ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗОВАНИХ СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Трансформація відео до тексту з використанням OpenCV»

на здобуття освітнього ступеня магістра
зі спеціальності 126 Інформаційні системи та технології
(код, найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Владислав Вислобіцький
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав:
здобувач вищої освіти Владислав Вислобіцький
група ІСДМ-61

Керівник:
науковий ступінь, Вікторія Шкапа
вчене звання доцент

Рецензент:
науковий ступінь, _____
вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення автоматизованих системOutfit

Ступінь вищої освіти Магістр

Спеціальність Інформаційні системи та технології

Освітньо-професійна програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедрою ІПЗАС

_____ Каміла СТОРЧАК
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Вислобіцькому Владиславу Андрійовичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Трансформація відео до тексту з використанням OpenCV
керівник кваліфікаційної роботи Вікторія Шкапа, доцент,

(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «19» 10.2023р. №145

2. Строк подання кваліфікаційної роботи «29» грудня 2023р.
3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, OpenCV, Tesseract, CMake, g++, C++, Kate
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   Дослідження домену комп'ютерного зору
   Аналіз OpenCV
   Аналіз інтсрументів розробки трансформації відео до тексту
   Розробка трансформації відео до тексту

5. Перелік графічного матеріалу: *презентація*

6. Дата видачі завдання «19» жовтня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	19.10-05.11.23	виконав
2	Дослідження домену комп'ютерного зору	05.11-12.11.23	виконав
3	Аналіз OpenCV	13.11-19.11.23	виконав
4	Аналіз інструментів розробки трансформації відео до тексту	20.11-25.11.23	виконав
5	Реалізації алгоритму трансформації відео до тексту	27.11-21.12.23	виконав
6	Розробка демонстраційних матеріалів	21.12-29.12.23	виконав

Здобувач вищої освіти _____ Владислав Вислобіцький
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Вікторія Шкапа
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

ПОДАННЯ
ГОЛОВІ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ
ЩОДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ
на здобуття освітнього ступеня магістра

Направляється здобувач Вислобіцький В.А. до захисту кваліфікаційної роботи
(прізвище та ініціали)
за спеціальністю 126 Інформаційні системи та технлогії
(код, найменування спеціальності)
освітньо-професійної програми 126 Інформаційні системи та технлогії
(назва)
на тему: «Трансформація відео до тексту з використанням OpenCV». Кваліфікаційна робота і рецензія додаються.

Директор ННІІТ _____ Бондарчук А.П.
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Висновок керівника кваліфікаційної роботи

Здобувач(ка) дослідив та описав сферу комп'ютерного зору. Розглянув та описав можливості та функціонал OpenCV. Описав інструменти розробки для тансформації відео до тексту. Розробив та реалізував алгоритм для трансформації відео до тексту з використанням OpenCV.

Все це дозволяє оцінити виконану кваліфікаційну роботу здобувача Вислобіцького Владислава Андрійовича на оцінку «відмінно» та присвоїти йому кваліфікацію магістр з інформаційних систем та технологій.
Керівник кваліфікаційної роботи _____ Шкапа В.В.
(підпис) *(Ім'я, ПРІЗВИЩЕ)*
«____» _____ 20___ року

Висновок кафедри про кваліфікаційну роботу

Кваліфікаційна робота розглянута. Здобувач Вислобіцький В.А. допускається до захисту даної роботи в Екзаменаційній комісії.
Завідувач кафедрою ІПЗАС _____ Сторчак І
(назва)* *(підпис)* *(Ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 92 стор., 52 рис., 30 джерел.

Мета роботи – реалізувати трансформацію відео до тексту.

Об'єкт дослідження – технології комп'ютерного зору.

Предмет дослідження – функціонал та характеристики технологій комп'ютерного зору.

Короткий зміст роботи: у роботі досліджено домен комп'ютерного зору. Проаналізовано основні можливості роботи OpenCV. Проаналізовані інструменти розробки. Побудовано та реалізовано трансформацію відео до тексту з використанням OpenCV мовою програмування C++.

КЛЮЧОВІ СЛОВА: OPENCV, TESSERACT, LEPTONICA, CMAKE, G++, C++, KATE, TERMINAL, BUILD, INSTALL, SOURCE CODE, REPOSITORY

ABSTRACT

Text part of the master's qualification work: 92 pages, 52 pictures, 30 sources.

The purpose of the work – Implement video to text conversion.

Object of research – Computer vision technology.

Subject of research – Functional capabilities and characteristics of computer vision technology.

Summary of the work – The work examines the domain of computer vision. The main capabilities of OpenCV have been analyzed. The main capabilities of development instruments have been analyzed as well. Built and implemented the transformation of video to text using OpenCV in the C++ programming language.

KEYWORDS: OPENCV, TESSERACT, LEPTONICA, CMAKE, G++, C++, KATE, TERMINAL, BUILD, INSTALL, SOURCE CODE, REPOSITORY

ВІДГУК РЕЦЕНЗЕНТА
на кваліфікаційну магістерську роботу

здобувача вищої освіти Вислобіцький Владислав Андрійович
(прізвище, ім'я, по батькові)

на тему «Трансформація відео до тексту з використанням OpenCV»

Актуальність.
Робота є актуальною у зв'язку з швидким розвитком та широким росповсюдженням використання технологій комп'ютерного зору у різноманітних сферах. OpenCV є ефективною та надійною технологією для створення різної складності рішень, які використовуються у багатьох сферах.

Позитивні сторони.
1. Зміст роботи повністю відповідає завданню, а поставлені задачі виконані у повному обсязі.
2. У роботі досліджені характеристики та функціональні можливості OpenCV .
3. Розглянута взаємодія OpenCV з Tesseract.

Недоліки.
1. Не розглянуті недоліки OpenCV.
2. Не розглянуті аналоги OpenCV.

Відзначені зауваження не впливають на загальну позитивну оцінку кваліфікаційної магістерської роботи.

Висновок: *кваліфікаційна магістерська робота заслуговує оцінку "відмінно", а здобувач Вислобіцький Владислав Андрійович заслуговує присвоєння кваліфікації:* магістр з інформаційних систем та технологій.

Рецензент: _____ _____
науковий ступінь, вчене звання ***підпис*** ***Ім'я, ПРІЗВИЩЕ***

ЗМІСТ

ВСТУП

Обгрунтування вибору теми та її актуальність: у зв'язку зі стрімким розвитком та розповсюдженням технологій штучного інтелекту дослідження проблем в області комп'ютерного зору є важливою місією, яка в результаті приносить користь у вигляді технологій, які суттєво полегшують та покращують різноманітні аспекти життя багатьох.

Ступінь вивчення проблеми: дослідження передової технології галузі комп'ютерного зору OpenCV призводить до покращення старих та створення нових алгоритмів, які використовуються для рішення різних задач. На основі OpenCV можливо створювати крос-платформні, ефективні та надійні рішення, які можна використовувати у різномантіних галзуях, як медецина, безпека, робототехніка. Трансформація відео до тексту з використанням OpenCV не є виключенням. Рішення даної проблеми дає змогу у реальному часі, використовуючи будь-який пристрій, у якого є камера, сканувати простір довкола та аналізувати будь-який текст, який попадається в область бачення камери пристроя. Потенційно, при подальшому дослідженні, дане рішення можливо вдоскноалити шляхом покращення точності розпізнавання візуальної інформації та додаванням нових моделей, які навчені розпізнавати десятки мов та різних людьских шрифтів.

Об'єкт дослідження - технології комп'ютерного зору.

Предмет дослідження - функціонал та характеристики технологій комп'ютерного зору.

Мета і завдання роботи - реалізувати трансформацію відео до тексту.

Наукова новизна роботи - використання сучасних технологій для впізнання тексту в реальному часі.

Практична значущість результатів дослідження — розроблено трансформацію відео до тексту з використанням OpenCV.

Апробація результатів магістерської роботи:

Вислобіцький В.А. “Створення графічного інтерфейсу для взаємодії з сервером трансформації відео до тексту”. Тези на Всеукраїнській науково-технічній конференції “Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу”. - Київ, 28 листопада 2023р.

Вислобіцький В.А. “Трансформація відео до тексту з використанням OpenCV”. Тези на Всеукраїнській науково-технічній конференції “Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу”. - Київ, 28 листопада 2023р.

Вислобіцький В.А. “Трансформація відео до тексту з використанням OpenCV”. Стаття на Всеукраїнській науково-технічній конференції “Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу”. - Київ, 28 листопада 2023р.

1 ДОСЛІДЖЕННЯ ДОМЕНУ КОМП'ЮТЕРНОГО ЗОРУ

Штучний інтелект відноситься до моедлювання людського інтелекту в машинах, які запрограмовані на мислення, навчання та вирішення проблем у людський манір. Штучний інтелект охоплює широкий спектр технологій і програм, і його можна розділити на два основні типи: вузький штучний інтелект (або слабкий штучний інтелект) і загальний штучний інтелект (або сильний штучний інтелект). Вузький штучний інтелект розроблений і навчений для конкретного завдання або вузького набору завдань. Він чудово виконує чітко визначену роботу, але йому бракує здатності узагальнювати свої знання в інших областях. Прикладами вузького штучного інтелекту можуть виступати персональні помічники (Siri), програмне забезпечення для розпізнавання зображень, системи рекомендацій (Netflix) та інструменти перекладу. Загальний штучний інтелект відноситься до типу унтілекту, який має здатність розуміти, вивчати та застосовувати знання в різноманітних завданнях, подібно до людського інтелекту. Наразі відбуваєьбся дослідження по створенню справжнього загального штучного інтелекту, який мав би когнитивні здібності, щоб розуміти та вивчити будь-яке інтелектуальне завдання, яке може виконати людина. Основними концепціями та техніками в штучному інтелекті виступають машинне навачння (Machine Learning), глибоке навчання (Deep Learning), обробка природної мови (Natural Language Processing), комп'ютерний зір (Computer Vision), робототехніка (Robotics). Машинне навчнання є підмножиною штучного інтелекту, яка передбачає розробку алгоритмів і статистичних моделей, які дозволяють комп'ютерам виконувати завдання без явного програмування. Типи манишшного навчання включають контрольоване навчання, неконтрольоване навчання та навчання з підкріпленням. Глибоке навчання є підмножиною штучного інтелекту, яке включає багаторівневі нейронні мережі (deep neural networks). Глибоке навчання успішно використовується у таких завданнях, як розпізнавання зображень і мови.

Обробка природної мови є розділом штучного інтелекту, який надає машинам здатність розуміти, інтерпретувати та створювати людську мову. Обробка природної мови використовується в програмах, як чат-боти, або перекладачі мов. Комп'ютерний зір є сферою штучного інтелекту, яка вивчає надання можливості машині розуміти візуальну інформацію з навколишнього середовища на манір людського зору.

Дана робота в основному присвячена вивченню та використанню технологій для роботи з комп'ютерним зором, але також частково перетинається з іншими галузями штучного інтелекту. Галузь комп'ютерного зору влючає в себе велику кількість завдань пов'язаних з інтерпритацією, аналізом та розмуінням візуальних даних як зображення та відео. Головна мета напрямку поглягає в тому, щоб надати машинам функціонал вилучення корисної інформації з візуальної інформації, яку можна в подальшому використовувати у різних задачах. Завдяки рішенню актуальних задач галузь комп'ютерного зору пропонує функціонал, який активно використовується в сферах медицини, безпеки, автомобільної промисловості, сільскому господарстві тощо. Сфера комп'ютерного зору використовується в таких задачах, як розпізнавання обличь, розпізнавання жестів, аналізі медичних зображень, автономних транспортних засобах, доповненій та віртуальній реальності тощо. Розпізнавання обличчя є технологією, яка використовує аналіз та ідентифікує обличчя на зображеннях або відео. Ця технологія використовується для безпеки, автентифікації та різноманітних додатків взаємодії людини з комп'ютером. Розпізнавання жестів є тлумаченням та реагуванням на людські жести, зняті камерами пристроїв. Розпізнавання жестів використовується в іграх, віртуальній реальності та інших інтерактивних системах. Використання технологій комп'ютерного зору в медицині допомогає в діагностиці та лікуванні захворювань за допомогою аналізу медичних зображень, таких як рентген, МРТ і КТ. Сфера комп'ютерного зору дозволяє

транспортним засоба сприймати та розуміти навколишнє середовище за допомогою візуальних даних. Це включає такі завдання, як визначення смуги руху, розпізнавання об’єктів і навігація. У доповненій та віртуальній реальності ком’ютерний зір дозволяє інтегрувати згенеровану комп’ютером інформації в режим реального світу або створювати захоплюючі віртуальні середовища за допомогою візуальних даних.

Галузь комп’ютерного зору вивчає такі речі, як формування зображень, обробка зображеннь, отримання особливостей зображеннь, представлення зображеннь, впізнаванням та виявленням різних об’єктів на зображенні, класифікацію зображень, сегментацією зображень тощо. Вивчення формування зображення необхідне для розуміння того, як формуються зображення. Комп’ютерний зір починається з отримання візуальних даних, як правило у формі зображень або відео. Розуміння того, як зображення або відео сформоване допомагає в обробці та інтерпритації візуальної інформації. Робота в галузі комп’ютерного зору починається з отримання візуальних даних, як правило, у формі зображень або відео. Розуміння того, як формуються зображення, допомагає в обробці та інтерпретації візуальної інформації. Обробка зображень необхідний крок для роботи з даними у сфері комп’ютерного зору. Завдяки цьому етапу необроблені зображення проходять процес використання на них різних фільтрів, нормалізацій, корекцій кольорів, змін розмірів, видалення шумів. Після обробки зображення позбувається усієї непотрібної інформації та передається у вигляді вхідних даних для подальших задач. Отримання особливостей зображення націлене на виявлення та виділення релевантних особливостей зображення. Такими особливостями можуть виступати текстури, краї, кути або більш складні візерунки. Представлення зображення включає конвертування зображення у формат, придатний для аналізу. Загальні представлення зображення включають градації сірого, кольорові гістограми або більш просуноті методи, як вбудовування

згорткової нейронної мережі. Розпізнавання об’єктів передбачає ідентифікацію об’єктів на зображенні, тоді як виявлення об’єктів не лише розпізнає, але й визначає місце розташування та окреслює виявлені об’єкти. Зазвичай при розпізнаванні та виявленні об’єктів використовуються такі методи, як каскади Хаара та підходи на основі глибокого навчання (Faster R-CNN, YOLO). Класифікація зображень займається призначенням міток або категорій всьому зображенню на основі його вмісту. Згорткові нейронні мережі є дуже успішними в задачах класифікації зображень. Сегментація зображення заємається поділом зображення на значущі сегменти або області. Існують такі типи сегментації, як семантична та екземплярна. Семантична призначає мітку кожному пікселю, тоді як екземплярна розрнізняє окремі екземпляри об’єктів.

Фундаментальними концепціями у галузі комп’ютерного зору є представлення зображення та обробка зображення. Представлення зображення — фундаментальна концепція комп’ютерного зору, яка пердбачає кодування візуальної інформації у форматі, придатному для обчислювального аналізу. Зображення зазвичай представляють як масиви пікселів, де кожен піксель відповідає невеликій одиниці зображення. Метою представлення зображення є захоплення основних візуальних особливостей сцени чи об’єкта таким чином, щоб полегшити подальшу обробку й аналіз. Основними компонентами при роботі з представленням зображення виступають пікселі, моделі кольору, розмір, роздільність, глибина кольору (розрядність, або бітова глибина), формат зображення, гістограми. Піксель — це найменша одиниця цифрового зображення. Це крихітний неподільний елемент із певною кольоровою інформацією. Для кольорових зображень кожен піксель складається з кількох каналів кольорів (наприклад червоний, зелений, синій для RGB зображення). Кольорові зображення можуть описуватися наступними модлелями кольорів: RGB, Grayscale, HSV, CMYK. RGB (Red, Green, Blue) найпоширеніша кольорова модель, у якій кольори представлені у вигляді комбінацій червного,

зеленого та синього кольорів. Grayscale — модель з одним кааналом, який представляє інтенсивність, де вищі значення вказують на яскравіші області. HSV (Hue, Saturation, Value) модель, що представляє кольори на основі їх відтінку, насиченості та яскравості. CMYK (Cyan, Magneta, Yellow, Black) модель, котра використовується у кольоровому друку. Кольори в такій моделі представляються як як комбінації чотирьох кольорів чорнила. Розмір зображення — це ширина та висота виміряна в пікселях. Роздільність зображення — це кількість деталей, яку містить зображення. Кількість деталей часто виражається в пікселях на дюйм (PPI) або точках на дюйм (DPI). Глибина кольору — кількість бітів, які використовуються для представлення кольору кожного пікселя. Вища глибина кольору дозволяє отримати більше кольорів і точніші градації кольорів. Формат зображення — розширення файлу зображення, яке визначає, як саме зображення опрацьовується системою. Найпопілярнішими форматами зображень виступають JPEG (Joint Photographic Experts Group), PNG (Portable Network Graphics), GIF (Graphics Interchange Format). Гістограми — це графічне представлення розподілу інтенсивності пікселів на зображенні. Гістограми допомагають зрозміти загальну яскравість і контрасніть зображення. Обробка зображення — ще один фундаментальний процес в сфері комп’ютерного зору. Обробка зображення включає маніпуляції та аналіз візуальних даних для вилучення значущої інформації. Вона охоплює широкий спектр методів покращення, трансформації та аналізу зображень. При обробці зображення існуються базові процеси, процеси націлені на колір зображення, покращення якості зображення, геометричні трансформації, сегментація. До базових процесів відноситься фільтрування, розмиття, виявлення контурів, морфологічні операції. Фільтрування передбачає застосування операцій згортання до зображень, часто з використанням фільтра або ядра. Загальні фільтри включають фільтри розмивання, підвищення різкості та визначення країв. Фільтри використовуються для згладжування або

посилення певних особливостей зображення. Розмиття зменшує різкість зображення, як правило, шляхом усереднення значень пікселів у околицях. Розмиття можна використати для видалення шуму, зменшення деталей або підготовці зображень до інших процесів обробки. Виявлення контурів використовується для визначення меж зображення шляхом виділення різких змін інтенсивності пікселів. Така обробка використовується для визначення розташування меж і форм об’єктів. Морфологічні операції на кшталт розширення та розмивання корисні для обробки бінарних зображень. Морфологічні операції використовуються при аналізі форм, зменшенні шуму та виділенні ознак зображення. До процесів, які направлені на роботу з кольором зображення відносяться конвертації кольорових моделей та вирівнювання гістограми. Конвертація кольорової моделі зображення — це процес перетворення зображень між різними кольоровими представленнями, наприклад змінити кольорову модель зображення з RGV у Grayscale. Така процедура корисна для адаптації інформації про колір для конкретних задач. Вирівнювання гістограми — це процес підвищення контрасноті зображення шляхом перерозподілу інтенсивності пікселів. Дана обробка покращує видимість деталей на зображенні із поганою контрасністю. До процесів покращення якості зображення відноситься регулювання контрасності, корекція гамми, зменшення шуму. Регулювання контрасності — це процес масштабування значень пікселів для збільшення або зменшення контрасності зображення, використовується для покращення видимості частин зображення. Корекція гами — це регулювання гами для керування яскравістю зображення, використовується для коригування нелінійних коливань інтенсивності. Зменешння шуму — це метод фільтрації для зменшення шуму, викликаного перешкодами в процесі отримання зображення. Дана обробка покарщує якість зображення та допомагає в подальшому аналізі. До геометричних трансформацій з зображенням відноситься обертання, масштабування,

викривлення. Обертання або масштабування використовуються для вирівнення та зміні розміру зображення. Викривлення — не лінійна деформація зображення, направлена на виправлення спотворень, викликаних різними умовами. До сегментації зображення відноситься обробка пороговим значенням та сегментація на основі регіону. Обробка пороговим значенням використовується для перетворення зображення у бінарний вигляд на основі порогових значень інтенсивності пікселів. Дана сегментація використовується для відокремлення предметів від фону. Сегментація на основі регіону поділяє зображення на області на основі певних критеріїв, таких як колір або текстура. Дана сегментація корисна у індетифікації та відокремленні окремих об’єктів зображення.

2 ХАРАКТЕРИСТИКА ТЕХНОЛОГІЇ OPENCV

OpenCV — бібліотека для роботи над задачами у галузі комп'ютерного зору з відкритим вихідним кодом. Бібліотека написана мовою програмування C та C++ та працює на різних платформах як Linux, Windows тощо. Дана бібліотека має інтерфейси у мовах Python, JavaScript, Matlab та інших, тому не прив'язує розробника до певної мови програмування, що є великою перевагою.

OpenCV розроблена з фокусом на додатки реального часу, через що великий акцент приділений ефективності обчислень у реальному часі. OpenCV у своїй основі використовує різноманті оптимізації, написані мовою програмування C, а також використовує перевагу багатоядерних процесорів. Окрім цього розробники додали можливість оптимізувати роботу бібліотеки OpenCV під різні архітектури. Наприклад можна використати бібліотеку IPP (Intel Integrated Perfomance Primitives), яка складається з низькорівневих оптимізованих процедур, які використовуються у багатьох алгоритмах. При наявності бібліотеки IPP можна налаштувати роботу бібліотеку OpenCV таким чином, щоб вона автоматично використовувала необхідний функціонал бібліотеки IPP у своїх функціях.

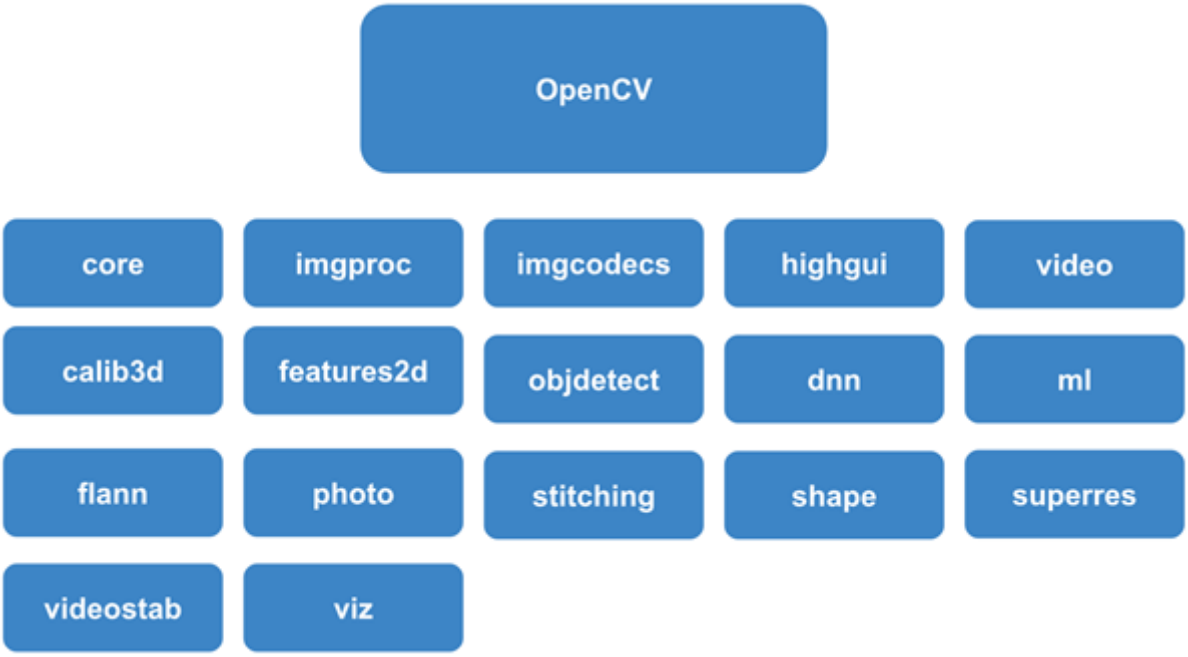


Рис. 2.1 - Модулі бібліотеки OpenCV

Однією з задач, яку вирішує OpenCV, є надання легкого та зрозумілого інтерфейсу, за яким прихований потужний функціонал, розробникам, які займаються розробкою додатків комп'ютерного зору. Завдяки простій інфраструктурі, яку надає бібілотека OpenCv, дана технологіє допомагає швидко створювати достатньо складні програми, які вирішують різноманітні задачі у облсаті комп'ютерного зору. Бібліотека OpenCV містить безліч функцій, які охоплюють різні сфери, включаючи медичну, безпеки, робототехніку та інші. Оскільки комп'ютерний зір та машинне навчання часто використовуються разом, OpenCV містить внутрішній модуль машиного навчання загального призначення (MLL). Даний модуль вирішує задачі розпізнавання образів та кластеризації. MLL є корисним інструментом у рішенні проблем комп'ютерного зору, на яких фокусується бібіліотека OpenCV.

Більшість практикуючих програмістів знають про деякі аспекти, але мало хто знає про всі способи використання в роботі комп'ютерного зору. Наприклад, більшість людей певною мірою знають про його використання в

відео спостереженні, багато хто також знає, що комп’ютерний зір все частіше використовується для зображень обрлбки зображень і відео в інтернеті. Дехто бачив використання комп’ютерного зору в ігрових інтерфейсах. Але мало хто усвідомлює що більшість аерофотозйомок і зображень вулиць (наприклад, у Google Street View) використовують методи калібрування камер та зшивання зображень. Деякі знають про нішеві додатки застосування в моніторингу безпеки, безпілотних літальних апаратах або біомедичному аналізі. Але мало хто усвідомлює, наскільки широко поширеним став машинний зір у виробництві: практично усе, що виробляється серійно, у якийсь момент автоматично перевіряється за допомогою комп'ютерного зору. З моменту випуску альфа-версії в січні 1999 року OpenCV використовувався в багатьох програмах, продуктах та дослідженнях. Ці програми включають оброблення спутникуових зображень, веб-карт, сканування зображень, шумозаглушення медичних зображень, системи безпеки та виявлення вторгнень, системи автоматичного моніторингу та безпеки, системи інспекції виробництва, калібрування камер, військове застосування в безпілотних літальніих наземних та підводні апарати. Комп’ютерний зір навіть використовувся в розпізнаванні музики, де методи зорового розпізнавання застосовувалися до звукових спектро-грам зображення.

2.1 ПРАКТИЧНЕ ЗАСТОСУВАННЯ OPENCV

```cpp
#include <opencv2/core.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui.hpp>
#include <iostream>

int main(int argc, char** argv)
{
  std::string image_path = argv[1];
  cv::Mat img = imread(image_path, cv::IMREAD_COLOR);
  if(img.empty())
  {
    std::cout << "Could not read the image: " << image_path << std::endl;
    return 1;
  }
  cv::imshow("Display window", img);
  cv::waitKey(0);
  return 0;
}
```

Рис. 2.2 — Відображення зображення бібілотекою OpenCV

На рисунку 2.2 представлене використання основних модулів бібліотеки OpenCV для роботи з зображенням. Бібліотека OpenCV містить різноманітні модулі. Кожен з модулів відповідає за надання функціоналу для рішення певних задач. Розглянемо покроково код на рисунку 2.2. OpenCV забезпечує набір інструментів для читання файлів зображень різноманітних розширень, відео файлів та роботу з камерою пристрою. Ці інструменти є частиною модуля HighGUI, який включений до бібліотеки OpenCV. В даному прикладі використовуються деякі інструменти для створення простої програми, яка відкриває зображення та відображає його на екрані.

int main(int argc, char** argv)

{

  std::string image_path = argv[1];

  cv::Mat img = cv::imread(image_path, cv::IMREAD_COLOR);

```
 if(img.empty())
 {
  std::cout << "Could not read the image: " << image_path << std::endl;
  return 1;
 }
 cv::imshow("Display window", img);
 cv::waitKey(0);
 return 0;
}
```

Коли програма буде скомпільованою та запущеною через командний рядок з вказівкою, у якості аргумента, шляху до зображення, дана програма завантажить зображення в пам’ять та відобразить на екрані. Після відображенні на екрані програма очікує моменту, коли користувач натисне на будь-яку клавішу, після чого програма одразу завершить своє виконання. Розглянемо крок за кроком дану програму.

```
std::string image_path = argv[1];
```

Цей рядок коду під час запуску програми через командний рядок отримує в якості аргумента для запуску програми вказаний шлях до зображення, яке необхідно завантажити та відобразити.

```
cv::Mat img = cv::imread(image_path, cv::IMREAD_COLOR);
```

Даний рядок завантажує зображення. Функція imread визначає формат файлу для завантаження основуючись на назві файлу. Ця фунція автоматично виділяє пам’ять, яка необхідна для збереження інформації про зображення. Варто зауважити, що функція imread може прочитати широку варіацію форматів зображень, включаючи BMP, DIB, JPEG, JPE, PNG, PMB, PGM, PPM, SR, RAS, TIFF. На результат роботи даної функції посилається ідентифікатор img з типом даних Mat. Даний тип даних репрезентує матрицю. OpenCV використвує даний тип для обробки усіх можливих варіантів зображень.

```
if(img.empty())
 {
   std::cout << "Could not read the image: " << image_path << std::endl;
   return 1;
 }
```

Даний блок коду перевіряє на успішність читання зображення. У випадку, якщо при запуску програми був вказаний не валідний шлях до зображення, спроба прочитати може відбутися без помилок, проте самі дані не будуть містити жодної інформації і зображення буде порожнім. Тому дана перевірка гарантує, що дані з зображення були отримані успішно і з даним зображенням можна продовжувати працювати в подальшій логіці програми.

```
cv::imshow("Display window", img);
```

Ще одна високорівнева функція бібліотеки OpenCV, imshow, відкриває вікно на екрані яке може містити та відображати зображення. Дана фінкція надана бібліотекою OpenCV HighGUI. Функція imshow прикрипляє назву “Display

window” до вікна. Усі майбутні виклики бібліотеки HighGUI, які будуть націлені на інтерактивність з даним вікном, будуть використовувати вказану назву вікна як посилання. За допомогою такого механізму, можливо працювати з багатьма вікнами одночасно, оскільки назва кожного вікна є унікальним ідентифікатором цього вікна.

```
cv::waitKey(0);
```

Функція waitKey просить програму зупинитися та чекати момент натиску на клавішу клавіатури. Якщо до функції передано позитивне значення у якості аргумента, програма буде чекати зазначену кількість часу в мілісекундах і після продовжить свою роботу, якщо нічого не було натиснуто. Якщо аргумент вказаний як нуль, або негативне значення, програма буде очікувати на натиск клавіші клавіатури нескінченно.

В даному прикладі не треба викликати спеціальні функції для звільнення ресурсів, які використовує програма, оскільки версія бібілотеки OpenCV 4.x реалізована таким чином, що в більшості випадків контролює використання ресурсів та автоматично звільнить їх, де необхідно.

3 ХАРАКТЕРИСТИКА ІНСТРУМЕНТІВ РОЗРОБКИ

Після того, як ми розглянули алгоритм трансформації відео до тексту, можна перейти безпосередньо до розробки проєкту. Етап розробки варто розбити на підпункти: ознайомлення з інструментами, налаштування інтсрументів, безспосередня розробка.

Варто розпочати з ознайолмення з мовою програмування, на якій буде написаний проєкт. C++ - універсальна та ефективна мова програмування, яка широко використовується в різних областях, включаючи вбудовані системи, системне програмування, розробку ігор тощо. C++ була створена Б’ярном Страуструпом на початку 1980-х років. C++ походить від мови програмування C, але має унікальні відмінності, наприклад підтримку об’єктно орієтованого програмування (ООП). За роки існування мова програмування C++ пройшла декілька ревізій, остання з яких має версію C++23.

Ключові особливості мови програмування C++:

- Мульти-парадигмість: C++ підтиримує процедурну, об’єктно-орієнтовану та загальні парадигми прогрумавання, що робить дану мову програмування надзвичайно універсальною.
- Крос-платформність: код написаний мовою програмування C++ може бути скомпільований і виконаний на різних платформах.
- Ефективність та низькорівневі можливості: C++ забезпечує низькорівневу роботу з пам’ятю за допомогою вказівників, яка є необхідною для програм, які націлені на максимальну продуктивність.
- Стандартна бібліотека шаблонів (Standart Template Library — STL): STL пропонує набір шаблоних класів і функцій, які використовуються у різних алгоритмах та структурах даних.

Розглянемо базовий синтаксис мови програмування C++. Код програми зазвичай починається з вхідної точки, функції, яка має назву main та може містити перед собою зарезервоване слово int. Це означає, що дана функція після своєї роботи поверне значення типу int, тобто ціле число. В даному випадку можна побачити, що всередині функції main після того, як відпрацює команда std::cout << "Hello, World!" << std::endl, яка виведе текст Hello, world, відпрацює команда return 0. У мові програмування C++ існує правило: функції можна реалізовувати таким чином, щоб вони завжди повертали число, яке буде сигналізувати, чи відпрацювала функція без помилок, чи ні. Якщо функція повертає значення 0, тоді помилок немає, інкаше виникла якась помилка. Завдяки такій реалізації можливо оброблювати помилки в коді, які зустрічаються під час роботи програми, що є дуже важливим моментом під час розробки. Також функція main є унікальна тим, що навіть позначивши, що вона повертає значення ціле число, рядок return 0 можна пропустити, це не буде помилкою, оскільки під час компіляції коду неявно дана команда буде додана.

Рис. 3.1 — Проста програма, яка виведе в консоль текст Hello, World

Мова програмування C++ підтримує коментарі. Коментарі — це спеціьаний синтаксис, який дозволяє в коді додавати нотатки, які несуть виключно інформаційне значення та не потрапляють до вихідного коду. Коментарі часто стають у нагоді, коли розробник має справу з комплексним кодом, оскільки за допомогою них можна залишати підсказки, які полегшуть розбір функціоналу.

C++ підтримує як коментарі в один рядок //, так і багаторядкові коментарі /**/.

```
1  // single-line comment
2
3  /*
4     ulti-line
5     comment
6  */
7
```

Рис. 3.2 — Створення комантарів в коді

C++ має статичну типізацію та містить багато типів даних. Під час розробки мовою програмування C++ ми постійно маємо справу з різними даними, які необхідно відрізняти одне від одного. Завдяки статичній типізації ми маємо змогу позначати, з якими саме даними працюємо в тому, чи іншому моменті. Це є великою перевагою, оскільки такий механізм покращує надійність роботи коду. Якщо ми позначили, що якась функція очікує аргументи певного типу, ми не зможемо викликати її з іншими аргументами, бо такий виклик призведе до помилки.

```
1  #include <string>
2
3  int dozen = 10;
4  double pi = 3.14;
5  std::string name = "Vlad";
6  char character = 'A';
7
```

Рис.3.3 — Значення та відповідні типи даних

C++ забезпечує розробника функціоналом для контролю над послідовністю виконнання коду. Використовуючи ключові слова if, else if, else,

switch та їх комбінації можна будувати блоки коду, які виконаються лише за настання певної умови. За допомогою ключових слів for, while, можна створювати блоки коду, які можуть виконуватися певну кількість разів, зазначену в момент створення блоку коду, або вираховану динамічно під час роботи коду.

```
#include <string>

int main()
{
  const int input = 10;
  const int limit = 0;
  if (input > limit)
  {
    printf("Input %d greater than %d\n", input, limit);
  }
  else if (input < limit)
  {
    printf("Input %d less than %d\n", input, limit);
  }
  else
  {
    printf("Input %d equals %d\n", input, limit);
  }
}
```

Рис.3.4 - Використання конструкцій if,else if, else

В даному варіанті наведений приклад написання коду з використанням if, else if, else, в якому запускається лише один блок коду відповідно до виконання вказаної умови. Після запуску цієї програми можна отримати повідомлення “Iput 10 greater than 0” у випадку, якщо умова, вказана у рядку if (input > limit), правдива, тобто якщо 10 > 0. У випдку input < limit отримаємо повідомлення “Input 10 less than 0”. Для всіх інших випадків, які явно не вказані в if, або else if, можна опрцювати у блоці else. В даному прикладі в блоці else перевіряється випадок, коли input == limit. У складніших випадках блок else відпрацьовував

би частіше, оскільки існувала би більша кількість умов, які явно не опрацьовуються if, або else if блоками.

```cpp
#include <string>
#include <iostream>

int main(void)
{
  const int dataLength = 10;
  int data[dataLength];
  int i = 0;
  while (i < dataLength)
  {
    data[i] = i;
    i++;
  }
  for (i = 0; i < dataLength; i++)
  {
    std::cout << data[i] << std::endl;
  }
}

```

Рис. 3.5 — Використання for та while

Цикли дозволяють повторювати необхідний фрагмент коду певну кількість разів, яку можна вказати під час оголошення блоку коду. Також можливий варіант, коли зазаделгіть невідома точна кількість ітерацій, тому цикл можна оголосити як нескінченний, проте в такому випадку необхідно обов'язково в коді вказати умову для завершення циклу, інакше такий цикл ніколи не завершиться, що є помилкою. В рисунку 3.5 показаний приклад використання циклів while та for. В даному коді є необхдність працювати з масивом, який буде зберігати 10 цілих чисел. Необхідно спочатку наповнити масив data даними, після чого виконати необхідні операції з цими даними. Без використання циклів довелося би спочатку прописати 10 разів один і той самий рядок коду, на кшталт data[0] = int, data[1] = int, в якому відрізнявся лише один

момент — індекс елемента, з яким працюємо в даний момент. Після чого довелося би знову повторити десять однакових рядків для отримання даних з масиву для подальших манціпуляцій. Використовуючи цикли можна автоматизувати даний процес. Цикл while дає змогу під час оголошення вказати умову завершення, але не передбачає створення зміних, які можна використовувати всередині циклу та не впливає на ці оголошені змінні після кожно ітерації, тому про такі нюанси варто потурбуватися заздалегіть. В даному випадку умовою завершення роботи циклу while буде момент, коли змінна для індексування не буде меньшою, ніж довжина масива. У якості індекса виступає змінна i, яка оголошена раніше циклу while. У якості коду, який буде змінювати індекс виступає рядог i++ - без цього рядка даний цикл буде нескінченним. Перед кожною ітерацією цикл перевіряє правдивість умови запуску блоку коду. Якщо жодним чином не впливати на дану умову, то можна отримати нескінченний цикл. Так відбувається тому, що кожен раз у перевірці фігурують значення зі змінної i, яке спочатку дорівнює нулю, та значення зі змінної довжини масива, яке є незмінним. Твердження про те, що i < dataLength, тобо 0 < 10 є правдивим, тому запсукаємо блок коду. Після відпрацювання блоку коду знову переходимо до перевірки, чи необхідно запустити блок коду. Якщо прибрати рядок i++, тоді перевірка i < dataLength завжди буде правдивою, оскільки змінні жодним чином не змінюються, а це призведе до нескінченного циклу. Створення циклу for надає більше можливостей, оскільки даний цикл передбачає місце для оголошення блокових змінних, одну з яких можна використовувати під час перевірки умови завершення роботи циклу, умову для завершення циклу, а також простір для маніпуляції над змінними, необхідіними для роботи циклу. Всі ці налаштування під час написанная циклу for відокремлені крапкою з комою. Також кожне з цих налаштуваннь є опційним, тобто в оголошенні циклу for можна пропустити одне з налаштувань. Відстуність кожного налатшування необхідно опрацювати

в коді для коректної роботи циклу. На рисунку 3.5 можна побачити використання циклу for: в першому налаштуванні звертаємося до змінної i, яку пов’язуємо зі значенням 0, щоб розпочати інтерування з першого елемента масива, далі вказуємо умову роботи цикла for — ітеруватися поки індекс меньший ніж довжина масива, після чого вказуємо яким чином змінюється індекс після кожної ітерації — рядком i++. Після оголошення циклу всередині фігурних дужок вказуємо блок коду, який необхідно виконати зазначену кількість разів. В даному випадку ітеруємося по масиву, який попередньо наповнили та переглядаємо кожне значення масива за відповідним індексом.

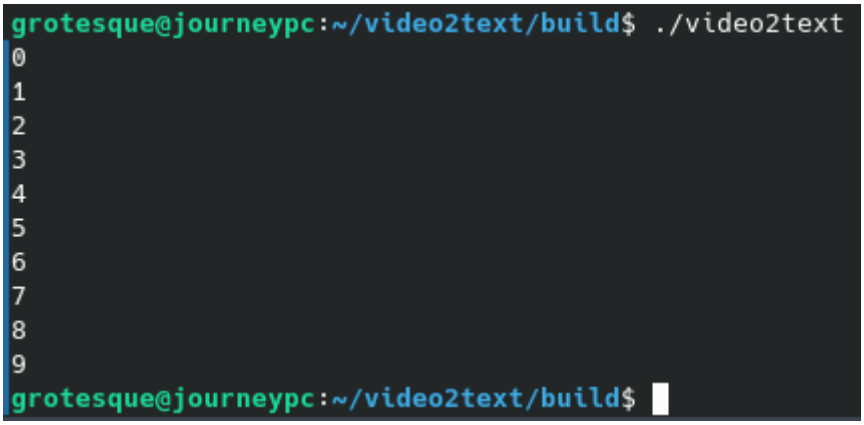


Рис. 3.6 — Результат роботи коду з рис.3.5

C++ підтримує функції, які дозволяють розбивати програму на сегменти коду, які можна виокремити одне від одного та використовувати в різних частинах програми. Зазвичай до фіункції виноситься певна частина коду, яка повторюється багато разів у програмі, після чого ця функції викликається де потрібно замість того, щоб дублювати код. Також під час написання коду функції варто створювати таким чином, щоб одна функція вирішувала одну задачу. Часто при роботі з функціями для вирішення тієї чи іншої задачі, яка

закладена в тілі функції, необхідно передати дані в середину функції. При оголошенні функції можна вказати, які параметри містить дана функція, тобто які дані очікує для успішного виклику. Також функцію можна реалізувати таким чином, що окрім даних, які можна передати до функції, можна отримати результат роботи функції, який використати в подальшій логіці. На рисунку 3.7 продемонстровано використання функцій. Для створення функції необхідно спочатку вказати тип значення, який вона повертає. Далі необхідно надати назву цій функції, через яку будемо її викликати. Після назви знаходяться круглі дужки, вони слугують для оголошення параметрів, які очікує дана функція. Після круглих дужок ідуть фігурні, які виокремлюють тіло функції, безпосередньо в якому знаходиться логіка функції. На рисунку 3.7 з 4 рядка по 6 можна побачити функцію з назвою log. Перед назвою log стоїть ключове слово void (порожнеча), яке вказує на те, що дана функція, після виконнаня своєї роботи, нічого не повертає.

```cpp
#include <string>
#include <iostream>

void log(void)
{
  std::cout << "Message" << std::endl;
}

void log(const std::string msg)
{
  std::cout << msg << std::endl;
}

int multiple(const int a, const int b)
{
    return a * b;
}

double multiple(const double a, const double b)
{
    return a * b;
}

int main(void)
{
  log();
  log("My message");
  const int first = multiple(5, 4);
  const double second = multiple(5.51, 3.96);
  std::cout << "First: " << first << "\nSecond: " << second << std::endl;
}

```

Рис.3.7 — Використання функції

Праворуч від назви log іде секція в круглих дужках, в яких повинні були бути параметри функції, натомість там знаходиться ключове слово void. Void в даному випадку сигналізує про те, що функція не приймає жодних параметрів. Якби void було відсутнім, це сигналізувало би про те, що дана функція приймає невизначену кількість параметрів. На рядку 26 рисунка 4.7 можна побачити, яким чином тепер викликати функцію log для відпрацювання. Необхідно звернутися за назвою ціє функції, після чого додати круглі дужки. Дана функція вивиде повідомлення Message. З рядка 9 по 12 визначена функція log, яка нічого не повертає після своєї роботи, натомість відрізняється від попереднього визначення тим, що очікує один параметр, який вказаний всередині круглих

дужок після назви функції. Даний парамет має назву msg, за цим ідентифікатором тепер можна звертатися для отримання переданого значення всередині функції. Також перед назвою змінної msg знаходиться тип даних, на який посилається дана змінна. В даному випадку ми вказуємо те, що змінна msg буде посилатися на значення типу string — рядок, текст. Цього достатньо для роботи даної функції, але бувають випадки, коли функція очікує на певні значення, проте використовує їх лише для читання, жодним чином не модифікуючи. Такий випадок можна вказати за допомогою ключового слова const, яке стоїть перед типом параметра. Таким чином дана функція очікує одне значення типу string, яке жодним чином не модифікує та після своє роботи нічого не повертає. На рядку 27 бачимо виклик функції: звертаємося за назвою функції log, далі в круглих дужках передаємо аргумент функції — значення типу string “My message”. Дана функція виводить передане до неї повідомлення. На цьому моменті могло виникнути питтання, - чи немає конфлікту між визначеннями функцій між 4 та 6 рядкамми, та з 9 по 12 рядок, оскільки дві функції містять однакову назву log? Відповідь — ні. C++ підтримує механізм перегрузку функції (function overloading). Даний механізм дозволяє двум і більше функціям мати одну назву, але відрізнятися параметрами. В момент виклику необхідної функції компілятор перевіряє, яку саме функцію необхідно викликати спираючись на аргументи, які передаються під час виклику до функції, після чого виконується необхідна логіка з потрібної функції. Таким чином не відбувається конфлікту. З рядка 14 по рядок 22 можна побачити визначення перегруженої функції multiple. Дана функція займається множенням переданих до неї значень. Функція у першому випадку очікує 2 цілих числових значення типу int, які не буде модифікувати, та поверне результат роботи у вигляді цілого числа. Повернене значення з функції описанл ключовим словом int, яке знаходиться лівіше від назви функції. У другому випадку функція multiple працює зі значеннями з плаваючою комою, які

описані типом double, та повертає результат їх множення, також число з плаваючою комою. На рядках 28 та 29 представлені виклики функції multiple з різними значеннями та опрацьований результат повернення функції. На 28 рядку функція multiple викликається з двума аргументами типу int, кожен переданий аргумент відокрмелюється комою: multiple(5, 4). Після відпрацювання функції результат роботи поміститься до змінної типу int з назвою first. На 29 рядку представлений аналогічний виклик функції multiple за винятком типу аргументів, які передаються до функції та типу змінної, яка посилається на результат роботи функції.

```
grotesque@journeypc:~/video2text/build$ ./video2text
Message
My message
First: 20
Second: 21.8196
grotesque@journeypc:~/video2text/build$
```

Рис.3.8 — Результат роботи коду рис.3.7

3.1 ОБ'ЄКТНО ОРІЄНОВНЕ ПРОГРАМУВАННЯ C++

C++ підтримує об'єктно орієнтовне програмування (ООП). Мета такого програмування полягає у тому, щоб пов'язати дані та функції, які працюють з цими даними та сховати ці дані від доступу на пряму з будь-якої іншої частини коду, окрім функцій, які спеціально визначені для роботи з цими даними. Основними концепціями у якості будівельних блоків ООП виступають класи, об'єкти, абстракція, наслідування, поліморфізм, інкапсуляція.

Клас — це спеціальний програмний код, який виступає у ролі шаблона для створення майбутніх екземплярів цього класа. Клас слід розглядати як механізм, який необхідний для опису можливостей майбутніх екземплярів класу. Екземпляр класу — об'єкт, що має властивості та функціонал описаний класом. Об'єкти використовуються для виконання бізнес логіки та взаємодії з

іншими об’єктами. Наприклад, якщо вам необхідно було створити банківську систему, то ви мали б справу з такми речами як користувач, рахунок користувача, транзакція між користувачами тощо. Такі речі можна було б описати трьома відповідними класами, в кожному з яких виокремети, які дані та які операції проводилися над тим, чи іншими даними. Клас оперує всередині себе внутрішнім станом — даними, які використовуються під час роботи та функції для роботи з цими даними. Кожен клас містить всередині себе поля та методи, які можуть називатися членами цього класу. Полями класу описуються властивості майбутніх екземплярів цього класу, в них зберігаються дані. Методи визначають логіку роботи майбутніх екзеплярів класу. Методаикористання сучасних технологій для впізнання тексту в реальному часі.ми називаються функції, які знаходяться всередині класу. На рисунку 3.9 наведений приклад коду з використанням класу. Даний клас передбачає те, що об’єкти, створені на його основі, будуть описувати можливості людей Для створення класу необхідно використати ключове слово class, після чого вказати назву цього класу, далі додати фігурні дужки всередині яких буде знаходитися уся необхідна логіка. За замовчування в C++ усі класи приватні, тобто усі поля та методи доступні для використання лише всередині цього класу, в межах фігурних дужок. Така механіка додає надійності роботи коду, оскільки гарантує, що ззовні відсутня змога отримати доступ до полів та методів всередині класу, тобто будь-хто випадково не вплине на роботу екземплярів клас в коді. Однак, оскільки на основі класів створюються об’єкти, які взаємодіють між собою, а за замовчування усі члени класу приватні, необхідно якимось чином вказати, які саме можливості класу доступні для використання ззовні. Для цього необхідно додати ключове слово public. Public — один з модифікаторів доступу, який вкзаує, що саме тепер доступно для використання не лише всередині класу, а й на зовні. На рисунку 3.9 на 6 рядку вказаний модифікатор доступу public, після чого йде набір полів та методів, які тепер

доступні для використання при звертанні до екземплярів класу. Даний клас описує властивості та можливості людини. Клас Human окреслює, що кожен екземпляр класу буде містити ім’я, вік та може розмовляти. На 7 рядку знаходиться поле класу name, яке очікує на значення типу string — текст. На 7 рядку поле з назвою age, яке очікує ціле число — вік людини.

```
1  #include <string>
2  #include <iostream>
3
4  class Human
5  {
6  public:
7    std::string name;
8    int age;
9
10   Human(std::string n, int a) : name(n), age(a) {};
11
12   void speak()
13   {
14     std::cout << name << " speaks..." << std::endl;
15   }
16 };
17
18 int main(void)
19 {
20   Human john("John", 30);
21   john.speak();
22   std::cout << john.age << std::endl;
23   Human alex("Alex", 25);
24   alex.speak();
25   std::cout << alex.age << std::endl;
26 }
27
```

Рис.3.9 — Використання класу

На 12 рядку створений метод класа speak, сенс існування якого полягає в тому, щоб надати можливість людині спілкуватися. Даний метод звертається до поля класу name, з якого отримує значення, яке виводить. Із важливого варто виокремити 10 рядок. На цьому рядку знаходиться спеціальний варіант функції в контексті опису класу — конструктор. Конструктор — функція, що викликається в момент створення екземпляра класа. Назва конструктора

позначається з великої літери. Як і будь-яка інша функція констурктор може приймати параметри. В даному випадку конструктор очікує назву людини та її вік, без цих даних екземпляр класу не створити. Конструктор може містити деяку логіку, яка запуститься в момент створення екземпляра класа, проте в даному варанті конструктор займається ініціалізацією полів класа. Тобто конструктор очікує, що він буде викликаний з певними аргументами, після чого значення будуть прив'язані до відповідних полів. Даний синтаксис показаний після закриваючої дужки списку параметрів конструктора, за якою слідує двокрапка : name(n), age(a) {};. Тобто n та a параметри конструктора Human, які пов'язуються з відповідними полями name та age. З 20 по 25 рядкі представлено створення екземплярів класа Human. Сам клас не запускає фізично код, а лише описує, як код буде працювати, тому до класу треба певним чином звернутися, щоб викликати необхідну логіку. На 20 та 23 рядках показано створення екземплярів класа Human. Дві змінні мають назву john та alex. Як і перед будь-якою змінною необхідно вказати тип даних, з яким ідентифікатор пов'язаний. В якості типу даних виступає назва класа, на основі якого створений об'єкт. Іншими словами ми стверджуємо, що на даних рядках змінні посилаються на дані в пам'яті, які виглядають так, як їх описує клас Human, тому тип цих зміних вказуємо як Human. Після назв john та alex ідуть круглі дужки, до яких передаються необхідні аргументи для ініціалізації. Як раз ці круглі дужки та передані аргументи є викликом конструктора класа Human. Після цих двух рядків маємо справу з двума об'єкатми, кожен з яких індивідуально оперує своїми даними та функціоналом. Можемо в цьому переконатися на 21, 22 та 23, 24 рядках. Спочатку на 21 рядку звертаємося до об'єкта john, у якого через крапку викликаємо метод speak. Аналогічну операцію проводимо з об'єктом alex на 23 рядку. На 22 рядку звертаємося до властивості age у об'єкта john та отримаємо необхідну інформацію для нашої логіки; повторюємо цю операцію на 24 рядку з об'єктом alex.

```
grotesque@journeypc:~/video2text/build$ ./video2text
John speaks...
30
Alex speaks...
25
grotesque@journeypc:~/video2text/build$
```

Рис.3.10 — Робота з об’єктами

Бачимо на рисунку 3.10 результат виконання коду: на основі одного класу Human створено декілька екземплярів класу, кожен з яких оперує своїм власним простіром даних, завдяки якому не відбувається конфлікту.

При роботі з системами, які будуються шляхом використання об’єктно орієнтованого програмування, часто можуть виникати сутації, коли одні об’єкти майже ідентичні до інших. Тобто класи, які описують такі об’єкти, включать в себе ідентичні частини коду, які дублюють одне одного. Для вирішення такої ситуації існує механіз під назвою наслідування. Наслідування дозволяє розширяти одні класи на основі інших. Простими словами, створючи певний клас і наслідуючи його від іншого, похідному класу надається доступ до визначеного функціоналу батьківського класа. Таким шляхом у ситуаціях, коли два класи близькі за своєю реалізацією, можна створити додатковий клас, до якого винести функціонал, що зустрічається в обох класах, після чого в цих класах залишається лише унікальна логіка, притамана цим класам. Завдяки наслідуванню можна будувати комплексні системи з тісною ієрархією класів, кожен елемент яких виконує певну визначену функцію, однак такий підхід при поганому дизайні системи призводить до ускладнень розробки та підтримки коду через тісний зв’язок між класами призводить, який призводить до ускладнень під час внесення змін до системи. На рисунку 3.11 представлений приклад використанням наслідування в мові програмування C++. В даному прикладі зображено наслідування від одного класу, хоча C++ підтримує мульти наслідування. На даному прикладі є базовий клас Human, який описує загальні

характеристики для усіх похідних класів. Похідними класами виступають Chef та Manager. Будуючи систему ми знаємо, що і повар, і менеджер є людьми з загальними характеристиками, які для того, щоб двічі не прописувати в обох класах можна винесити до базового класу Human, від якого наслідувати похідні класи Chef та Manager, в яких залишити лише унікальні властивості та функціонал в контексті цих об’єктів. Клас Chef описує лише об’єкт повара, який працює в деякому закладі та вміє готувати. Ці характеристики виражені полем std::string restaurant, яке зберігає назву ресторану, в якому працює шеф та методом cook всередині класу Chef. Аналогічний підхід використовується з класом Manger, в якому є поле std::string company, в якому буде зберігатися назва компанії, в якій працює менеджер та унікальний метод для екземплярів цього класу manage. Таким шляхом ми визначили те, що в нашій системі ми

```cpp
class Human
{
public:
  std::string name;
  int age;

  Human(std::string n, int a) : name(n), age(a) {};

  void speak(void)
  {
    std::cout << name << " speaks..." << std::endl;
  }
};

class Chef : public Human
{
public:
  std::string restaurant;

  Chef(std::string n, int a, std::string r) : Human(n, a), restaurant(r) {};

  void cook(void)
  {
    std::cout << "Chef " << name << " is coocking at " << restaurant << std::endl;
  }
};

class Manager : public Human
{
public:
  std::string company;

  Manager(std::string n, int a, std::string c) : Human(n, a), company(c) {};

  void manage(void)
  {
    std::cout << "Manager " << name << " is managing at " << company << std::endl;
  }
};

int main(void)
{
  Chef john("John", 30, "Some restaurant");
  Manager alex("Alex", 25, "Some company");
  john.speak();
  john.cook();
  alex.speak();
  alex.manage();
}

```

Рис. 3.11 — Наслідування класів

працюємо з об’єктами, які виступають у ролі людей, але за допомогою наслідування вказали, що у нас не просто є об’єкти люди, а існують конкретно

визначені категорії людей як повар, або менеджер. З точки зору використання наслідування в коді варто відмітити деілька моментів. По-перше на рядках 18 та 31 показно, як розширити похідний клас від базового. Після ключового слова class та назви класу необхідно додати двокрапку після якої додати ключове слово public, за яким вказати назву класа, від якого наслідємося. Наприклад рядок 18 class Chef : public Human сигналізує про те, що клас Chef наслідується від класа Human. Аналогічну річ сигналізує рядок 31 class Manager : public Human в якому класс Manager наслідується від Human. По-друге варто звернути увагу на те, як в похідних класах реалізовані конструктори. На рядку 23 визначений конструктор класа Chef, який має три параметри: ім'я повара, вік та ресторан. Аналогічну конструкцію можна побачити на рядку 36 де конструктор класа Manager очікує ім'я менеджера, вік та компанію, в якій працює менеджер. Після чого необхідно викликати конструктор базового класа. Цей момент в коді знаходиться після закриваючої круглої дужки параметрів похідного конструктора, за якою слідує двокрапка, після якої стоїть виклик конструктора базового класа з необхідними аргументами. У класі Chef це Chef(std::string n, int age, std::string restaurant) : Human(n, a), у класі Manager(std::string n, int a, std::string company) : Human(n, a). Після виклику батьківського конструктора в даному випадку слідує ініціалізація відповідних полів необхідними значеннями, після чого йде тіло конструктора, яке відокремлене фігурними дужками. В похідних класах можна пропускати виклик базового констуктора у випадках, коли для виклику базового конструктора не потрібно передавати жодних аргументів. В такому випадку виклик базового конструктора буде виконуватися неявно. В даному прикладі з поваром та менеджером базовий клас людини очікує необхідні аргументи, тому в похідних класах необхідно явно вказати, що такі конструктори очікують окрім унікальних значень, притаманних даним класам, значення необхідні для виклику базового конструктора. На 45, 46 рядках відбувається ініціалізація екземплярів класів

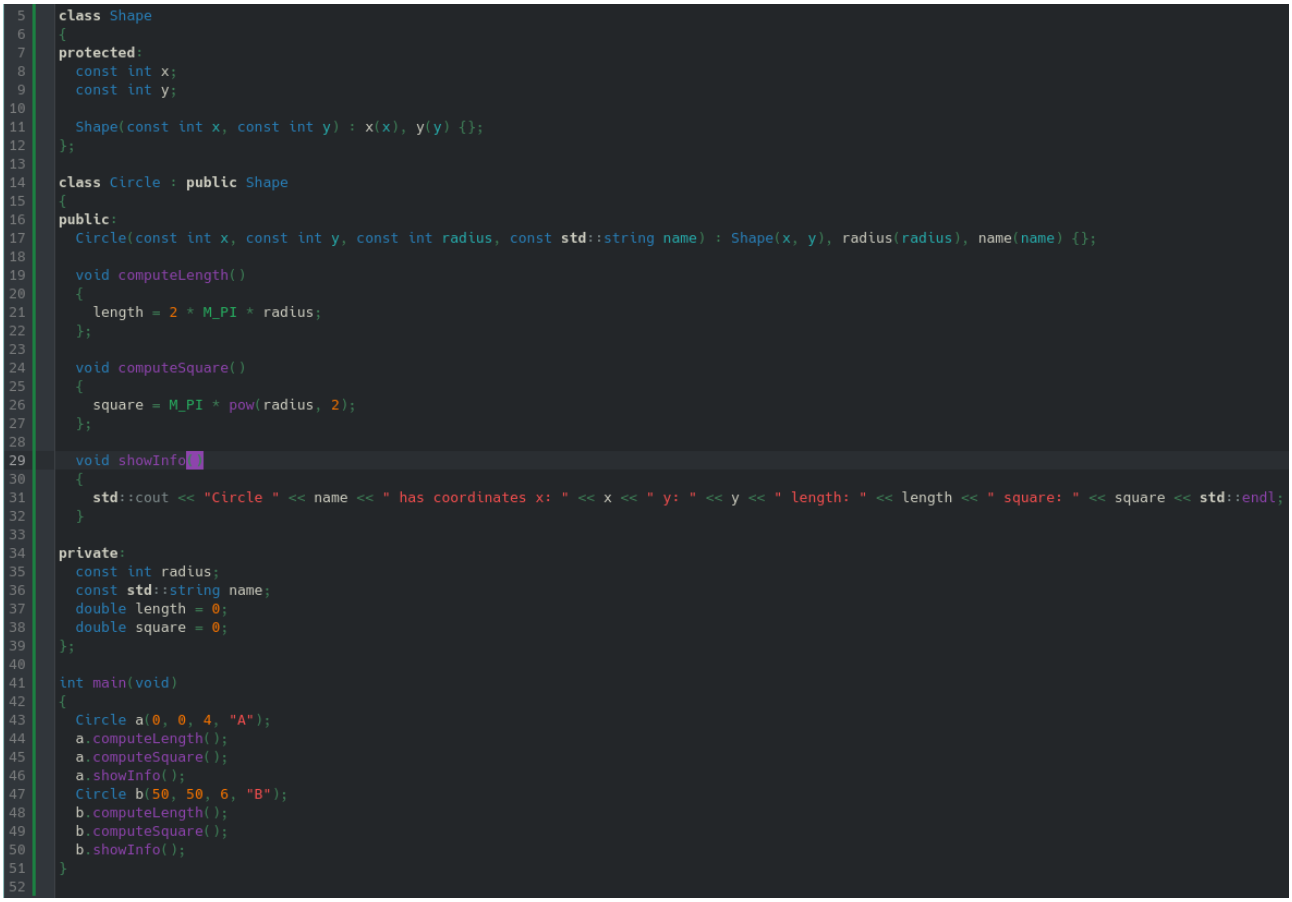
Chef та Manager. Можна побачити, що змінні john та alex тепер мають не тип Human, а конкретніші типи Chef та Manager. На рядках з 48 по 51 відбувається виклик методів у об’єктів john та alex.

```
grotesque@journeypc:~/video2text/build$ ./video2text
John speaks...
Chef John is coocking at Some restaurant
Alex speaks...
Manager Alex is managing at Some company
grotesque@journeypc:~/video2text/build$
```

Рис. 3.12 — Робота екземплярів похідних класів

Можна побачити, що об’єкт john містить унікальний для себе метод cook та загальний метод для усіх людей спік; те саме стосується і об’єкта alex, у якого викликається притаманний метод для менеджерів manage та загальний для усіх людей метод speak.

За замовчуванням в мові програмування C++ усі члени класу є приватними. Тобто доступ до них є лише всередині класу, неможливо звернутися до властивостей та методів екземплярів класу. Для керування доступом до функціоналу існують спеціальні модифікатори доступу такі як public, private та protected. Модифікатор доступу public позначає те, що усі члени класу є публічними та доступні як до виклику ззовні, так і зсередини структури класу. Модифікатор доступу private означає, що члени класу є приватними та достіпні лише всередині структури класу.



```
class Shape
{
protected:
  const int x;
  const int y;

  Shape(const int x, const int y) : x(x), y(y) {};
};

class Circle : public Shape
{
public:
  Circle(const int x, const int y, const int radius, const std::string name) : Shape(x, y), radius(radius), name(name) {};

  void computeLength()
  {
    length = 2 * M_PI * radius;
  };

  void computeSquare()
  {
    square = M_PI * pow(radius, 2);
  };

  void showInfo()
  {
    std::cout << "Circle " << name << " has coordinates x: " << x << " y: " << y << " length: " << length << " square: " << square << std::endl;
  }

private:
  const int radius;
  const std::string name;
  double length = 0;
  double square = 0;
};

int main(void)
{
  Circle a(0, 0, 4, "A");
  a.computeLength();
  a.computeSquare();
  a.showInfo();
  Circle b(50, 50, 6, "B");
  b.computeLength();
  b.computeSquare();
  b.showInfo();
}
```

Рис . 3.13 — Використання модифікаторів доступу

За замовчування під час визначення, всередині структури класу, усі члени приватні. Модифікатор доступу protected позначає члени класу наступним чином: усі члени класу позначені як protected доступні всередині структури класу, в якому оголошені, доступні всередині структури будь-якого похідного класа, який розширяє базовий клас, в якому оголошені protected члени, та забороняє звертання до protected членів через екземпляри класу.

На рисунку 3.13 зображений код використання усіх трьох модифікаторів доступу. В даному коді ми маємо клас Shape та клас Circle. Клас Shape містить два protected поля x та y, які будуть зберігати координати фігури та protected конструктор, який неможливо викликати напряму, тобто рядок Shape shape(0, 0)

видасть помилку, оскільки модифікатор доступу protected заборонює доступ до членів класу ззовні цього класу. Клас Circle наслідує клас Shape. Даний клас містить публічний конструктор для створення екземплярів класу Circle публічний метод computeLength для підрахунку довжини кола, публічний метод computeSquare для підрахунку площі кола та публічний метод showInfo, який виводить усю необхідну інформацію стовоно даного кола. Нижче публічних членів розташовані приватні члени. В даному прикладі приватними членами виступає поле radius для збереження радіуса кола, name для збереження назви кола, length для довжини кола, яке за замовченням дорівнює 0, поки не буде викликан метод computelength, який вирахує поточну довжини кола та поле square, яке за замовчуванням дорівнює 0, поки не буде викликан метод computeSquare для підрахунку площі поточного кола. В методі showInfo знаходиться функціонал, який використовує одночасно усі модифікатори доступу. Сам метод є публічним та доступен ззвоні класу, на рядках 46 та 50 можна побачити, що у екземплярів класу a та b через крапку викликається даний метод. Сам метод реалізований таким чином, що виводить інформацію про поточне коле. Справа в тому, що вивід цієї інформації включає в себе звертання до приватних членів класу Circle як length, square а також до наслідуваних полів x та y. Ззовні класу Circle дані члени недоступні. Якщо спробувати звернутися у об’єкта a через точку до координати x, або до його радіуса, отримаємо помилку, оскільки модифікатори доступу блокують дану поведінку. На рядках з 43 по 50 демонструється створення двух екземплярів класу Circle a та b.

```
grotesque@journeypc:~/video2text/build$ ./video2text
Circle A has coordinates x: 0 y: 0 length: 25.1327 square: 50.2655
Circle B has coordinates x: 50 y: 50 length: 37.6991 square: 113.097
grotesque@journeypc:~/video2text/build$
```

Рис. 3.14 — Результат роботи коду з рис. 3.13

Після створення у кожного з об’єктів викликається спочатку метод computeLength для підрахунку довжини кола, після викликається метод computeSuqare для підрахунку площі кола, після чого викликається метод showInfo для демонстрації результату усіх підрахунків.

Ще однією важливою концепцію при роботі з об’єктно орієнотавним кодом є поліморфізм. Поліморфізм — фундаментальна концепція об’єктно орієнтованого програмування, яка дозволяє об’єкти різних похідних типів розглядати та опрацьовувати як об’єкти одного базового типу. Такий підхід дозволяє мати один інтерфейс, який дозволяє відображати різні типи об’єктів, забезпечуючи один загальний спосіб для роботи з об’єктами різних типів. Поліморфізм розділяють на статичний (compile time) та динамічний (runtime). Статичний поліморфізм відбувається на етапі компліяції програми. Він може виражатися у вигляді перегрузки функції. Перегружена функція (function overloading) один з видів статичного поліморфізму, в якій кілька функцій мають однакові назви, але відрізняються параметрами. Компілятор визначає яку саме функцію необхідно викликати спираючись на передані аргмунти під час виклику функції. Приклади перегрузки функції можна поглянути на рисунку 3.7.

```cpp
#include <string>
#include <iostream>
#include <math.h>

class Number {
public:
    double value;

    Number operator+(const Number& other) {
        Number result;
        result.value = this->value + other.value;
        return result;
    }
};

int main(void)
{
  Number first{1.25};
  Number second{1.5};
  Number sum = first + second;
  std::cout << sum.value << std::endl;
}

```

Рис. 3.15 — Перегрузка оператора +

Другим прикладом статичного поліморфізму виступає перегруження оператора. Перегруження оператора — механізм, який дозволяє вказати власну поведінку роботи певного оператора при взаємодії цього оператора з типами даних визначеними розробником.

На рисунку 3.15 представлений приклад статичного поліморфізму у вигляді перегрузки оператора + для визначеного типу Number. Даний тип визначений класом Number, в якому є поле value типу double для збереження значень з плаваючою комою. З 9 по 13 рядок представлена реалізація перевантаження оператора + при роботі з типом Number. В даному випадку ми вказуємо те, що кожен раз, коли в коді оператор + буде використовуватися з операндми типу Number, даний код буде розглядатися як сума значень, які беруться з кожного операнда з поля value та плюсуються разом. Після чого

даний вираз буде повертати екземпляр класа Number, в якому в полі value буде знаходитися результат суми двох операндів. На 18 та 19 рядках створюються два об’єкти, до кожного з яких поміщається передане значення. На 20 рядку показан приклад використання перегруженого оператора +. В даному випадку оператор + відпрацьовує з операндами first та second та повертає результат роботи у вигляді значення типу Number, яке записується до змінної sum. На рядку 21 виводимо результат роботи.

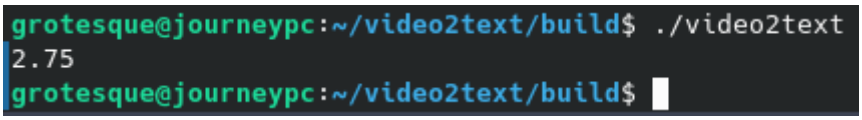


Рис. 3.16 — Результат роботи коду з рис. 3.15

Другим типом поліморфізму є динамічний поліморфізм (runtime). Такий поліморфізм досягається за рахунок віртуальних функцій, або справжніх віртуальних функцій. В мові програмування C++ віртуальною функцією називається функція, яка оголошена в базовому класі та позначена ключовим словом virtual. Така функція дозволяє похідним класам впроваджувати власну реалізацію цієї функції. Момент виклику необхідної функції відбувається під час роботи програми (runtime) та основується безпосередньо на типі об’єкта, у якого викликається метод. На рисунку 3.18 представлений приклад використання віртуальної функції. В даному прикладі визначений базовий клас Shape, всередині якого оголошена публічна віртуальна функція draw. Далі представленйи клас Circle, який є похідним від базового класа Shape, тобто розширяє його. Всередині класу Shape оголошений метод draw, який модифікований ключовим словом override. Дана модифікація вказує на те, що похідний клас Circle впропаджує власну реалізацію метода draw. Таким чином коли в коді відбувається виклик метода draw у об’єкта shape, викликається саме похідний метод draw у класа Circle, а не метод draw у базового класа Shape.

```
grotesque@journeypc:~/video2text/build$ ./video2text
Circle draw method
grotesque@journeypc:~/video2text/build$
```

Рис 3.17 — Результат роботи віртуальної функції

```cpp
#include <string>
#include <iostream>
#include <math.h>

class Shape
{
public:
  virtual ~Shape() {};

  virtual void draw()
  {
    std::cout << "Shape draw method" << std::endl;
  }
};

class Circle : public Shape
{
public:
  void draw() override
  {
    std::cout << "Circle draw method" << std::endl;
  }
};

int main(void)
{
  Shape* shape = new Circle;
  shape->draw();
  delete shape;
}
```

Рис. 3.18 — Приклад застосування віртуальної функції

Справжньою віртауальною функцією називається функція, яка оголошена всередині базового класу, але не має реалізації. Клас, який містить хоча б одну справжню віртуальну функцію називається абстрактним класом. На основі абстрактного класу не можна створювати екземпляри цього класу. Абстрактний клас використовується для наслідування похідним класом, який зобов’язаний реалізувати усі справжні віртуальні функції базового класа . На рисунку 3.19 наведений приклад використання абстрактного класа. В даному коді є абстрактний клас AbstractShape. Даний клас містить справжню віртуальну функцію draw, яке відрізняється від віртуальної функції тим, що після сигнатури функції стоїть оператор = та значення 0. Даний запис сповіщає про те, що даний метод є чистою віртуальною функцією, яка не може містити реалізації. Також в коді існує клас Circle, який наслідує абстрактний клас AbstractShape. Якщо в похідному класі наслідувати базовий класс та не реалізувати усі чисті віртуальні функції, даний код при компіляції видаст помилку, тому клас Circle обов’язково надає власну реалізацію метода draw.

```cpp
#include <string>
#include <iostream>
#include <math.h>

class AbstractShape
{
public:
  virtual ~AbstractShape() {};

  virtual void draw() = 0;
};

class Circle : public AbstractShape
{
public:
  void draw() override
  {
    std::cout << "Circle draw method" << std::endl;
  }
};

int main(void)
{
  AbstractShape* shape = new Circle;
  shape->draw();
  delete shape;
}
```

Рис. 3.19 — Приклад справжньої віртуальної функції

```
grotesque@journeypc:~/video2text/build$ ./video2text
Circle draw method
grotesque@journeypc:~/video2text/build$
```

Рис. 3.20 — Результат роботи справжньої віртуальної функції

Поліморфізм в мові програмування C++ надає потужні механізми для створення гнучкого, розширюваного коду, який можна перевикористоувати у різних частинах системи. Ця концепція об’єктно орієнтованого програмування дозволяє писати код в більш абстрактний спосіб, покращуючи адаптивність системи до змін та нововведень.

Однією з багатьох особливостей, якими володіє мова програмування C++ є шаблонне програмування (template programming). Таке програмування є потужним та гнучким інструментом, який дозволяє створювати класи та

функції, які можна використовувати для роботи з різноманітними даними. Шаблони дозволяють визначати класи та функції без прив'язки до певних типів даних, над якими вони проводять операції. Замість цього тип даних визначається в момент, коли використовується шаблон, дозволяючи створити фрагмент коду, який можна багаторазовов використовувати та який забезпечує надійність використання конкретного типу даних в конкретній ситуації. Шаблонне програмування часто використовується при роботі з алгоритмами та стрктурами даних, які створені спеціально для того, щоб мати справу з різними типами даних.

```
1  #include <string>
2  #include <iostream>
3  #include <math.h>
4
5  template <typename T>
6  T multiple(T a, T b) {
7      return a * b;
8  }
9
10 int main(void)
11 {
12   const int a = multiple(4, 5);
13   const double b = multiple(14.28, 15.96);
14   std::cout << a << "\n" << b << std::endl;
15 }
16
```

Рис. 3.21 — Приклад створення шаблону функції

В C++ існує два головних різновидів шаблонів: функцій та класів. Шаблони функцій дозволяють визначати універсальні функції, які можуть оперувати над різними типами даних.

На рисунку 3.21 на рядках з 5 по 8 представлено визначення шаблону функції з назвою multiple. Дана функція містить спеціальний синтаксис у вигляді запису template <typename T>. Даний запис сигналізує про те, що дана функція є шаблонною. В цьому записі важливим моментом є шаблонний параметер T, який відображає універсальний тип даних. Таким чином функція

multiple може бути використана з різними типами даних та повертати різні типи даних. Компілятор згенерує спеціальний варіант функції під кожен необхідний тип даних, з якими дана функція працює. На рядках 12 та 13 можна побачити, як функція multiple спочатку викликається з аргументами типу int, після чого на наступномо рядку функція multiple викликається з аргументами типу double. Обидва виклики працюють без помилок та повертають необхідний результат.

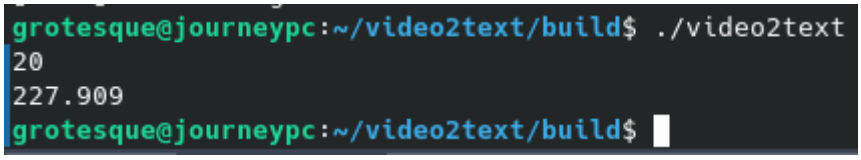


Рис. 3.22 — Робота шаблону функції

Шаблони класів дозволяють визначати універсальні класи, які можуть працювати з різними типами даних. На рисунку 3.24 представлений код, в якому наведений приклад використання шаблону класу. На 6 рядку оголошений клас Transaction, над яким на 5 рядку стоїти запис template <typename T>, який сповіщає про те, що це шаблон класу. Літера T відображає універасльний параметер в шаблоні. Даний клас в середині себе містить приватне поле currency, тип якого вказаний як літера T. Сенс в тому, що кожен раз, коли буде створений екземпляр класа Transaction, буде вказаний тип даних, з яким працює даний екземпляр класу, тому поле curency кожен раз буде містити відповідний зазначений тип. На рякдах 40 та 41 можна побачити ініціалізацію екземплярів класу Transaction. Важливим моментом є вказання типу, з яким даний екземпляр класу буде працювати. На рядку 40 об’єкт транзакції буде працювати з типом даних Dollar. Даний синтаксис вказаний в записі <Dollar>. На рядку 41 об’єкт транзакції буде працювати з типом даних Euro. Таким чином в транзакції з назвою dollarTrans поле каренсі буде містити значення типу Dollar, в той час

як в транзакції з назвою euroTrans поле currency буде містити тип даних Euro. Також в класі Transaction визначений метод getCurrency. Даний метод є простим прикладом використання шаблонів. Цей метод повертає значення з поля currency. Тип цього значення також вказаний як T, тобто залежить від тих даних, з якими працює дана транзакція. Таким чином ми створили шаблон класу, який незалежно від типу даних, має однакову поведінку та передбачену поведінку.

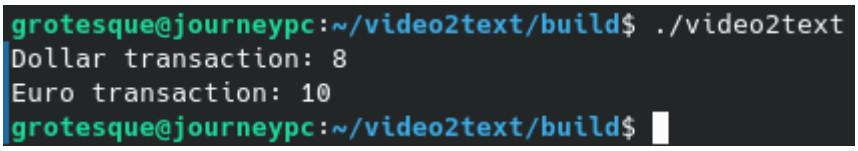


Рис. 3.23 — Робота шаблону класу

```cpp
#include <string>
#include <iostream>
#include <math.h>

template <typename T>
class Transaction
{
public:
  Transaction(T currency) : currency(currency) {}

  T getCurrency()
  {
    return currency;
  }

private:
  T currency;
};

class Dollar
{
public:
  int value;

  Dollar(int value) : value(value) {}
};

class Euro
{
public:
  int value;

  Euro(int value) : value(value) {}
};

int main(void)
{
  Dollar dollar{8};
  Euro euro{10};
  Transaction<Dollar> dollarTrans{dollar};
  Transaction<Euro> euroTrans{euro};
  std::cout << "Dollar transaction: " << dollarTrans.getCurrency().value << std::endl;
  std::cout << "Euro transaction: " << euroTrans.getCurrency().value << std::endl;
}

```

Рис. 3.24 — Використання шблону класу

Даний приклад є простим та демонстраційним, у реальних випадках на основі різних типів можна було б реалізувати по різному логіку метода getCurrency та інших методів всередині класа Transaction. Шаблонне програмування є однією з ключових особливостей мови програмування C++,

яка покращує гнучкість коду та його повторне використання. Цей механізм дозволяє створювати універсальні алгоритми та структури даних підртимуючи надійне використання різних типів. Шаблони активно використовуються в бібліотеці STL (Standart Template Libraries), яка надає легкий у використанні функціонал для роботи з багатьма типами.

3.2 ПАРАЛЕЛЬНЕ ПРОГРАМУВАННЯ C++

Також не можна обійти стороною концепцію під назвою паралелізм. Паралелізм в мові програмування C++ відноситься до можливості програми виконувати декілька задач, або потоків паралельно. Паралелізм є однією з важливих концепцій сучасного програмування, яка надає змогу реалізовувати програми, котрі можуть виконувати задачі паралельно. Використовуючи паралельність можна розпреділяти навантаження між різними потоками, тим самим покращуючи продуктивність роботи таких програм. В C++ існують механізми, які використовуються під час написання паралелізму, а саме thread, mutex, conditional variable та atomic operations. Thread (потік) — найменша одиниця під час виконаня програми. Потік представляє собою послідовність інструкцій які можна запланувати для виконання паралельно разом з іншими потоками.

```cpp
#include <iostream>
#include <thread>

void task(void)
{
  std::cout << "Thread works" << std::endl;
}

int main(void)
{
  std::thread t(task);
  std::cout << "Main works" << std::endl;
  t.join();
}
```

p

Рис. 3.25 — Використання потоку

На рисунку 3.25 зображено використання потоків. Для того, щоб створити потік, спочатку необхідно включити спеціальний файл заголовок thread. Після чого необхідно створити функцію, яка буде виконуватися в потоці. На 4 рядку визначена функція task, яка буде передана у вигляді аргумента під час створення потока. На 11 рядку відбувається ініціалізація потока. Для створення потоку необхідно передати функцію, яку цей потік буде виконувати. В даному випадку це функція task. Також у створеного потока обов'язково необхідно викликати метод join для того, щоб дочекатися кінця роботи потоку.

```
grotesque@journeypc:~/video2text/build$ ./video2text
Main works
Thread works
grotesque@journeypc:~/video2text/build$
```

Рис. 3.26 — Робота потоку

Mutex (Mutual Exclusion) один з механізмів, який використовується під час роботи з паралельним програмування. Даний механізм використовується в ситуаціях, коли необхідно захистити дані від одночасного звертання декількох

потоків до них. Даний механізм забезпечує те, що лише один потік за раз може використовувати дані для роботи.

```cpp
#include <iostream>
#include <thread>
#include <mutex>
#include <vector>

std::mutex mutex;

std::vector<int> nums;

void fill(const int size)
{
  std::lock_guard<std::mutex> lock(mutex);
  for (int i = 0; i < size; i++)
  {
    nums.push_back(i);
  }
}

void print(void)
{
  std::lock_guard<std::mutex> lock(mutex);
  for (size_t i = 0; i < nums.size(); i++)
  {
    std::cout << nums[i] << std::endl;
  }
}

int main(void)
{
  std::thread t1(fill, 10000);
  std::thread t2(print);
  t1.join();
  t2.join();
}
```

Рис. 3.27 — Використання mutex

На рисунку 3.27 зображено використання mutex разом з thread. Першим кроком необхідно включити header file mutex. Даний код підключення знаходиться на 3 рядку. Далі необхідно створити зміну типу mutex, яку необхідно буде передавати до спеціального об’єкта типу lock_guard. Дана глобальна змінна знаходиться на 6 рядку. На 8 рядку знаходиться вектор, який

буде зберігати числа. Даний вектор використовується у якості демонстрації mutex. Фукнкція fill, оголошена на 10 рядку, очікує у якості аргумента майбутню кількість елементів вектора. Далі дана функція запускає цикл на кількість ітерацій, які визначає передане значення. Цей цикл наповнює елементами вектор. Дана функція fill запускається в окремому потоці, що можна побачаити на 30 рядку, де сворюється потік з назвою t1, до якого передається функція fill, яка буде виконуватися в потоці та значення 10000, яке передається у якості аргументам до функції fill. Паралельно потоку t1 на 31 рядку створюється потік t2, який буде виконувати функцію print. Дана функція оголошена на 19 рядку. Вона займається тим, що виводить усі елементи вектора nums. Проблема одразу полягає в тому, що два потоки одночасно посилаються на один той самий ресурс. Перший поток заповнює вектор паралельно другому який намагається почати виводити елементи ще не до кінця наповненого вектора. Для вирішення цієї проблеми як раз підходить mutex. Додавши код std::lock_guard<std::mutex> lock(mutex) до функцій fill та print на 12 та 21 рядках відповідно ми забезпечуємо послідовність звертання до вектора nums. Спочатку відпрацює функція fill, яка наповнить вектор та звільне даний ресурс для використання потоком print.

Рис. 3.28 — Робота mutex з thread

Код на рисунку 3.27 займається тим, що спочатку в потоці t1 наповнить вектор nums 10000 елементами, після чого потік t2 виведе всі елементи вектора nums.

Conditional variable один з механізмів, який використовується під час

```cpp
#include <iostream>
#include <thread>
#include <mutex>
#include <condition_variable>
#include <string>

std::mutex mutex;
std::condition_variable condition;
std::string data;
bool ready = false;
bool processed = false;

void task()
{
  std::unique_lock<std::mutex> lock(mutex);
  condition.wait(lock, []{ return ready; });
  std::cout << "Task is processing" << std::endl;
  data += ", some additional data after processing";
  processed = true;
  std::cout << "Task is completed" << std::endl;
  lock.unlock();
  condition.notify_one();
}

int main(void)
{
  std::thread t1(task);
  data = "Some data";
  {
    std::lock_guard<std::mutex> lock(mutex);
    ready = true;
    std::cout << "Dats is ready for processing by task" << std::endl;
  }
  condition.notify_one();
  {
    std::unique_lock<std::mutex> lock(mutex);
    condition.wait(lock, []{ return processed; });
  }
  std::cout << "Data after processing by task: " << data << std::endl;
  t1.join();
}

```

Рис. 3.29 — Використання conditional_variable

роботи з паралельним програмуванням. Даний мехназім необхідний для того, щоб заблокувати один чи більше thread об’єктів до моменту, поки інший thread не завершить модифікувати спільні для усіх потоків дані та не сповістить conditional_variable об’єкт про завершення роботи.

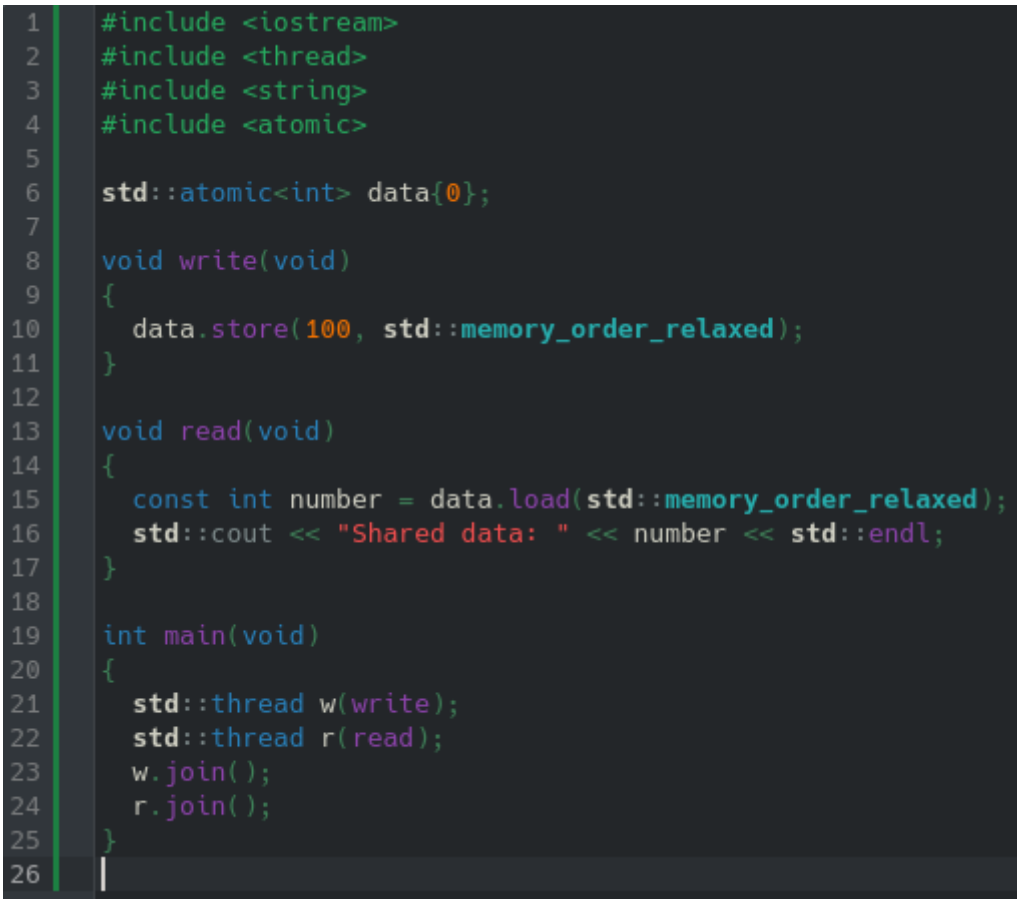
На рисунку 3.29 представлений приклад коду з використанням conditional_variable. В даному коді існує декілька глобальних змінних. Змінна data типу string зберігає текст, який буде модифікуватися протягом роботи потоку, в якому буде виконуватися функція task. Змінні ready та processed містять значення типу bool та виступають у ролі сигналів, які необхідні для сповіщення про те, що той, чи інший ресурс доступний для використання. Змінна mutex необхдіна для роботи механізмів по захищенню спільних даних від одночасного звертання. Змінна condition містить значення типу conditional_variable, через яке ми будемо керувати сповіщеннями про доступність того чи іншого ресурсу. Вхідною точкою є функція main. В даній функції першим кроком відбувається створення об’єкта типу thread, якому передається для виконання функція task. Після ініціалізації змінної t1 ініціалізуємо змінну data текстовим значенням “Some data”. Далі на рядках з 30 по 34 створюємо об’єкт тип lock_guard, яким визначаємо власника для доступу для спільних даних, змінюємо значення глобальної змінної ready на true, далі виводимо повідомлення “Data is ready for processing by task”, після чого звертаємося до об’єкта condition, у якого викликаємо метод notify_one. Ця логіка необхідна для того, щоб сповістити thread t1 про те, що тепер можна виконувати свою роботу. Справа в тому, що у функції task, яка оголошена 13 рядку існує код, який блокує виконання коду в даному потоці до моменту, поки умова для продовження виконання не буде правдива. Цей функціонал очікування роботи знаходиться на 16 рядку. Після запуску дана функція виводить повідомлення “Task is processing”, конкатенує зачення в змінній data, додаючи до нього текст “, some additional data after processing”, перемикає

змінну processed в стан правдивості, яка є умовою для того, щоб продовжити виконання коду всередині головного потоку, який, рядком 37 поставив себе на паузу до моменту настання необхідної умови продовження роботи, далі виводить текст “Task is completed”, викликає метод unlock у об’єкта lock та викликає метод notify_one у об’єкта condition. Далі виконнаня повертається до головного потоку, в якому виводиться повідомлення “Data after processing ” з рузльтатом роботи потоку, який знаходиться в змінній data.

```
grotesque@journeypc:~/video2text/build$ ./video2text
Dats is ready for processing by task
Task is processing
Task is completed
Data after processing by task: Some data, some additional data after processing
grotesque@journeypc:~/video2text/build$
```

Рис. 3.30 - Робота conditional_variable

Atomic ще один з механізмів, який надає стандартна бібліотека мови програмування C++ для роботи з паралельним програмуванням. Даний механізм забезпечує те, що операції над спільними даними є неподільними та стійкі до втручання з інших потоків. Atomic можна використоувати в ситуаціях, коли не портрібно явно вказувати механізми для синхронізації роботи потоків. Також Atomic надає змогу використовувати різні способи доступу до пам’яті.



```cpp
#include <iostream>
#include <thread>
#include <string>
#include <atomic>

std::atomic<int> data{0};

void write(void)
{
  data.store(100, std::memory_order_relaxed);
}

void read(void)
{
  const int number = data.load(std::memory_order_relaxed);
  std::cout << "Shared data: " << number << std::endl;
}

int main(void)
{
  std::thread w(write);
  std::thread r(read);
  w.join();
  r.join();
}
```

Рис. 3.31- Використанням atomic

На рисунку 3.31 наведений приклад використання atomic. На 6 рядку знаходиться глобальна зміна data, яка використовується для атомарних операцій над даними типу int. На 8 рядку оголошена функція write, яка через метод store змінює значення в data. Також за допомогою константи memory_order_relaxed вказуємо, яким саме чином хочемо звертатися до пам’яті. На 8 рядку оголошена функція read, яка заємається тим, що отримує поточні дані з змінної data, після чого виводить отримане значення. Функція main створює потоки для запису та читання, які запускає та очікує результат виконання.

```
grotesque@journeypc:~/video2text/build$ ./video2text
Shared data: 100
grotesque@journeypc:~/video2text/build$
```

Рис. 3.32 — Робота atomic

Розглянувши основний функціонал мови програмування C++, який використовується у розробці трансформації відео до тексту, варто відзначити інтсрумент, який вихідний код, написаний у редакторі, перетворює в програму, яку можна запустити та отримати бажаний результат роботи. Даний інтсрумент має назву G++. G++ - це комплітор, який розроблений спеціально для роботи з мовою програмування C++. Компілятор G++ входить до колекції компіляторів GNU (GNU Compiler Colection). Дана клолекція компіляторів для різних мов програмування є проєктом з відкритим вихідним кодом, яка підтримується інтузіастами з різних куточків планети. Використання GCC є безкоштовним. Даний проєкт доступний на багатьох платформах, включаючи Unix подібні операційні системи Linux, macOS та Windows (інструмент MinGW). G++ зазвичай використовується через командний рядок. Користувачі надають файли з вихідним кодом та додаткові флаги компіляції у якості аргументів для створення потрібного результату компіляції. Приклад використання компілятора G++: g++ hello_world.cpp -o hello_world. В даному рядку виконується звертання до компілятора G++. Коміплятор очікує на вхід файл з вихідним кодом, необхідний для компіляції. В даному випадку таким файлом є hello_world.cpp. Після чого знаходиться флаг (аргумент) -o для налаштування роботи компіятора. Даний флаг вказує компілятору, до якого файлу необхідно помістити результат компіляції. В результаті компіляції, якщо не виникло помилок, створюється файл hello_world, який можна через командний рядок запустити та переглянути результат роботи. Процеси, які виконує компілятор G++:

- Процеси компіляції включають декілька стадій: попередня обробка (preprocessing), компіляція (compilation), збірка (assembly), лінкування (linking).
- Попередня обробка відбувається програмою cpp (C Preprocessor). Дана програма виконує необхідні кроки для обробки, після чого передає код до компілятора.
- Компілятор генерує асемблерний код, який асемблер збирає в машинний код.
- Нарешті комонувальник (linker) поєднує скомпліьований код із необхідними бібліотеками для створення файлу, який можна запустити.

G++ підтримує широкий асортимент опцій, якій можуть використовуватися для контролю процесу компіляції. Ці опції можуть вказувати для необхідності використання рівні оптимізації коду, піключення різних бібліотек тощо. Даний рядок: g++ -03 -std=c++17 hello_world.cpp -o hello_world вказує за допомогою флагу -03 рівень оптимізації, -std=c++17 стандарт мови програмування C++, на якому ця програма написана, hello_world.cpp файл з вихідним кодом, -o hello_world файл для запуску програми в командному рядку. G++ зазвичай використовуєтьсся в поєднанні з такими системами збірки, як Make, CMake або іншими. G++ підтримує різні функції налагодження (debugging) та профілювання (profiling). Додавши параметер -g при використанні g++ в командному рядку можна включити додаткову інформацію до вихідного фалйу, яка полегшує налагодження файлу при використанні інструментів для налагодження, як GDB (GNU Debugger).

При роботі з великими проєктами стає все складніше підтримувати усі необхідні параметри для компіляції коду, використовувати сторонні бібліотеки,

тестувати код. Для цього існують інструменти, які значено полегшують дані процеси. В розробці трансформації відео до тексту використовується інструмент під назвою CMake. CMake крос-платформна система збірки з відкритим вихідним кодом, яка забезпечує унифікований процес для створення та керування проєктами програмного забезпечення. CMake широко використовується в розробці мовою програмування C++, але підтримує і інші мови програмування. CMake допомагає створювати незалежні від платформи файли збірки, які дозволяють створювати проєкти під різні операційні системи використовуючи різні компілятори. Основні характеристики CMake:

- Підтримка різних платформ: CMake генерує файли збірки для різних платформ, включаючи Linux, macOS і Windows. Це дозволяє розробникам один раз написати універсальні конфігуруючі файли CMakeLists.txt, після чого використовувати їх в різних системах.
- Підтримка різних мов програмування: CMake зазвичай асоціюється з C++ проєктами, однак інструмент підтримує широкий спектр мов програмування, включаючи C, Fortran, Python тощо. Ця підтримка робить можливим залучення CMake у проєктах, які одночасно використовують декілька мов програмування.
- Генерація конфігураційних файлів: CMake створює крос-платформні конфігураційні файли для збірок під різні системи та IDE, як Makefiles, Ninjia, Visual Studio solutions.
- Конфігураційні файли: CMake використовує CMakeLists.txt файли для налаштування побудови проєкту. Ці файли містять директиви та команди для позначення файлів з вихідним кодом, залежностей проєкту, налаштування компліятора та інших налаштувань пов’язаних зі збіркою проєкту.

- Незалежна збірка: CMake виконує усів процеси збірки проєкту в окремій папці, яку можна налаштувати. Завдяки такому підходу можливом мати різні збірки з різними налаштуваннями у різних каталогах, залишаючи головну директорю з вихідним кодом чистою та недоторканою.
- Модульність: CMake підтримує організацію проєктів модулями. Проєкт може бути розбитий на на меньші підпроєкти, кожен з яких містить свій CMakeLists.txt. Модульний підхід дозволяє краще організовувати код в проєкті.
- Тестування: CMake забезпечує вбудовану підтриму для створення та запуску тестів під різні платформи. Налаштування тестів та спеціальних тестових скриптів конфігурується всередині CMakeLists.txt файлів.
- Керування залежностями: CMake включає базову підтриму по пошуку та використанню зовнішних бібліотек у проєкті.

Розглянемо базові кроки для використання CMake в проєкті. Першим кроком необхідно створити CMakeLists.txt файл в гкорневій папці проєкту для налаштування проєкту та його властивостей. Далі необхідно запустити CMake для генерації файлів збірки проєкту. Цей крок конфігурує збірку проєкту, включає необхідні флаги для компіляції та налаштовує залежності проєкта. Команда для генерації виглядає наступним чином cmake path/to/project/source. Наступним кроком необхідно зібрати проєкт на основі згенерованих конфігураційних файлів. Результат збірки проєкту краще винести до окремої папки, наприклад build, щоб не забруднювати корневу папку проєкту. Для цього необхідно свторити папку build командою mkdir build. Після кроку з генераціією конфігураційних файлів у корневій папці проєкту та створенні папки для збірки можна зібрати проєкт. Необхідно перейти до папки build, після чого запустити всередині неї наступну команду cmake —build . .Дана команді помістить результати збірки в поточну папку. Після цього етапу в

папці build повинен з’явитися файл для виконнання, який можна запустити через командний рядок. Далі існує необов’язковий крок у вигляді встановлення файлів для виконання для використання на рівні системи. Для цього необхідно виконати команду cmake —install вказавши шлях до бінарних файлів.

```
1  cmake_minimum_required(VERSION 3.15...3.28)
2
3  project(HelloWorld VERSION 1.0)
4
5  add_executable(hello_world main.cpp)
6
```

Рис. 3.33 - Базовий CMakeLists.txt

На рисунку 3.33 наведений приклад базової конфігурації проєкту. В даному файлі зазанчаено на першому рядку налаштуванням cmake_minimum_required мінімальна необхідна версія CMake для роботи з нашим проєктом. На рядку 3 командою project вказуємо назву та версію проєкту. На 5 рядку командою add_executable вказуємо назву майбутнього файлу для виконання, в даному випадку файл буде називатися hello_ world і саме за цією назвою в командному рядку можна буде звернутися до цього файлу та запустити його. Окрім назви файлу для викононня необхідно вказати вхідну точку у коді, яка знаходиться в файлі main.cpp.

На цьому етапі використовуючи мову програмування C++ та інструмент збірки CMake можна розпочати розробку проєкту, який використовує функціонал стандартної бібліотеки std мови програмування C++. Однак трансформація відео до тексту в даній реалізації потребує двох зовнішних залежностей проєкту, а саме: OpenCV та Tesseract. Детальніше ознайомитися з OpenCV можна в розділі 2 Характеристики технології OpenCV, тому охарактеризуємо бібліотеку Tesseract. Tesseract — це рушій для оптичного розпізнавання символів (OCR) із відкритим вихідним кодом. OCR — це технологія, яка перетворює різні типи документів, як скановані паперові

документи, PDF файли або зображення, зняті цифровою камерою, у дані, доступні для редагування та пошуку. Tesseract спеціально розроблений для розпізнавання тексту в зображеннях, що робить його потужним інструментом для різноманітних програм. Tesseract в залежностях використовує бібліотеку Leptonica, яка необхідна для внутрішньої логіки роботи бібліотеки Tesseract. Також для роботи Tesseract в папку, в якій буде знаходитися рушій, необхідно помістити натреновані моделі LSTM.

Фінальним кроком необхідно використати інтсрумент Git та платформу GitHub. Через Git відбувається керування файлами під час розробки. GitHub використовується для створення віддаленого репозиторія, до якого локально, через Git, будуть відбуватися запити для виконання певних процедур в контексті файлів проєкту. Git — це розподілена система контролю версій, яка відстежує зміни в будь-якому наборі комп'ютерних файлів і зазвичай використовується для координації роботи програмістів, які спільно розробляють вихідний код під час розробки програмного забезпечення. Основними перевагами є швидкість роботи, забезпечення цілісності даних та підтримка розподілених нелінійних робочих процесів. GitHub — це платформа для розробників на основі штучного інтелекту, яка дозволяє розробникам створювати, зберігати свій код і керувати ним. Платформа використовує програмне забезпечення Git, забезпечуючи розподілений контроль версій Git плюс контроль доступу, відстеження помилок, запити функцій програмного забезпечення, керування завданнями, постійну інтеграцію та інформаційну документацію для кожного проєкту.

4 РОЗРОБКА ТРАНСФОРМАЦІЇ ВІДЕО ДО ТЕКСТУ

Розробка трансформації відео до тексту даної версії ведеться під Debian 12 GNU/Linux, тому усі команди та показані процеси будуть притаманії Linux, однак загальна концепція схожа з іншими операційними системами. Першим кроком перевіримо наявність компілятора G++ та системи контролю версій Git. За замовчуванням дані інструменти повині бути присутніми в дистрибутиві. Перед встановленням нового програмного забезпечення варто оновити усі пакети системи Debian та переконатися, щоб вони були останніх версій. Для цього в терміналі необхідно виконати команду sudo apt update && sudo apt upgraed. Після виконнаня даної команди повинна з’явитися інформація стосовного стану пакетів системи. Після перевірки стану пакетів системи спробуємо звернутися інструментів G++ та Git з простими запитами, наприклад отримати поточну версію інструментів, щоб перевірити їх роботу. Команда для отримання версії компілятора G++ буде виглядати g++ -v. Команда для отримання версії Git буде git -v.

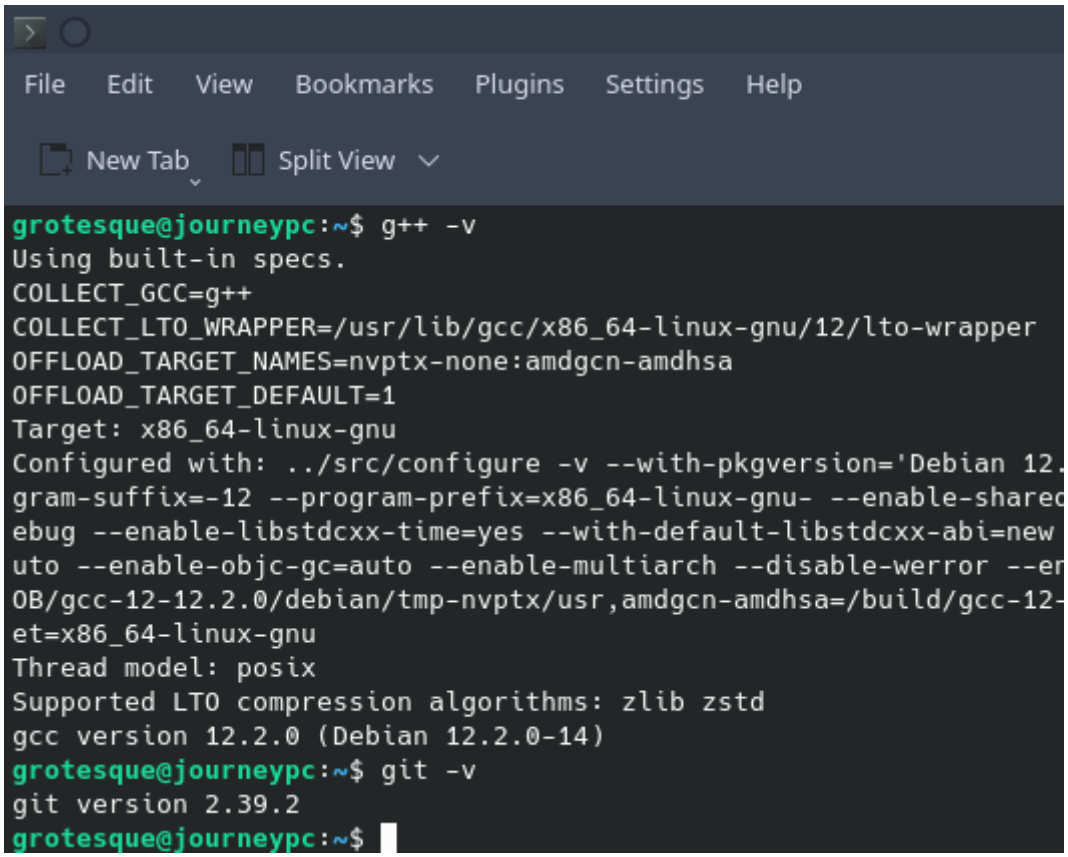


Рис. 4.1 - Перевірка роботи G++ та Git

На рисунку 4.1 представлено результат команд g++ -v та git -v. Даний вивід до терміналу показує необхідну інформацію стовоно встановленної версії інструмента та сигналізує про те, що інструменти правильно встановлені та готові до використання. У випадку, якщо ісінтрументи не були знайдені, одним із способів, як можна встановити дані інструменти, це прописати команду в терміналі sudo apt install g++ git.

Після перевірки роботи компілятора G++ та Git необхідно встановити систему збірки CMake та перевірити її роботу. Необхідна команда для цього sudo apt install cmake. Після чого перевірими версію інструмента CMake командою cmake --version.

```
grotesque@journeypc:~$ cmake --version
cmake version 3.25.1

CMake suite maintained and supported by Kitware (kitware.com/cmake).
grotesque@journeypc:~$
```

Рис. 4.2 — Перевірка роботи CMake

Тепер необхідно встановити локально бібіліотеки OpenCV та Tesseract. Дані бібліотеки можна встановити командами через термінал з репозиторію пакетів, а можна зкомпілювати з відкритого вихідного коду. Варіант з компіляцією з відриктого вихідного коду потребує більше часу, але в результаті надає найновіший функціонал, які запровадили розробники. Даний варіант встановлення бібіліотек відбвується шляхом компіляції вихідного коду і встанволення бінарних файлів до системи. Розпочнемо з бібліотеки Tesseract. Репозиторій з відкритим вихідним кодом даної бібліотеки знаходитсья за посиланням https://github.com/tesseract-ocr/tesseract. Першим ділом необхідно отримати локально вміст цього віддаленого репозиторя. Для цього необхідно завантажити на локальну машину архів з вмістом віддаленого репозиторія, після чого архів треба розкрити та усі дані помістити в необхідну папку. Для цього в терміналі необхідно виконати команду unzip ~/Downloads/tesseract.zip ~/. Усі команди виконуються в домашній папці користувача, тому щоб перевірити, чи створилася папка з tesseract, необхідно виконати в терміналі команду ls та побачити в поточній дерикторії новостворену папку tesseract. Далі необхідно виконати набір команд, які представленні в репозиторії GitHub Tesseract під конкретну систему, в даному випадку Debian. Бібліотека Tesseract залежить від деяких компонентів, а саме компілятора під мови програмування C та C++, інструментів автоматизації GNU: autoconf, automake, libtool, інструмента pkg-config, бібліотки Leptonica, необов'язкових залежностей zlib, libpng, libjpeg, libtiff, giflib, openjpeg, webp, archive, curl. Якщо дані біліотеки не встановлені, тоді необхідно виконати наступні команди в терміналі:

- sudo apt install g++ або sudo apt install clang++

- sudo apt install autoconf automake libtool
- sudo apt install pkg-config
- sudo apt install libpng-dev
- sudo apt install libjpeg8-dev
- sudo apt install libtiff5-dev
- sudo apt install zlib1g-dev
- sudo apt install libwebpdemux2 libwebp-dev
- sudo apt install libopenjp2-7-dev
- sudo apt install libgif-dev
- sudo apt install libarchive-dev libcurl4-openssl-dev

Якщо необхідно необхідно встановити інструменти для тренування, тоді необхідно виконати наступні команди:

- sudo apt install libicu-dev
- sudo apt install libpango1.0-dev
- sudo apt install libcairo2-dev

Також необхідно встановити бібліотеку Leptonica та переконатися, що файли заголовків для розробки встановлені раніше, ніж будемо компілювати бібліотеку Tesseract. Команда для встановлення Leptonica виглядає наступним чином: sudo apt install libleptonica-dev.

Тепер перейдемо до компіляції та встановлення до системи Tesseract. Спочатку необхідно перейти до папки tesseract в домашній дерикторії користувача, або в тій, в яку склонували з репозиторія tesseract. Виконати це можно в терміналі командою cd ~/tesseract. Після чого необхідно виконати команди:

- ./autogen.sh
- ./configure
- make -j6
- sudo make install
- sudo ldconfig
- make training -j6
- sudo make training-install

Останнім кроком необхідно завантижити файл LSTM модель для англійської мови, оскільки дана версія трансформації відео до тексту працює з текстом написаним англійською мовою. Для цього перейдемо за посиланням https://github.com/tesseract-ocr/tessdata.git. В даному репозиторії GitHub знаходяться натреновані моделі для різних мов, нам необхідно знайти файл eng.traineddata та завантажити його. Це можна виконати як в браузері руками, так і через термінал утилітою curl. Після цього необхідно файл eng.traineddata помістити до папки tessdata. Для цього необхідно виконати команду в терміналі mv ~/Downloads/eng.traineddata /usr/local/share/tessdata/.

Після усіх кроків необхідно перевірити роботу Tesseract. Для цього виконаємо команду tesseract -v та поглянемо результат роботи.

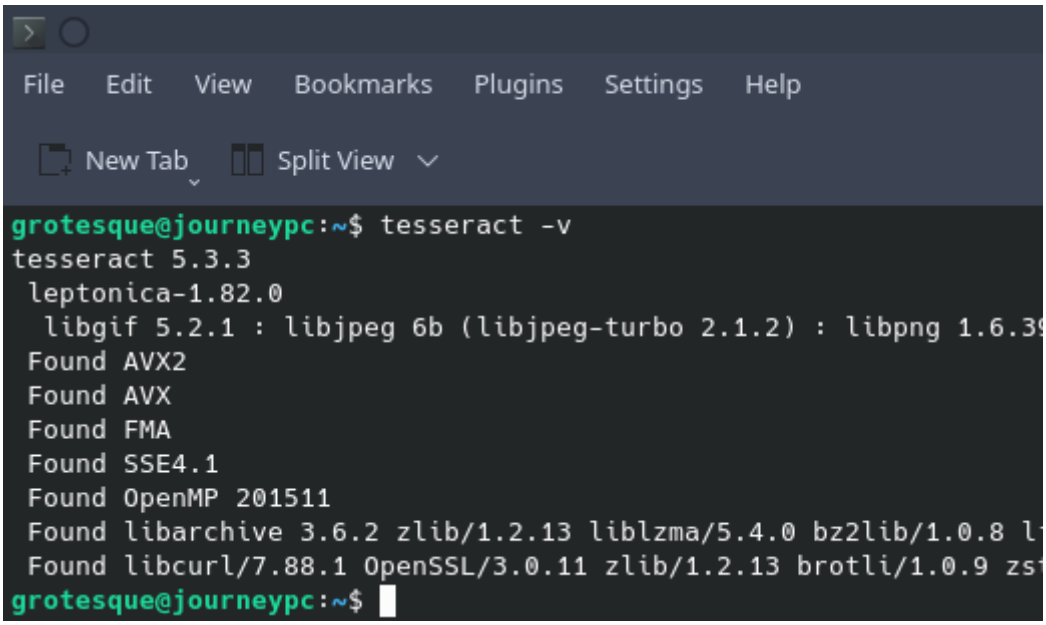


Рис. 4.3 — Перевірка роботи tesseract

Тепер перейдемо до компіляції з відкритого вихідного коду бібліотеки OpenCV. Переходимо за посиланням https://github.com/opencv/opencv.git до GitHub репозиторія та завантажуємо архів з усіма файлами віддаленого репозиторія, або можемо виконати даний запит через термінал утилітами wget або curl. Після чого розархівуємо файл та покладемо увесь вміст до папки opencv-4.x в домашній папці користувача командою unzip ~/Downloads/opencv-4.x ~/. Перевіримо створення папки opencv-4.x в корневій папці користувача командою ls. Далі необхідно перейти до папки opencv-4.x та створити всередині папку build. Виконати це можно прописавшин наступні команди:

- cd ~/opencv-4.x
- mdkir -p build && cd build

Після створення та переходу до папки build необхідно сконфігурувати даний проєкт утилітою CMake. Після чого на основі конфігураційних файлів дану бібліотеку можна скомпілювати. Наступні команди виконають генерацію необхідних збірочних скриптів та скомпілюють проєкт.

- cmake ../opencv

- make -j6

Після успішної компіляції всередині папки build повинні з’явитися папки bin та lib. Перевірити наявність цих папо можна командами.

- ls bin
- ls lib

Останнім кроком необхідно бінарні файли встановити на рівні системи. Наступна команда помістить з невеликими налаштуваннями бібліотеки та бінарні файли бібліотеки opencv за шляхом за замовчуванням /usr/local. Даний варіант не рекомендується, оскільки можуть виникнути конфлікти з інишими системними пакетами, також через те, що таким шляхом opencv не буде інтегрована в реєстр системних пакетів, відповідно просто так автоматично opencv не вдасться видалити. Для зміни місця вістановлення opencv необхідно використати флаг зі значенням -DCMAKE_INSTALL_PREFIX=$HOME/.local. Дана комбінація флага і значення встановить opencv за вказаним шляхом. Проте, якщо в системі один користувач і немає конфліктів з іншими пакетами, opencv можна встановити глобально для усієї системи. Для встановлення необхідно одну з двух наступних команд:

- sudo make install: встановить opencv глобально до /usr/local
- sudo make install -DCMAKE_INSTALL_PREFIX=$HOME/.local: встановить opencv до локальної дерикторії поточного користувача.

Після усіх команд перевіримо успішне встановлення бібліотеки opencv. Для цього необхідно перевірити вміст системної папки за замовчування, куди були встановлені файли opencv, або вміст тієї папки, яку вказали під час встановленнями. Виконати це можна наступними командами:

- ls /usr/local/bin | grep opencv або ls ~/.local/bin | grep opencv

- ls /usr/local/lib | grep opencv або ls ~/.local/bin | grep opencv

Дані команди перевірять вміст папок bin та lib і виведуть список лише тих файлів, у яких в назві зустрічається слово opencv.

```
grotesque@journeypc:~$ ls /usr/local/bin/ | grep opencv
opencv_annotation
opencv_interactive-calibration
opencv_model_diagnostics
opencv_version
opencv_visualisation
setup_vars_opencv4.sh
grotesque@journeypc:~$ ls /usr/local/lib/ | grep opencv
libopencv_calib3d.so
libopencv_calib3d.so.408
libopencv_calib3d.so.4.8.0
libopencv_core.so
libopencv_core.so.408
libopencv_core.so.4.8.0
libopencv_dnn.so
libopencv_dnn.so.408
libopencv_dnn.so.4.8.0
```

Рис. 4.4 — Перевірка встановлення бібліотеки opencv

Тепер, коли є усі необхідні компоненти для реаліазції проєкту трансформації відео до тексту, можна перейти до написання коду. Першим кроком створимо репозиторій на платформі GitHub, у якості назви репозиторій буде містити video2text. Після створення репозиторія платформа GitHub видасть способи, як підключитися локально до даного віддаленого репозиторія. Існують 2 способи: https та ssh. Відповідно обирається необхпідний тип підключення та копіюється адреса підключення, яка необхідна для подальшого налаштування локальної роботи з Git. Цей етап є важливим, оскільки платформа GitHub переймається безпекою коду користувачів, тому від того, яким способом локальний репозиторій буде ппов’язаний з віддаленим, буде залежати спосіб автентифікації. У випадку, якщо обраний варіант є https з’єднанням, найпростішим способом буде скористатися GiHub Manager, або GitHub Desktop, або Personal Access Token (PAT). GitHub Manager виконає автентифікацію на вашому пристрої, після чого не доведеться кожного разу під

час відправлення змін до віддаленого репозиторія проходити перевірку на особистість. Варіант з використанням GitHub Desktop також вирішує цю проблему, до того ж ще й є зручним візуальним інструментом для керування усіма необхідними операціями всередині репозиторія. Варіант з персональним токеном доступу потребує більше кроків, але є гнучкішим та не потребує встановлення будь-якого додаткового програмного забезпеченням. Даний варіант трансформації відео до тексту використовує ssh з'єднання. Для автентифікації в такому з'єднанні необхідно згенерувати пару локальних ключів, після чого прив'язати локальний публічний ключ до вашого акаунта в GitHub, після чого необхідно пройти перевірку та отримати змогу також відправляти зміни до віддаленого репозиторія. Детальніше стовоно налаштувань автентифікації за посиланням https://docs.github.com/en/authentication/connecting-to-github-with-ssh.

Після створення віддаленого репозиторія та налаштування автентифікації перейдемо до локальної папки проєкту. В цій папці створимо додаткові папки та помістимо усі необхідні файли. Назва папки буде video2text, тому створимо дану папку командою:

- mkdir -p video2text

Це буде корнева папка проєкту, в якій будуть міститися усі необхідні папки та файли для роботи проєкту. Під час роботи всередині даної дерикторії можуть виникати конфлікти через відсутність доступу до контенту всередині даної дерикторії. Для рішення цієї потенційної проблеми необхідно системі вказати, що поточний користувач повністю володіє усіма правами для користування потрібною папкою. Команда для цього виглядає наступним чином:

- sudo chown -R $USER ~/video2text

Після створення корневої папки video2text та налаштування прав доступу до ціє дерикторії, помістимо всередину папки video2text папки src та build. Для цього виконаємо команду:

- mkdir -p src build

Папка src буде зберігати вихідний код проєкту, написаний мовою прогармування C++. Папка build буде використовуватися для зберігання усіх необхідних файлів зборщика проєкта CMake та бінарний файл програми, який можна запустити. Далі необхідно створити два файли: .gitignore та CMakeLists.txt. Команди для цього наступні:

- touch ~/video2text/.gitignore
- touch ~/video2text/src/CMakeLists.txt

Файл .gitingore необхідний для того, щоб інструмент Git ігнорував певні файли. Такі файли, або дерикторії, можна вказати всередині даного файлу. Завдяки цьому файлу маємо змогу ігнорувати та не відправляти до віддаленого репозиторія зазначені файли та папки. Одразу до файла .gitignore помістимо назви папок build та .cache, після чого збережемо зміни у файлі. Тепер Git буде ігнорувати вміст вказаних папок та не відправить їх до віддаленого репозиторія.

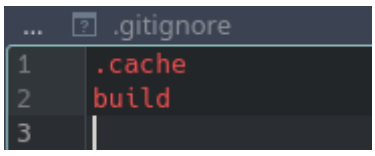


Рис. 4.5 — Налаштування .gitignore

Файл CMakeLists.txt всередині папки src є вхідною точкою для конфігурації зборщика проєкту CMake. Даний файл виглядає наступним чином:

```
1  cmake_minimum_required(VERSION 3.15...3.28)
2  project(Video2Text VERSION 1.0)
3
4  add_library(video2text_compiler_flags INTERFACE)
5  target_compile_features(video2text_compiler_flags INTERFACE cxx_std_11)
6
7  set(gcc_like_cxx "$<COMPILE_LANG_AND_ID:CXX,ARMClang,AppleClang,Clang,GNU,LCC>")
8  set(msvc_cxx "$<COMPILE_LANG_AND_ID:CXX,MSVC>")
9  target_compile_options(video2text_compiler_flags INTERFACE
10   "$<${gcc_like_cxx}:$<BUILD_INTERFACE:-Wall;-Wextra;-Wshadow;-Wformat=2;-Wunused>>"
11   "$<${msvc_cxx}:$<BUILD_INTERFACE:-W3>>"
12 )
13
14 set(CMAKE_EXPORT_COMPILE_COMMANDS ON)
15
16 find_package(OpenCV REQUIRED)
17 MESSAGE(STATUS "OpenCV included dirs ${OpenCV_INCLUDE_DIRS}")
18 MESSAGE(STATUS "OpenCV linked libraries ${OpenCV_LIBS}")
19
20 find_package(PkgConfig REQUIRED)
21 pkg_search_module(Tesseract REQUIRED tesseract)
22 pkg_search_module(Leptonica REQUIRED lept)
23
24 add_subdirectory(video)
25 add_subdirectory(image)
26 add_subdirectory(ocr)
27
28 add_executable(video2text main.cpp)
29
30 target_link_libraries(video2text PUBLIC video image ocr video2text_compiler_flags)
31
32 target_include_directories(video2text PUBLIC ${PROJECT_BINARY_DIR})
33
```

Рис. 4.6 - Конфігурація CMakeLists.txt в дерикторії src

На рисунку 4.6 представлено налаштування вхідної точки для зборщика проєкта CMake. Першим кроком для вхідного файла необхідно вказати обов'язкові команди, які вкажуть, яка версія CMake необхідна для збірки проєкта, а також назву та версію проєкта. В даному файлі необхідні команди знаходяться на рядках 1 та 2:

- cmake_minimum_required(VERSION 3.15...3.28): дана команда вказує, що для збірки даного проєкта необхідний CMake версії від 3.15 до 3.28.
- project(Video2text VERSION 1.0): дана команда встановлює назву та версію проєкта.

На 4 та 5 рядках створюється допоміжна бібліотека video2text_compiler_flags, яка позначена як INTERFACE. Дана бібліотека не буде компілюватися, а буде поширювати налаштування компіляції для інших бібліотек в проєкті:

- add_library(video2text_compiler_flags): додає бібліотеку video2text_compiler_flags та позначає її як INTERFACE.
- target_compiler_features(video2text_compiler_flags INTERFACE cxx_std_11): вказує, що дану бібліотека використовує 11 стандарт C++.

З 7 по 12 рядки йдуть налаштування використання компіляторів та налаштування самої компіляції:

- set(gcc_like_cxx "$<COMPILE_LANG_AND_ID:CXX,ARMClang,AppleClang,Clang,GNU,LCC>") set(msvc_cxx "$<COMPILE_LANG_AND_ID:CXX,MSVC>"): вказують використання компіляторів
- target_compile_options(video2text_compiler_flags INTERFACE "$<${gcc_like_cxx}:$<BUILD_INTERFACE:-Wall;-Wextra;-Wshadow;-Wformat=2;-Wunused>>" "$<${msvc_cxx}:$<BUILD_INTERFACE:-W3>>"): позначає додаткові правила при компіляції, наприклад буде відображатися попередження при невикористанній змінній

На 14 рядку знаходиться команда, яка необхідна для створення спеціального файла, в якому будуть вказані налаштування компіляції. Даний файл необхідний для роботи LSP сервера в текстовому редакторі.

- set(CMAKE_EXPORT_COMPILE_COMMANDS ON): експортує команди компіляції

З 16 по 18 рядок відбувається підключення бібліотек OpenCV до проєкту:

- find_package(OpenCV REQUIRED): команда вказує, що обов’язково треба знайти файли пакет opencv.
- MESSAGE(STATUS "OpenCV included dirs ${OpenCV_INCLUDE_DIRS}"): допоміжна команда, яка виводить усі включені дерикторії opencv.
- MESSAGE(STATUS "OpenCV linked libraries ${OpenCV_LIBS}"): допоміжна команда виводить пов’язані біліотеки opencv.

З 20 по 22 рядки відбувається підключення бібліотеки Tesseract до проєкту. Дане підключення відрізняється тим, що Tesseract не має конфігураційного файлу CMake, завдяки якому можна було б автоматично знайти необхідну бібліотеку командою find_package, як у випадку з OpenCV. Тому спочатку відбувається підключення утиліти PkgConfig, після чого PkgConfig дає змогу використати pkg_search_module для підключення необхідних модулів:

- find_package(PkgConfig REQUIRED): команда вказує, що обов’язково необхідно знайти пакет PkgConfig.
- pkg_search_module(Tesseract REQUIRED tesseract): команда вказує, що обов’язково треба знайти модуль Tesseract.
- pkg_search_module(Leptonica REQUIRED lept): команда вказує, що обо’вякзково необхідно знайти модуль Leptonica.

З 24 по 26 рядок йдуть команди, які вказують зборщику проєкта CMake про те, що даний проєкт включає піддерикторії:

- add_subdirectory(video): команда додає піддерикторію video.
- add_subdirectory(image): команда додає піддерикторію image.
- add_subdirectory(ocr): команда додає піддерикторію ocr.

З 28 по 32 рядки йдуть команди, які встановлюють назву бінарного файла — результат компіляції та пов’язані з цим файлом бібліотеки.

- add_executable(video2text main.cpp): команда вказує, що вхідною точкою для виконнання програми слугує файл main.cpp та після компіляції буде створений файл video2text, який можна буде запустити через термінал.
- target_link_libraries(video2text PUBLIC video image ocr video2text_compiler_flags): команда пов’язує об’єкт компіляції з необхідними бібліотеками для роботи.
- target_include_directories(video2text PUBLIC ${PROJECT_BINARY_DIR}): команда вказує компілятору, де шукати файли для включення.

З головною конфігурацією зборщика CMake завершили, тепер необхідно створити піддерикторії проєкту та заповнити їх необхідними файлами. Даний проєкт включає в себе три піддерикторії: video, image, ocr. Необхідно перейти до папки src та виконати в терміналі наступну команду:

- cd src && mkdir -p video image ocr: переходимо до дерикторії src, після чого сторюємо дерикторії video, image та ocr.

Тепер в кожну з дерикторій необхідно помістити наступні файли: файл заголовок, файл з реалізацією коду та конфігураційний файл для даної дерикторії. Для цього необхідно виконати наступні команди в терміналі:

- cd video && touch video.hpp video.cpp CMakeLists.txt.
- cd image && touch image.hpp image.cpp CMakeLists.txt.
- cd ocr && touch ocr.hpp ocr.cpp CMakeLists.txt.

Тепер залишилося додати вхідну точку проєкту — файл main.cpp. Команда в терміналі виглядає наступним чином:

- cd src && touch main.cpp

Після виконнаня описаних команд файлова структура проєкту повинна виглядати наступним чином: дерикторія src повинна містити конфігураційний файл CMakeLists.txt, файл main.cpp та три дерикторії, в кожній з яких повинно знаходитися по три файли.

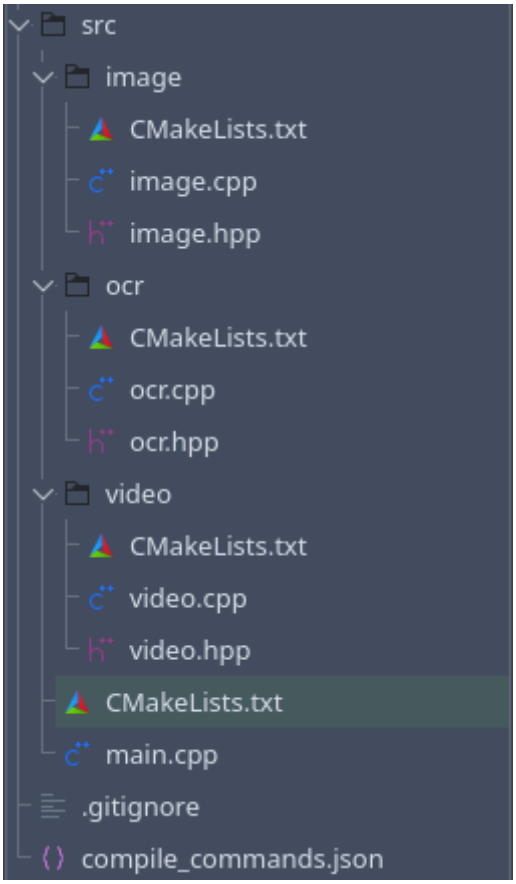


Рис. 4.7 — Файлова структура проєкта

Тепер розпочнемо помодульно реалізовувати частини проєкту для трансформації відео до тексту. Вхідною точкою є файл main.cpp в якому реалізовані усі необхідні функції для роботи з відео та пошуку тексту, які використовують функціонал з інших модулів.

```cpp
#include <fstream>
#include <filesystem>
#include <thread>
#include "video/video.hpp"
#include "image/image.hpp"
#include "ocr/ocr.hpp"

void videoHandler(const cv::Mat& frame);

void extractText(const cv::Mat& frame, std::vector<cv::Rect>& contours);

int main(void)
{
  const cv::Mat frame = video::live(videoHandler);
  std::vector<cv::Rect> contours = image::findContours(frame);
  extractText(frame, contours);
}

void videoHandler(const cv::Mat& frame)
{
  const cv::Mat frameClone = frame.clone();
  const std::vector<cv::Rect> contours = image::findContours(frameClone);
  image::renderContours(frameClone, contours);
  cv::imshow("Frame", frameClone);
}

void extractText(const cv::Mat& frame, std::vector<cv::Rect>& contours)
{
  const std::string outputPath = std::filesystem::current_path().generic_string() + "/found_text.txt";
  std::filesystem::remove(outputPath);
  std::ofstream output(outputPath);
  for (std::vector<cv::Rect>::reverse_iterator i = contours.rbegin(); i != contours.rend(); ++i)
  {
    const cv::Mat roi = frame(*i);
    const std::string text = ocr::findText(roi);
    output << text << std::endl;
  }
  output.close();
}

```

Рис. 4.8 - Функціонал main.cpp

На рисунку представлений код з вхідної точки усієї логіки проєкту. Розпочнемо з функції main, яка знаходиться з 12 по 17 рядки. Дана функція не містить параметрів, це позначено словом void в списку параметрів функції. На 14 рядку відбуваєть виклик функції live із модуля video. До реалізації цього модуля перейдемо пізніше. В якості аргумента до функції live передається посилання на функцію videoHandler. Функція videoHandler в даному випадку виступає в ролі callback функції. Сенс такої функції полягає в тому, що вона може бути оголошена в одному місці коду (одному файлі), передатися у якості аргумента, після чого бути викликаною в іншій частині коду (іншому файлі). Після роботи функції live вона поверне останній кадр відео, на який буде посилатися змінна frame. На 15 рядку відбувається виклик функції findContours із модуля image.

До реалізації цього модуля перейдемо пізніше. Дана фунція приймає параметр кадр відео, після чого повертає вектор із об’єктів типу cv::Rect. Даний вектор містить потенційно необхідні частини зображення з відео, на якому може знаходитися текст. Наступним кроком на 16 рядку йде виклик функції extractText. До даної функції у якості аргументів передається останній кадр відео та потенційні облсаті з текстом на цьому кадрі. Ця функції займається отриманням тексту з кадру, після чого зберігає увесь знайдений текст до текстового файлу. Резюмуючи: функція main запускає відео стрім в реальному часі, кожен кадр відео оброблюється функцією videoHandler, по завершенню відео отримаємо отсанній кадр даного відео, після чого передаємо його до функції findCountours, яка поверне усі потенційні області з текстом на цьому кадрі, після чого передаємо останній кадр відео та потенційні області до функції extractText, яка проходиться по всім областям зображення, функцією findText із модуля ocr знаходить текст та записує знайдений текст до текстового файлу.

Тепер перейдемо до функції videoHandler, яка визначена з 19 по 25 рядок. Функція videoHandler оброблює кожен кадр відео реального часу, робить клон цього кадру, знаходить потенційні області з текстом на кадрі, малює ці області на клоні кадру, після чого даний клон відображає в графічному вікні. Дана функція нічого не повертає та містить один параметер:

- const cv::Mat& frame: функція videoHandler очікує посилання на кадр відео.
- const cv::Mat frameClone = frame.clone(): клонує отриманий кадр для подальших маніпуляцій.
- const std::vector<cv::Rect> contours = image::findContours(frameClone): викликається функція findContours з модуля image, до якої передається

клон кадру. Функція findContours повертає вектор прямокутників, на які посилається змінна contours.

- image::renderContours(frameClone, contours): киликаємо функцію renderContours, до якої передаємо клон кадру та знайдені контури. Дана функція намалює контури на зображенні.
- cv::imshow("Frame", frameClone): функція із бібліотеки OpenCV для створення графічного вікна, в якому буде відображатися передане зображення.

Залишилося розглянути функцію extractText, яка визначена з 27 по 38 рядок. Дана функція приймає зображення з відео та контури з текстом на зображенні. Функція extractText видаляє файл found_text.txt, в якому знаходився текст останнього запуску програми, після чого створює файл found_text.txt, до якого запишиться знайдений текст поточного результату виконнаня програми. Далі запускається цикл for, який проходиться з кінця до початку по кожному елементу вектора, передає даний елемент у якості аргумента до функції findText модуля ocr, після чого отриманий текст записує до файлу. По завершенню роботи цикла викликається метод close для сигналізації про те, що робота з файлом завершена та можно звільнити виділені для цього ресурси.

- const std::string outputPath = std::filesystem::current_path().generic_string() + "/found_text.txt": змінна, яка містить шлях в системі до файла found_text.txt.
- std::filesystem::remove(outputPath): видаляємо файл за вказаним шляхом.
- std::ofstream output(outputPath): створюємо файл за вказаним шляхом, до якого в циклі будемо робити запис.

- for (std::vector<cv::Rect>::reverse_iterator i = contours.rbegin(); i != contours.rend(); ++i)

  {

  const cv::Mat roi = frame(*i);

  const std::string text = ocr::findText(roi);

  output << text << std::endl;

  }: проходимося від останнього до першого елемента в векторі, кожен елемент є певною областю на зображенні, тому записом frame(*i) вирізаємо певну область зображення, яка нас цікавить, передаємо дану область зображення до функції findText, яка повертає текст, який виявила на переданій області зображення, записуємо виявлений текст до файла found_text.txt.

- output.close(): по завершенню роботи цикла звільняємо ресурси для роботи з файлом.

Тепер, коли ми розглянули функціонал у файлі main.cpp, можемо рухатися далі та розібрати функціонал із модулів video, image та ocr. Усі вони ідентичні на рівні файлової структури. Переглянемо файл CMakeLists.txt всередині дерикторії video.

```
1  add_library(video video.cpp)
2
3  target_include_directories(video INTERFACE ${CMAKE_CURRENT_SOURCE_DIR})
4
5  target_link_libraries(video PUBLIC ${OpenCV_LIBS} video2text_compiler_flags)
6
```

Рис. 4.9 — CmakeListst.txt для бібліотеки video

На рисунку 4.9 представлено налаштування конфігураційного файла CMakeLists.txt для зборщика CMake для дерикторії video. Даний файл вказує ціль компіляції, файл для компіляції, які дерикторії включати, коли бібліотека використовується, які бібліотеки підключити для компіляції об’єкта.

- add_library(video video.cpp): команда вказує скомпілювати файл video.cpp та на виході отримати об’єкт video.
- target_include_directories(video INTERFACE ${CMAKE_CURRENT_SOURCE_DIR}): вказує необхідні дерикторії для включення для компіляції поточного об’єкта video.
- target_link_libraries(video PUBLIC ${OpenCV_LIBS} video2text_compiler_flags): вказує бібліотеки або флаги, які необхідно використати для лінкування даного об’єкта

```
1  #include <iostream>
2  #include <functional>
3  #include <opencv2/imgcodecs.hpp>
4  #include <opencv2/imgproc.hpp>
5  #include <opencv2/video.hpp>
6  #include <opencv2/videoio/videoio.hpp>
7  #include <opencv2/highgui.hpp>
8  #include "../image/image.hpp"
9
10 namespace video
11 {
12   cv::Mat live(std::function<void(const cv::Mat&)>);
13 }
14
```

Рис. 4.10 - Оголошення функціоналу video.hpp

На рисунку 4.10 представлений деклараційний файл video.hpp, в якому оголошений увесь функціонал, який буде реалізований у файлі video.cpp, а також включені необхідні для реалізації функціоналу деклараційні файли:

#include <iostream> - підключаємо бібліотеку iostream.

#include <functional> - підключаємо бібліотеку functional.

#include <opencv2/imgcodecs.hpp> - підключаємо бібліотеку imgcodecs з opencv для кодування зображень.

#include <opencv2/imgproc.hpp> - підключаємо бібліотеку imgproc з opencv для обробки зображень.

#include <opencv2/video.hpp> - підключаємо бібліотеку video з opencv для роботи з відео.

#include <opencv2/videoio/videoio.hpp> - підключаємо бібліотеку videoio з opencv для роботи з вхідними/вихідними відео потоками пристроя.

#include <opencv2/highgui.hpp> - підключаємо бібліотеку highgui з opencv для роботи з графічним інтерфейсом.

#include "../image/image.hpp" - підключаємо власну бібліотеку image для роботи з зображенням.

```
namespace video
{
  cv::Mat live(std::function<void(const cv::Mat&)>);
}
```

Оголошуємо простір імен video, в якому знаходиться уся необхідна логіка, яка стосується роботи з відео. В даному просторі оголошуємо функцію live. Функція live очікує посилання на об’єкт типу Mat. Даний тип береться з бібліотеки OpenCV. Функція live повертає значення типу Mat.

На рисунку 4.11 представлена реалізація модуля video. В даному модулі реалізована функція live, яка використовує VideoCapture клас із бібліотеки

OpenCV для ороботи з камерою девайса. Функція live в нескінченному циклі звертається до камери девайса та отримує зображення з камери. На кожне отримання зображення функція live викликає callback функцію onFrame, до якої передає поточне зображення. Після завершення роботи функція live повертає останнє отримане зображення від камери.

```
cv::Mat frame;

cv::VideoCapture videoCapture;

const int deviceID = 0;

const int apiID = cv::CAP_ANY;

videoCapture.open(deviceID, apiID);
```

Даний блок коду створює об’єкт типу Mat в змінній frame, до якого записується зображення, отримане з камери девайсу. Після чого іде створення об’єкта типу VideoCapture в змінній videoCapture. Даний клас реалізований бібілотекою OpenCV та дозволяє звернутися до камери пристрою та отримати зображення. Далі ідуть змінні deviceID та apiID, які необхідні у якості аргументів для налаштування взаємодії з камерою пристрою. Рядок videoCapture.open(deviceID, apiID) виклиткає метод open у об’єкта videoCapture. Дана функція сповіщає систему про те, що програмі необхідно розпочати взаємодію з виявленою камерою девайсу.

```cpp
#include "video.hpp"

cv::Mat video::live(const std::function<void(const cv::Mat&)> onFrame)
{
  cv::Mat frame;
  cv::VideoCapture videoCapture;
  const int deviceID = 0;
  const int apiID = cv::CAP_ANY;
  videoCapture.open(deviceID, apiID);
  if (!videoCapture.isOpened())
  {
    std::cerr << "Unable to open camera\n";
    std::exit(-1);
  }
  for (;;)
  {
    videoCapture.read(frame);
    if (frame.empty())
    {
      std::cerr << "Blank frame\n";
      break;
    }
    onFrame(frame);
    if (cv::waitKey(5) >= 0)
    {
      break;
    }
  }
  return frame;
}

```

Рис. 4.11 — Реалізація функціоналу video.cpp

```
if (!videoCapture.isOpened())
 {
   std::cerr << "Unable to open camera\n";
   std::exit(-1);
 }
```

Після відкриття камери пристрою необхідно переконатися, чи була спроба скористатися камерою девайсу успішною. Даний блок коду звертається до об’єктка videoCapture, у якого викликає метод isOpened. У разі, якщо виклик метода open у об’єкта videoCapture пройшов без помилок, метод isOpen повинен повернути true, інакше false. У випадку, якщо відкриття камери не було успішним, метод isOpened поверне false. У такому випадку оператор ! значення false перетворить на протилежне true. Якщо дана перевірка буде true, тоді блок коду всередині if запуститься. Блок коду всередині перевірки виведе повідомлення з помилкою та завершить виконання програми.

```
for (;;)
{
   videoCapture.read(frame);
   if (frame.empty())
   {
    std::cerr << "Blank frame\n";
    break;
   }
   onFrame(frame);
   if (cv::waitKey(5) >= 0)
   {
    break;
   }
}
```

return frame;

Даний блок коду запускає нескінченний цикл, який завершиться лише за певних умов. У даному циклі відбувається наступна логіка: у об’єкта videoCapture викликаємо метод read, до якого передаємо змінну frame, яка посилається на облсать в пам’яті, до якої записується отиримане зображення з камери. Після спроби отримати зображення з камери виконається перевірка, чи не є порожнім отримане зображення. У випадку, якщо зображення порожнє, виводиться помилка та завершується виконання циклу ключовим словом break; Якщо вдалося отримати зображення з камери, викликається функція onFrame, до якої передається отримане зображення. Після виклику функції onFrame знаходиться перевірка, яка звертається до функції бібліотеки OpenCV waitKey. Дана функція приблизно очікує вказану кількість мілісекунд на подію, коли відбудеться натискання на будь-яку клавішу, після чого поверне код натиснутої клавіші. Якщо клавіша була натиснута, тоді функція поеврне код клавіші, яку натиснули, який буде більше або дорівнювати нулю. Натиск на клавішу сигналізує про завершення циклу. Після завершення циклу рядком return frame повертаємо із функції live останнє отримане зображення з камери.

```
1  add_library(image image.cpp)
2
3  target_include_directories(image INTERFACE ${CMAKE_CURRENT_SOURCE_DIR})
4
5  target_link_libraries(image PUBLIC ${OpenCV_LIBS} video2text_compiler_flags)
6
```

Рис. 4.12 - CMakeLists.txt для модуля image

На рисунку 4.12 представлено налаштування конфігураційного файла CMakeLists.txt для зборщика CMake для дерикторії image. Даний файл вказує

ціль компіляції, файл для компіляції, які дерикторії включати, коли бібліотека використовується, які бібліотеки підключити для компіляції об’єкта.

add_library(image image.cpp): команда вказує скомпілювати файл video.cpp та на виході отримати об’єкт image.

target_include_directories(image INTERFACE ${CMAKE_CURRENT_SOURCE_DIR}): вказує необхідні дерикторії для включення для компіляції поточного об’єкта image.

target_link_libraries(video PUBLIC ${OpenCV_LIBS} video2text_compiler_flags): вказує бібліотеки або флаги, які необхідно використати для лінкування даного об’єкта image

```
1  #include <opencv2/imgcodecs.hpp>
2  #include <opencv2/imgproc.hpp>
3
4  namespace image
5  {
6    std::vector<cv::Rect> findContours(const cv::Mat&);
7    void renderContours(const cv::Mat&, const std::vector<cv::Rect>&);
8  }
9
```

Рис. 4.13 — Оголошення функціоналу image.hpp

На рисунку 4.13 представлений деклараційний файл image.hpp, в якому оголошений увесь функціонал, який буде реалізований у файлі image.cpp, а також включені необхідні для реалізації функціоналу деклараційні файли:

#include <opencv2/imgcodecs.hpp> - підключаємо бібліотеку imgcodecs з opencv для кодування зображень.

#include <opencv2/imgproc.hpp> - підключаємо бібліотеку imgproc з opencv для обробки зображень.

namespace image

```
{
  std::vector<cv::Rect> findContours(const cv::Mat&);
  void renderContours(const cv::Mat&, const std::vector<cv::Rect>&);
}
```

Оголошуємо простір імен image, в якому описуємо функції findContours та renderContours. Функція findContours очікує в якості параметра посилання на об’єкт типу Mat. Функція findContours повертає вектор об’єктів типу Rect. Тип Rect отрмується з бібліотеки OpenCV. Функція renderContours очікує два параметра: перший параметер посилання на об’єкт типу Mat, другий параметер — посилання на вектор, який зберігає об’єкти типу Rect.

```cpp
#include "image.hpp"

void image::renderContours(const cv::Mat& frame, const std::vector<cv::Rect>& contours)
{
  for (size_t i = 0; i < contours.size(); i++)
  {
    cv::rectangle(frame, contours[i], cv::Scalar(0, 255, 0), 1, 8, 0);
  }
}

std::vector<cv::Rect> image::findContours(const cv::Mat& frame)
{
  cv::Mat gray, sobel, threshold, kernel;
  cv::cvtColor(frame, gray, cv::COLOR_BGR2GRAY);
  cv::Sobel(gray, sobel, CV_8U, 1, 0, 3, 1, 0, cv::BORDER_DEFAULT);
  cv::threshold(sobel, threshold, 0, 255, cv::THRESH_OTSU + cv::THRESH_BINARY);
  kernel = cv::getStructuringElement(cv::MORPH_RECT, cv::Size(15, 5));
  cv::morphologyEx(threshold, threshold, cv::MORPH_CLOSE, kernel);
  std::vector<std::vector<cv::Point>> contours;
  cv::findContours(threshold, contours, 0, 1);
  std::vector<std::vector<cv::Point>> contoursPoly(contours.size());
  std::vector<cv::Rect> bound;
  for (size_t i = 0; i < contours.size(); i++)
  {
    if (contours[i].size() > 100)
    {
      cv::approxPolyDP(cv::Mat(contours[i]), contoursPoly[i], 3, true);
      cv::Rect approxy(cv::boundingRect(cv::Mat(contoursPoly[i])));
      if (approxy.width > approxy.height)
      {
        bound.push_back(approxy);
      }
    }
  }
  return bound;
}

```

Рис. 4.14 — Реалізація функціоналу image.cpp

На рисунку 4.14 представлена реалізація функції renderContours та findContours модуля image.

void image::renderContours(const cv::Mat& frame, const std::vector<cv::Rect>& contours)

{

for (size_t i = 0; i < contours.size(); i++)

{

cv::rectangle(frame, contours[i], cv::Scalar(0, 255, 0), 1, 8, 0);

```
  }
}
```

Функція renderContours займається тим, що малює прямоктники на переданому зображенні. Функція renderContours має два параметри: frame — для посилання на об’єкт типу Mat, contours — для посилання на вектор об’єктів типу Rect. Функція renderContours нічого не повертає. В тілі функції renderContours знаходиться цикл, яким ітеруємося по вектору контурів. На кожну ітерацію цикла викликається функція із бібліотеки opencv rectangle. Функція rectangle приймає зображення, на якому необхідно намалювати прямоктник, також дані прямокутника (розмір, координати), колір ліній прямокутника, товщину граней прямокутника.

Функція findContours займається тим, що знаходить необхідні області на переданому зображенні. Функція findContours має параметер frame — для посилання на об’єкт типу Mat. Функція findContours повертає вектор об’єктів типу Rect.

```
cv::Mat gray, sobel, threshold, kernel;
cv::cvtColor(frame, gray, cv::COLOR_BGR2GRAY);
cv::Sobel(gray, sobel, CV_8U, 1, 0, 3, 1, 0, cv::BORDER_DEFAULT);
cv::threshold(sobel, threshold, 0, 255, cv::THRESH_OTSU + cv::THRESH_BINARY);
kernel = cv::getStructuringElement(cv::MORPH_RECT, cv::Size(15, 5));
cv::morphologyEx(threshold, threshold, cv::MORPH_CLOSE, kernel);
std::vector<std::vector<cv::Point>> contours;
cv::findContours(threshold, contours, 0, 1);
```

При виклику функції findContours до неї передається зображення, яке повинно пройти етапи обробки, після чого бути проаналізованим. Функція всередині має декілька об’єктів тип Mat gray, sobel, threshold, kernel. Кожен з цих об’єктів буде містити інформацію про зображення після кожного етапу обробки. Спочатку оригінальне зображення frame функцією бібліотеки OpenCV cvtColor конвертується до сірої моделі кольору. Дана трансформація необхідна для того, щоб спростити зображення, відкинувши інформацію про кольорию. Результат даної конвертації записується до змінної gray. Наступним кроком зображення gray потрапляє до функції бібліотеки OpenCV Sobel. Дана функція на основі переданого зображення отримує нове, в якому підкреслює межі. Нове зображення записується до змінної sobel. Наступним кроком отримане зображення sobel оброблюється функцією бібліотеки OpenCV threshold. Дана функція створює бінарне зображення, в якому є лише білі та чорні пікселі. Дана функція необхідна для того, щоб виділити області, які потенційно можуть містити текст. На нове бінарне зображення посилається змінна threshold. Наступним кроком отримане бінарне зображення передається до функції бібліотеки OpenCV morphologyEx, яка застосовує морофлогічні трансформації над отриманим зображенням. Дана функція необхідна для того, щоб отримати нове зображення, в якому видалені будь-які маленькі об’єкти. На нове зображення посилається змінна threshold. Після даних трансформацій зображення threshold передається до функції бібліотеки OpenCV findContours. Дана функція виявляє контури об’єктів, які представляються у вигляді послідовностей точок. Об’єкт точка представлений бібліотекою OpenCV типом Point. Функція findContours записує результат роботи до змінної contours, яка є вектором векторів, які оперують точками.

```
std::vector<std::vector<cv::Point>> contoursPoly(contours.size());

std::vector<cv::Rect> bound;
```

```
for (size_t i = 0; i < contours.size(); i++)
{
   if (contours[i].size() > 100)
   {
      cv::approxPolyDP(cv::Mat(contours[i]), contoursPoly[i], 3, true);
      cv::Rect approxy(cv::boundingRect(cv::Mat(contoursPoly[i])));
      if (approxy.width > approxy.height)
      {
        bound.push_back(approxy);
      }
   }
}
return bound;
```

Після знаходження усіх контурів необхідно пройтися по кожному контуру та відсіяти непотрібні. Логіка по роботі з контурами всередині циклу буде запускатися лише у випадку, якщо кількість елементів в контурі більше 100. Якщо умова правдива, тоді запсукається подальша логіка у вигляді функції бібліотеки OpenCV approxPolyDP. Дана функція отримує на вході об’єкт типу Mat, який створюється на основі вектору точок. Дана функції апроксимує криву (послідовність із точок) з вказаною точністю та зазначає, що крива повинна бути замкненою. Результат роботи функції у вигляді вектора точок записується до вектору contoursPoly. Наступним кроком відпрацювує логіка, яка за допомгою функції бібліотеки OpenCV boundingRect на бінарному зображені прямокутниками приблизно відокремлює необхідні області. В результаті якщо

область більша по ширині, ніж висоті, така область записується до вектора bound. Після завершення роботи циклу функція findContours повертає вектор прямокутників, які можуть вказувати на потенційні області з текстом.

```
1  add_library(ocr ocr.cpp)
2
3  target_include_directories(ocr INTERFACE ${CMAKE_CURRENT_SOURCE_DIR})
4
5  target_link_libraries(ocr PUBLIC ${OpenCV_LIBS} tesseract lept video2text_compiler_flags)
6
```

Рис. 4.15 - CMakeLists.txt для модуля ocr

На рисунку 4.15 представлено налаштування конфігураційного файла CMakeLists.txt для зборщика CMake для дерикторії ocr. Даний файл вказує ціль компіляції, файл для компіляції, які дерикторії включати, коли бібліотека використовується, які бібліотеки підключити для компіляції об’єкта.

add_library(ocr ocr.cpp) - команда вказує скомпілювати файл ocr.cpp та на виході отримати об’єкт ocr.

target_include_directories(ocr INTERFACE ${CMAKE_CURRENT_SOURCE_DIR}) - вказує необхідні дерикторії для включення для компіляції поточного об’єкта ocr.

target_link_libraries(ocr PUBLIC ${OpenCV_LIBS} tesseract lept video2text_compiler_flags) - вказує бібліотеки або флаги, які необхідно використати для лінкування даного об’єкта ocr.

```
#include <string>
#include <leptonica/allheaders.h>
#include <tesseract/baseapi.h>
#include <opencv2/imgproc.hpp>

namespace ocr
{
    std::string findText(const cv::Mat&);
}

```

Рис. 4.16 — Оголошення функціоналу ocr.hpp

На рисунку 4.16 представлений деклараційний файл ocr.hpp, в якому оголошений увесь функціонал, який буде реалізований у файлі ocr.cpp, а також включені необхідні для реалізації функціоналу деклараційні файли:

#include <string> - для роботи з текстом.

#include <leptonica/allheaders.h> - необхідний файл для роботи бібліотеки tesseract.

#include <tesseract/baseapi.h> - підключення бібліотеки baseapi з tesseract для взаємодії з функціоналом tesseract.

#include <opencv2/imgproc.hpp> - підключення бібліотеки improc з opencv для роботи з зображенням.

namespace ocr

{

std::string findText(const cv::Mat&);

}

Оголошення простору імен ocr, в якому оголошена функція findText. Функція findText очікує у якості параметра посилання на б’єкт типу Mat. Функція findText повертає значення типу string (текст).

```cpp
#include "ocr.hpp"

std::string ocr::findText(const cv::Mat& img)
{
  tesseract::TessBaseAPI *api = new tesseract::TessBaseAPI;
  api->Init(NULL, "eng", tesseract::OEM_LSTM_ONLY);
  api->SetImage(img.data, img.cols, img.rows, 3, img.step);
  std::string text = std::string(api->GetUTF8Text());
  api->End();
  delete api;
  return text;
}

```

Рис. 4.17 — Реалізація функцоналу ocr.cpp

На рисунку 4.17 представлена реалізація функції findText. Функція findText використовує бібліотеку Tesseract для визначення тексту в переданому зображенні. Дана функція має параметер img, до якого буде передане посилання на об’єкт типу Mat (зображення). Дана функція по завершенню роботи поверне значення типу string. В тілі функції знаходиться вказівник на об’єкт в памя’ті типу TessBaseApi. У даного об’єкта викликається метод Init для налаштування роботи Tesseract для певної LSTM моделі. Наступним кроком викликається метод SetImage для передачі зображення для аналіза. Після викликається метод GetUTF8Text, який повертає виявлений текст на зображенні. Останніми кроками викликається метод End, який викликає усю необхідну логіку для завершення роботи Tesseract, після чого відбувається очищення пам’яті командою delete api. По завершенню роботи функція findText повертає текст.

ВИСНОВКИ

Під час виконання роботи мною було досліджено напрямок штучного інтелекту, було проаналізовано галузь штучного інтелекту комп'ютерний зір, було визначено основні концепції комп'ютерного зору, було описано застосування ком'ютерного бачення у різних сферах. Мною було проаналізовано технологію OpenCV. Було описано характеристики OpenCV та наведені приклади використання різних функцій різних модулів бібліотеки OpenCV під час розробки трансформації відео до тексту. Були визначені основні функції, які повинна реалізовувати трансформація відео до тексту на основі OpenCV, а саме:

- Крос-платформність.
- Швидкість.
- Надійсність.

Усі поставлені функції у трансформації відео до тексту були реалізовані.

Також було проаналізовано технологію Tesseract. Було описано характеристики та наведений приклад використання функціоналу біліотеки Tesseract під час розробки трансформації відео до тексту на основі OpenCV.

Мною було проаналізовано сучасні інструменти розробки для створення програмного забезпечення в галузі комп'ютерного зору. Були описані основні характеристики та наведені приклади використання різного функціоналу мови програмування C++. За допомогою описаних можливостей мови програмування C++ була досягнута як швдикість роботи трансформації відео до тексту, так і її надійність.

Мною було описано використання зброщика проєкта CMake. За допомогою інстурмента CMake налаштування проєкта та його розробку вдалося максимально автоматизувати, що позитивно вплинуло на розширення проєкта,

його тестування та інтеграцію додаткових бібліотек та інструментів. Використовуючи систему контролю версій Git та платформу GitHub увесь вихідний код проєкта вдалося систематизувати, захистити від випадкових втрат та непердбачуваних змін.

В результаті було розроблено трансформацію відео до тексту з використанням OpenCV.

ПЕРЕЛІК ПОСИЛАНЬ

1. Learn C++ - https://en.cppreference.com/w/
2. C++ Data Types - https://en.cppreference.com/w/cpp/language/types
3. C++ Flow Control - https://en.cppreference.com/book/intro/control
4. C++ Functions - https://en.cppreference.com/w/cpp/utility/functional/function
5. C++ Classes - https://en.cppreference.com/w/cpp/language/classes
6. C++ Access Specifiers - https://en.cppreference.com/w/cpp/language/access
7. C++ Inheritance - https://en.cppreference.com/book/intro/inheritance
8. C++ Abstract Class - https://en.cppreference.com/w/cpp/language/abstract_class
9. C++ Polymorphism - https://cplusplus.com/doc/tutorial/polymorphism/
10. C++ Thread - https://en.cppreference.com/w/cpp/thread/thread
11. C++ Mutex - https://en.cppreference.com/w/cpp/thread/mutex
12. C++ Conditional variable - https://en.cppreference.com/w/cpp/thread/condition_variable
13. C++ Atomic - https://en.cppreference.com/w/cpp/atomic/atomic
14. C++ Operator overloading - https://en.cppreference.com/w/cpp/language/operators
15. C++ Templates - https://en.cppreference.com/w/cpp/language/templates
16. CMake Doc - https://cmake.org/cmake/help/latest/
17. OpenCV GitHub Repository - https://github.com/opencv/opencv
18. OpenCV Docs - https://docs.opencv.org/4.x/

19. Learn OpenCV - https://www.bogotobogo.com/cplusplus/files/OReilly%20Learning%20OpenCV.pdf

20. OpenCV Core functionality - https://docs.opencv.org/4.x/d0/de1/group__core.html

21. OpenCV Image file reading and writing - https://docs.opencv.org/4.x/d4/da8/group__imgcodecs.html

22. OpenCV Image processing - https://docs.opencv.org/4.x/d7/dbd/group__imgproc.html

23. OpenCV Video I/O - https://docs.opencv.org/4.x/dd/de7/group__videoio.html

24. OpenCV HighGUI - https://docs.opencv.org/4.x/d7/dfc/group__highgui.html

25. Tesseract GitHub Repository - https://github.com/tesseract-ocr/tesseract

26. Tesseract Tessdata GitHub Repository https://github.com/tesseract-ocr/tessdata

27. Debian GNU/Linux - https://www.debian.org/

28. Git - https://git-scm.com/

29. GitHub - https://github.com/

30. Video2text GitHub Repository - https://github.com/Mole-Miner/video2text

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)

Державний університет інформаційно-комунікаційних технологій

Кафедра інженерії програмного забезпечення автоматизованих систем

КВАЛІФІКАЦІЙНА РОБОТА
на тему:
“Трансформація відео до тексту з використанням OpenCV”

На здобуття освітнього ступеня магістра
Зі спеціальності 126 Інформаційні системи та технології
Освітньо-професійної програми Інформаційні системи та технології

Виконав: здобувач вищої освіти гр.ІСДМ-61
Вислобіцький Владислав
Керівник: доцент, Шкапа Вікторія

Київ - 2023

Актуальність теми: у зв'язку з швидким розвитком та широким росповсюдженням використання сфер штучного інтелекту, як комп’ютерний зір, дана тема, яка розкриває на практиці використання передової технології комп’ютерного зору OpenCV для трансформації відео до тексту є актуальною.
Об’єкт дослідження: технології комп’ютерного зору.
Предмет дослідження: функціонал та характеристики технологій комп’ютерного зору.
Мета дослідження: реалізувати трансформацію відео до тексту.
Завдання дослідження:

- Дослідити домен комп’ютерного зору.
- Проаналізувати OpenCV.
- Проаналізувати інструменти розробки трансформації відео до тексту.
- Реалізувати трансформацію відео до тексту.

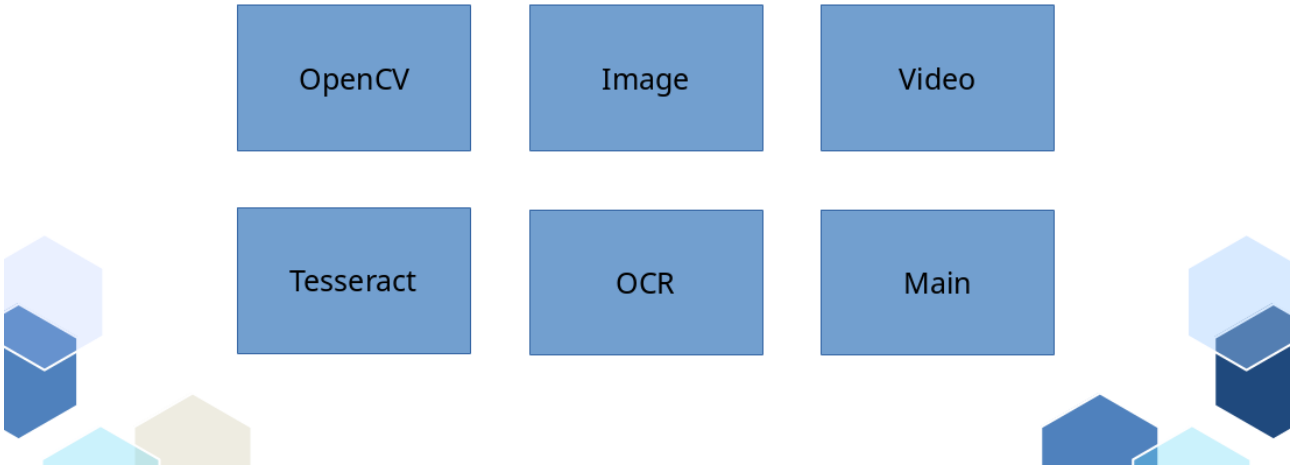
OpenCV

Open Computer Vision

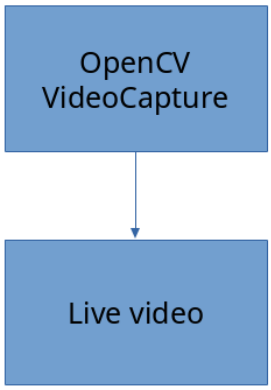
Tesseract

Tesseract OCR Engine

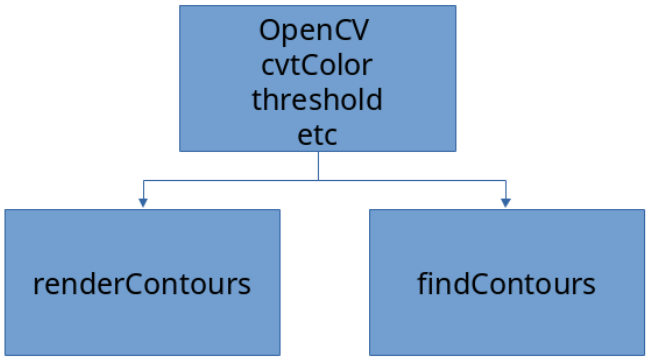
Модулі трансформації відео до тексту



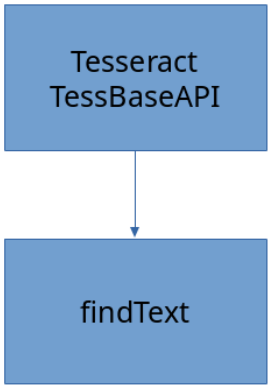
Модуль роботи з відео



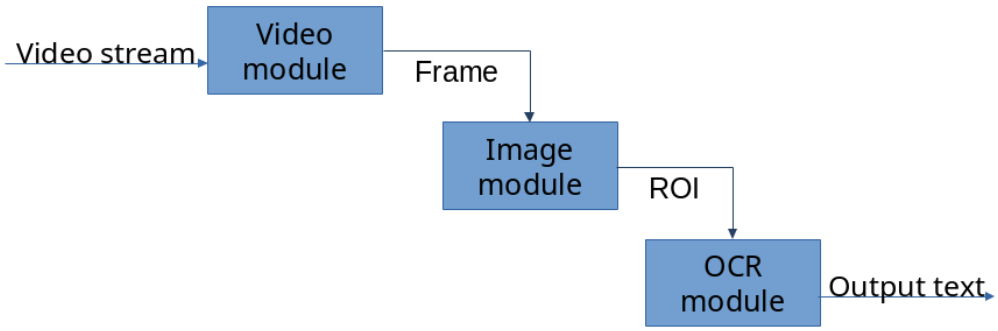
Модуль роботи з зображенням



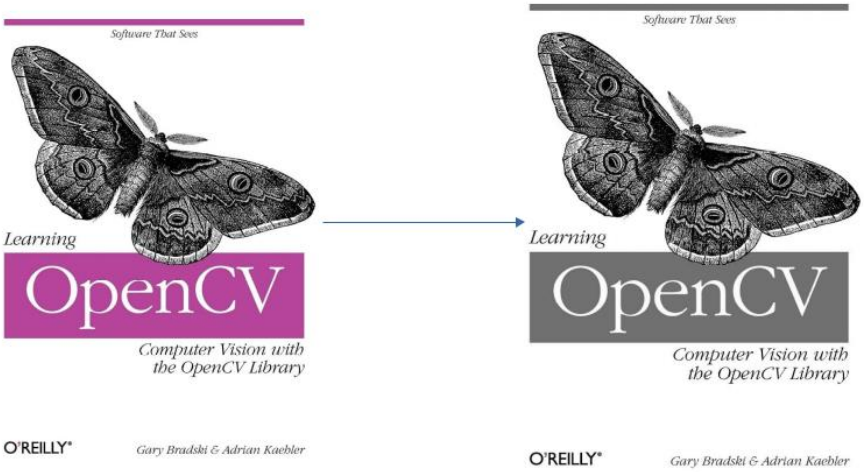
Модуль виявлення тексту



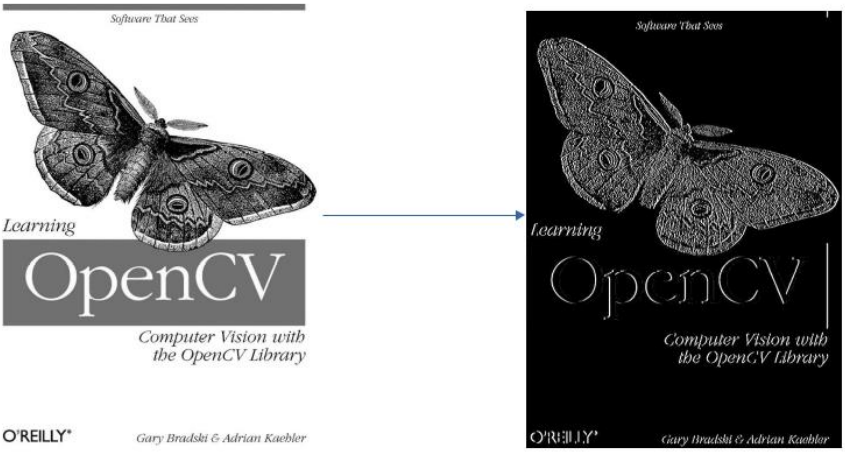
Взаємодія модулів



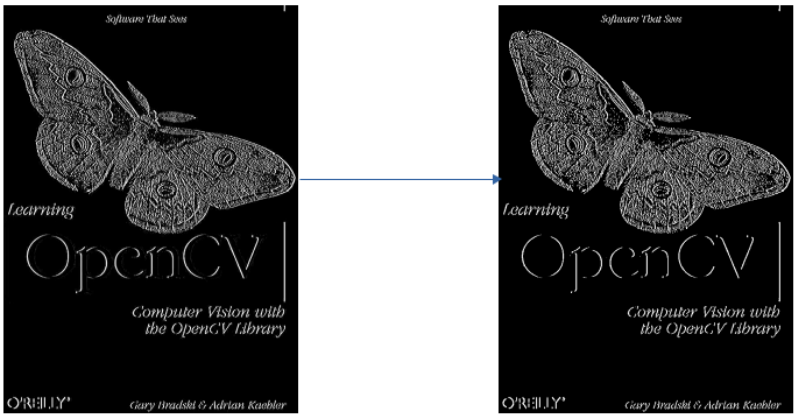
OpenCV cvtColor



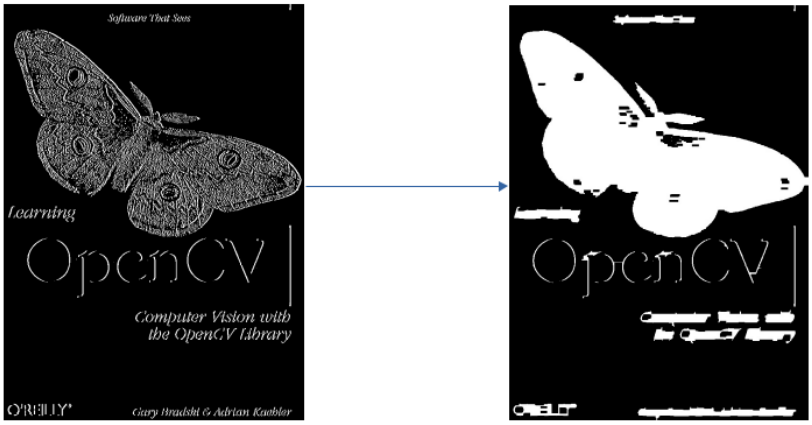
OpenCV Sobel



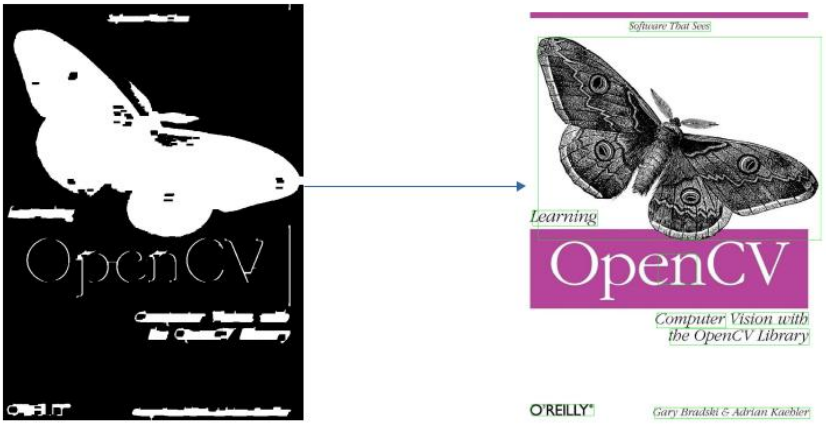
OpenCV threshold



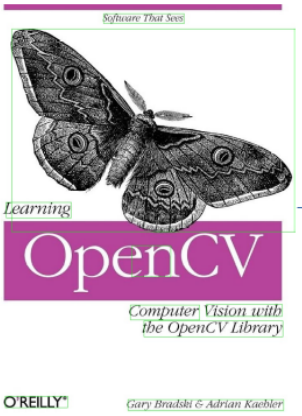
OpenCV morphologyEx

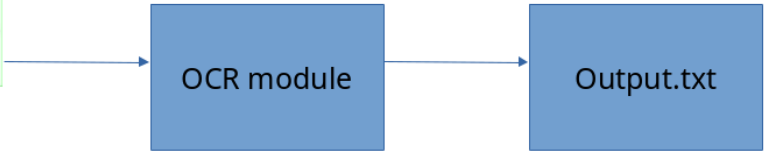


OpenCV findCountours



TessBaseAPI





Висновки

- У кваліфікаційній магістерській роботі мною було проводено дослідження домену комп'ютерного зору.
- Було проаналізовано можливості та функціонал OpenCV.
- Було проаналізовано інструменти розробки.
- Створено алгоритм, на основі якого було реалізовано трансформацію відео до тексту.

Апробація результатів дослідження:

1. Вислобіцький В.А. “Трансформація відео до тексту з використанням OpenCV”. Стаття на Всеукраїнській науково-технічній конференції “Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу”. - Київ, 28 листопада 2023р.
2. Вислобіцький В.А. “Створення графічного інтерфейсу для взаємодії з сервером трансформації відео до тексту”. Тези на Всеукраїнській науково-технічній конференції “Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу”. - Київ, 28 листопада 2023р.
3. Вислобіцький В.А. “Трансформація відео до тексту з використанням OpenCV”. Тези на Всеукраїнській науково-технічній конференції “Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу”. - Київ, 28 листопада 2023р.

Дякую за увагу!