Міністерство освіти і науки України
Національний університет “Львівська політехніка”
Інститут комп’ютерних наук та інформаційних технологій

Кафедра САПР

Лабораторна робота №1

з дисципліни: “Розпізнавання образів і комп’ютерний зір”

на тему:

“Робота з зображеннями”

Виконав:
Ст. групи ПП-44
Верещак Б. О.
Прийняв:
Асис. Мельник М. Р.

Львів - 2025

Мета: навчитися виконувати елементарні дії з зображеннями, перетворювати матриці зображень.

Лабораторне завдання

1. Відкрити відеофайл із записом руху автомобілів.
2. Зберегти з відео файлу серію з 5-ти зображень.
3. Відкрити зображення, з розширенням .jpg;
4. Визначити розмір зображення;
5. Збережіть інформаційні поля зображення в структурні змінні.
6. Перетворіть зображення в чорнобіле.
7. Провести дослідження медіанного фільтру.
8. Змістити два послідовні кадри відносно себе таким чином щоб різниця віднімання цих кадрів була найменшою.
9. Описати всі функції, які використовувалися для вище перелічених завдань.

Кроки виконання завдання:

Крок 1: Відкриття відеофайлу

Для виконання лабораторної роботи я вирішив використовувати мову програмування C++, так як я б хотів її вивчити за виконанням робіт з цього предмету.

У C++ є декілька способів використовувати компютерний зір, наприклад ffmpeg, gstreamer, OpenCV й інші. Для курсу лабораторних робіт я обрав OpenCV, так як це найпопулярніша бібліотека для CV(computer vision) на C++.

Для відтворення відео в цій бібліотеці використовується клас VideoCapture з параметром назви файла, та наступного створення вікна, де цей файл можна переглянути. Перегляд відео працює пофреймово, тобто ми попорядку загружуємо кожен фрейм з відео (30 fps). Результати видно на *Рис. 1. Показ відео через OpenCV*.

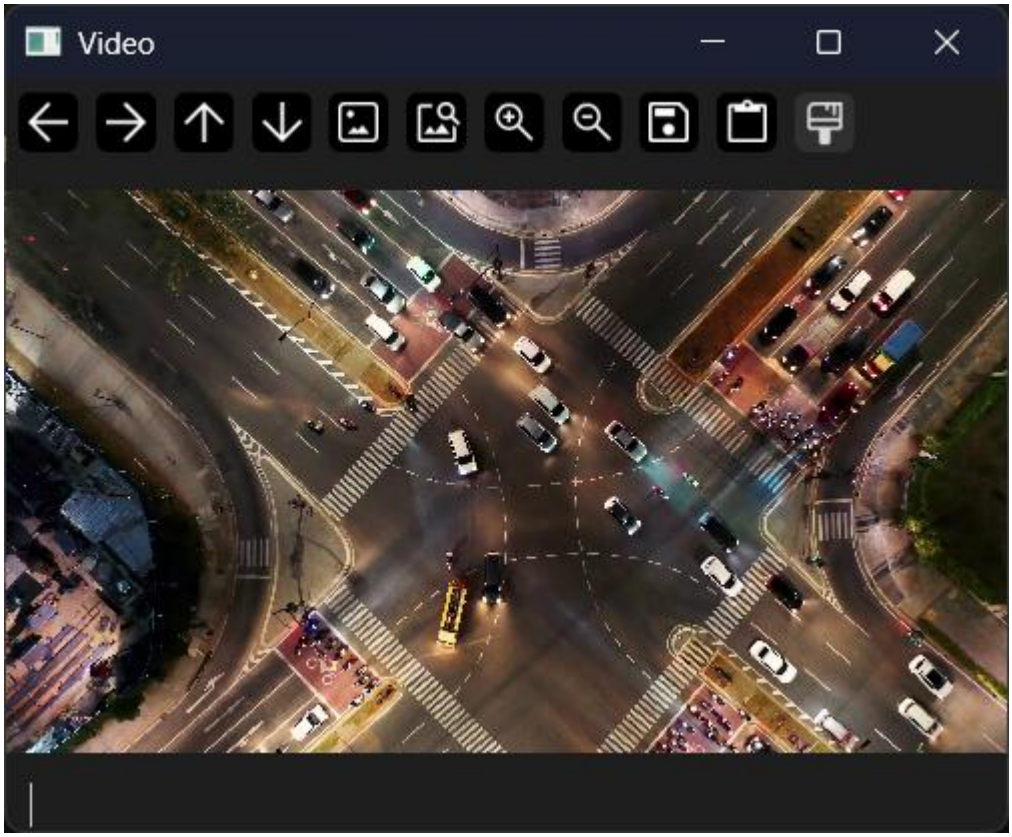

Рис. 1. Показ відео через OpenCV

Код програми:

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

using namespace cv;
using namespace std;

int main() {
    VideoCapture cap("cars.mp4");
    if (!cap.isOpened()) {
        cout << "Не вдалося відкрити відео!" << endl;
        return -1;
    }

    // Створюємо нове вікно одразу
    namedWindow("Video", WINDOW_NORMAL); // WINDOW_NORMAL дозволяє змінювати розмір
    resizeWindow("Video", 800, 600);  // можна задати початковий розмір

    Mat frame;
    while (true) {
        cap >> frame;
        if (frame.empty()) break;

        imshow("Video", frame);  // показуємо кадр
        if (waitKey(30) == 27)  // вихід по ESC
```

```
            break;
        }

        destroyAllWindows();
        return 0;
}
```

Крок 2-5: Зберегти серію з 5-ти зображень, відкрити, визначити розмір зображення та зберегти поля в структурну змінну

Тут вже потрібно витягнути 5 послідовних кадрів і зберегти їх у файли. Для цього я використовував операцію читання кадру, зберіг їх на диск через imwrite, та створив вікно для для показу кожного кадру.

Для відкриття файлу використовувалася подібна до минулої функція imread, після чого знову показую зображення в новому вікні.

Для визначення розміру зображення використовувалися вбудовані змінні cols і rows для ширини і висоти відповідно. Також ше витягувалися кількість шарів про кольори (канали) і все це зберігалося в новій структурі ImageInfo. Результат можна побачити на Рис. 2. Перший послідовний кадр

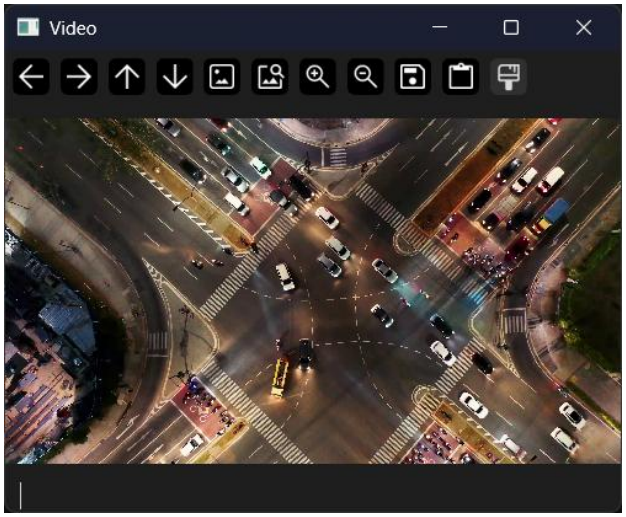

Рис. 2. Перший послідовний кадр

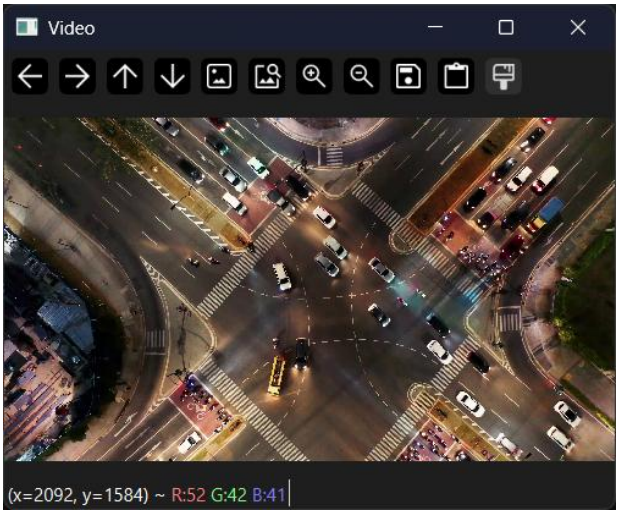

Рис. 3. Другий послідовний кадр

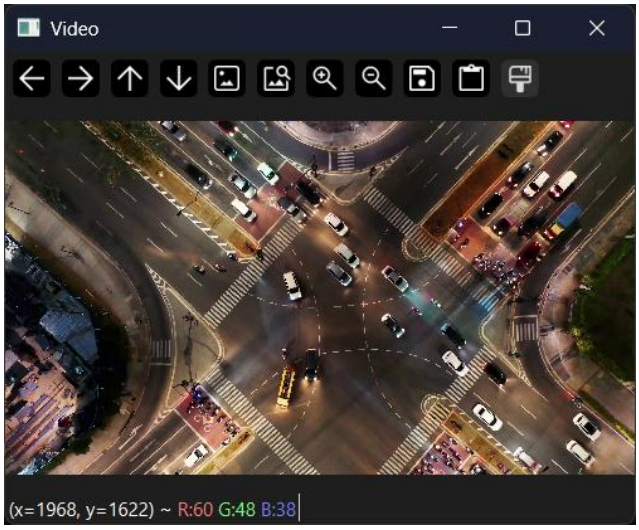

Рис. 4. Третій послідовний кадр

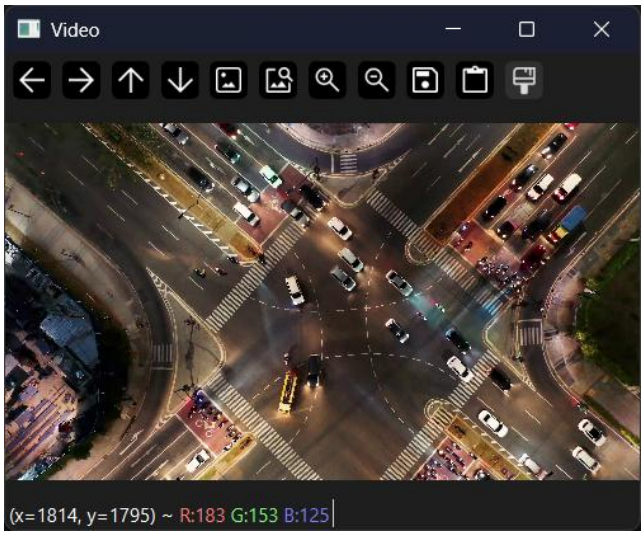

Рис. 5. Четвертий послідовний кадр

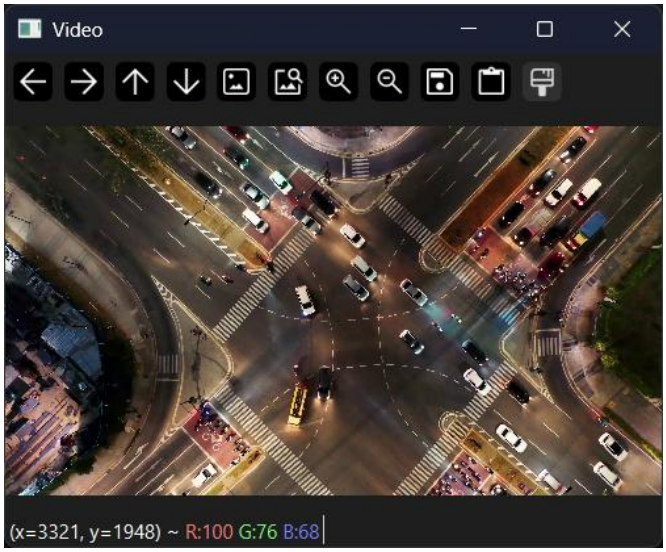

Рис. 6. Останній фрейм відео

```
Saved frame0.jpg
Saved frame1.jpg
Saved frame2.jpg
Saved frame3.jpg
Saved frame4.jpg
frame0.jpg : 3840x2160, channels=3
frame1.jpg : 3840x2160, channels=3
frame2.jpg : 3840x2160, channels=3
frame3.jpg : 3840x2160, channels=3
frame4.jpg : 3840x2160, channels=3
```

Рис. 7. інформація про зобравження

Код програми:

```
struct ImageInfo {
    int width;
    int height;
    int channels;
};
vector<string> saveFrames(VideoCapture& cap, int count = 5, const string& prefix =
"frame") {
    vector<string> filenames;
    Mat frame;
    for (int i = 0; i < count; ++i) {
        cap >> frame;
        if (frame.empty()) break;
        string filename = prefix + to_string(i) + ".jpg";
        imwrite(filename, frame);
        filenames.push_back(filename);

        namedWindow("Video", WINDOW_NORMAL);
        imshow("Video", frame);
        waitKey(0);
```

```
        cout << "Saved " << filename << endl;
    }
    return filenames;
}
Mat openImage(const string& filename) {
    Mat img = imread(filename);
    if (img.empty()) {
        cerr << "Cant open image: " << filename << endl;
    } else {
        // Відобразити кадр
        namedWindow("Image", WINDOW_NORMAL);
        imshow("Image", img);
        waitKey(0);
    }
    return img;
}

ImageInfo getImageInfo(const Mat& img) {
    ImageInfo info;
    info.width = img.cols;
    info.height = img.rows;
    info.channels = img.channels();
    return info;
}
```

Крок 6: Перетворення зображення в чорно-біле

Для конвертації зображення в чорно-біле, я використовую функцію, перетворює вхідне зображення з одного колірного простору в інший – cvtcolor, і показую результат перетворення в новому вікні. Результат видно на Рис. 8. Перетворене зображення через чорно-білий фільтр

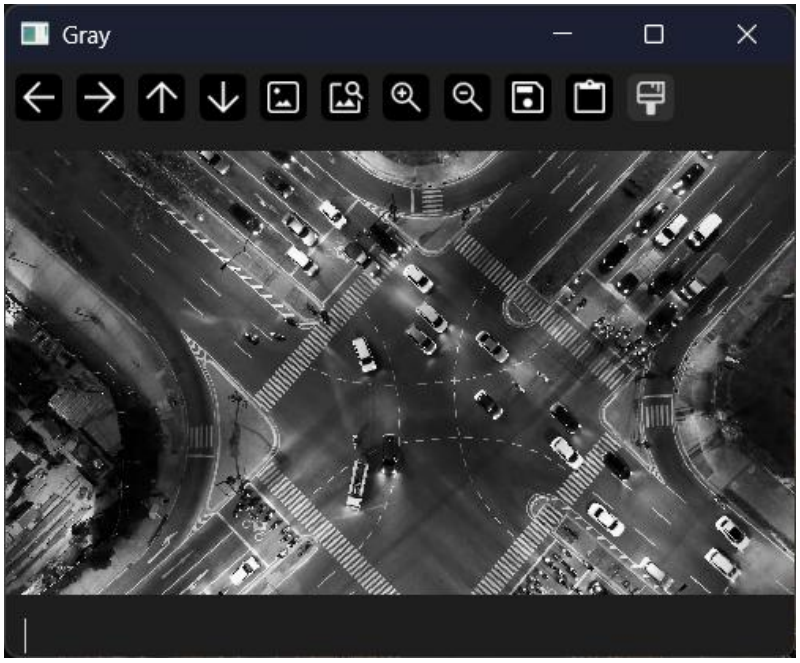

Рис. 8. Перетворене зображення через чорно-білий фільтр

Крок 7: Дослідження медіанного фільтру

На цьому кроці потрібно було застосувати медіанний фільтр, який розмиває шум на зображенні. В якості розміра ядра було обрано 3 пікселі, тобто навколо кожного пікселя було створене уявне вікно 3 на 3 і для того пікселя вибиралася медіана за яскравістю з того вікна. Результат можна розмиття можна побачити на Рис. 9.

Рис. 9. Накладання медіанного фільтру

Код програми:

```
Mat applyMedianFilter(const Mat& img, int ksize = 3) {
    Mat filtered;
    medianBlur(img, filtered, ksize);

    // Показати
    namedWindow("Median Filter", WINDOW_NORMAL);
    imshow("Median Filter", filtered);
    waitKey(0);

    return filtered;
}
```

Крок 8: Зміщення послідовних кадрів

На даному кроці наша мета знайти найкращий зсув другого кадру, щодо першого, щоб сума абсолютної різниці пікселів була мінімальною.

Реалізація працює так: ініціалізовуємо початкові змінні, перебираємо через цикл зсув по dx і dy. Після цього для кожного зсуву використовуємо афінне перетворення (переміщення) на другий фрейм. Далі обчислюємо абсолютну різницю і сумуємо всі пікселі різниці і перевіряємо чи ця сума найменша. Після всіх перевірок повертаємо значення при якому найменша різниця.

Так вийшло що найменша різниця між фреймами при статичній камері в більшості випадків = 0, тому і так результати. Різницю можна глянути на Рис. 13.

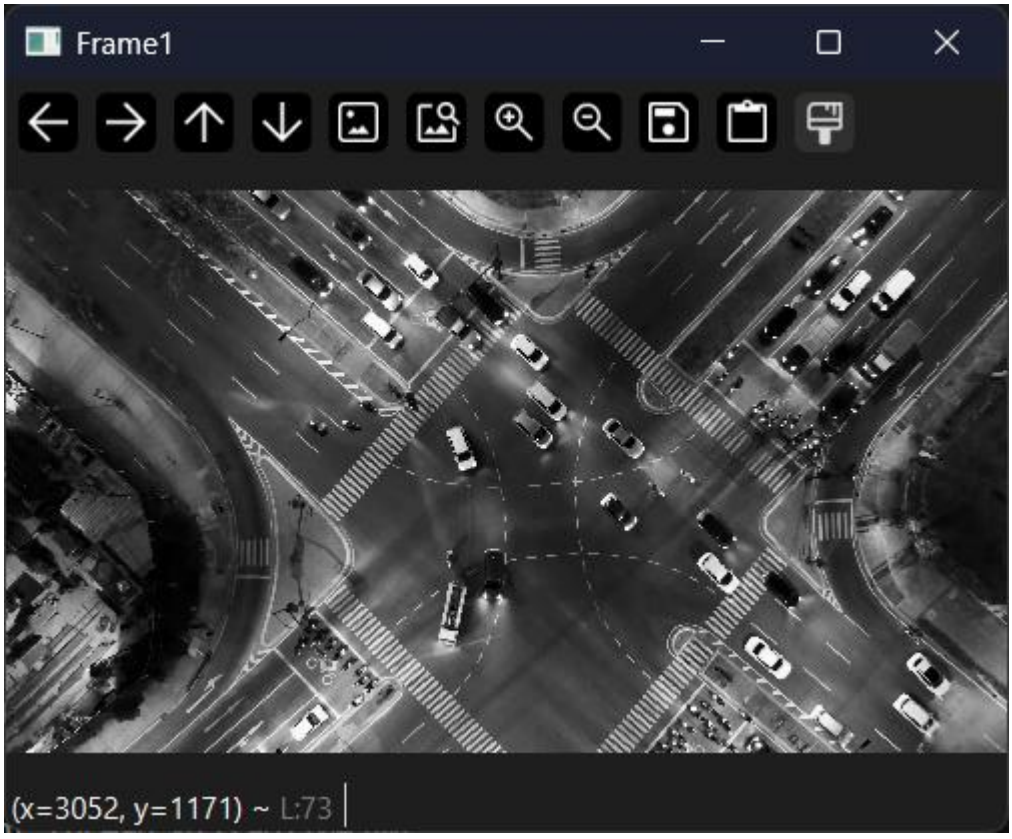

Рис. 10. Перший кадр

Рис. 11. Другий кадр до зсуву

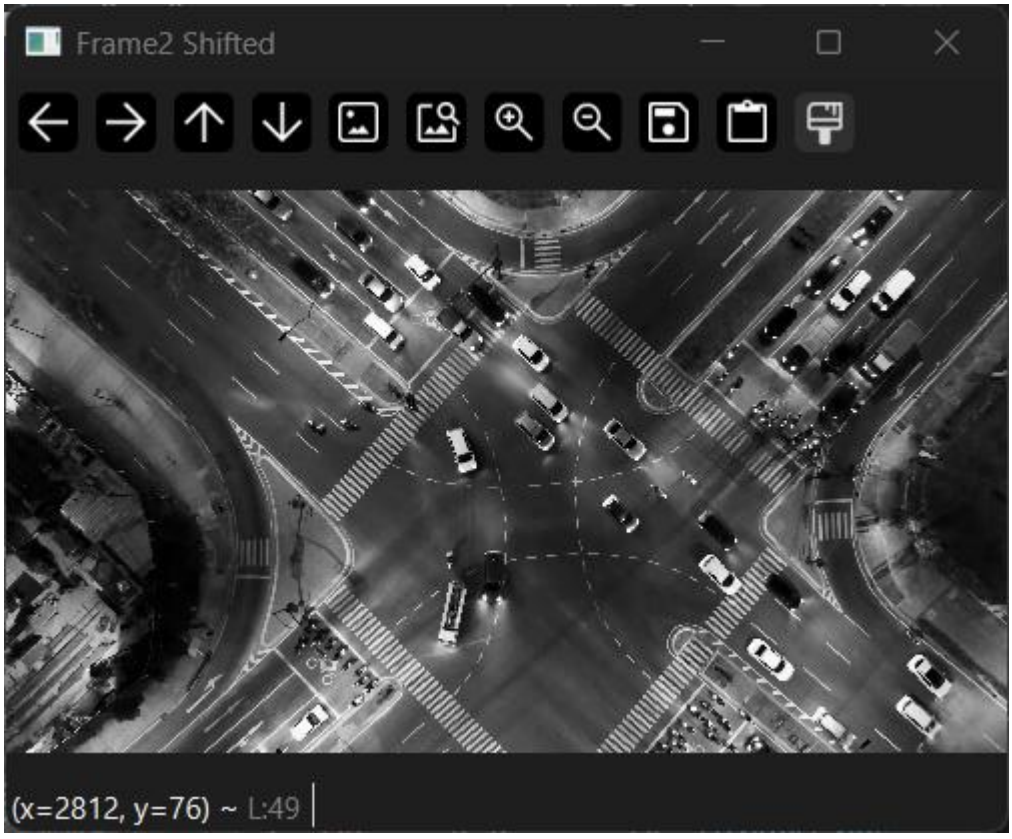

Рис. 12. Другий кадр після зсуву

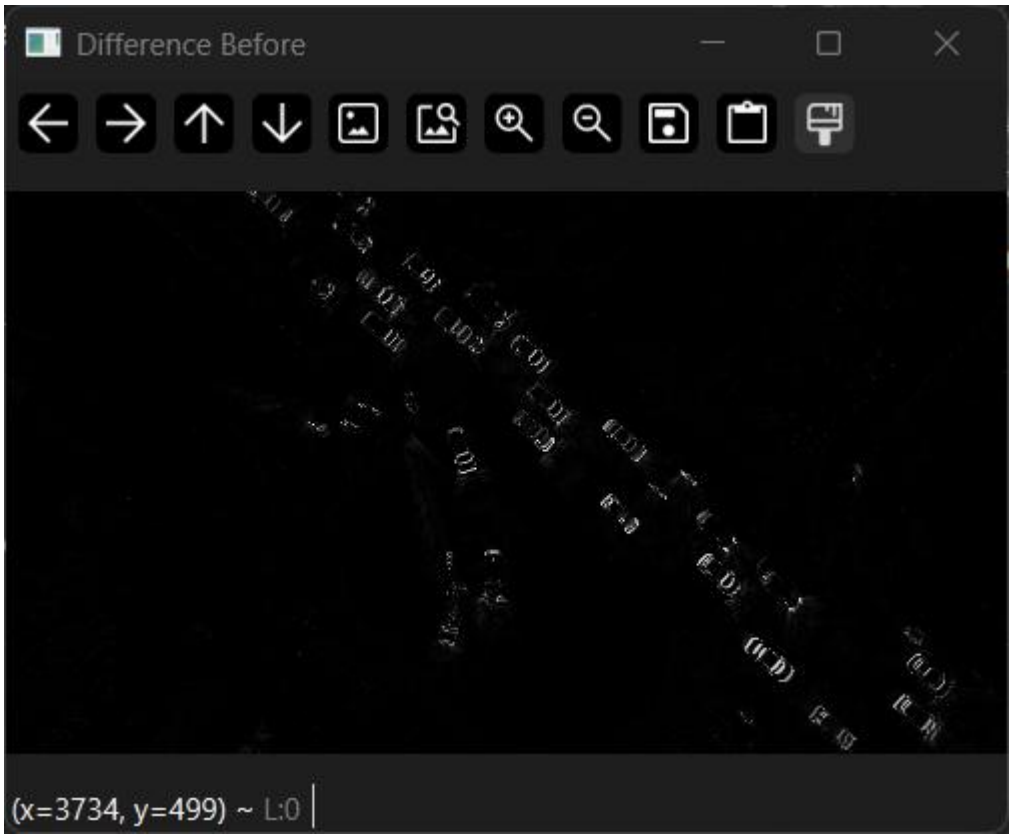

Рис. 13. Різниця до зсуву

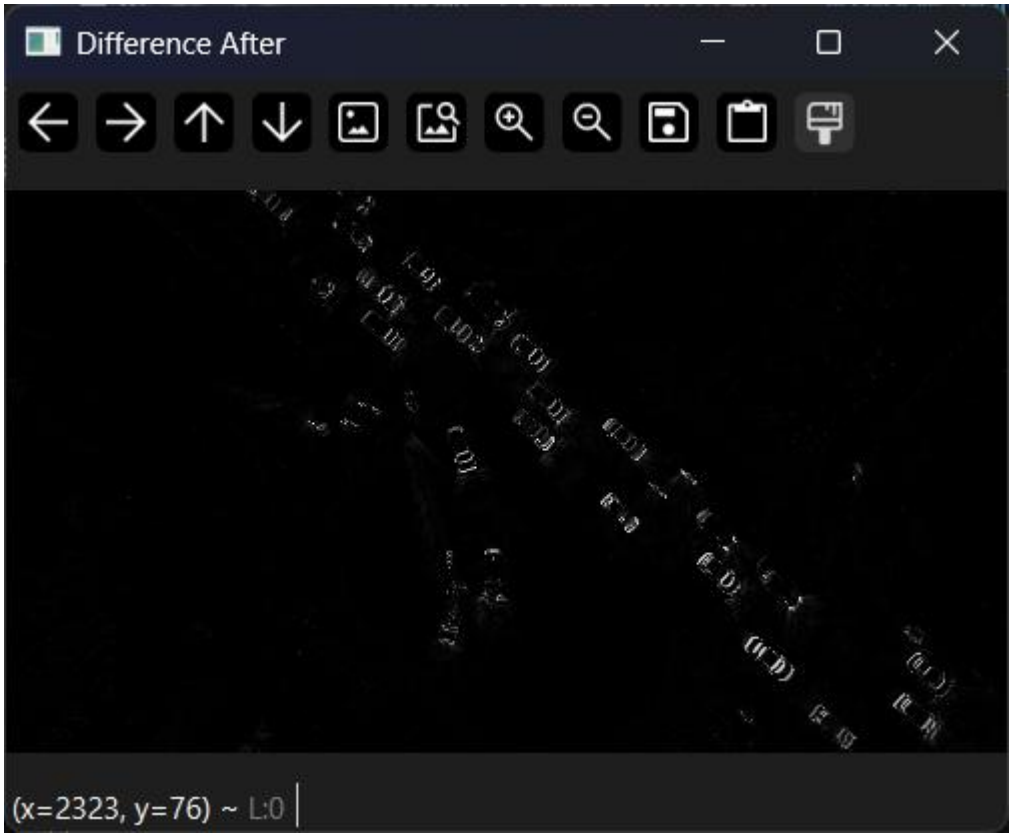

Рис. 14. Різниця після зсуву

Код програми:

```
Point findMinimalDifferenceShift(const Mat& frame1, const Mat& frame2, int maxShift = 5)
{
   // Перетворюємо кадри в grayscale
   Mat gray1, gray2;
   if(frame1.channels() == 3) cvtColor(frame1, gray1, COLOR_BGR2GRAY);
   else gray1 = frame1.clone();
   if(frame2.channels() == 3) cvtColor(frame2, gray2, COLOR_BGR2GRAY);
   else gray2 = frame2.clone();

   int minSum = numeric_limits<int>::max();
   Point bestShift(0,0);

   Mat diffOriginal;
   absdiff(gray1, gray2, diffOriginal); // різниця до зміщення

   // Перебираємо всі зсуви
   for(int dx=-maxShift; dx<=maxShift; dx++)
   {
      for(int dy=-maxShift; dy<=maxShift; dy++)
      {
         Mat shifted;
         Mat translationMat = (Mat_<double>(2,3) << 1,0,dx, 0,1,dy);
         warpAffine(gray2, shifted, translationMat, gray2.size());
```

```
            Mat diff;
            absdiff(gray1, shifted, diff);
            int sumDiff = sum(diff)[0];

            if(sumDiff < minSum){
                minSum = sumDiff;
                bestShift = Point(dx,dy);
            }
        }
    }

    // Показуємо кадри та різниці
    Mat shiftedBest;
    Mat translationMatBest = (Mat_<double>(2,3) << 1,0,bestShift.x, 0,1,bestShift.y);
    warpAffine(gray2, shiftedBest, translationMatBest, gray2.size());

    Mat diffAfter;
    absdiff(gray1, shiftedBest, diffAfter);

    namedWindow("Frame1", WINDOW_NORMAL); imshow("Frame1", gray1);
      namedWindow("Frame2 Original", WINDOW_NORMAL); imshow("Frame2 Original", gray2);
       namedWindow("Frame2 Shifted", WINDOW_NORMAL); imshow("Frame2 Shifted", shiftedBest);
    namedWindow("Difference Before", WINDOW_NORMAL); imshow("Difference Before", diffOriginal);
     namedWindow("Difference After", WINDOW_NORMAL); imshow("Difference After", diffAfter);
    waitKey(0); // чекаємо натискання клавіші

    cout << "Best shift: dx=" << bestShift.x << ", dy=" << bestShift.y << endl;
    return bestShift;
}
```

Крок 9: Опис функціональностей

Основні класи сервісів

ImageProcessingService – основний клас для обробки зображень і кадрів відео, включає методи для витягування кадрів, аналізу параметрів, конвертації в grayscale та пошуку оптимального зсуву між кадрами.

ImageAlignment – утилітарний клас з методами для вирівнювання зображень і розрахунку мінімальної різниці між ними.

OpenCV функції

- cv::VideoCapture – читання відеофайлів і потоків.

- cv::imread – зчитування зображень з файлу.
- cv::imwrite – збереження зображень у файл.
- cv::cvtColor – конвертація кольорового простору (BGR → Gray).
- cv::warpAffine – зсув кадру для пошуку мінімальної різниці.
- cv::absdiff – обчислення абсолютної різниці між двома кадрами.
- cv::medianBlur – застосування медіанного фільтру.
- cv::namedWindow / cv::imshow / cv::waitKey / cv::destroyAllWindows – створення та показ вікон зображень, очікування клавіші, закриття всіх вікон.

Службові методи ImageProcessingService

- ExtractVideoFrames – збереження кадрів з відео у файли .jpg.
- AnalyzeImageInfo – отримання ширини, висоти та кількості каналів зображення.
- ConvertToGrayscale – конвертація кольорового зображення у чорно-біле.
- ApplyMedianFilter – застосування медіанного фільтру до зображення.
- FindMinimalDifferenceShift – пошук найкращого зсуву другого кадру для мінімальної різниці.

Допоміжні класи та структури

- ImageInfo – структура для зберігання інформації про зображення (ширина, висота, кількість каналів).

Висновки

У ході виконання лабораторної роботи були успішно реалізовані усі етапи обробки зображень та відеопотоків. Набуто практичних навичок роботи з бібліотекою OpenCV для C++, включаючи витягування кадрів з відеофайлів, аналіз параметрів зображень, конвертацію колірних просторів та реалізацію алгоритмів обробки.

Експерименти показали, що медіаний фільтр ефективний для видалення шуму, однак час обробки значно зростає зі збільшенням розміру ядра фільтру. Для ядра 3×3 час обробки становить кілька мілісекунд, тоді як для ядра 9×9 це може бути кількість десятків мілісекунд залежно від розміру зображення.

Алгоритм вирівнювання послідовних кадрів показав себе погано при статичній камері. Але метод легко масштабується на більші діапазони зсувів або може бути оптимізований за допомогою більш складних алгоритмів, таких як кореляція або фаз-кореляція.

Реалізовані функції та класи можуть служити основою для розробки більш складних систем комп’ютерного зору, включаючи об’єктив відстеження, детектування руху та аналіз відеопотоків.