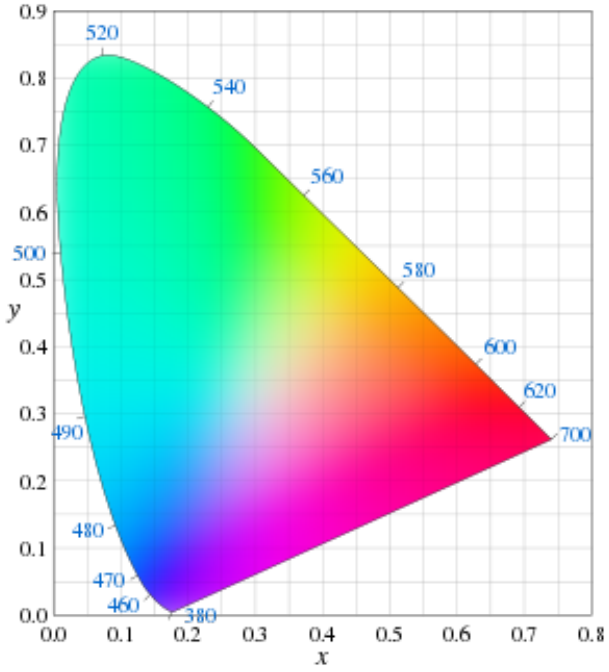


Рисунок 2.2 – Хроматична діаграма з довжинами хвиль кольорів

2.2.1 RGB

RGB (скорочено від англ. Red, Green, Blue — червоний, зелений, синій) — адитивна колірна модель, що описує спосіб синтезу кольору, за якою червоне, зелене та синє світло накладаються разом, змішуючись у різноманітні кольори. Широко застосовується в техніці, що відтворює зображення за допомогою випромінення світла.

У даній моделі колір кодується градаціями складових каналів (Red, Green, Blue). Тому за збільшення величини градації котрогось каналу — зростає його інтенсивність під час синтезу.

Кількість градацій кожного каналу залежить від розрядності бітового значення RGB. Зазвичай використовують 24-бітну модель, у котрій визначається по 8 біт на кожен канал, і тому кількість градацій дорівнює 256, що дозволяє закодувати 2563 = 16 777 216 кольорів.

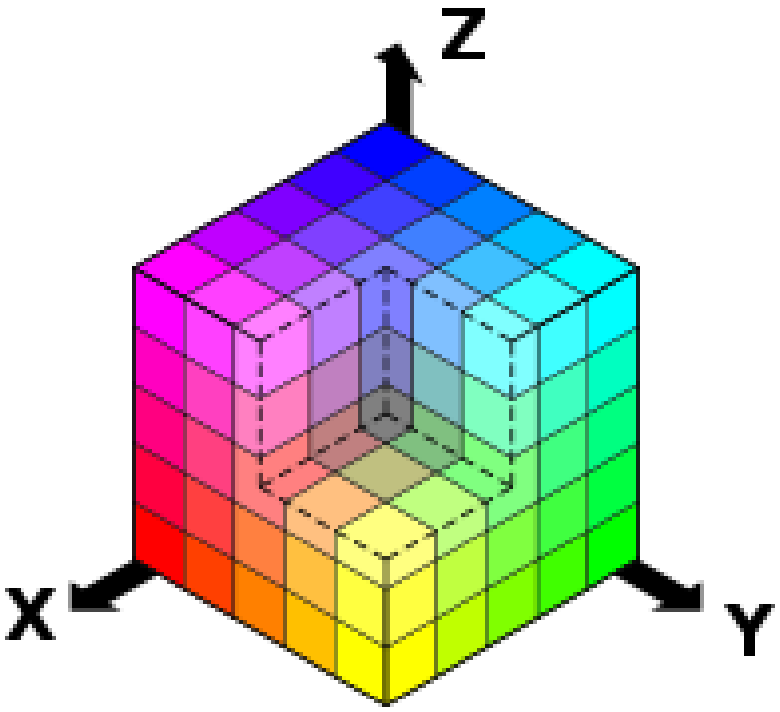


Рисунок 2.3 – Тривимірне представлення моделі RGB

Колірна модель RGB призначена сприймати, представляти та відображати зображення в електронних системах, таких як телебачення та комп’ютери, хоча її також застосовували у традиційній фотографії. Вже до електронного віку, модель RGB мала за собою серйозну теорію, засновану на сприйнятті кольорів людиною.

Типово приладами із RGB-входом є кольоровий телевізор і відеокамера, сканер і цифровий фотоапарат.

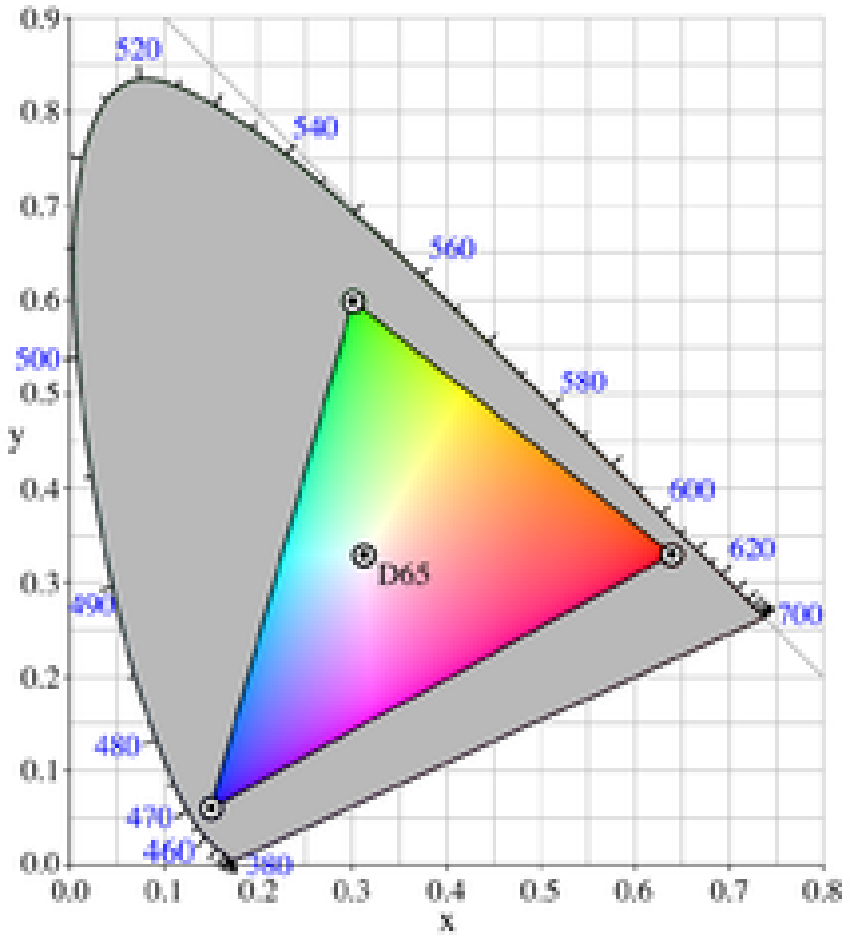


Рисунок 2.4 – Обмеженість моделі по можливості передачі кольору

Переваги моделі:

а) Апаратна близкість із монітором, сканером, проектором, іншими пристроями;
б) Велика кольорова гама, близька до можливостей людського зору;
в) Доступність багатьох функцій обробки зображення (фільтрів) у програмах растрової графіки;
г) Невеликий (порівняно до моделі CMYK) обсяг, проте ширший спектр кольорів.

Недоліки:

а) Обмеженість моделі (Рис 2.4);
б) Немає явного відділення люмінантної компоненти - для аналізу яскравості пікселя потрібно робити додаткові перетворення.

2.2.2 HSV

HSB — колірна модель, що використовується тільки для оформлення векторних і текстових об’єктів документа. Описує колірний простір, заснований на трьох характеристиках кольору: колірному тоні (Hue), насиченості (Saturation) і яскравості (Brightness).

- Hue — колірний тон, (наприклад, червоний, зелений або синьо-блакитний). Варіюється в межах 0-360°, але іноді приводиться до діапазону 0-100 або 0-1. У Windows весь колірний спектр ділиться на 240 відтінків (що можна спостерігати в редакторі палітри MS Paint), тобто тут «Hue» зводиться до діапазону 0-240 (відтінок 240 відсутній, оскільки він дублював би 0).
- Saturation — насиченість. Варіюється в межах 0-100 або 0-1. Чим більший цей параметр, тим «чистіший» колір, тому цей параметр іноді називають чистотою кольору. А чим ближчий цей параметр до нуля, тим ближчий колір до нейтрального сірого.
- Value (значення кольору) або Brightness — яскравість. Також задається в межах 0-100 або 0-1.

Модель була створена Елві Реєм Смітом, одним із засновників Pixar, в 1978 році. Вона є нелінійним перетворенням моделі RGB.

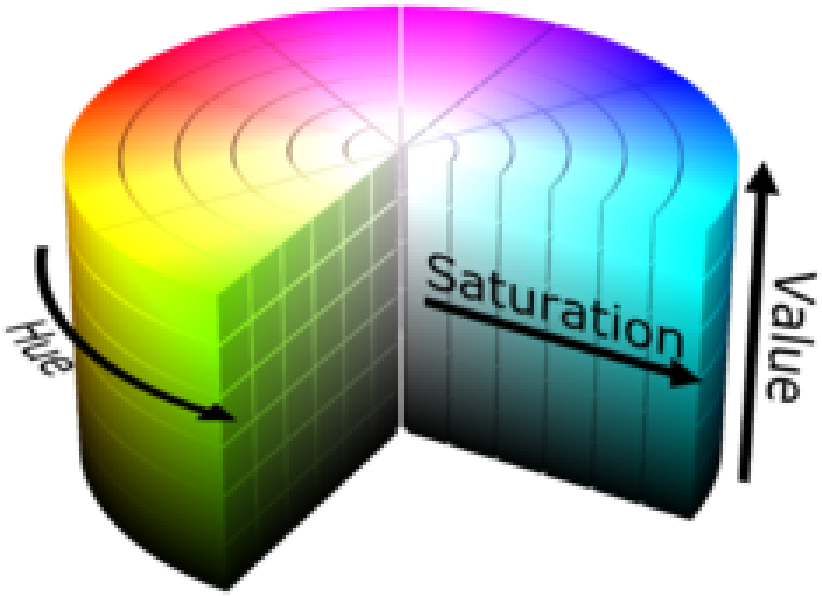


Рисунок 2.5 – Циліндричне представлення колірного простору

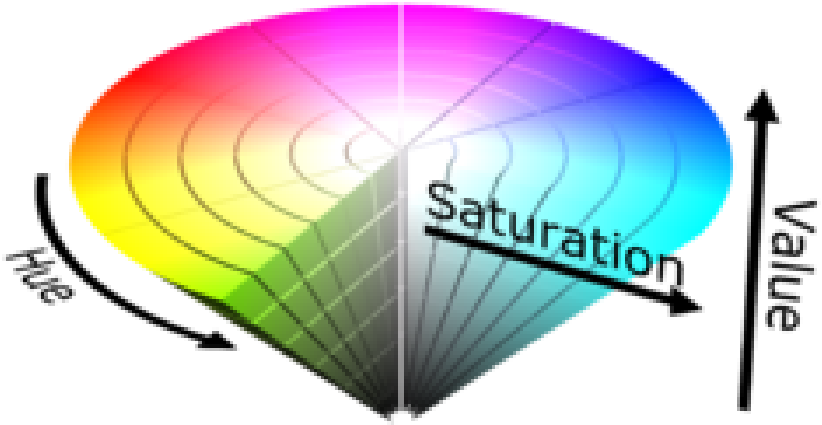


Рисунок 2.6 – Конічне представлення колірного простору

Колір, представлений в HSV, залежить від пристрою, на який він буде виведений, так як HSV — перетворення моделі RGB, яка теж залежить від пристрою. Для отримання коду кольору, не залежного від пристрою, використовується модель Lab.

2.2.3 Lab

Корірний простір Lab використовує як параметри світлосилу, відношення зеленого до червоного та відношення синього до жовтого. Ці три параметри утворюють тривимірний простір, точки якого відповідають певним кольорам.

Цей колірний простір розроблявся як апаратно-незалежний, тобто він задає кольори без врахування особливостей відтворення кольорів. Має три параметри для опису кольору: світлосила L (англ. Lightness) та два хроматичні параметри. Перший (умовно позначений латинською літерою a) вказує на співвідношення зеленої і червоної складової кольору, другий (позначений літерою b) — співвідношення синьої та жовтої складової.

На відміну від кольорових просторів RGB чи CMYK, які є, по суті, набором апаратних даних для відтворення кольору на папері чи на екрані монітора, Lab однозначно визначає колір. Тому Lab широко використовується в програмному забезпеченні для обробки зображень у якості проміжного кольорового простору, через яке проходить конвертування даних між іншими кольоровими просторами(наприклад, з RGB сканера в CMYK печатного процесу). При цьому особливі властивості Lab зробили редагування в цьому просторі потужним інструментом корекції кольору.

Завдяки характеру визначення кольору в Lab з’являється можливість окремо впливати на яскравість, контраст зображення і на його колір. У багатьох випадках це дозволяє прискорити обробку зображень. Lab надає можливість вибіркового впливу на окремі кольори в зображенні, посилення кольорового контрасту, незамінними є можливості, які цей колірний простір надають для боротьби із шумом на цифрових фотографіях.

2.2.4 YCrCb

YCrCb являється ще одним колірним простором в якому виділяється люмінантна складова кольору.

$$
\begin{aligned}
Y &= K_R * R + (1 - K_R - K_B) * G + K_B * B \\
Cb &= \frac{1}{2} * \frac{B - Y}{1 - K_B} \\
Cr &= \frac{1}{2} * \frac{R - Y}{1 - K_R}
\end{aligned}
\tag{2.3}
$$

де R, G, B - координати кольору у форматі RGB, K_R, K_B - деякі константи.

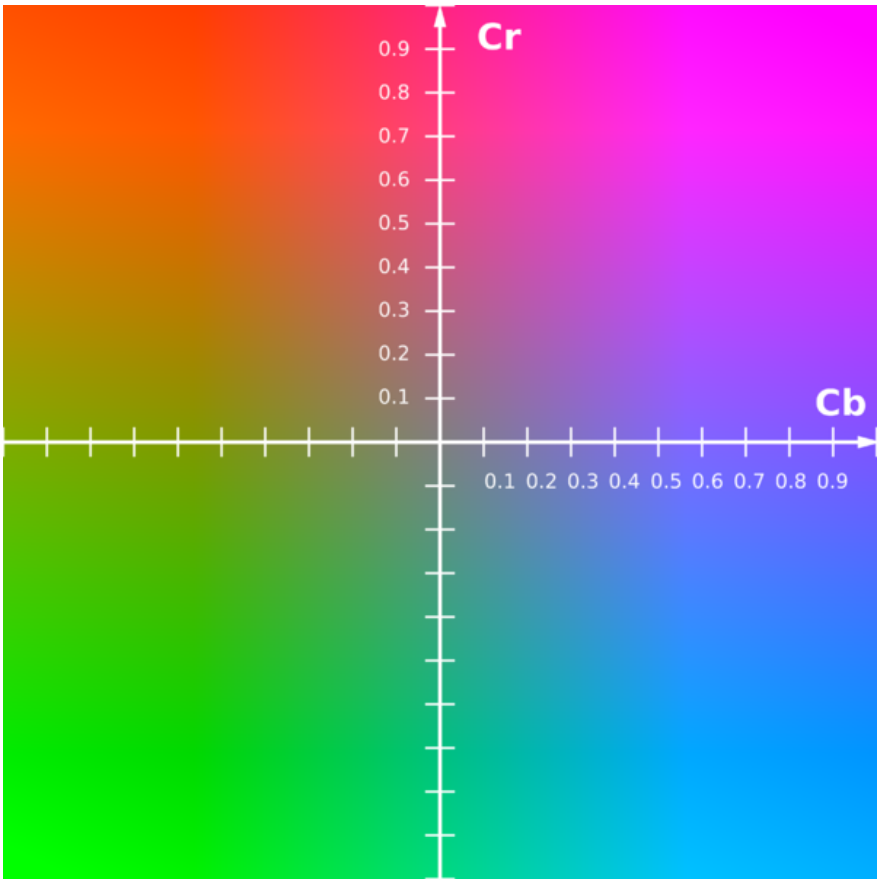


Рисунок 2.7 – Площина CbCr при Y = 0.5

В комп’ютерній графіці часто YCrCb покоординатно переводять з відрізка $[0, 1]$ у множину $[0, ..., 255]$ щоб кожну координату зберігати як беззнакове ціле число розміром 1 байт. Звісно це вносить деякі похибки, проте в такому форматі легче працювати з кольором.

2.3 Попередня обробка зображень

Попередня обробка зображень - перший етап у будь-якій роботі над зображеннями чи відео оскільки у більшості випадків від того, наскільки правильно буде вона проведена, залежить стабільність роботи алгоритмів у подальшому.

Основні причини чому попередня обробка зображень обов’язкова:

а) Видалення шумів - випадкові чи невипадкові шуми можуть з’являтися за багатьох причин і дуже сильно впливати на точність роботи системи. Також слід розуміти, що достатньо якісні камери дорого коштують і тому система, яка не буде враховувати випадкові шуми на простих камерах працювати буде погано або навіть кардинально неправильно. До того ж система по локалізації людської руки буде більш затребуваною якщо зможе достатньо добре працювати на простих вебкамерах ноутбуків

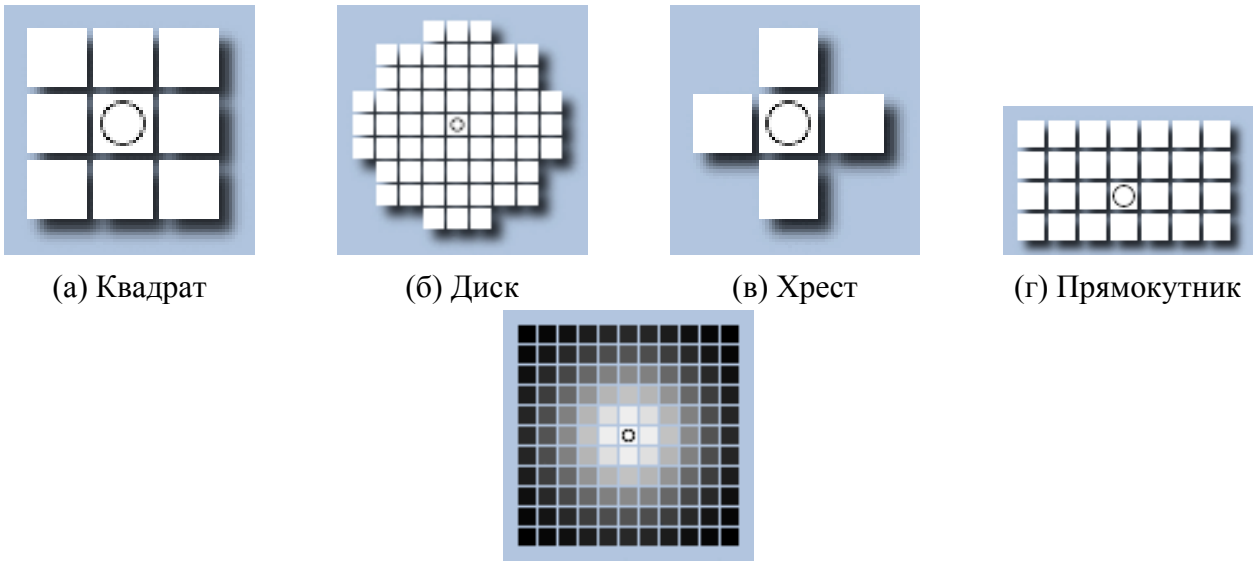
б) Стабілізація форм об’єктів - іноді шуми можуть бути не точкові, а мати деяку форму та спотворювати форму досліджуваних об’єктів на відео. Такі випадки також потрібно враховувати оскільки для видалення таких дефектів використовуються окремі підходи

2.3.1 Морфологічні перетворення зображень

Математична морфологія — це наука, яка вивчає методи і алгоритми аналізу і обробки геометричних структур, основана на теорії множин, топології і випадкових функцій. Застосовується при обробці цифрових зображень, але також може бути застосована до графів, полігональної сітки, стереометрії і бататьох інших просторових структур.

Морфологічні операції виконуються над двома зображеннями: вхідним зображенням і спеціальним, яке залежить від операції і типу виконуваної задачі. Таке спеціальне зображення в математичній морфології називається структурним елементом, примітивом чи ядром. Структурний елемент являє собою деяке двійкове зображення. Він може бути довільного розміру і структури, але за звичай розмір такого зображення має розмір 3x3, 4x4, 5x5 пі-

кселів, тобто значно менше вхідного зображення. Частіше за все використовуються симетричні елементи, такі як прямокутник фіксованого розміру чи круг заданого діаметру. В кожному елементі виділяють особливу точку, яку називають початковою (origin). Вона може вибиратися в будь-якому місці зображення, але найчастіше це центральний піксель.



(а) Квадрат (б) Диск (в) Хрест (г) Прямокутник

(д) Гаусівське ядро

Рисунок 2.8 – Основні структурні елементи

Структурні елементи 2.8а, 2.8б, 2.8в та 2.8г складаються лише з 0 та 1. Ядро 2.8д відрізняється тим, що 1 лише у центрі і далі з віддаленням від центра елементи мають величину, що виражається гаусівським законом розподілу [7].

Основними морфологічними операціями є:

а) Ерозія
б) Дилація
в) Розкриття
г) Закриття
д) Перенос

2.3.1.1 Ерозія

$$A \ominus B = z \in E | B_z \subseteq A \tag{2.4}$$

Інакше кажучи, ерозія множини A по примітиву B, це таке геометричне місце точок для всіх таких позицій точок центру z, при зсуві яких множина B цілком міститься в A.

(а) До

(б) Після

Рисунок 2.9 – Ілюстрація операції ерозії

2.3.1.2 Дилація

$$A \oplus B = z \in E | B_z^{\delta} \cap A \neq \emptyset \tag{2.5}$$

При цьому дилація множини A по структурному елементу B це множина всіх таких переміщень z, при яких множини A і B співпадають принаймні в одному елементі.

(а) До

(б) Після

Рисунок 2.10 – Ілюстрація операції дилації

2.3.1.3 Розкриття

$$A \circ B = (A \ominus B) \oplus B \tag{2.6}$$

Таким чином, розкриття множини A про структурному елементу B знаходиться як ерозія A по B, результат котрої піддається дилації по тому ж структурному елементу B. В загальному випадку розкриття сгладжує контури об'єкту, усуває візькі перешийки і ліквідує виступи невеликої ширини.

2.3.1.4 Закриття

$$A \bullet B = (A \oplus B) \ominus B \tag{2.7}$$

В результаті операції закриття відбувається згладження відрізків контурів, але, на відміну від розкриття, в загальному випадку заповнюються невеликі розриви і довгі заглибини малої ширини, а також ліквідуються невеликі отвори і заповнюються проміжки контуру.

2.3.1.5 Перенос

$$A_s = a + s | a \in A, \forall s \in E \tag{2.8}$$

Перенос можна визначити за допомогою упорядкованої пари чисел (x,y), де x – кількість пікселів зміщення вздовж осі X, а y – рух вздовж осі Y

2.3.2 Видалення шумів шляхом зглажування

Шуми на цифрових зображеннях доволі поширене явище, особливо якщо зображення було отримане у нестабільних умовах освітлення чи з неякісної камери. Нехтування цим етапом попередньої обробки може призвести до того, що алгоритми також будуть видавати зашумлений результат з великої кількістю дефектів.

Згорткове ядро являє собою матрицю та має деяку особоливу точку в матриці, яку називають центром чи початком. Частіше за все цю точку не задають, а вважають что нею є центр матриці оскільки будь-яке ядро з центром відмінним від центра матриці можна привести до ядра, можливо більшого розміру, щоб його центр співпадав з центром матриці.

Згортка матриці A розмірів $m \times n$ з ядром K розмірів $a \times b$ та центром у точці (c_1, c_2) є матриця R, яка будується таким чином:

$$R_{i,j} = \sum_{x=1}^{a} \sum_{y=1}^{b} A_{i-c_1+x, j-c_2+y} * K_{x,y}, \ i = 1..m, j = 1..n \tag{2.9}$$

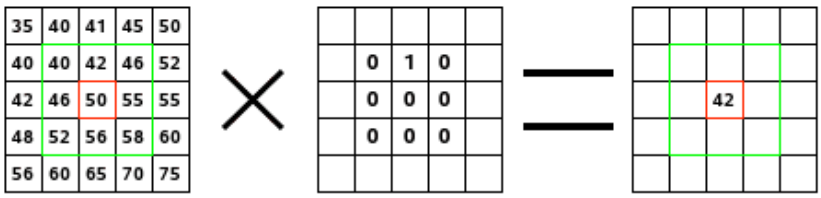


Рисунок 2.11 – Ілюстрація згортки

На Рис. 2.11 показаний приклад ядра зсуву. Такі ядра називають ядрами зсуву. Вони мають лише одну 1 в матриці, а вектор зсуву можна отримати пободувавши вектор з початком в центрі матриці, а кінцем в одиниці.

Дивлячись на формулу 2.9 можна побачити, що на краях зображення з'являється невизначеність, якої можна позбутися доповнивши матрицю A до розмірів $m + 2a \times n + 2b/$.

Доповнювати можна по-різному: нулями, дублювати крайні елементи чи проводити апроксимацію.

Види зглажування выдрізняються лише ядром згортки та операцією над елементами згортки. Як правило операція - сума. Такі фільтри називають лінійними.

$$K = \frac{1}{K_{width} \cdot K_{height}} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \\ . & . & . & \dots & 1 \\ . & . & . & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}$$

Рисунок 2.12 – Box filter

$$G_{x,y} = Ae^{\frac{-(x-\mu_x)^2}{2\sigma_x^2} + \frac{-(y-\mu_y)^2}{2\sigma_y^2}} \tag{2.10}$$

Прикладами лінійних фільтрів є:

а) Box filter - прямокутний фільтр. Його ядро має вигляд як на Рис. 2.12;

б) Гаусівський фільтр. Елементи матриці мають вигляд щільності двовимірного гаусівскього розподілу 2.10.

Прикладами нелінійних фільтрів є:

а) Фільтр медіаною - операція над елементами згортки є не сума, а вибір медіани серед них;

б) Білатеральний фільтр - достатньо потужний фільтр, який дає дуже хороше розмиття зображення, проте він працює значно довше ніж перераховані вище і саме це є його основним недоліком [8].

2.4 Основні алгоритми локалізації людської руки на відео

2.4.1 Віднімання фону

Віднімання фону дуже поширений алгоритм, який використовується у ситуаціях, коли відомо, що камера відносно нерухома. Завдяки йому можна

достатньо просто виділити фон із зображення, а значить і передній план чи просто об’єкти, які динамічно рухаються на відео.

Часто використовується для обробки відео з дорожніх відеорегістраторів чи IP камер, які розташовані у людних місцях. Відніманням фону можна миттево отримати усі рухомі об’єкти на відео - машини, пішоходи тощо. Також є певні модифікації цього алгоритму для виділення навіть об’єктів, що не рухаються певний час, проте вони більш пристосовані під окремі задачі.

2.4.1.1 Математичні основи

$$P[F(t)] = P[I(t)] - P[B] \tag{2.11}$$

де $P[B]$ - зображення фону, $P[I(t)]$ - поточне зображення, $P[F(t)]$ - різниця між поточним зображенням та фоном.

За умови незмінності фону можна працювати з формулою (2.11).

Для відслідковування рухів динамічних об’єктів також використовується рівняння:

$$\|P[F(t)] - P[F(t+1)]\| > Treshold \tag{2.12}$$

де $Treshold$ - деяка задана порогова величина.

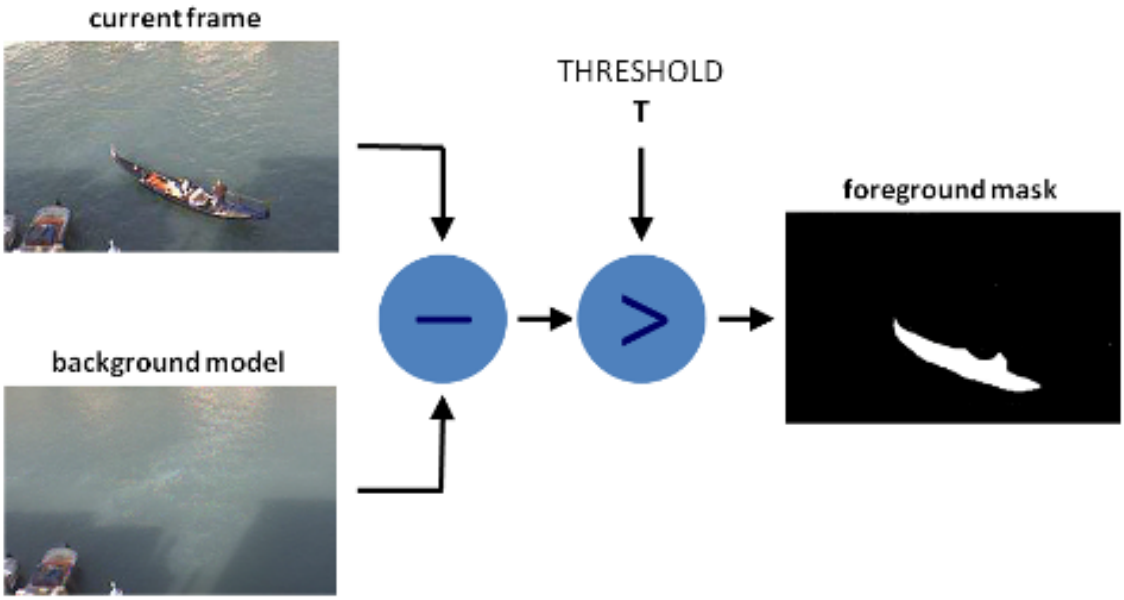


Рисунок 2.13 – Ілюстрація роботи простого алгоритма віднімання фону

Тобто маючи попередньо зображення фону можна отримати передній план зображення просто віднявши від поточного зображення поелементно зображення фону.

Також існують підходи, за яких зберігається не фонове зображення, а історія N останніх зображень. Такі алгоритми називають змішаними.

Нехай у деякий момент $t = t^*$ у пам'яті зберігається історія N останніх фреймів відео $P[I(t^* - 1)], ..., P[I(t^* - N)]$, $\lambda_1, ..., \lambda_N : \sum_{i=1}^{N} \lambda_i = 1$ - ваги, тоді фонове зображення може бути виражене як:

$$P[F(t^*)] = \sum_{i=1}^{N} \lambda_i * P[I(t^* - i)] \tag{2.13}$$

Від вибору λ_i залежить значимість віддалених у часі фреймів у фоновому зображенні.Якщо взяти їх усіх рівними $\frac{1}{N}$, то такий алгоритм називають змішаною моделю середнього значення віднімання фону.

Виділення переднього фону можна записати як:

$$Foreground[I(t^*)] = |P[F(t^*)] - P[I(t^*)]| > Threshold \tag{2.14}$$

$|P[F(t^*)] - P[I(t^*)]| > Threshold$ у формулі 2.14 слід розуміти як поелементна різниця матриць і поелементне порівняння цієї різниці з певною пороговою величиною. Тобто на виході отримане бінарне зображення - матриця з 0 та 1.

2.4.1.2 Особливості реалізації

У роботі використовується змішана гаусівська модель віднімання фону, що описана у [9] та [10]. Така модель краща за звичайну модель середнього значення через те, що чим більше віддалена від поточного зображення фрейм, тим менша його значимість на данний момент. Також гаусівский розподіл дуже добре себе показує у багатьох ситуаціях. Звісно можна вибрати коефіцієнти згідно з рядом фібоначі, проте гаусівський розподіл показує кращі результати.

2.4.2 Байесовський класифікатор

2.4.2.1 Теоретичні засади

Цей підхід зустрічається у літературі дуже часто і основна формула, що описує модель, на перший погляд незрозуміла:

$$P(s|c) = \frac{P(c|s) * P(s)}{P(c)} \tag{2.15}$$

Це звичайний запис теореми Байеса для знаходження апостеріорних ймовірностей.

$P(s|c)$ - ймовірність того, що піксель належить до множини кольорів шуканого об'єкта за умови що піксель має колір c. $P(s|c)$ в свою чергу має обернене формулювання - ймовірність того, що піксель приймає значення c за умови що він належить до множини кольорів об'єкта. $P(s)$ - ймовірність того, що піксель належить до множини кольорів об'єкта. $P(c)$ - загальна ймовірність того, що піксель має колір c.

2.4.2.2 Пристосування до задачі розпізнавання кольорів об'єкта

З формули 2.15 не зовсім зрозуміло як взагалі шукати 3 ймовірності, що присутні у правій частині рівності.

Основне припущення цього підходу - кольори окремих пікселей на зображенні незалежні один від одного. За такого припущення можна сказати, що $P(s)/P(c)$ - деяка загальна константа якою можна знехтувати поставивши її рівною 1, а $P(c|s) = m/n$, де m - кількість пікселів кольору c, що належать об'єкту, n - загальна кількість пікселів, що належала об'єкту.

Приймаючи до уваги, що в задачі розпізнавання людської руки на відео класифікатор використовується лише для 2 із 3 компонент колірного простору, в якому є виділення люмінантної складової в окрему компоненту, то результатом навчання класифікатора слід очікувати деяку матрицю ймовірностей.

Основною ідеєю корекції моделі є те, що навчання могло проводитись не зовсім якісно і тому деякі ймовірності можуть бути лише поодинокими

шумами. Матрицю ймовірностей можна розцінювати як деяке зображення та його обробку можна також розцінювати як обробку специфічного зображення.

2.4.2.3 Корекція моделі шляхом видалення поодиноких ймовірностей

Перший підхід до обробки матриці ймовірностей має прив'язку лише до того чи нульова певна ймовірність.

$$A[i,j] = (P[i,j] > 0) \tag{2.16}$$

де P - матриця ймовірностей.

Матриця A складається лише з 0 та 1. Наступним кроком є застосування згортки з ядром, що складається лише з 1 та має форму круга з радіусом R. Це ядро ненормоване, тобто не виконується $\sum k_{i,j} = 1, \forall i,j \;\; k_{i,j} > 0$. Таким способом буде отримана матриця N, що її (i,j) елемент приймає значення кількості його ненульових сусідів у радіусі R навколо нього. Тобто побудована матриця N на позиції (i,j) містить число ненульових ймовірностей матриці P у крузі з центром в (i,j) та радіуса R. Заключним кроком є занулювання усіх ймовірностей, що мають недостатню кількість ненульових сусідів у відсотках від площі круга згорткового ядра.

Таким чином після цієї процедури вигляд матриці ймовірностей більш заокруглений та не містить поодиноких ймовірностей.

2.4.2.4 Корекція моделі шляхом видалення слабких ймовірностей

Ідея цього способу фільтрації прийшла після деякого аналіз матриці ймовірностей та помічання факту, що ідеально руку на зображенні виділити неможливо. Для цього потрібно повністю стирати краї руки, що небажано, проте саме ці краї вносять у модель деяку область ймовірностей, яка має пік в декількох точках, що знаходяться на малій відстані одна від одної, а в інших ймовірності порядка $10^{-4} - 10^{-2}$.

Таким чином пропонується провести розмиття ядром середнього арифметичного та видалити усі ймовірності, що менші за задану порогову величину.

Цей алгоритм дозволяє видалити скупчення ймовірностей, що присутнє у кожній моделі і відповідає саме кольорам руки, які знаходяться на її границі. Кольори на границі ближче всього до чорного кольору оскільки саме в них є перепад освітлення і вони лише зпотворюють класифікатор.

2.4.2.5 Удосконалення процесу навчання класифікатора

Оскільки процес навчання класифікатора доволі тривалий через те, що треба вручну обробляти велику кількість фотографій, то є сенс подумати над іншими можливими шляхами навчання.

У цій роботі використовується камера Intel Realsense F200, яка окрім кольорової камери має ще і сенсор глибини. Завдяки вбудованій у драйвер можливості отримувати синхронізовану карту глибини зображення є спосіб удосконалити та кардинально пришвидшити навчання класифікатора.

Карта глибини - матриця з цілими типу uint16 (16 бітне ціле беззнакове число). 0 використовується для позначення неможливості визначити відстань.

Таким чином можна фільтрувати колірне зображення за відстанню та встановити межу 0.5м і усі пікселі, відстані до яких більша за встановлену границю, будуть мати чорний колір, який використовується для позначення пікселя, який не бере участь у навчанні.

2.4.3 Обробка відеопотоку камери глибини

Завдяки камері Intel Realsense F200 є можливість працювати з картами глибини зображення, проте для локалізації руки та подальшого аналізу жестів в цьому випадку дані про кольори насправді непотрібні. Вимикання колірного відеопотоку значно збільшує швидкість отримання зображення оскільки не виконується синхронізація двох відеопотоків, яка у драйвері навіть не оптимізована.

Обробка карт глибини ідентична до того, що вище було зазначено для кольорових зображень:

а) Erode, dilate - для стабілізації геометричних контурів об’єктів;
б) Розмиття - видалення шумів оскільки інфрачервоний сенсор цієї камери працює не зовсім стабільно та часто можна помітити випадкові шуми.

Далі у загальному випадку за допомоги SimpleBlobDetector з бібліотеки opencv можна отримати усі blob елементи зображення та провести фільтрацію по певним параметрам - площа, кількість геометричних дефектів контура об’єкта тощо.

2.5 Критерії оцінки якості роботи системи

Для оцінки адекватності роботи алгоритмів запропоновано ввести такі 2 критерії:

а) Відсоток пікселів, що належать руці та були помічені алгоритмом як ті, що належать руці;
б) Відсоток пікселів, які не належать руці, проте були помічені алгоритмом як ті, що належать.

Цих критеріїв буде достатньо для порівняння алгоритмів.

Перевірка алгоритмів буде проводитись в найкращих для кожного алгоритма умовах:

а) Віднімання фону - рука рухається весь час на відео, сторонніх об’єктів, що рухаються, немає;
б) Байесовський класифікатор - при тих самих умовах освітлення що і проводилось навчання та з умовою, що лице не знаходиться у полі зору камери (оскільки можна провести детекцію лиця на відео і потім вилучити його з досліджуваних областей;
в) Камера глибини - у полі зору на відстані до 0.5 метрів окрім руки інших об’єктів немає.

2.6 Висновки за розділом

У цьому розділі розглянуті теоретичні засади колірних просторів, попередньої обробки зображень та вибраних алгоритмів локалізації людської руки на відео.

Окрім класичного метода віднімання фону розглянуто байесовський класифікатор та запропоновані модифікації моделі. Також через відносну незручність класичного методу навчання класифікатора запропонований метод швидкого навчання за допомоги камери Intel Realsense F200. Останнім було розглянуто більш сучасний метод локалізації руки на відео, що потребує камеру від компанії Intel, як приклад сучасного підходу до розв’язання поставленої проблеми.

Введені критерії, які повною мірою оцінюють найважливіші аспекти проблеми - якість розпізнавання руки та помилки розпізнавання.

РОЗДІЛ 3 ПРАКТИЧНА ЧАСТИНА

3.1 Результати роботи та аналіз оптимальних умов роботи алгоритмів

У роботі реалізовані такі модулі:

а) train_from_folder - модуль навчання байесовського класифікатора з вказаної папки, в ній мають міститися кольорові фотографії та маски для кожної відповідно. Назви мають бути формату [номер]_i.[формат] для зображення, та [номер]_m.[формат] - маска;

б) train_from_realsense - модуль навчання байесовського класифікатора за допомоги камери Intel Realsense.

в) process_opencv - обробка відеопотоку за допомоги байесовського класифікатора та отримання відео за допомоги бібліотеки opencv. Бібліотека opencv накладає деякі обмеження на максимальну кількість фреймів за секунду, тому було вирішено реалізувати варіант з використанням драйвера Intel Realsense так як в ньому такого обмеження немає;

г) process_realsense - обробка відеопотоку за допомоги байесовського класифікатора та отримання відео за допомоги драйвера Intel Realsense;

д) show_model_representation - модуль, що призначений для читання байесовської моделі з файла та виведення зображень ілюстрацій навчання;

е) filter_model - модуль для роботи з моделями, в якому реалізовані усі зазначені у практичній частині перетворення матриці ймовірностей.

Усі програмні модулі виконані у консольному середовищі з виведенням відеопотоків в окремі вікна для ілюстрації роботи алгоритмів.

3.1.1 Віднімання фону

Віднімання фону достатньо показало достатньо непогані результати.

(а) Поточне кольорове зображення

(б) Маска виділеного переднього плану

Рисунок 3.1 – Ілюстрація роботи алгоритма віднімання фону

Значення першого критерія - 91.37
Значення другого критерія - 0.03

(а) Поточне кольорове зображення

(б) Маска виділеного переднього плану

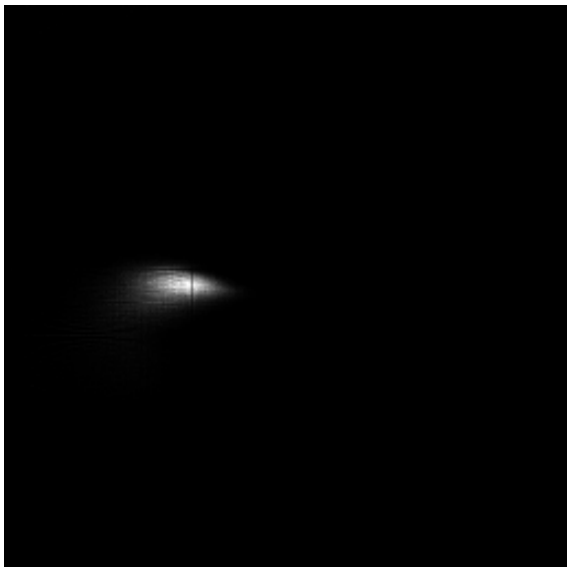
Рисунок 3.2 – Ілюстрація роботи алгоритма віднімання фону

Значення першого критерія - 98.26
Значення другого критерія - 0.74

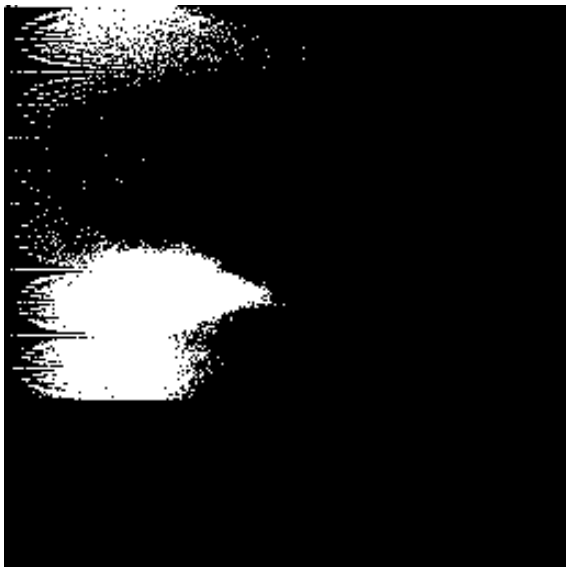
3.1.2 Байесовський класифікатор

3.1.2.1 Класична реалізація

Після навчання класифікатора можна отримати графічне представлення матриці ймовірностей як два зображення градації сірого.



(а) Матриця ймовірностей



(б) Матриця ненульових ймовірностей

Рисунок 3.3 – Результат навчання класифікатора

На Рис. 3.3а показана матриця ймовірностей в градації сірого. Ймовірності переводяться методом домноження їх на 255. На Рис. 3.3б показані усі елементри матриці ймовірностей, що мають ненульові значення.

(а) Кольорове зображення

(б) Відфільтроване

Рисунок 3.4 – Ілюстрація роботи байесовського класифікатора

Значення першого критерія - 84.62

Значення другого критерія - 4.68

(а) Кольорове зображення

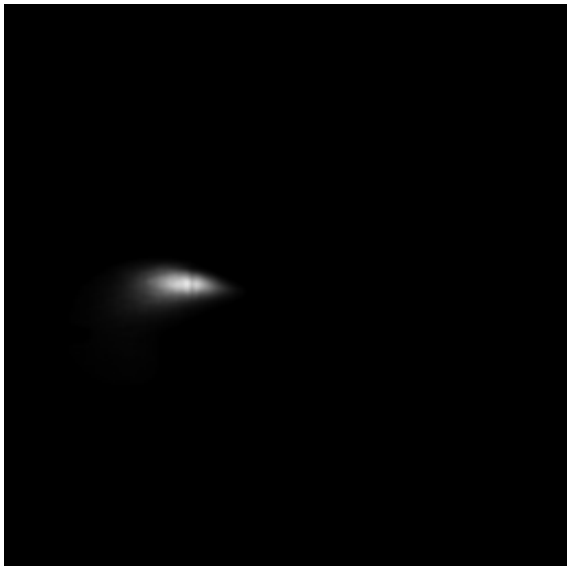
(б) Відфільтроване

Рисунок 3.5 – Ілюстрація роботи байесовського класифікатора

Значення першого критерія - 76.21
Значення другого критерія - 3.84

3.1.2.2 Корекція класифікатора

Застосувавши корекції ймовірностей матриця ймовірностей прийняла такий вид:



(а) Матриця ймовірностей

(б) Матриця ненульових ймовірностей

Рисунок 3.6 – Результат обробки матриці ймовірностей класифікатора

Значення першого критерія покращились майже на 3%. Значення другого критерія практично не змінилися.

3.1.3 Камера глибини

Камера глибини показує найкращі результати по першому притерію з мінімальним значенням 97%, проте по другому критерію досягає 3%.

Такі результати по другому критерію можна пояснити тим, що на границі руки інфрачервоні промені заломлюються і тому значення нестабільні.

Також у ході роботи були виявлені помилки у драйвері камери, що приводили до появу фантомної руки при роботі одночасто з двома відеопотоками - кольору та глибини. Через помилки у частині, яка відповідає за синхронізацію цих відеопотоків, на кольоровомі зображенні з’являється фантом найближчого до камери об’єкта, а на зображенні глибини також фантом такої ж форми, проте значення пікселей у його області камера помічає як ”неможливо визначити відстань”.

3.2 Порівняння алгоритмів

Усі алгоритми по першому критерію подолали планку 75%, а в середньому 80%. По другому критерію усі алгоритми не переходять межі у 5%. Це значить, що вони не генерують великі помилкові області, які помічаються як ті, що містять людську руку.

3.2.1 Віднімання фону

Сильні сторони:

а) Хороша точність - у межах 95% по першому критерію;
б) Відсутність складних обчислень - кожна ітерація отримання поточного фрему супроводжується обрахуванням лінійної комбінації заданої кількості зображень;
в) Відсутня прив’язка до спеціального обладнання.

Слабкі сторони:

а) Зі збільшенням кількості рухомих об’єктів на відео алгоритм помічає їх усі, а тому з’являється потреба у більш складному алгоритмі, який буде по формі класифікувати об’єкти;
б) Чутливий до зміни освітлення - зміна освітлення призведе до невизначеної поведінки протягом повного перепису історії моделі віднімання фону;
в) Зміна положення камери повністю руйнує логіку алгоритму та призводить до невизначеної поведінки.

3.2.2 Байесовський класифікатор

Сильні сторони:
а) Достатня точність у середньому до 80%;
б) Відсутність складних обчислень - класифікація пікселя зводиться лише до перевірки ймовірності його кольору бути кольором людської шкіри;
в) Відсутня прив’язка до спеціального обладнання;
г) Рух камери у загальному випадку не впливає на класифікацію.
Слабкі сторони:
а) Для стійкості до зміни характеру освітлення потрібно тренувати додаткові моделі та на етапі класифікації проводити вибір однієї з них;
б) Через прив’язку до кольору шкіри помічає усі області, де її знаходить і тому у загальному випадку потрібно проводити їх обробку.

3.2.3 Камера глибини

Сильні сторони:
а) Найкраща точність - до 97% по першому критерію;
б) У випадках непотрібності кольорового каналу відео відсутність складних обчислень;
в) Рух камери не впливає на роботу алгоритму взагалі.
Слабкі сторони:
а) Потрібна спеціальна камера;

б) У разі необхідності кольорового каналу значно збільнується кількість обчислень через синхронізацію двох відеопотоків.

3.3 Обгрунтування вибору платформи та мови програмування

На початкових етапах виконання роботи основною мовою програмування для реалізації поставлених задач була мова Python. Ця мова була вибрана через простоту та зручність розробки, проте для реалізації байесовського класифікатора вона не підходить оскільки швидкість доступу до елементів матриці np.ndarray дуже низька, а класифікація пікселів побудована саме на цьому.

Таким чином було вирішено зробити аналогічну реалізацію на C++ та в разі великої різниці у часі обробки класифікатором одного фрейма продовжити розробку на C++.

Реалізація байесовського класифікатора на мові Python в секунду обробляла майже 2 фрейма, в той час як C++ версія коливалася в межах 300-400 фреймів в секунду.

3.4 Висновки до розділу

У цьому розділі представлені практичні результати реалізованих алгоритмів.Кожен підхід має свої сильні та слабкі сторони, які були описані у порівнянні алгоритмів.

Віднімання фону дало відносно непогані результати в середньому 94% по першому притерію, та 0.5% по другому за умов що на відео не були присутні ішні рухомі об’єкти крім руки. Цей алгоритм можна запускати на бюджетних ПК.

Байесовський класифікатор дав найгірші результати - в середньому 80% по першому критерію та 4% по другому. Проте його сильними сторонами є те, що класифікація взагалі не потребує ніяких обчислень, а лише доступу до відповідного елемента матриці. Також в опрацьованих джерелах були наведені підходи до пристосування класифікатора до змінних умов освітле-

ння, що у загальному випадку може покращити середні показники критеріїв за стабільних умов.

Робота з камерою глибини показала найкращі показники ... . За умови отримання лише відеопотоку з камери глибини без кольорового відеопотоку цей підхід також не потребує складних обчислень. У випадку, коли потрібні обидва видеопотоки, виникає проблема синхронізації двох відеопотоків і швидкість отримання фреймів збільшується на два порядки. Основним недоліком цього підходу є те, що потрібно використовувати спеціальне обладнання.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться оцінка основних характеристик програмного продукту, призначеного для обробки вхідного відеопотоку і локалізації людської руки. Інтерфейс користувача був розроблений за допомогою мови програмування C++ та opencv.

Програмний продукт призначено для використання на персональних комп'ютерах під управлінням операційних систем Windows або Linux.

Нижче наведено аналіз різних варіантів реалізації модулю з метою вибору оптимальної, з огляду при цьому як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. Як прямі, так і побічні витрати розподіляються по продуктам та послугам у залежності від потрібних на кожному етапі виробництва обсягів ресурсів. Виконані на цих етапах дії у контексті метода ФВА називаються функціями.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Фактично, цей метод працює за таким алгоритмом:

- визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На цьому ж етапі оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат.
- для кожної функції визначаються повні річні витрати й кількість робочих часів.

- для кожної функції на основі оцінок попереднього пункту визначається кількісна характеристика джерел витрат.
- після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

4.1 Постановка задачі техніко-економічного аналізу

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки.

Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

- програмний продукт повинен функціонувати на персональних комп'ютерах зі стандартним набором компонент;
- забезпечувати високу швидкість обробки великих об'ємів даних у реальному часі;
- забезпечувати зручність і простоту взаємодії з користувачем або з розробником програмного забезпечення у випадку використання його як модуля;
- передбачати мінімальні витрати на впровадження програмного продукту.

4.1.1 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного продукту, який аналізує процес за вхідними даними та будує його модель для подальшого прогнозування. Виходячи з конкретної мети, можна виділити наступні основні функції ПП:

F_1 вибір мови програмування;

F_2 інтерфейс користувача;

F_3 вибір камери для роботи.

Кожна з основних функцій може мати декілька варіантів реалізації.

- Функція F1 :

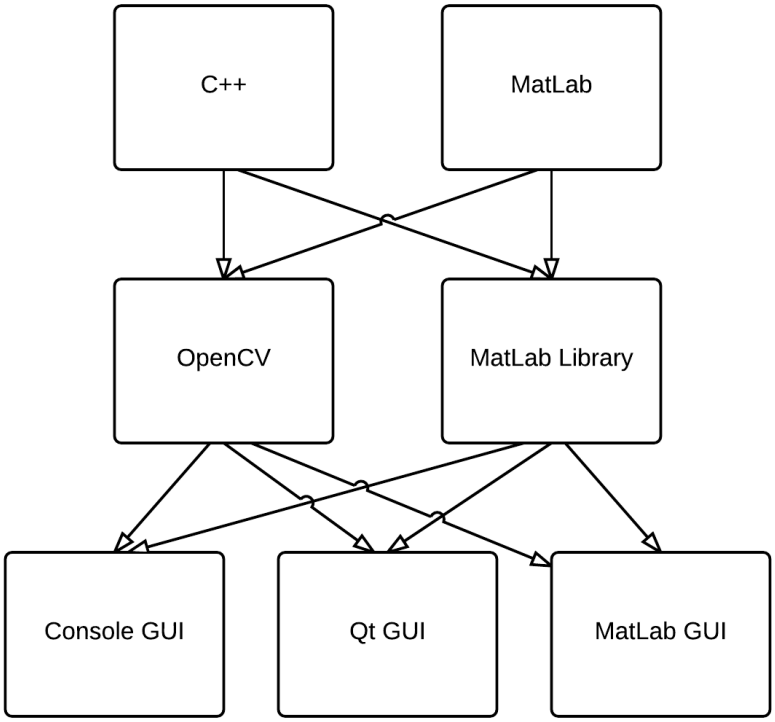


Рисунок 4.1 – Морфологічна карта

а) мова програмування Python;
б) мова програмування C++;

- Функція F2:
  а) інтерфейс користувача, створений на opencv;
  б) інтерфейс користувача як консольний додаток.
- Функція F3:
  а) проста веб камера;
  б) Intel Realsense F200.

4.1.2 Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1). На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (таблиця 4.1).

Морфологічна карта відражає всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

Таблиця 4.1 – Позитивно-негативна матриця

Основні функції	Варіанти реалізації	Переваги	Недоліки
F1	А	Займає менше часу при написанні коду	Менша швидкість написання коду
	Б	Швидкодійний, кросплатформений	Менша швидкість написання коду
F2	А	Проста розробка	Недостатній функціонал
	Б	Простота інтерфейса	Неможливість динамічної зміни робочих параметрів
F3	А	Доступність	Лише камера RGB
	Б	Потужний SDK, depth камера	Висока ціна

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

4.1.2.1 Функція F_1

Оскільки для нас важлива швидкість роботи, варіант а) має бути відкинутий.

4.1.2.2 Функція F_2

Оскільки для нас важлива простота, варіант а) має бути відкинутий.

4.1.2.3 Функція F_3

Вибір двох різних камер основна ідея проекту і тому вважаємо варіанти а) та b) гідними розгляду.

Таким чином, будемо розглядати такі варіанти реалізації ПП:

а) $F_{1б} - F_{2б} - F_{3а}$

б) $F_{1б} - F_{2б} - F_{3б}$

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.2 Обґрунтування системи параметрів ПП

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

4.2.1 Опис параметрів

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

X_1 - швидкодія мови програмування;

X_2 - об'єм пам'яті для збереження даних;

X_3 - час обробки даних;

X_4 - потенційний об'єм програмного коду.

X_1 відображає швидкодію операцій залежно від обраної мови програмування.

X_2 відображає об'єм оперативної пам'яті персонального комп'ютера, що необхідний для збереження та обробки даних під час виконання програми.

X_3 відображає час, який витрачається на дії.

X_4 показує кількість програмного коду який необхідно створити безпосередньо розробнику.

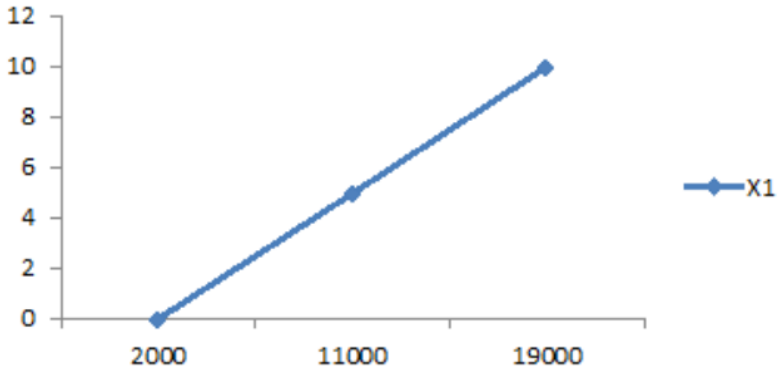


Рисунок 4.2 – Швидкодія мови програмування X_1

4.2.2 Кількісна оцінка параметрів

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у табл. 4.2

Таблиця 4.2 – Основні параметри ПП

Назва параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X_1	оп/мс	19000	11000	2000
Об’єм оперативної пам’яті	X_2	мб	32	16	8
Час обробки зображення	X_3	мс	1000	420	60
Об’єм програмного коду	X_4	строк	2000	1500	1000

За даними таблиці 4.2 будуються графічні характеристики параметрів: рис. 4.2 - рис. 4.5

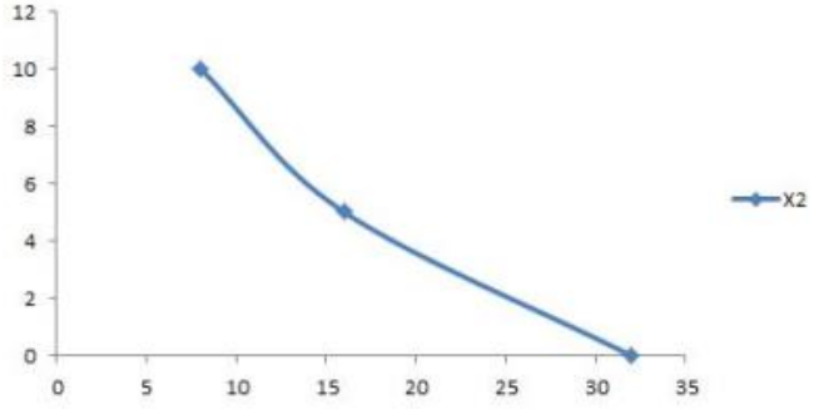


Рисунок 4.3 – Об’єм оперативної пам’яті X_2

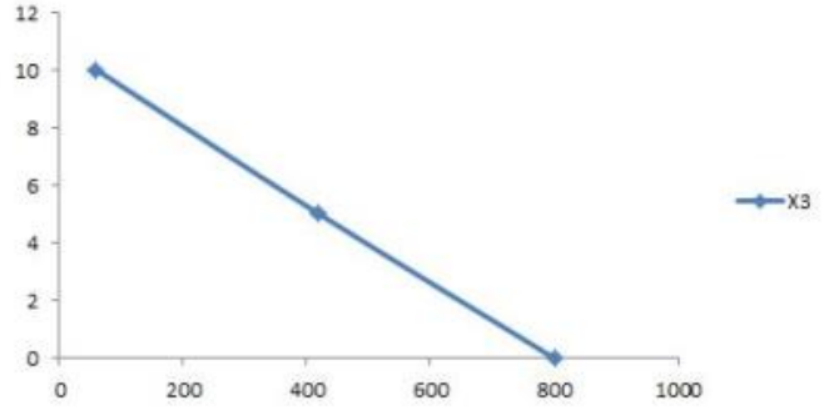


Рисунок 4.4 – Час, який витрачається на дії X_3

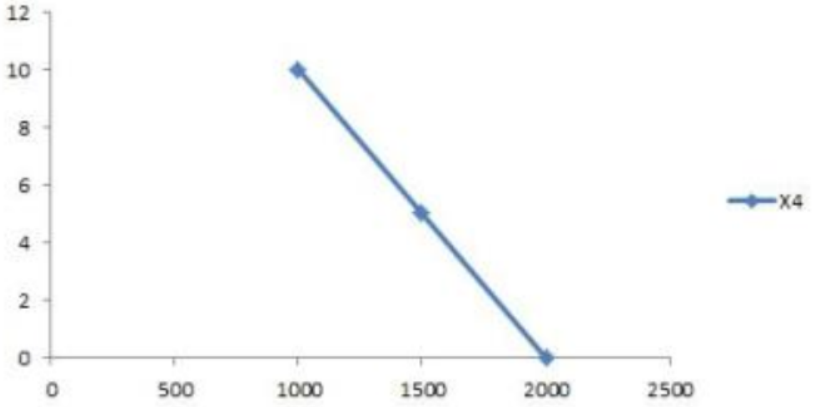


Рисунок 4.5 – Об’єм програмного коду X_4

4.2.3 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при оптичному аналізі нотних листів.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Результати експертного ранжування параметрів

Позначення	Назва	Ранг за оцінкою експерта							Сума рангів R_i	Відх. Δ	Δ^2
		1	2	3	4	5	6	7			
X_1, оп/мс	Швидкодія мови програмування	4	3	4	4	4	4	4	27	0.75	0.56
X_2, мб	Об'єм оперативної пам'яті	4	4	4	3	4	3	3	25	-1.25	1.56
X_3, мс	Час, який витрачається на дії	2	2	1	2	1	2	2	12	-14.25	203.06
X_4, строк	Об'єм програмного коду	5	6	6	6	6	6	6	41	14.75	217.56
Разом		15	15	15	15	15	15	15	105	0	420.75

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R = \sum_{i=1}^{n} R_i = \frac{Nn(n+1)}{2} = 105 \tag{4.1}$$

де n - число оцінюваних параметрів, N - число експертів

б) середня сума рангів:

$$T = \frac{1}{n}\sum_{i,j} R_{ij} = 24.5 \tag{4.2}$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T \qquad i = 1 \dots n \tag{4.3}$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta^2 = 35 \tag{4.4}$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12 \cdot S}{N^2(n^3 - n)} \tag{4.5}$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0.67.

Скориставшись результатами ранжування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4

Таблиця 4.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Чисельне значення
	1	2	3	4	5	6	7		
X1 і X2	=	>	=	<	=	<	<	<	0.5
X1 і X3	<	<	<	<	<	<	<	<	0.5
X1 і X4	>	>	>	>	>	>	>	>	1.5
X2 і X3	<	<	<	<	<	<	<	<	0.5
X2 і X4	>	>	>	>	>	>	>	>	1.5
X3 і X4	>	>	>	>	>	>	>	>	1.5

Числове значення, що визначає ступінь переваги i–го параметра над j–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5, & X_i > X_j \\ 1.0, & X_i = X_j \\ 0.5, & X_i < X_j \end{cases} \tag{4.6}$$

З отриманих оцінок складемо матрицю $A = ||a_{ij}||$

Для кожного параметра зробимо розрахунок вагомості $K_{\text{в}}^{(i)}$ за наступною формулою:

$$K_{\text{в}}^{(i)} = \frac{b_i}{\sum_{j=1}^{n} b_j} \quad \text{, де} \quad b_i = \sum_{j=1}^{n} a_{ij} \tag{4.7}$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%).На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{в}}^{(i)} = \frac{b'_i}{\sum_{j=1}^{n} b'_j} \quad \text{, де} \quad b'_i = \sum_{j=1}^{n} a_{ij} \cdot b_j \tag{4.8}$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів

Ітерації					1		2		3	
Параметри X_i	X_1	X_2	X_3	X_4	b_i	$K_{\text{в}}^{(i)}$	b_i^1	$K_{\text{в}}^{(i)}$	b_i^2	$K_{\text{в}}^{(i)}$
X_1	1.0	0.5	0.5	1.5	3.5	0.219	22.25	0.216	100	0.215
X_2	1.5	1.0	0.5	1.5	4.5	0.281	27.25	0.282	124.3	0.283
X_3	1.5	1.5	1.0	1.5	5.5	0.344	34.25	0.347	156	0.348
X_4	0.5	0.5	0.5	1.0	2.5	0.156	14.25	0.155	64.75	0.154
Всього:					16	1	98	1	445	1

4.3 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів Х2(об'єм пам'яті для збереження даних) та Х1 (швидкодія мови програмування) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра Х3 (час обробки даних) обрано не найгіршим (не максимальним), тобто це значення відповідає або варіанту а) 800 мс або варіанту б) 80мс.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується за формулою (4.9) Результати наведемо в таблиці 4.6

$$K_K(j) = \sum_{i=1}^{n} K_{\text{в}}^{(i,j)} B^{(i,j)} \tag{4.9}$$

де n - кількість параметрів, $K_{\text{в}}^{(i)}$ - коефіцієнт вагомості i-го параметра, $B^{(i,j)}$ - оцінка i-го параметра в балах.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
$F_1(X_1)$	А	11000	3.6	0.215	0.774
$F_2(X_2)$	А	16	3.4	0.283	0.962
$F_3(X_3, X_4)$	А	800	2.4	0.348	0.835
	Б	80	1	0.154	0.154

За даними з таблиці 4.6 за формулою

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + ... + K_{\text{ТУ}}[F_{zk}] \tag{4.10}$$

визначаємо рівень якості кожного з варіантів:

$$\begin{aligned} K_{K1} &= 0.774 + 0.962 + 0.835 = 2.57 \\ K_{K2} &= 0.774 + 0.962 + 0.154 = 1.89 \end{aligned}$$

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.4 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

а) Розробка проекту програмного продукту;

б) Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Загальна трудомісткість обчислюється як

$$\mathrm{T_O} = \mathrm{T_P} \cdot \mathrm{K_П} \cdot \mathrm{K_{СК}} \cdot \mathrm{K_М} \cdot \mathrm{K_{СТ}} \cdot \mathrm{K_{СТ.М}} \tag{4.11}$$

де $\mathrm{T_P}$ - трудомісткість розробки ПП; $\mathrm{K_П}$ - поправочний коефіцієнт; $\mathrm{K_{СК}}$ - коефіцієнт на складність вхідної інформації; $\mathrm{K_М}$ - коефіцієнт рівня мови програмування; $\mathrm{K_{СТ}}$ - коефіцієнт використання стандартних модулів і прикладних програм; $\mathrm{K_{СТ.М}}$ - коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $\mathrm{T_P} = 90$ людино-днів. Поправний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $\mathrm{K_П} = 1.7$. Поправний коефіцієнт, який враховує складність контролю вхідної та

вихідної інформації для всіх завдань рівний 1: $К_{СК} = 1$. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо ц за допомогою коефіцієнта $К_{СТ} = 0.8$. Тоді, за формулою (4.11), загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 90 \cdot 1.7 \cdot 0.8 = 122.4 \quad \text{людино-днів}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р = 27$ людино-днів, $К_П = 0.9$, $К_{СК} = 1$, $К_{СТ} = 0.8$:

$$T_2 = 27 \cdot 0.9 \cdot 0.8 = 19.44 \quad \text{людино-днів}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$\begin{aligned} T_1 = &\quad (122.4 + 19.44 + 4.8 + 19.44) \cdot 8 \quad = 1328.64 \text{ людино-годин} \\ T_2 = &\quad (122.4 + 19.44 + 6.91 + 19.44) \cdot 8 \quad = 1345.52 \text{ людино-годин} \end{aligned}$$

Найбільш високу трудомісткість має варіант 2. В розробці беруть участь два програмісти з окладом 7000 грн., один аналітик-програміст з окладом 9500 грн. Визначимо годинну зарплату за формулою:

$$C_h = \frac{M}{T_m t} \text{ грн} \tag{4.12}$$

де – місячний оклад працівників; T_m – кількість робочих днів тиждень; t – кількість робочих годин в день.

$$C_h = \frac{7000 + 7000 + 9500}{3 \cdot 21 \cdot 8} = 46.62 \text{ грн} \tag{4.13}$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{зп} = C_h \cdot T_i \cdot К_{КД} \tag{4.14}$$

де C_h– величина погодинної оплати праці програміста; T_i – трудомісткість відповідного завдання; $К_{КД}$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

а) $С_{зп} = 46.62 \cdot 1328.64 \cdot 1.2 = 74340.57$ грн

б) $С_{зп} = 46.62 \cdot 1345.52 \cdot 1.2 = 75285.04$ грн

Відрахування на єдиний соціальний внесок в залежності від групи професійного ризику (II клас) становить 22%:

а) $С_{від} = С_{зп} \cdot 0.22 = 74340.57 \cdot 0.22 = 16354.92$ грн

б) $С_{від} = С_{зп} \cdot 0.22 = 75285.04 \cdot 0.22 = 16562.70$ грн

Тепер визначимо витрати на оплату однієї машино-години — $С_М$

Так як одна ЕОМ обслуговує одного програміста з окладом 7000 грн., з коефіцієнтом зайнятості 0.2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot M \cdot К_З = 12 \cdot 7000 \cdot 0.2 = 16800 \text{ грн}$$

З урахуванням додаткової заробітної плати:

$$С_{зп} = С_г \cdot (1 + K) = 16800 \cdot (1 + 0.2) = 20160 \text{ грн}$$

Відрахування на єдиний соціальний внесок:

$$С_{від} = С_{зп} \cdot 0.22 = 20160 \cdot 0.22 = 4435.2 \text{ грн}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 20000 грн.

$$С_а = К_{тм} \cdot К_а \cdot Ц_{пр} = 1.15 \cdot 0.25 \cdot 20000 = 5750 \text{ грн}$$

де $К_{тм}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; $К_а$ – річна норма амортизації; $Ц_{пр}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_р = К_{тм} \cdot Ц_{пр} \cdot К_р = 1.15 \cdot 20000 \cdot 0.05 = 1150 \text{ грн}$$

де $К_р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{еф} = (Д_к - Д_в - Д_с - Д_р) \cdot t \cdot К_в = \\ = (365 - 104 - 8 - 16) \cdot 8 \cdot 0.9 = 1706.4 \text{ годин}$$

де $Д_к$ – календарна кількість днів у році; $Д_в, Д_с$ – відповідно кількість вихідних та святкових днів; $Д_р$ – кількість днів планових ремонтів устаткування; t – кількість робочих годин в день; $К_в$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ел} = Т_{еф} \cdot N_c \cdot К_з \cdot Ц_{ен} = 1706.4 \cdot 0.156 \cdot 0.9733 \cdot 2.018 = 523.82 \text{ грн}$$

де N_c – середньо-споживча потужність приладу; $К_з$ – коефіцієнтом зайнятості приладу; $Ц_{ен}$ – тариф за 1 КВт-годину електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_н = Ц_{пр} \cdot 0.67 = 20000 \cdot 0.67 = 13400 \text{ грн}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{екс} = С_{зп} + С_{від} + С_а + С_р + С_{ел} + С_н$$

$$С_{екс1} = 20160 + 4435.2 + 5750 + 1150 + 523.82 + 13400 = 45419.02 \text{ грн}$$
$$С_{екс2} = 20160 + 4435.2 + 13586 + 1150 + 472.84 + 13400 = 53204.03 \text{ грн}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{м\text{-}г1} = С_{екс}/Т_{еф} = 45419.02/1706.4 = 26.61 \text{ грн/год}$$
$$С_{м\text{-}г1} = С_{екс}/Т_{еф} = 53204.03/1706.4 = 31.17 \text{ грн/год}$$

В нашому випадку всі роботи пов’язані з розробкою програмного продукту ведуться на ЕОМ. Витрати на оплату машинного часу розраховуються за наступною формулою:

$$\text{С}_\text{м} = \text{С}_\text{м-г} \cdot T$$

В залежності від обраного варіанта реалізації, витрати на оплату машинного часу складають:

а) $\text{С}_\text{м} = 26.61 \cdot 1328.64 = 35355.11$ грн

б) $\text{С}_\text{м} = 31.17 \cdot 1345.52 = 41939.85$ грн

Накладні витрати складають 67% від заробітної плати:

$$\text{С}_\text{н} = \text{С}_\text{зп} \cdot 0.67$$

а) $\text{С}_\text{н} = 74340.57 \cdot 0.67 = 49808.18$ грн

б) $\text{С}_\text{н} = 75285.04 \cdot 0.67 = 50440.98$ грн

Отже, вартість розробки ПП за варіантами становить:

$$\text{С}_\text{пп} = \text{С}_\text{зп} + \text{С}_\text{від} + \text{С}_\text{м} + \text{С}_\text{н}$$

а) $\text{С}_\text{пп} = 74340.57 + 27335.02 + 35355.11 + 49808.18 = 186838.88$ грн

б) $\text{С}_\text{пп} = 75285.04 + 27682.31 + 41939.85 + 50440.98 = 195348.18$ грн

4.5 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$\text{К}_\text{терj} = \text{К}_\text{Кj} \,/\, \text{С}_\text{Фj} \tag{4.15}$$

а) $\text{К}_\text{тер1} = 2.57/186838.88 = 1.37 \cdot 10^{-5}$;

б) $\text{К}_\text{тер2} = 1.89/195348.18 = 0.96 \cdot 10^{-5}$;

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $\text{К}_\text{тер1} = 1.37 \cdot 10^{-5}$.

4.6 Висновки до розділу

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП.

Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $K_{тер} = 0.16 \cdot 10^{-4}$.

Цей варіант реалізації програмного продукту має такі параметри:

- мова програмування – C++;
- інтерфейс користувача у консольному варіанті;
- камера RGB;

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, непоганий функціонал та швидкодію.

ВИСНОВКИ ПО РОБОТІ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

У ході роботи були реалізовани 3 кардинально різні підходи:

а) Віднімання фону;
б) Байесовський класифікатор;
в) Аналіз відеопотоку камери глибини.

Віднімання фону та використання камери глибини дали дуже точні результати по обом критеріям, проте у загальному випадку віднімання фону саме в задачі локалізації руки на відео використовувати недоцільно через прив’язку до фону та рухомих об’єктів. Обробка відеопотоку камери глибини також дає можливість точно виділити людську руку, проте постають проблеми ціни таких камер та синхронізації depth та кольорового відеопотоків.

Реалізація байесовського класифікатора дозволяє зменшити кількість обчислень та використовувати його на веб камерах бюджетних ноутбуків.

У подальшому є сенс реалізувати запропоновані в опрацьованих джерелах методи пристосування моделі до змінних умов освітлення і це дозволить розробити доступну систему розпізнавання жестів навіть на бюджетних ноутбуках, що на данний момент ще ніким не реалізовано.

СПИСОК ЛІТЕРАТУРИ

1. Ogihara, Akio. Hand Region Extraction by Background Subtraction with Renewable Background for Hand Gesture Recognition / Akio Ogihara // International Symposium on Intelligent Signal Processing and Communications. — 2006. — Pp. 227 – 230.
2. Taylor, Michael J. — Adaptive skin segmentation via feature-based face detection. — Master's thesis, School of Computer Science, 2014.
3. Adaptive skin detection under unconstrained lightning conditions using a bigaussian model and illumination estimation / Jian-Hua Zheng, Chong-Yang Hao, Yang-Yu Fan, Xian-Yong Zang // Image Anal Stereol. — 2005. — Vol. 24, no. 3. — Pp. 21–33.
4. Георгиевич, Мурлин Алексей. Алгоритм и методы обнаружения и распознавания жестов руки на видео в режиме реального времени / Мурлин Алексей Георгиевич, Пиотровский Дмитрий Леонидович, Руденко Елизавета Алексеевна // Научный журнал КубГАУ. — 2014. — Т. 3, № 97. — С. 10.
5. Н.Х., Фан. Распознавание жестов на видеопоследовательности в режиме реального времени на основе применения метода Виолы-Джонса, алгоритма CAMSHIFT, вейвлет-преобразования и метода главних компонент / Фан Н.Х., Буй Т.Т., Спицын В.Г. // Управление, вычислительная техника и информатика. — 2013. — Pp. 103–110.
6. Kandel, Eric. Principles of Neural Science / Eric Kandel, James Schwartz, Thomas Jessell. — 4 edition. — McGraw-Hill, New York, 2000. — Pp. 577–80.
7. Eroding and Dilating. [Електронний ресурс]. — Режим доступу: http://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html.
8. Bilateral Filtering for Gray and Color Images. [Електронний ресурс]. — Режим доступу: http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MANDUCHI1/Bilateral_Filtering.html.

9. Zivkovic, Zoran. Efficient adaptive density estimation per image pixel for the task of background subtraction / Zoran Zivkovic, Ferdinand van der Heijden // Pattern recognition letters. — 2006. — Vol. 27, no. 7. — P. 773–780.
10. Zivkovic, Zoran. Improved adaptive gaussian mixture model for background subtraction / Zoran Zivkovic // Proceedings of the 17th International Conference. — 2004. — Vol. 2, no. 5. — P. 28–31.

ДОДАТОК А. **ІЛЮСТРАТИВНІ МАТЕРІАЛИ ДОПОВІДІ**

Розпізнавання людської руки на відео

студент 4-го курсу
КА-21, Одобеску Владислав

Інститут прикладного системного аналізу
керівник: доц. Дідковська Марина Віталіївна

Актуальність роботи

Сфери використання:

- Computer-human interaction systems;
- Робота з мовою жестів;
- Динамічні рухові додатки.

Актуальність роботи полягає у тому, що:

- Виводить взаємодію з ПК на новий рівень;
- У зв'язку з розвиненням сфери віртуальної да доповненої реальностей, з'являється потреба у винайдені оптимальних шляхів взаємодії користувача та системи;
- Використовується нова камера Intel Realsense F200.

Постановка задачі

Мета роботи

Розробка системи розпізнавання людської руки на відео

Об'єкт дослідження

Цифровий відеопотік з RGB чи depth камери

Предмет дослідження

Методи локалізації людської руки на відео

Постановка задачі

Поставлені задачі

- попередня обробка зображення:
  - видалення шумів
  - згладжування
- обробка RGB чи depth зображення та виділення об'єктів, що можна класифікувати як людська рука

Підходи

Видалення шумів

- морфологічні операції над зображенням (erode та dilate)
- зглажування (медіаною, Гауса)

Виділення людської руки

- Віднімання фону;
- Байесовський класифікатор;
- Обробка відеопотоку з depth камери;

Алгоритми

1. Віднімання фону (медіана, Гауса)
2. Байесовський класифікатор
    1. Класична реалізація
    2. Поправки ймовірностей
    3. Удосконалений метод навчання
3. Розпізнавання на основі сенсора глибини (Intel Realsense F200 camera)

Результати роботи

Висновки

- Проаналізовано існуючі методи попередньої обробки зображення;
- Реалізовано та проведено порівняльний аналіз трьох підходів з урахуванням обмежень на середовище роботи камер;
- Розроблена система:
  - проводить попередню обробку зображення та видаляє випадкові шуми;
  - виділяє ділянки на яких ймовірно знаходиться людська рука та трансформує зображення у бінарне;
  - проводить обробку цих ділянок.

Шляхи подальшого розвитку

- комбінування деяких підходів;
- інтеграція цієї системи у систему по розпізнаванню статичних та динамічних жестів;
- оптимізація трудомістих обчислень з метою кращого пристосування системи для обробки real-time відеопотоку.

Дякую за увагу.

ДОДАТОК Б. **ЛІСТИНГ КОДУ**

appendicies/code/background_subtraction/background_subtraction.cpp

```
#include <opencv2/opencv.hpp>

using namespace cv;
using namespace std;


int main(int argc, char* argv[])
{
    cv::VideoCapture capture(0);

    namedWindow("Frame");
    namedWindow("Background");
    namedWindow("FG Mask MOG 2");

    Mat kernel = getStructuringElement(CV_SHAPE_ELLIPSE,Size(5,5));
    Mat frame;
    Mat background;
    Mat fgMaskMOG2;
    Ptr<BackgroundSubtractorMOG2> pMOG2 = createBackgroundSubtractorMOG2(1000,
    30, false);
    pMOG2->setVarInit(15);
    pMOG2->setVarMin(4);
    pMOG2->setVarMax(75);

    unsigned n = 0;

    while( capture.read(frame) ){

        pMOG2->apply(frame, fgMaskMOG2);
        pMOG2->getBackgroundImage(background);

        dilate(fgMaskMOG2,fgMaskMOG2,kernel);
        erode(fgMaskMOG2,fgMaskMOG2,kernel);

        imshow("Frame", frame);
        imshow("Background", background);
        imshow("FG Mask MOG 2", fgMaskMOG2);

        int key = cv::waitKey(1) & 0xFF;
        if (key == 27)
            break;
        else if (key == 32){
            ++n;
```

```
            imwrite("frame"+std::to_string(n)+".png", frame);
            imwrite("hand"+std::to_string(n)+".png", fgMaskMOG2);
        }
    }
    destroyAllWindows();
    return 0;
}
```

appendicies/code/bayesian_classificator/train_from_folder.cpp

```
#include <fstream>
#include <iostream>

#include <experimental/filesystem>

#include <opencv2/opencv.hpp>

#include "classificator.hpp"


namespace fs = std::experimental::filesystem;

int main(int argc, char* argv[]) {
    assert(argc>3);

    Bayesian bayesian(256, 256);

    std::string output_folder = argv[1];
    std::string train_folder = argv[2];
    std::string filename = argv[3];

    fs::path path_to_output_folder(output_folder);
    path_to_output_folder/=filename;
    fs::path path_to_train_folder(train_folder);

    bayesian.train_from_folder<0,1>(path_to_train_folder);

    auto bayesian_m = bayesian.model();

    std::ofstream file(path_to_output_folder.string());
    bayesian_m.save_to_file(file);
    bayesian_m.gaussian_blur(5,5,1,1);
    cv::Mat color_map = bayesian_m.representation();
    cv::imwrite(path_to_output_folder.string() + ".jpg",color_map);

    cv::imshow("color map",color_map);
    cv::waitKey(0);
```

```
    return 0;
}
```

appendicies/code/bayesian_classificator/train_from_realsense.cpp

```
#include <string>

#include "classificator.hpp"

#define DEBUG

const uint16_t height = 480;
const uint16_t width = 640;

std::string color_window = "color";

auto struct_element = cv::getStructuringElement(cv::MorphShapes::MORPH_RECT,
    cv::Size(4, 4));


// in : cv::Mat of uint16_t
std::pair<double, double> find_min_max(cv::Mat in, bool ignoreZero = true) {
    double min = std::numeric_limits<double>::max(), max =
    std::numeric_limits<double>::min();
    if (ignoreZero) {
        for (int column = 0; column < in.cols; column++)
            for (int row = 0; row < in.rows; row++) {
                double elem = in.at<uint16_t>(row, column);
                if (elem > max)
                    max = elem;
                if (elem > 0 && elem < min)
                    min = elem;
            }
        // if all elements were ignored ( all zeros )
        if (min == std::numeric_limits<double>::max() && max ==
    std::numeric_limits<double>::min()){
            min = 0;
            max = 0;
        }
    }
    else {
        for (int column = 0; column < in.cols; column++)
            for (int row = 0; row < in.rows; row++) {
                double elem = in.at<uint16_t>(row, column);
                if (elem > max)
                    max = elem;
```

```
39                  if (elem < min)
                        min = elem;
                }
        }
        return std::make_pair(min, max);
44  };

    void filter_color_image_by_depth(cv::Mat &color, cv::Mat &depth, uint16_t
        max_distance) {
        assert(color.rows == depth.rows && color.cols == depth.cols);
        for (int column = 0; column < color.cols; column++)
49          for (int row = 0; row < color.rows; row++) {
                uint16_t current_depth = depth.at<uint16_t>(row, column);
                if (current_depth > max_distance | current_depth == 0)
                    color.at<cv::Vec3b>(row, column) = cv::Vec3b(0, 0, 0);
            }
54  }

    int main(int argc, char* argv[]) {
        assert(argc>2);
        std::string filepath = argv[1];
59      size_t distance = std::stoull(argv[2]);

        cv::namedWindow(color_window, 1);

        Bayesian b(256,256);
64

        rs::device *dev = nullptr;

        try {
            rs::context ctx;
69

            auto devices_count = ctx.get_device_count();
            if (devices_count == 0) {
                std::cout << "No devices found" << std::endl;
                return -1;
74          }

            dev = ctx.get_device(0);

            dev->set_option(rs::option::f200_laser_power, 15);
79          dev->set_option(rs::option::f200_motion_range, 50);
            dev->set_option(rs::option::f200_confidence_threshold, 15);
            dev->set_option(rs::option::f200_filter_option, 6);
            dev->set_option(rs::option::f200_accuracy, 3);

84          dev->enable_stream(rs::stream::depth, width, height, rs::format::z16, 30);
```

```
        dev->enable_stream(rs::stream::color, width, height, rs::format::bgr8,
    30);
        dev->start();

        cv::Mat depth_frame;
        cv::Mat color_frame;
        cv::Mat hsv_frame;

        while (true) {
            dev->wait_for_frames();
#ifdef DEBUG
            std::chrono::duration<double> duration;
            std::chrono::system_clock::time_point start =
    std::chrono::system_clock::now();
#endif
            depth_frame = cv::Mat(height, width, CV_16UC1, const_cast<void
    *>(dev->get_frame_data(rs::stream::depth)));
            color_frame = cv::Mat(height, width, CV_8UC3, const_cast<void
    *>(dev->get_frame_data(rs::stream::color_aligned_to_depth)));
#ifdef DEBUG
            duration = std::chrono::system_clock::now() - start;
            std::cout << "time elapsed for getting frames " << duration.count()
    << std::endl;
            start = std::chrono::system_clock::now();
#endif
            cv::blur(color_frame,color_frame,cv::Size(3,3));

            cv::erode(depth_frame, depth_frame, struct_element);
            //cv::dilate(depth_frame, depth_frame, struct_element);

#ifdef DIST_DEBUG
            auto minmax = find_min_max(depth_frame);
                std::cout << "min=" << minmax.first << " | max=" << minmax.second
    << std::endl;
#endif

            filter_color_image_by_depth(color_frame, depth_frame, distance);


#ifdef DEBUG
            duration = std::chrono::system_clock::now() - start;
            std::cout << "time elapsed for image processing frames " <<
    duration.count() << std::endl;
#endif

            cv::imshow(color_window, color_frame);

```

```
            // i'm rly mad cuz this function returns every run different codes
    just wtf O_O
            int key = cv::waitKey(30) & 0xFF;
            if (key == 27)
                break;
            else if (key == 32){
                cv::cvtColor(color_frame,hsv_frame,cv::COLOR_BGR2HSV);
                b.train_from_image<0,1>(hsv_frame,hsv_pixel::black);
            }

        }
        auto b_model = b.model();
        std::ofstream out(filepath);
        b_model.save_to_file(out);
        dev->stop();
    }
    catch (rs::error error) {
        std::cout << error.what() << std::endl;
        return -1;
    }
    return 0;
}
```

appendicies/code/bayesian_classificator/process_opencv.cpp

```
#include <fstream>
#include <chrono>

#include <experimental/filesystem>

#include <opencv2/opencv.hpp>

#include "classificator.hpp"

namespace fs = std::experimental::filesystem;

std::string color_window = "color";
std::string classified_window = "color";

int main(int argc, char* argv[]) {
    assert(argc>3);

    std::string filepath = argv[1];
    size_t camera = std::stoll(argv[2]);
    double treshold = std::stod(argv[3]);
```

```
23      std::ifstream file(filepath);

        BayesianModel b_m = std::move(BayesianModel::load_from_file(file));

        cv::imshow("repr",b_m.representation());
28      cv::imshow("gen␣repr",b_m.general_representation());


        cv::VideoCapture cap(camera);
        cap.set(CV_CAP_PROP_FPS, 30);
33

        cv::Mat color_frame;
        cv::Mat hsv_frame;
        cv::Mat classified_frame;
38      cv::Mat result;
        cv::Mat struct_element =
       cv::getStructuringElement(cv::MORPH_ELLIPSE,cv::Size(10,10));

        size_t n = 0;
        std::chrono::system_clock::time_point start =
       std::chrono::system_clock::now();
43      std::chrono::high_resolution_clock::time_point start_time;
        std::chrono::duration<double> duration;
        while(cap.isOpened()){
            start_time = std::chrono::high_resolution_clock::now();
            cap >> color_frame;
48          cv::cvtColor(color_frame,hsv_frame,cv::COLOR_BGR2HSV);
            duration = std::chrono::high_resolution_clock::now() - start_time;
            std::cout << duration.count() << std::endl;

            start_time = std::chrono::high_resolution_clock::now();
53          classified_frame = b_m.classify<0,1>(hsv_frame,treshold);
            duration = std::chrono::high_resolution_clock::now() - start_time;
            std::cout << duration.count() << std::endl<< std::endl<< std::endl;

            //cv::GaussianBlur(classified,result,cv::Size(5,5),0.2,0.2);
58          //cv::medianBlur(result,result,5);


            //cv::dilate(result,result,struct_element);
            //cv::erode(result,result,struct_element);
63          //cv::imshow("result",result);

            cv::imshow(color_window,color_frame);
            cv::imshow(classified_window,classified_frame);
```

```
68          int key = cv::waitKey(1) & 0xFF;
            if (key == 27)
                break;
            n++;
        }
73      duration = std::chrono::system_clock::now() - start;
        std::cout << "Average␣FPS␣" << n/duration.count() << std::endl;

        return 0;
    }
```

appendicies/code/bayesian_classificator/process_realsense.cpp

```
    #include <string>

3   #include "classificator.hpp"

    #define DEBUG

    const uint16_t height = 480;
8   const uint16_t width = 640;

    std::string color_window = "color";
    std::string classified_window = "color";

13  auto struct_element = cv::getStructuringElement(cv::MorphShapes::MORPH_RECT,
        cv::Size(4, 4));

    size_t n = 0;

    // in : cv::Mat of uint16_t
18  std::pair<double, double> find_min_max(cv::Mat in, bool ignoreZero = true) {
        double min = std::numeric_limits<double>::max(), max =
        std::numeric_limits<double>::min();
        if (ignoreZero) {
            for (int column = 0; column < in.cols; column++)
                for (int row = 0; row < in.rows; row++) {
23                  double elem = in.at<uint16_t>(row, column);
                    if (elem > max)
                        max = elem;
                    if (elem > 0 && elem < min)
                        min = elem;
28              }
            // if all elements were ignored ( all zeros )
            if (min == std::numeric_limits<double>::max() && max ==
        std::numeric_limits<double>::min()){
```

```
            min = 0;
            max = 0;
        }
    }
    else {
        for (int column = 0; column < in.cols; column++)
            for (int row = 0; row < in.rows; row++) {
                double elem = in.at<uint16_t>(row, column);
                if (elem > max)
                    max = elem;
                if (elem < min)
                    min = elem;
            }
    }
    return std::make_pair(min, max);
};

void filter_color_image_by_depth(cv::Mat &color, cv::Mat &depth, uint16_t
    max_distance) {
    assert(color.rows == depth.rows && color.cols == depth.cols);
    for (int column = 0; column < color.cols; column++)
        for (int row = 0; row < color.rows; row++) {
            uint16_t current_depth = depth.at<uint16_t>(row, column);
            if (current_depth > max_distance | current_depth == 0)
                color.at<cv::Vec3b>(row, column) = cv::Vec3b(0, 0, 0);
        }
}

int main(int argc, char* argv[]) {
    assert(argc>2);

    std::string filepath = argv[1];
    std::cout << argv[2];
    double treshold = std::stod(argv[2]);

    std::ifstream file(filepath);

    BayesianModel b_m = std::move(BayesianModel::load_from_file(file));

    cv::imshow("repr",b_m.representation());
    cv::imshow("gen␣repr",b_m.general_representation());


    cv::namedWindow(color_window, 1);
    cv::namedWindow(classified_window, 1);

    rs::device *dev = nullptr;
```

```
78      try {
            rs::context ctx;

            auto devices_count = ctx.get_device_count();
            if (devices_count == 0) {
83              std::cout << "No␣devices␣found" << std::endl;
                return -1;
            }

            dev = ctx.get_device(0);
88
            dev->enable_stream(rs::stream::color, width, height, rs::format::bgr8,
        30);
            dev->start();

            cv::Mat color_frame;
93          cv::Mat hsv_frame;
            cv::Mat classified_frame;
            cv::Mat kernel =
        cv::getStructuringElement(CV_SHAPE_ELLIPSE,cv::Size(5,5));

            while (true) {
98              dev->wait_for_frames();
    #ifdef DEBUG
                std::chrono::duration<double> duration;
                std::chrono::system_clock::time_point start =
        std::chrono::system_clock::now();
    #endif
103             color_frame = cv::Mat(height, width, CV_8UC3, const_cast<void
        *>(dev->get_frame_data(rs::stream::color)));
    #ifdef DEBUG
                duration = std::chrono::system_clock::now() - start;
                std::cout << "time␣elapsed␣for␣getting␣frames␣" << duration.count()
        << std::endl;
                start = std::chrono::system_clock::now();
108 #endif

                cv::cvtColor(color_frame,hsv_frame,cv::COLOR_BGR2HSV);
                classified_frame = std::move(b_m.classify<0,1>(hsv_frame,treshold));
                cv::dilate(classified_frame,classified_frame,kernel);
113             cv::erode(classified_frame,classified_frame,kernel);

    #ifdef DEBUG
                duration = std::chrono::system_clock::now() - start;
                std::cout << "time␣elapsed␣for␣image␣processing␣" << duration.count()
        << std::endl;
```

```
#endif


            cv::imshow(color_window, color_frame);
            cv::imshow(classified_window, classified_frame);

            // i'm rly mad cuz this function returns every run different codes just wtf O_O
            int key = cv::waitKey(30) & 0xFF;
            if (key == 27)
                break;
            else if (key == 32) {
                ++n;
                cv::imwrite("color" + std::to_string(n) + ".png", color_frame);
                cv::imwrite("mask" + std::to_string(n) + ".png", classified_frame);
            }
        }
        dev->stop();
    }
    catch (rs::error error) {
        std::cout << error.what() << std::endl;
        return -1;
    }
    return 0;
}
```

appendicies/code/bayesian_classificator/filter_model.cpp

```
#include <fstream>
#include <iostream>

#include <opencv2/opencv.hpp>

#include "classificator.hpp"

int main(int argc, char* argv[]){
    assert(argc>4);
    std::string in_filepath = argv[1];
    std::string out_filepath = argv[2];
    std::string command = argv[3];
    std::vector<std::string> params;
    for(size_t i = 4; i < argc; ++i)
        params.push_back(std::string(argv[i]));

    BayesianModel b_m = std::move(BayesianModel::load_from_file(in_filepath));

```

```
    cv::imshow("before repr", b_m.representation());
    cv::imshow("before gen_repr", b_m.general_representation());

    if(command == "normalize") {
        b_m.normalize_probabilities();
    }
    else if(command == "threshold"){

    b_m.threshold_small_probabilities_blobs(std::stoull(params[0]),std::stod(params[1]));
    }
    else if(command == "filter"){

    b_m.filter_random_probabilities(std::stoull(params[0]),std::stod(params[1]));
    }
    else if(command == "erode"){
        b_m.erode(std::stoull(params[0]),std::stoull(params[1]));
    }
    else if(command == "dilate"){
        b_m.dilate(std::stoull(params[0]),std::stoull(params[1]));
    }
    else if(command == "blur"){
        b_m.blur(std::stoull(params[0]),std::stoull(params[1]));
    }
    else if(command == "gblur") {

    b_m.gaussian_blur(std::stoull(params[0]),std::stoull(params[1]),std::stod(params[2]),st
    }
    else if(command == "dec") {
        auto result =
    b_m.decomposition(std::stod(params[0]),std::stod(params[1]));
        for (size_t i = 0; i < result.size(); ++i)
            result[i].save_to_file(out_filepath+std::to_string(i));
    }

    b_m.save_to_file(out_filepath);

    cv::imshow("after repr", b_m.representation());
    cv::imshow("after gen_repr", b_m.general_representation());
    cv::waitKey(0);
}
```

appendicies/code/bayesian_classificator/show_model_representation.cpp

```
#include <fstream>

#include <opencv2/opencv.hpp>
```

```
#include "classificator.hpp"

int main(int argc, char* argv[]){
    assert(argc>1);
    std::string filepath = argv[1];
    BayesianModel b_m = std::move(BayesianModel::load_from_file(filepath));
    cv::imwrite("repr.png",b_m.representation());
    auto reprep = std::move(b_m.general_representation());
    cv::imwrite("gen_repr.png",reprep);
    cv::waitKey(0);
}
```

appendicies/code/realsense/realsense.cpp

```
#include <iostream>
#include <chrono>

#include <opencv2/opencv.hpp>
#include <librealsense/rs.hpp>


#define DEBUG

#undef DIST_DEBUG

rs::device *dev = nullptr;


const uint16_t height = 480;
const uint16_t width = 640;

auto struct_element = cv::getStructuringElement(cv::MorphShapes::MORPH_RECT,
    cv::Size(4, 4));

namespace options {
    uint16_t current_distance;
};

namespace option_window {
    std::string name = "options";
    std::string depth = "depth";

    void depth_update(int value, void *ptr) {
        options::current_distance = uint16_t(value);
    }

    void create() {
```

```
        cv::namedWindow(option_window::name, cv::WINDOW_NORMAL);
        cv::createTrackbar(option_window::depth, option_window::name, nullptr,
    30000, option_window::depth_update,
                           nullptr);
    }
}

// in : cv::Mat of uint16_t
std::pair<double, double> find_min_max(cv::Mat in, bool ignoreZero = true) {
    double min = std::numeric_limits<double>::max(), max =
    std::numeric_limits<double>::min();
    if (ignoreZero) {
        for (int column = 0; column < in.cols; column++)
            for (int row = 0; row < in.rows; row++) {
                double elem = in.at<uint16_t>(row, column);
                if (elem > max)
                    max = elem;
                if (elem > 0 && elem < min)
                    min = elem;
            }
        // if all elements were ignored ( all zeros )
        if (min == std::numeric_limits<double>::max() && max ==
    std::numeric_limits<double>::min()) {
            min = 0;
            max = 0;
        }
    }
    else {
        for (int column = 0; column < in.cols; column++)
            for (int row = 0; row < in.rows; row++) {
                double elem = in.at<uint16_t>(row, column);
                if (elem > max)
                    max = elem;
                if (elem < min)
                    min = elem;
            }
    }
    return std::make_pair(min, max);
};

void filter_color_image_by_depth(cv::Mat &color, cv::Mat &depth, uint16_t
    max_distance) {
    assert(color.rows == depth.rows && color.cols == depth.cols);
    for (int column = 0; column < color.cols; column++)
        for (int row = 0; row < color.rows; row++) {
            uint16_t current_depth = depth.at<uint16_t>(row, column);
            if (current_depth > max_distance | current_depth == 0)
```

```
                color.at<cv::Vec3b>(row, column) = cv::Vec3b(0, 0, 0);
            }
        }

int main() {

    option_window::create();

    try {
        rs::context ctx;

        auto devices_count = ctx.get_device_count();
        if (devices_count == 0) {
            std::cout << "No devices found" << std::endl;
            return -1;
        }

        dev = ctx.get_device(0);

        dev->set_option(rs::option::f200_laser_power, 15);
        dev->set_option(rs::option::f200_motion_range, 100);
        dev->set_option(rs::option::f200_confidence_threshold, 15);
        dev->set_option(rs::option::f200_filter_option, 6);
        dev->set_option(rs::option::f200_accuracy, 3);

        dev->enable_stream(rs::stream::depth, width, height, rs::format::z16, 30);
        dev->enable_stream(rs::stream::color, width, height, rs::format::bgr8,
    30);
        dev->start();

        cv::Mat depth_frame;
        cv::Mat color_frame;
        cv::Mat color_frame_filtered;

        size_t n = 0;

        while (true) {
            dev->wait_for_frames();
#ifdef DEBUG
            std::chrono::duration<double> duration;
            std::chrono::system_clock::time_point start =
    std::chrono::system_clock::now();
#endif
            depth_frame = cv::Mat(height, width, CV_16UC1, const_cast<void
    *>(dev->get_frame_data(rs::stream::depth)));
            color_frame_filtered = cv::Mat(height, width, CV_8UC3,
```

```
                                    const_cast<void
    *>(dev->get_frame_data(rs::stream::color_aligned_to_depth)));
            color_frame_filtered.copyTo(color_frame);
#ifdef DEBUG
            duration = std::chrono::system_clock::now() - start;
            std::cout << "time elapsed for getting frames " << duration.count()
    << std::endl;
            start = std::chrono::system_clock::now();
#endif
            cv::erode(depth_frame, depth_frame, struct_element);
            cv::dilate(depth_frame, depth_frame, struct_element);

#ifdef DIST_DEBUG
            auto minmax = find_min_max(depth_frame);
                std::cout << "min=" << minmax.first << " | max=" << minmax.second
    << std::endl;
#endif

            filter_color_image_by_depth(color_frame_filtered, depth_frame,
    options::current_distance);


#ifdef DEBUG
            duration = std::chrono::system_clock::now() - start;
            std::cout << "time elapsed for image processing frames " <<
    duration.count() << std::endl;
#endif

            cv::imshow("depth", depth_frame);
            cv::imshow("color", color_frame);
            cv::imshow("filtered", color_frame_filtered);

            // i'm rly mad cuz this function returns every run different codes
    just wtf O_O
            int key = cv::waitKey(30) & 0xFF;
            if (key == 27)
                break;
            else if (key == 32) {
                ++n;
                cv::imwrite("color" + std::to_string(n) + ".png", color_frame);
                cv::imwrite("filtered" + std::to_string(n) + ".png",
    color_frame_filtered);
                cv::imwrite("depth" + std::to_string(n) + ".png", depth_frame);
            }
        }

    }
```

```
159     catch (rs::error error) {
            std::cout << error.what() << std::endl;
            return -1;
        }
        dev->stop();
164     return 0;
}
```

Заголовочні файли

appendicies/code/bayesian_classificator/continuous_matrix.hpp

```
#ifndef CONTINUOUS_MATRIX_HPP
#define CONTINUOUS_MATRIX_HPP

#include <iostream>
5 #include <cstring>

#include <boost/serialization/version.hpp>
#include <boost/serialization/split_member.hpp>


10
template<class T,class S = size_t>
class continuous_matrix {

    friend class boost::serialization::access;

15
    typedef T value_type;

    typedef T* pointer;
    typedef const pointer const_pointer;

20
    typedef value_type& reference;
    typedef const reference const_reference;

    typedef S size_type;

25
    typedef pointer iterator;
    typedef const_pointer const_iterator;

    size_type _rows;
30  size_type _cols;
    size_type _total;
    pointer _data;

    template<class Archive>
```

```
    void save(Archive & ar, const unsigned int version) const
    {
        ar & _rows;
        ar & _cols;
        ar & _total;
        auto elem = cbegin(), last = cend();
        for(;elem != last; elem++)
            ar & *elem;
    }

    template<class Archive>
    void load(Archive & ar, const unsigned int version)
    {
        ar & _rows;
        ar & _cols;
        ar & _total;

        if (_data != nullptr ){
            delete[] _data;
        }
        _data = new value_type[_total];

        auto elem = cbegin(), last = cend();
        for(;elem != last; elem++)
            ar & *elem;
    }

    BOOST_SERIALIZATION_SPLIT_MEMBER()
public:

    continuous_matrix() : _data(nullptr), _rows(0), _cols(0), _total(0) { }

    continuous_matrix(size_type rows, size_type cols, value_type value = 0) :
    _rows(rows), _cols(cols), _total(rows * cols) {
        _data = new value_type[_total];
        std::fill(begin(), end(),value);
        for(auto c = begin(); c!=end(); c++)
            assert(*c == value);
    }

    continuous_matrix(size_type rows, size_type cols, const_pointer data) :
    _rows(rows), _cols(cols), _total(rows * cols) {
        _data = new value_type[_total];
        std::memcpy(_data, data, _total * sizeof(value_type));
    }
```

```
    continuous_matrix(const continuous_matrix &matrix) : _rows(matrix._rows),
_cols(matrix._cols), _total(matrix._rows * matrix._cols) {
        _data = new value_type[_total];
        std::memcpy(_data, matrix._data, _total * sizeof(value_type));
    }

    continuous_matrix(continuous_matrix && matrix) : _rows(matrix._rows),
_cols(matrix._cols), _total(matrix._total) {
        _data = matrix._data;
        matrix._rows = matrix._cols = matrix._total = 0;
        matrix._data = nullptr;
    }

    continuous_matrix& operator=(continuous_matrix&& matrix){
        _rows = matrix._rows;
        _cols = matrix._cols;
        _total = matrix._total;
        _data = matrix._data;
        matrix._rows = matrix._cols = matrix._total = 0;
        matrix._data = nullptr;
    }

    inline reference operator()(size_type row, size_type col) {
        return _data[row * _cols + col];
    }

    inline const_reference operator()(size_type row, size_type col) const{
        return _data[row * _cols + col];
    }

    inline reference get(size_type row, size_type col) const {
        return _data[row * _cols + col];
    }

    inline void set(size_type row, size_type col, const_reference value) {
        _data[row * _cols + col] = value;
    }

    void increment(size_type row, size_type col){
        _data[row * _cols + col]++;
    }

    ~continuous_matrix() {
        if (_data != nullptr)
            delete[] _data;
    }
```

```
    inline size_type rows() const{
125         return _rows;
    }

    inline size_type cols() const{
        return _cols;
130     }

    inline pointer data() {
        return _data;
    }
135
    inline pointer data() const {
        return _data;
    }

140 inline iterator begin() {
        return _data;
    }

    inline iterator end() {
145         return _data+_total;
    }

    inline const_iterator cbegin() const {
        return _data;
150 }

    inline const_iterator cend() const {
        return _data+_total;
    }
155
    void resize(size_type rows, size_type cols){
        _rows = rows;
        _cols = _cols;
        _total = _rows*_cols;
160     if ( _data != nullptr )
            delete[] _data;
        _data = new value_type[_total];
    }

165 size_type count_non_zero(){
        size_type non_zero = 0;
        for(auto c = begin(); c!=end(); c++)
            if(*c != 0)
                non_zero++;
170     return non_zero;
```

```
    }


};

typedef continuous_matrix<bool>     bmatrix;
typedef continuous_matrix<uint8_t>  umatrix;
typedef continuous_matrix<size_t>   smatrix;
typedef continuous_matrix<double>   dmatrix;

#endif //CONTINUOUS_MATRIX_HPP
```

appendicies/code/bayesian_classificator/general_defines.hpp

```
#ifndef GENERAL_DEFINES
#define GENERAL_DEFINES

#include <opencv2/opencv.hpp>
#include <librealsense/rs.hpp>

typedef cv::Vec3b vec3b;

template<class UIntegerType>
struct point{
    UIntegerType x,y;
    point() = default;
    point(const point& p) = default;
    point(point&& p) = default;
    point& operator=(const point&) = default;
    point(const UIntegerType& x_, const UIntegerType& y_):x(x_),y(y_) {};
};

typedef point<size_t> spoint;

namespace gray_pixel{
    const uint8_t black = 0;
    const uint8_t white = 255;
};

namespace rgb_pixel{
    const vec3b black(0,0,0);
    const vec3b white(255,255,255);
};

namespace hsv_pixel{
    const vec3b black(0,0,0);
```

```
    const vec3b white(0,0,255);
};

#endif //GENERAL_DEFINES
```

appendicies/code/bayesian_classificator/classificator.hpp

```
#ifndef CLASSIFICATOR
#define CLASSIFICATOR

#include <vector>
#include <algorithm>
#include <string>
#include <utility>
#include <fstream>
#include <queue>

#include <experimental/filesystem>

#include <boost/archive/text_oarchive.hpp>
#include <boost/archive/text_iarchive.hpp>

#include "continuous_matrix.hpp"
#include "general_defines.hpp"


namespace fs = std::experimental::filesystem;

spoint push_wave(const dmatrix& probs, bmatrix& visited, dmatrix& out, spoint
    start);

class Bayesian;
class BayesianModel;


BayesianModel model_union(const BayesianModel& bm1, const BayesianModel& bm2);


class BayesianModel{
    friend boost::serialization::access;
    friend BayesianModel model_union(const BayesianModel& bm1, const
    BayesianModel& bm2);

    size_t first_dim;
    size_t second_dim;
    dmatrix probs;
```

```
39      // used for serialization
        template<class Archive>
        inline void serialize(Archive & ar, const unsigned int version){
            ar & first_dim;
            ar & second_dim;
44          ar & probs;
        }

        BayesianModel() {};
    public:
49
        static BayesianModel load_from_file(std::ifstream & stream);
        static BayesianModel load_from_file(const std::string& path);

        BayesianModel(BayesianModel&&);
54      BayesianModel(const smatrix& counts);
        BayesianModel(dmatrix&& prob_in);

        void save_to_file(std::ofstream &stream);
        void save_to_file(const std::string& path);
59
        cv::Mat representation();

        cv::Mat general_representation();

64      /*
         *
         */
        template <uint first,uint second>
        cv::Mat classify(const cv::Mat& in, double threshold){
69          cv::Mat out(in.size(),CV_8UC1);
            auto i = in.begin<vec3b>(), end = in.end<vec3b>();
            auto o = out.begin<uint8_t>();
            for (; i != end; ++i,++o) {
                const vec3b& pixel = *i;
74              if (probs(pixel[first],pixel[second]) > threshold)
                    *o = gray_pixel::white;
                else
                    *o = gray_pixel::black;
            }
79          return out;
        };

        void normalize_probabilities(){
            auto coef = 1.0 / *std::max_element(probs.cbegin(),probs.cend());
84          auto curr = probs.begin(), last = probs.end();
            for(; curr!=last; curr++)
```

```
            *curr*=coef;
        }

89      void threshold_small_probabilities_blobs(uint8_t size, double prob_threshold){
            cv::Mat process(first_dim,second_dim,CV_64FC1,probs.data());
            cv::Mat mask;

            cv::blur(process,mask,cv::Size(size,size));
94          cv::inRange(mask,0,prob_threshold,mask);

            process.setTo(.0,mask);
        }

99      /*
         * size — uint8_t size of elliptic kernel with a=b=size
         *
         * percents — double in range of [0,1] part of the kernel area to accept as
    not noise probability
         * adequate tip is to use percents not less than 0.25 ( 1/4 of the circle )
    because 0.5 may erase
104      * edge probilities of big probabilities blobs
         */
        void filter_random_probabilities(uint8_t size, double part){
            if (size > 16) {
                std::cerr << "filter_random_probabilities:␣size␣>␣16" << std::endl;
109             return;
            }
            cv::Mat kernel = cv::getStructuringElement(cv::MORPH_ELLIPSE,
    cv::Size(size,size));
            cv::Mat process(first_dim,second_dim,CV_64FC1,probs.data());
            uint8_t lower_bound = cv::countNonZero(kernel) * part;
114

            continuous_matrix<uint8_t> cmm(first_dim,second_dim);
            auto p_b = probs.cbegin(),p_e = probs.cend();
            auto cmm_b = cmm.begin();
            for(;p_b!=p_e;++p_b,++cmm_b) {
119             if (*p_b > 0) {
                    *cmm_b = 1;
                }
            }
            cv::Mat mask(first_dim,second_dim,CV_8U,cmm.data());
124         cv::filter2D(mask,mask,-1,kernel);
            cv::inRange(mask,0,lower_bound,mask);

            process.setTo(.0,mask);
        }
129
```

```
void erode(uint8_t a,uint8_t b){
    cv::Mat process(first_dim,second_dim,CV_64FC1,probs.data());
    cv::Mat kernel = cv::getStructuringElement(cv::MORPH_ELLIPSE,
cv::Size(a,b));
    cv::erode(process,process,kernel);
}

void dilate(uint8_t a,uint8_t b){
    cv::Mat process(first_dim,second_dim,CV_64FC1,probs.data());
    cv::Mat kernel = cv::getStructuringElement(cv::MORPH_ELLIPSE,
cv::Size(a,b));
    cv::dilate(process,process,kernel);
}

void blur(uint8_t a,uint8_t b){
    cv::Mat process(first_dim,second_dim,CV_64FC1,probs.data());
    cv::blur(process,process,cv::Size(a,b));
}

void gaussian_blur(size_t a ,size_t b , double sigmax, double sigmay){
    cv::Mat process(first_dim,second_dim,CV_64FC1,probs.data());
    cv::GaussianBlur(process,process,cv::Size(a,b),sigmax,sigmay);
}

std::vector<BayesianModel> decomposition(double min_area, double
min_distance){
    auto params = cv::SimpleBlobDetector::Params();
    params.minDistBetweenBlobs = min_distance;
    params.filterByInertia = false;
    params.filterByConvexity = false;
    params.filterByColor = false;
    params.filterByCircularity = false;
    params.filterByArea = true;
    params.minArea = min_area;
    params.maxArea = std::numeric_limits<double>::max();
    auto blob_detector = cv::SimpleBlobDetector::create(params);

    auto gen_repr = std::move(this->general_representation());

    std::vector< cv::KeyPoint > keypoints;
    blob_detector->detect(gen_repr,keypoints);


cv::drawKeypoints(gen_repr,keypoints,gen_repr,cv::Scalar::all(-1),cv::DrawMatchesFlags
    cv::imshow("LALALA", gen_repr);

    std::vector<BayesianModel> result;
```

```
            result.reserve(keypoints.size());

            bmatrix visited(first_dim,second_dim);

            std::cout << keypoints.size() << std::endl;

            for(auto & keypoint : keypoints){
                auto x = keypoint.pt.x,y=keypoint.pt.y;
                dmatrix probs_blob(first_dim,second_dim);
                push_wave(probs,visited,probs_blob,spoint(y,x));
                auto current_model = BayesianModel(std::move(probs_blob));
                result.push_back(std::move(current_model));
            }
            //for (auto & elem : result)
            //    elem.normalize_probabilities();
            return result;
        }

        size_t non_zero(){
            return probs.count_non_zero();
        }
};

class Bayesian{
    friend boost::serialization::access;

    size_t first_dim;
    size_t second_dim;
    smatrix counts;

    // used for serialization
    template<class Archive>
    void serialize(Archive & ar, const unsigned int version){
        ar & first_dim;
        ar & second_dim;
        ar & counts;
    }

    Bayesian() {};
public:

    static Bayesian load_from_file(std::ifstream & stream);
    static Bayesian load_from_file(const std::string& path);

    Bayesian(Bayesian&&) = default;
    Bayesian(const size_t first, const size_t second) : first_dim(first),
    second_dim(second), counts(first,second) {};
```

```
219
        void save_to_file(std::ofstream & stream);
        void save_to_file(const std::string& path);

        template <uint first,uint second>
224     void train_from_folder(const fs::path &path, uchar color_space_conversion =
        cv::COLOR_BGR2HSV) {
            std::vector<std::pair<fs::path, fs::path>> train_set;
            {
                fs::directory_iterator entries(path);
                train_set.resize(std::distance(fs::begin(entries), fs::end(entries))
        / 2);
229         }
            for (auto &entry : fs::directory_iterator(path)) {
                std::stringstream str_stream(entry.path().filename());
                std::string temp;
                std::getline(str_stream, temp, '_');
234             size_t n = std::stoi(temp);
                char c = str_stream.get();
                if (c == 'i')
                    train_set[n - 1].first = std::move(const_cast<fs::path
        &>(entry.path()));
                else
239                 train_set[n - 1].second = std::move(const_cast<fs::path
        &>(entry.path()));
            }
            for (auto &set : train_set) {
                cv::Mat img = cv::imread(set.first.string());
                cv::Mat mask = cv::imread(set.second.string());
244             cv::cvtColor(img, img, color_space_conversion);
                auto i = img.begin<vec3b>(), end = img.end<vec3b>();
                auto m = mask.begin<vec3b>();
                for (; i != end; ++i, ++m) {
                    vec3b &i_pixel = *i, m_pixel = *m;
249                 if (m_pixel != rgb_pixel::white)
                        counts.increment(i_pixel[first],i_pixel[second]);
                }
            }
        };
254

        template <uint first,uint second>
        void train_from_image(const cv::Mat& image, const vec3b& ignore_color) {
            auto i = image.begin<vec3b>(), end = image.end<vec3b>();
            for (; i != end; ++i) {
259             const vec3b& i_pixel = *i;
                if (i_pixel != ignore_color)
                    counts.increment(i_pixel[first], i_pixel[second]);
```

```
        }
    };

    BayesianModel model() const {
        return BayesianModel(counts);
    }
};



#endif
```

Файли реалізації

appendicies/code/bayesian_classificator/continuous_matrix.cpp

appendicies/code/bayesian_classificator/classificator.cpp

```
#include "classificator.hpp"

spoint push_wave(const dmatrix& probs, bmatrix& visited, dmatrix& out, spoint
    start){
    std::queue<spoint> curr_q;
    auto height = probs.rows()-1, width = probs.cols()-1;
    curr_q.push(start);
    spoint max_prob = start;
    double max_prob_value = probs(start.x, start.y);
    visited(start.x, start.y) = true;
    while (!curr_q.empty()) {
        spoint current_point = std::move(curr_q.front());
        curr_q.pop();
        auto& x = current_point.x, y = current_point.y;

        auto prob = probs(x,y);
        out(x, y) = prob;

        if (x > 1 && !visited(x - 1, y) && probs(x - 1, y) > 0) {
            curr_q.push(std::move(spoint(x - 1, y)));
            visited(x - 1, y) = true;
        }
        if (x < width && !visited(x + 1, y) && probs(x + 1, y) > 0) {
            curr_q.push(std::move(spoint(x + 1, y)));
            visited(x + 1, y) = true;
```

```
            }
            if (y > 1 && !visited(x, y — 1) && probs(x, y — 1) > 0) {
                curr_q.push(std::move(spoint(x, y — 1)));
                visited(x, y — 1) = true;
            }
            if (y < height && !visited(x, y + 1) && probs(x, y + 1) > 0) {
                curr_q.push(std::move(spoint(x, y + 1)));
                visited(x, y + 1) = true;
            }

            if (prob > max_prob_value) {
                max_prob_value = prob;
                max_prob = std::move(current_point);
            }
        }
        return max_prob;
    }

    BayesianModel BayesianModel::load_from_file(std::ifstream &stream) {
        boost::archive::text_iarchive in_archive(stream);
        BayesianModel b_m;
        in_archive >> b_m;
        return b_m;
    }

    BayesianModel BayesianModel::load_from_file(const std::string& path) {
        std::ifstream stream(path);
        boost::archive::text_iarchive in_archive(stream);
        BayesianModel b_m;
        in_archive >> b_m;
        return b_m;
    }


    BayesianModel::BayesianModel(const smatrix& counts) : first_dim(counts.rows()),
        second_dim(counts.cols()), probs(first_dim,second_dim) {
        size_t m = *std::max_element(counts.cbegin(), counts.cend());
        double multiplier = 1.0 / m;
        for (size_t i = 0; i < first_dim; ++i)
            for (size_t j = 0; j < second_dim; ++j)
                probs(i,j) = counts.get(i,j) * multiplier;
    }

    BayesianModel::BayesianModel(BayesianModel&& model) : first_dim(model.first_dim),
        second_dim(model.second_dim){
        std::swap(probs,model.probs);
    }
```

```
BayesianModel::BayesianModel(dmatrix&& probs_in) : first_dim(probs_in.rows()),
    second_dim(probs_in.cols()){
    probs = std::move(probs_in);
}

void BayesianModel::save_to_file(std::ofstream &stream) {
    boost::archive::text_oarchive out_archive(stream);
    out_archive << *this;
}

void BayesianModel::save_to_file(const std::string& path) {
    std::ofstream stream(path);
    boost::archive::text_oarchive out_archive(stream);
    out_archive << *this;
}

cv::Mat BayesianModel::representation() {
    cv::Mat result(cv::Size(first_dim,second_dim),CV_8UC1);
    for(size_t i = 0; i<first_dim; ++i)
        for(size_t j = 0; j<second_dim; ++j)
            result.at<uint8_t>(i,j) = probs(i,j)*255;
    return result;
}

cv::Mat BayesianModel::general_representation() {
    cv::Mat result(cv::Size(first_dim,second_dim),CV_8UC1);
    for(size_t i = 0; i<first_dim; ++i)
        for(size_t j = 0; j<second_dim; ++j)
            if (probs(i,j)> 0)
                result.at<uint8_t>(i,j) = 255;
    return result;
}


Bayesian Bayesian::load_from_file(std::ifstream &stream) {
    boost::archive::text_iarchive in_archive(stream);
    Bayesian b;
    in_archive >> b;
    return b;
}

Bayesian Bayesian::load_from_file(const std::string& path) {
    std::ifstream stream(path);
    boost::archive::text_iarchive in_archive(stream);
    Bayesian b;
    in_archive >> b;
```

```
    return b;
}

void Bayesian::save_to_file(std::ofstream &stream) {
    boost::archive::text_oarchive out_archive(stream);
    out_archive << *this;
}

void Bayesian::save_to_file(const std::string& path) {
    std::ofstream stream(path);
    boost::archive::text_oarchive out_archive(stream);
    out_archive << *this;
}

template <class BinaryFunction>
BayesianModel model_union(const BayesianModel& bm1, const BayesianModel& bm2,
    BinaryFunction func = std::max){
    assert(bm1.first_dim==bm2.first_dim && bm1.second_dim == bm2.second_dim);
    BayesianModel result(bm1.first_dim,bm1.second_dim);

    auto bm1_b = bm1.probs.cbegin(),bm1_e = bm1.probs.cend();
    auto bm2_b = bm2.probs.cbegin(), res_b = result.probs.begin();

    for(;bm1_b!=bm1_e;++bm1_b,++bm2_b,++res_b)
        *res_b == func(*bm1_b,*bm2_b);
    return result;
}
```