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

Кафедра САПР

Лабораторна робота №3

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

на тему:

“Трекінг (супроводження) об’єктів”

Виконав:
ст. групи ПП-44
Богдан ВЕРЕЩАК
Прийняв:
доц. Михайло МЕЛЬНИК

Львів - 2025

Мета: навчитися встановлювати відповідності між об'єктами або їх частинами на послідовності кадрів, а також визначати їх траєкторію руху і швидкість.

Лабораторне завдання

1. Встановити відповідності між об'єктами або їх частинами на послідовності кадрів.
2. Визначати траєкторію руху.
3. Визначити швидкість переміщення об’єктів на серії послідовних зображень у пікселях.

Кроки виконання завдання:

Крок 1: Встановлення відповідностей між об’єктами, або їх частинами на послідовності кадрів

На першому етапі відбувається встановлення відповідностей між об’єктами на послідовності кадрів. Для цього спершу виявляються контури рухомих об’єктів, визначаються їхні центри мас і створюються об’єкти типу movingObj, які зберігають інформацію про центр, попереднє положення, траєкторію, швидкість і колір. Кожному об’єкту присвоюється унікальний ID, що дозволяє відстежувати його протягом усього відео. Це забезпечує основу для подальшого відстеження руху і дозволяє коректно зіставляти об’єкти між кадрами навіть при невеликих зміщеннях.

На основі коду з минулої лабораторної роботи, виявлення руху здійснюється за допомогою функції absdiff, яка обчислює абсолютну різницю між двома послідовними кадрами, дозволяючи виділити змінені пікселі, після чого знаходяться маски, які ми згладжуємо, що видно на Рис. 2 і знаходяться центри маси рухомих об’єктів. На основі цих центрів ініціалізуються об’єкти типу movingObj. Функція initObjects створює ці об’єкти і присвоює їм початкові значення, готуючи систему до відстеження, результат якого можна побачити на Рис. 1.

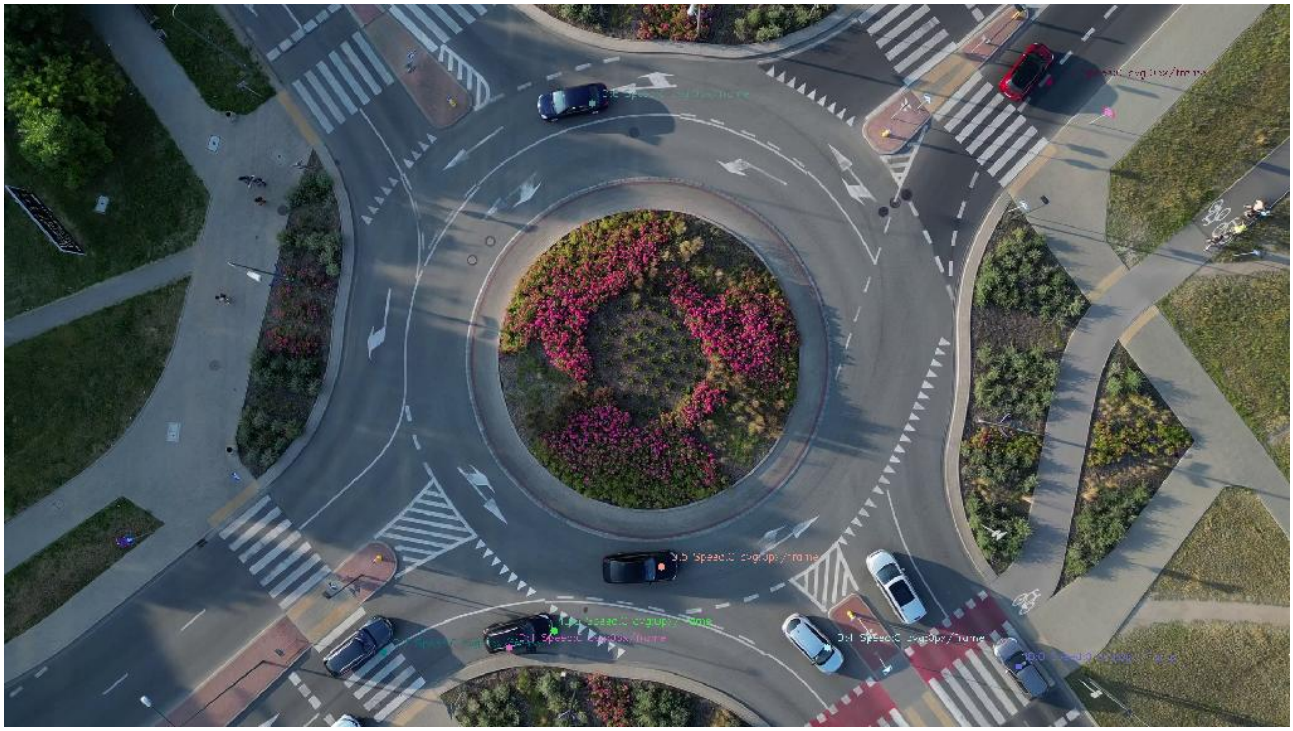

Рис. 1. Початковий кадр з початковими відповідностями між об’єктами

Рис. 2. Згладжена маска двох послідовних фреймів

Код програми:

```
struct movingObj
{
    int id;
    Point2f center;
    Point2f prevCenter;
    Rect boundingBox;
    double area;
```

```
    vector<Point2f> trajectory;
    double speed;
    double averageSpeed;
    Scalar color;
};

vector<movingObj> initObjects(const vector<Point2f>& centers) {
    vector<movingObj> objs;
    int idCounter = 0;
    for (auto& c : centers) {
        movingObj obj;
        obj.id = idCounter++;
        obj.center = c;
        obj.prevCenter = c;
        obj.speed = 0;
        obj.averageSpeed = 0;
        obj.trajectory.push_back(c);
        obj.color = Scalar(rand()%256, rand()%256, rand()%256);
        objs.push_back(obj);
    }
    return objs;
}
```

Крок 2: Визначення траєкторії руху

На другому етапі визначається траєкторія руху об’єктів. Кожен об’єкт оновлюється за допомогою функції updateObjects, яка знаходить найближчий центр до поточного положення об’єкта через findClosest, що обчислює евклідову відстань (norm) між точками і повертає індекс найближчого центру. Якщо відповідність знайдена, координати об’єкта оновлюються, до траєкторії додається нове положення, промалювання траєкторії видно на Рис. 3. Для уникнення накладень і подвоєних надписів застосовується функція removeOverlappingObjects, яка проходить по всіх об’єктах і видаляє ті, що знаходяться на одному й тому самому місці або ближче за порогову відстань, використовуючи norm для визначення відстані між центрами. Це дозволяє підтримувати чистоту трекінгу і коректне відображення тільки одного об’єкта в кожній позиції.

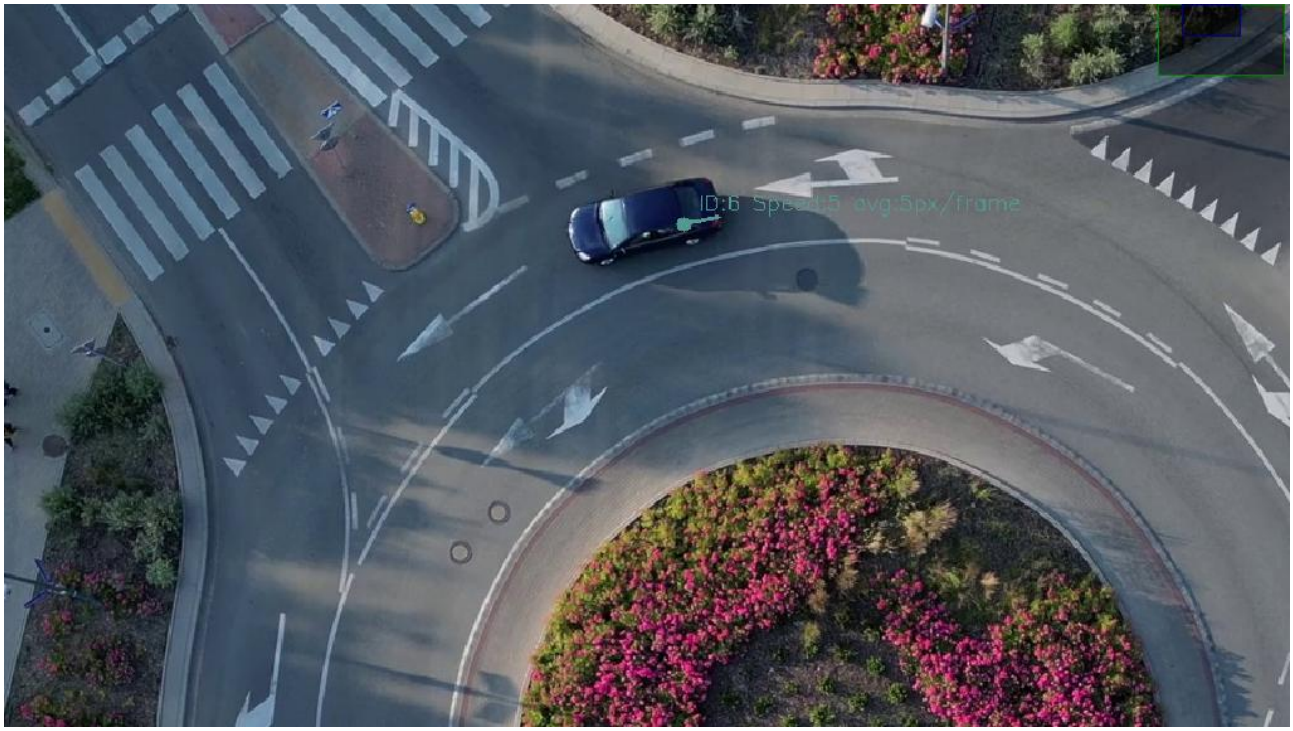

Рис. 3. Показ роботи траєкторії руху

Код програми:

```
void updateObjects(vector<movingObj>& objs, const vector<Point2f>& centers) {
    vector<bool> matched(centers.size(), false);

    for (auto& obj : objs) {
        int idx = findClosest(obj.center, centers);
        if (idx != -1) {
            obj.prevCenter = obj.center;
            obj.center = centers[idx];
            obj.trajectory.push_back(centers[idx]);
            obj.speed = norm(obj.center - obj.prevCenter);
            obj.averageSpeed = (obj.averageSpeed * (obj.trajectory.size() - 2) + obj.speed)
/ (obj.trajectory.size() - 1);
            matched[idx] = true;
        }
    }

    int nextID = objs.size();
    for (int j = 0; j < centers.size(); ++j) {
        if (!matched[j]) {
            movingObj newObj;
            newObj.id = nextID++;
            newObj.center = centers[j];
            newObj.prevCenter = centers[j];
            newObj.speed = 0;
            newObj.averageSpeed = 0;
            newObj.trajectory.push_back(centers[j]);
            newObj.color = Scalar(rand()%256, rand()%256, rand()%256);
            objs.push_back(newObj);
```

```
        }
    }
}
```

Крок 3: Визначення швидкості переміщення

На третьому етапі визначається швидкість переміщення об’єктів та їх графічне відображення. Швидкість об’єкта обчислюється як відстань між поточним і попереднім положенням. Середня швидкість обчислюється з врахуванням усієї траєкторії. Функція drawTracking малює траєкторії руху, використовуючи line для відображення послідовних положень, і circle для позначення центру об’єкта. Текстова інформація з ID, швидкістю та середньою швидкістю додається через putText. Таким чином на кожному кадрі наочно показується рух об’єктів, їхні траєкторії та параметри швидкості. Всі графічні операції виконуються на копії кадру, щоб зробити візуалізацію більш наочною.

Рис. 4. Фінальний результат визначення напрямку та швидкості рухомих об’єктів

Код програми:

```
void removeOverlappingObjects(vector<movingObj>& objs, double minDist = 10.0)
{
    for (size_t i = 0; i < objs.size(); ++i) {
        for (size_t j = i + 1; j < objs.size(); ) {
            if (norm(objs[i].center - objs[j].center) < minDist) {
                objs.erase(objs.begin() + j);
            } else {
                ++j;
```

```
        }
      }
   }
}

void drawTracking(Mat& frame, const vector<movingObj>& objs) {
   for (auto& obj : objs) {
      if (obj.trajectory.size() > 1) {
         for (int t = 1; t < obj.trajectory.size(); ++t) {
            line(frame, obj.trajectory[t-1], obj.trajectory[t], obj.color, 2);
         }
      }
      string label = "ID:" + to_string(obj.id) +
                  " Speed:" + to_string(cvRound(obj.speed)) +
                  " avg:" + to_string(cvRound(obj.averageSpeed)) + "px/frame";

                     putText(frame, label, obj.center + Point2f(10, -10),
FONT_HERSHEY_SIMPLEX, 0.5, obj.color, 1);
      circle(frame, obj.center, 5, obj.color, -1);
   }
}

int main() {
   VideoCapture cap("Traffic.mp4");
   if (!cap.isOpened()) {
      cout << "Cant open video!" << endl;
      return -1;
   }

   vector<string> filenames = saveFrames(cap, 100);
   vector<Mat> frames;

   for (auto& f : filenames) {
      frames.push_back(openImage(f, false, true));
   }

   vector<movingObj> objects;
   bool initialized = false;
     Ptr<BackgroundSubtractor>  bg  =  createBackgroundSubtractorMOG2(200,20,
false);

   for (int i = 1; i < frames.size(); ++i) {
      Mat diff = calcFrameDiff(frames[i-1], frames[i]);
      namedWindow("Diff", WINDOW_NORMAL);
      imshow("Diff",diff);


      Mat fgMask;
```

```
        bg->apply(frames[i], fgMask);
        inRange(fgMask, Scalar(200), Scalar(255), fgMask);

        Mat mask = cleanMask(diff);

        namedWindow("Mask", WINDOW_NORMAL);
        imshow("Mask",mask);
        vector<vector<Point>> contours = findMovingObjContours(mask);
        vector<Point2f> centers = getContourCenters(contours);
        if (!initialized) {
            objects = initObjects(centers);
            initialized = true;
        } else {
            updateObjects(objects, centers);
            removeOverlappingObjects(objects, 15.0);
        }

        Mat frameColor = openImage(filenames[i], false, false);
        drawTracking(frameColor, objects);
        namedWindow("Tracking", WINDOW_NORMAL);
        imshow("Tracking", frameColor);
        if (waitKey(0) == 27) break;
    }

    destroyAllWindows();
    return 0;
}
```

Крок 5 Додатково: Додати стабілізацію відео потоку

На цьому етапі до системи відстеження було додано стабілізацію відео. Її основна мета – усунути небажані коливання камери, які могли спотворювати визначення траєкторій та швидкостей рухомих об’єктів. Для цього застосовується метод оцінки оптичного потоку між послідовними кадрами, що дозволяє обчислити середнє зміщення зображення (dx, dy). Далі це зміщення згладжується експоненційним ковзним середнім (EMA), щоб усунути різкі стрибки.

Після розрахунку корекційного зсуву формується матриця афінного перетворення, яка компенсує рух камери шляхом протилежного зсуву кадру. Завдяки цьому отримане стабілізоване відео не містить тремтіння, а об’єкти на ньому переміщуються рівномірно та плавно.

Для більшої точності стабілізації передбачено обчислення зміщення двома методами – через оптичний потік (calcOpticalFlowPyrLK) і через фазову кореляцію (phaseCorrelate). Якщо кількість якісних точок недостатня, система автоматично переходить до другого способу, що підвищує надійність стабілізації.

Додатково реалізовано візуальну сітку для аналізу результатів, що складається з чотирьох вікон: стабілізованого відео з трекінгом, моделі фону, очищеної маски переднього плану та тіней. Це дозволяє оцінити вплив стабілізації на якість детекції та відстеження.

Разом з стабілізаціє я розраховував фон зображення за допомогою createBackgroundSubtractorMOG2, яка одночасно з тим виявляла тіні, хоча не дуже ефективно, які я потім віднімав від маски зображення.

У результаті застосування стабілізації траєкторії об’єктів стали більш рівними, швидкості – точнішими, а кількість помилкових виявлень суттєво зменшилася. Це значно покращило загальну якість аналізу руху транспортних засобів на відео.

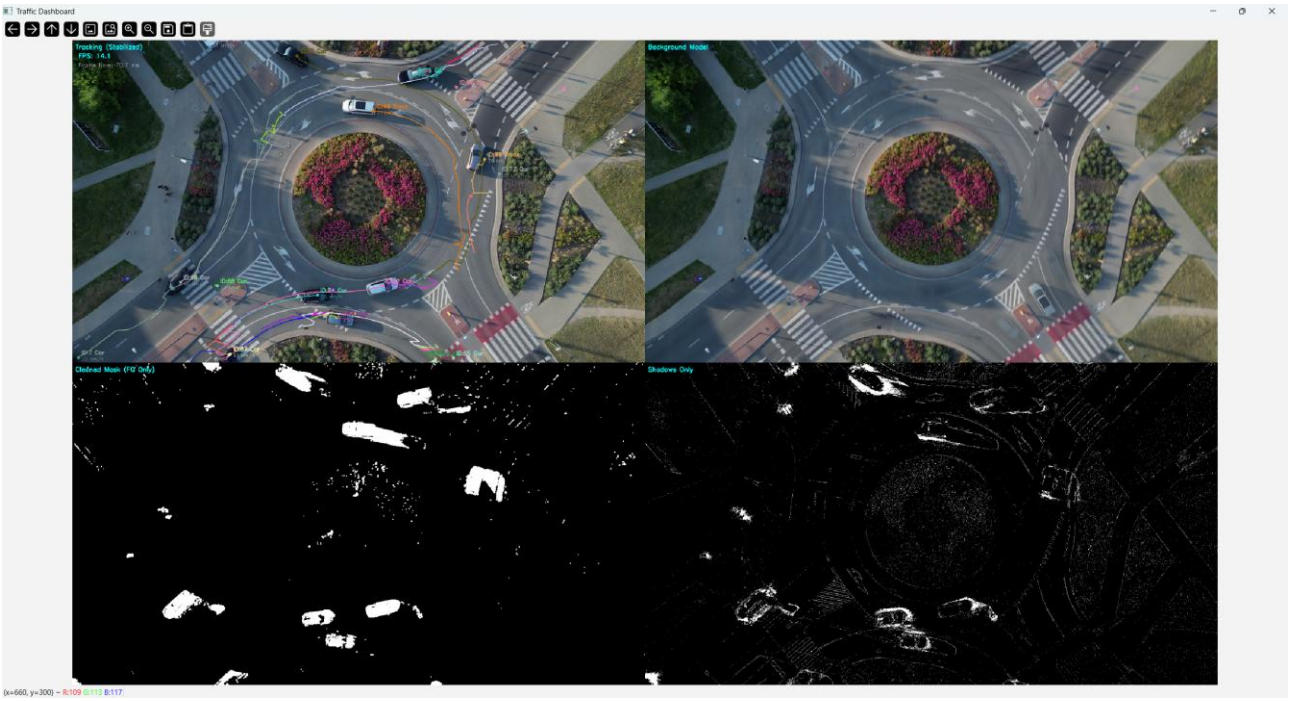

Рис. 5. Результати знаходження маски, фону, тіней та відстеження об’єктів після стабілізації

Код програми:

```
// --- 1. Стабілізація ---
    Mat cur_frame = frame.clone();
    Mat cur_grey;
    cvtColor(cur_frame, cur_grey, COLOR_BGR2GRAY);

    Mat fgMaskPrev;
    bg->apply(prev_frame, fgMaskPrev);
    Mat staticMaskPrev;
    threshold(fgMaskPrev, staticMaskPrev, 200, 255, THRESH_BINARY_INV);

    if (prev_pts.size() < 80) {
       vector<Point2f> candidates;
       goodFeaturesToTrack(prev_grey, candidates, 100, 0.01, 8, staticMaskPrev);
       prev_pts = candidates;
    }
```

```
        vector<Point2f> cur_pts;
        vector<uchar> status;
        vector<float> err;

        if (!prev_pts.empty()) {
            calcOpticalFlowPyrLK(prev_grey, cur_grey, prev_pts, cur_pts, status, err,
Size(25,25), 3);
        }

        vector<double> dxs, dys;
        for (size_t i = 0; i < status.size(); ++i) {
            if (status[i]) {
                double dx = cur_pts[i].x - prev_pts[i].x;
                double dy = cur_pts[i].y - prev_pts[i].y;
                dxs.push_back(dx);
                dys.push_back(dy);
            }
        }

        double dx = 0.0, dy = 0.0;
        bool haveGoodEstimate = false;

        if (dxs.size() >= 6) {
            auto median = [](vector<double>& v) -> double {
                size_t n = v.size();
                nth_element(v.begin(), v.begin() + n/2, v.end());
                double med = v[n/2];
                if (n % 2 == 0) {
                    double a = *max_element(v.begin(), v.begin() + n/2);
                    med = (med + a) / 2.0;
                }
                return med;
            };
            dx = median(dxs);
            dy = median(dys);
            haveGoodEstimate = true;
        } else {
            Mat f1, f2;
            prev_grey.convertTo(f1, CV_32F);
            cur_grey.convertTo(f2, CV_32F);
            Point2d p = phaseCorrelate(f1, f2);
            dx = p.x;
            dy = p.y;
            if (std::hypot(dx, dy) < 1000.0) haveGoodEstimate = true;
        }

        // Згладжування (EMA)
```

```
smooth_dx = ALPHA * smooth_dx + (1.0 - ALPHA) * dx;
smooth_dy = ALPHA * smooth_dy + (1.0 - ALPHA) * dy;

// Побудова матриці трансформації для чистого зміщення
Mat transformation_matrix = Mat::eye(2, 3, CV_64F);
transformation_matrix.at<double>(0, 2) = -smooth_dx;
transformation_matrix.at<double>(1, 2) = -smooth_dy;

Mat stabilized_frame;
        warpAffine(cur_frame, stabilized_frame, transformation_matrix, cur_frame.size(), INTER_LINEAR, BORDER_REPLICATE);
```

Висновки

У ході виконання лабораторної роботи я навчився встановлювати відповідності між об’єктами або їх частинами на послідовності кадрів, визначати їх траєкторію руху та швидкість. Реалізований алгоритм дозволяє відстежувати об’єкти на відео, зберігаючи їх унікальні ідентифікатори, оновлюючи положення та обчислюючи швидкість переміщення.

Додатково було виконано стабілізацію відеопотоку, що дозволило зменшити вплив коливань камери та покращити точність визначення рухомих об’єктів. Завдяки цьому траєкторії стали більш плавними, а результати відстеження – стабільнішими та ближчими до реальних переміщень об’єктів.