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
Верещак Б. О.
Прийняв:
Асис. Мельник М. Р.

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.

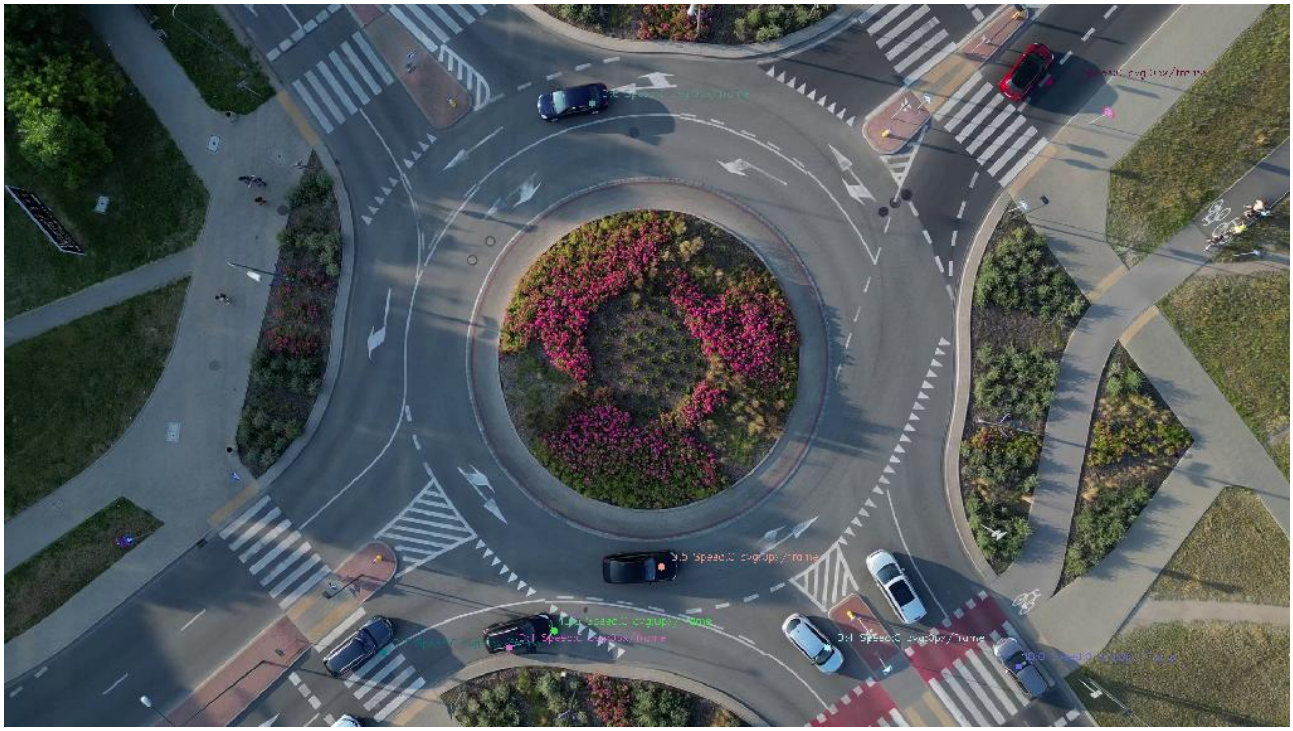

Рис. 1. Початковий кадр з початковими відповідностями між об’єктами

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.

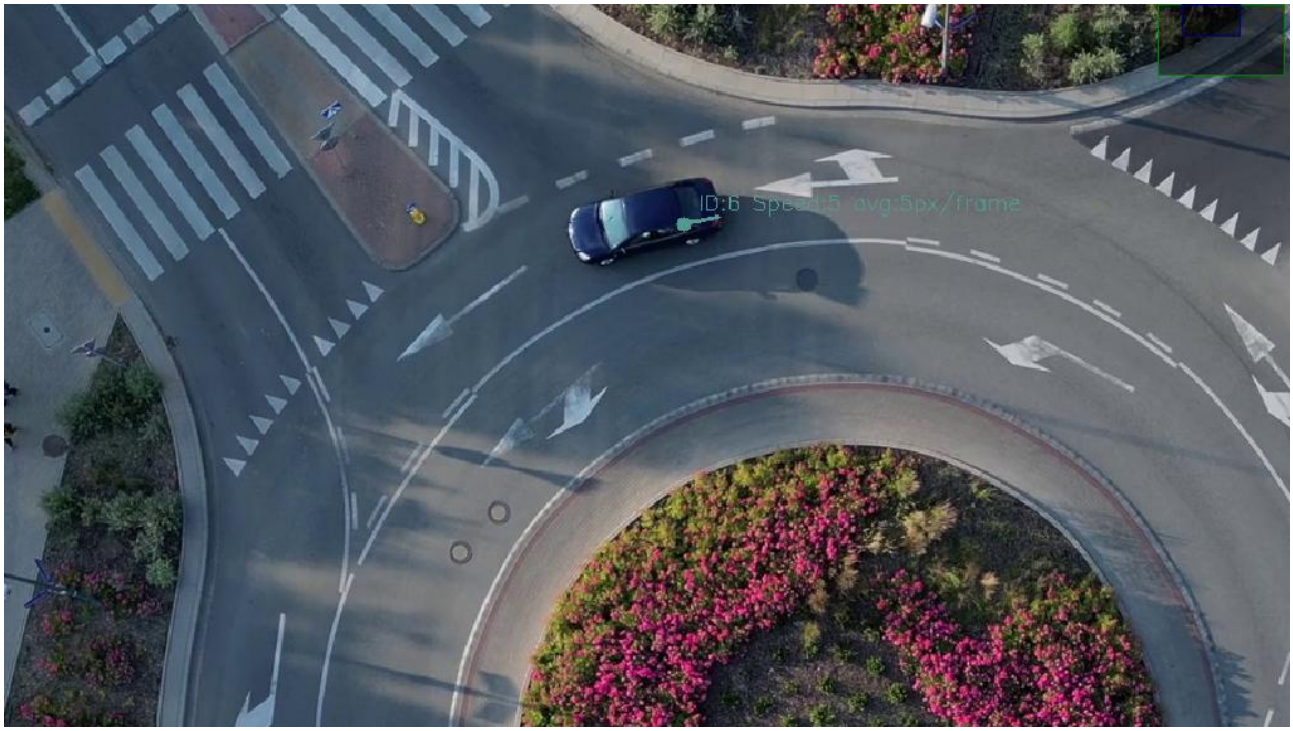

Рис. 3. Показ роботи траєкторії руху

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;
}
```

Висновки