**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ТВ**

**отчЁт**

**по лабораторной работе № 2**

**по дисциплине «Интеллектуальный анализ данных»**

**Тема: Сегментация движущихся объектов на основе оценки энергии движения**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 9105 |  | Шаривзянов Д. Р. |
|  |  | Чугунов Р. |
| Преподаватель |  | Поздеев А. А. |

Санкт-Петербург

2024

СЕГМЕНТАЦИЯ ДВИЖУЩИХСЯ ОБЪЕКТОВ НА ОСНОВЕ ОЦЕНКИ ЭНЕРГИИ ДВИЖЕНИЯ

**Цель работы:** моделирование алгоритма сегментации движущегося объекта с помощью оценки энергии движения. Исходными данными являются два кадра видеопоследовательности со статичным фоном, на которых присутствует перемещающийся объект.

1. Код программы

#ifndef MOVEMENT\_ENERGY\_H

#define MOVEMENT\_ENERGY\_H

#include <vector>

#include <opencv2\opencv.hpp>

#include "logger.hpp"

using namespace cv;

using namespace std;

class Image {

public:

    Mat bgr1;

    Mat bgr2;

    Mat gray1;

    Mat gray2;

    Mat diff;

    Mat diffAndEdges;

    Mat edges;

    Mat morph;

    vector<int> horizontalProj;

    vector<int> verticalProj;

    Mat hist\_hor;

    Mat hist\_ver;

    Mat hist\_dia;

    int minLength = 100;    // длина сегмента

    int thresh = 100;       // пороговое значение для определения сегментов

};

/\*\*

 \* @brief Функция для морфологической обработки изображения

 \*

 \* Функция morphProc() предназначена для морфологической обработки

 \* двоичного изображения. Она удаляет шумы, оставляя только общие

 \* контуры объектов на изображении.

 \*

 \* @param src двоичное изображение, которое подвергается

 \*            морфологической обработке

 \* @param dst двоичное изображение, которое является результатом

 \*            морфологической обработки

 \*/

void morphProc(const Mat& src, Mat& dst) {

    // Применяем морфологическую операцию "закрытие" для удаления шумов

    // размер ядра - 31 на 31, центр ядра - (16, 16)

    Mat element = cv::getStructuringElement(MORPH\_RECT, Size(31, 31), Point(16, 16));

    morphologyEx(src, dst, MORPH\_ERODE, element);

}

/\*\*

 \* @brief Calculate horizontal and vertical projections of a binary image

 \*

 \* @param [in] image Binary image

 \* @param [out] horizontalProjection Vector of horizontal projection values

 \* @param [out] verticalProjection Vector of vertical projection values

 \*/

void calculateProjections(const Mat& image, std::vector<int>& horizontalProjection, std::vector<int>& verticalProjection)

{

    // Инициализизация нулевого вектора проекций изображения с размером, соответсвующим высоте и ширине изображения

    horizontalProjection.resize(image.rows, 0);

    verticalProjection.resize(image.cols, 0);

    // Подсчет ненулевых пикселей по строкам и столбцам

    for (int row = 0; row < image.rows; ++row)

        for (int col = 0; col < image.cols; ++col)

            if (image.at<uchar>(row, col) > 0) {

                ++horizontalProjection[row];

                ++verticalProjection[col];

            }

}

/\*\*

 \* @brief Выделяет проекции по заданному порогу

 \*

 \* @param[in] proj Вектор проекции

 \* @param[out] segments Вектор сегментов

 \* @param[in] thresh Пороговое значение

 \* @param[in] minLength Минимальная длина сегмента

 \*/

void findProjections(const vector<int>& proj, vector<pair<int, int>>& segments, int thresh, int minLength) {

    int start = -1; // Начало сегмента

    // Проходим по вектору проекции

    for (int i = 0; i < proj.size(); ++i) {

        // Если значение проекции больше порога, то

        if (proj[i] > thresh) {

            // Если старт не установлен, то

            if (start == -1)

                start = i; // Установка стартового значения

        }

        else

            // Если старт установлен, то

            if (start != -1) {

                // Если длина сегмента больше пороговой длинны, то

                if (i - start >= minLength)

                    segments.push\_back({start, i - 1}); // Сохраняем сегмент

                start = -1; // Сброс

            }

    }

    // Проверка на конец массива

    if (start != -1 && proj.size() - start >= minLength)

        segments.push\_back({start, proj.size() - 1}); // Сохраняем последний сегмент

}

/\*\*

 \* @brief Функция отображения гистограмм горизонтальной и вертикальной проекций

 \*

 \* Данная функция создает и отображает гистограммы горизонтальной и вертикальной

 \* проекций двоичного изображения. Гистограммы нормализуются и отображаются на

 \* белом фоне. На гистограммах также отображается уровень отсечки, заданный

 \* пользователем.

 \*

 \* @param img Структура Image, содержащая параметры изображения

 \* @param horizontalProj Вектор значений горизонтальной проекции

 \* @param verticalProj Вектор значений вертикальной проекции

 \*/

void histShow(const Image& img, const vector<int>& horizontalProj, const vector<int>& verticalProj) {

    int hist\_height = 200; // Высота гистограммы

    // Создание черных изображений для гистограмм

    Mat histImageH = Mat::zeros(hist\_height, horizontalProj.size(), CV\_8UC3);

    Mat histImageV = Mat::zeros(hist\_height, verticalProj.size(), CV\_8UC3);

    // Заливка изображений белым цветом

    histImageH.setTo(Scalar(255, 255, 255));

    histImageV.setTo(Scalar(255, 255, 255));

    // Нахождение максимальных значений проекций для нормализации

    int horProjMax = \*max\_element(horizontalProj.begin(), horizontalProj.end());

    int vertProjMax = \*max\_element(verticalProj.begin(), verticalProj.end());

    // Отрисовка гистограммы горизонтальной проекции, если максимум больше нуля

    if (horProjMax > 0)

        for (int i = 0; i < horizontalProj.size(); ++i)

            line(histImageH, Point(i, hist\_height), Point(i, hist\_height - (horizontalProj[i] \* hist\_height / horProjMax)), Scalar(0, 0, 0));

    // Отрисовка гистограммы вертикальной проекции, если максимум больше нуля

    if (vertProjMax > 0)

        for (int i = 0; i < verticalProj.size(); ++i)

            line(histImageV, Point(i, hist\_height), Point(i, hist\_height - (verticalProj[i] \* hist\_height / vertProjMax)), Scalar(0, 0, 0));

    // Определение и отрисовка уровня отсечки на гистограммах

    int thresholdYh = hist\_height - (img.thresh \* hist\_height / horProjMax);

    int thresholdYv = hist\_height - (img.thresh \* hist\_height / vertProjMax);

    line(histImageH, Point(0, thresholdYh), Point(horizontalProj.size(), thresholdYh), Scalar(255, 0, 0), 2);

    line(histImageV, Point(0, thresholdYv), Point(verticalProj.size(), thresholdYv), Scalar(255, 0, 0), 2);

    // Отображение гистограмм в отдельных окнах

    imshow("Horizontal Projection", histImageH);

    imshow("Vertical Projection", histImageV);

}

/\*\*

 \* @brief Сегментация изображения на основе проекций

 \*

 \* Данная функция проводит сегментацию двоичного изображения на основе

 \* горизонтальной и вертикальной проекций. Она находит сегменты на каждой

 \* проекции, а затем выделяет прямоугольники, пересекающиеся сегменты

 \* которых образуют итоговые сегменты изображения.

 \*

 \* @param img Структура Image, содержащая параметры изображения

 \*

 \* @return Вектор прямоугольников, образующих сегменты изображения

 \*/

vector<Rect> segmentImage(const Image& img) {

    // Векторы для хранения проекций

    // vector<int> horizontalProj, verticalProj;

    vector<int> horizontalProj = img.horizontalProj;

    vector<int> verticalProj = img.verticalProj;

    calculateProjections(img.morph, horizontalProj, verticalProj);

    // logger.info("Horizontal projection: {}", fmt::format("[{}]", fmt::join(horizontalProj, ", ")));

    // logger.info("Vertical projection: {}", fmt::format("[{}]", fmt::join(verticalProj, ", ")));

    // Векторы для хранения сегментов

    vector<pair<int, int>> horizontalSegments, verticalSegments;

    // находим сегменты по горизонтальной и вертикальной проекциям

    findProjections(horizontalProj, horizontalSegments, img.thresh, img.minLength);

    findProjections(verticalProj, verticalSegments, img.thresh, img.minLength);

    // отображение гистограмм

    histShow(img, horizontalProj, verticalProj);

    // Вектор для хранения прямоугольников

    vector<Rect> rectangles;

    // выделяем прямоугольники на основе проекций

    for (const auto& hSeg : horizontalSegments)

        for (const auto& vSeg : verticalSegments)

            rectangles.push\_back(Rect(Point(vSeg.first, hSeg.first), Point(vSeg.second, hSeg.second)));

    return rectangles; // возвращаем список прямоугольников

}

/\*\*

 \* @brief Обновляет сегментированное изображение

 \*

 \* Функция updateSegmentedImage() создает сегментированное изображение на основе входных данных,

 \* рисуя прямоугольники вокруг обнаруженных сегментов.

 \*

 \* @param img Структура Image, содержащая параметры изображения

 \*/

void updateSegmentedImage(const Image& img) {

    // Клонируем второе изображение для последующего редактирования

    Mat result = img.bgr2.clone();

    // Получаем список прямоугольников, образующих сегменты изображения

    vector<Rect> rectangles = segmentImage(img);

    // Рисуем прямоугольники вокруг сегментов

    for (const auto& rect : rectangles)

        rectangle(result, rect, Scalar(255, 0, 0), 2);

    // Отображаем сегментированное изображение

    imshow("Segmented Image", result);

}

void onMinLengthChange(int, void\* userdata) {

    Image\* img = static\_cast<Image\*>(userdata);

    updateSegmentedImage(\*img);

}

void onThresholdChange(int, void\* userdata) {

    Image\* img = static\_cast<Image\*>(userdata);

    updateSegmentedImage(\*img);

}

/\*\*

 \* @brief Функция для детекции движения на двух изображениях

 \*

 \* Функция lab2\_MovementEnergy() предназначена для детекции движения на двух

 \* изображениях. Она вычитает из второго изображения первое, получая изображение

 \* разности. Затем она получает контурное препарата с помощью алгоритма Собеля,

 \* применяет логическое И к изображению разности и контурному препарату,

 \* проводит морфологическую обработку с помощью операции "закрытие" и

 \* инвертирует полученное изображение.

 \*

 \* @param img\_bgr1 первое изображение в формате BGR

 \* @param img\_bgr2 второе изображение в формате BGR

 \*/

void lab2\_MovementEnergy(Mat& img\_bgr1, Mat& img\_bgr2) {

    Image img;

    img.bgr1 = img\_bgr1;

    img.bgr2 = img\_bgr2;

    // конвертируем изображения в градации серого

    cvtColor(img\_bgr1, img.gray1, COLOR\_BGR2GRAY);

    cvtColor(img\_bgr2, img.gray2, COLOR\_BGR2GRAY);

    // отображаем изображения

    imshow("img bgr 1", img.bgr1);

    imshow("img bgr 2", img.bgr2);

    // вычитаем из второго изображения первое, получая изображение разности

    absdiff(img.gray1, img.gray2, img.diff);

    imshow("absdiff", img.diff);

    // оператор Собеля

    // Mat grad\_x, grad\_y;

    // Mat abs\_grad\_x, abs\_grad\_y;

    // int scale = 1;

    // int delta = 0;

    // int ddepth = CV\_16S;

    // Sobel(img.gray2, grad\_x, ddepth, 1, 0, 3, scale, delta, BORDER\_DEFAULT);

    // Sobel(img.gray2, grad\_y, ddepth, 0, 1, 3, scale, delta, BORDER\_DEFAULT);

    // convertScaleAbs(grad\_x, abs\_grad\_x);

    // convertScaleAbs(grad\_y, abs\_grad\_y);

    // addWeighted(abs\_grad\_x, 0.5, abs\_grad\_y, 0.5, 0, img.edges);

    // оператор Кэнни

    Canny(img.gray2, img.edges, 50, 200);

    // отображаем контурное препарата

    imshow("edges", img.edges);

    // применяем логическое И

    bitwise\_and(img.diff, img.edges, img.diffAndEdges);

    imshow("absdiff AND edges", img.diffAndEdges);

    // задаем пороговое значение

    threshold(img.diffAndEdges, img.diffAndEdges, 5, 255, THRESH\_BINARY\_INV);

    // проводим морфологическую обработку

    morphProc(img.diffAndEdges, img.morph);

    imshow("absdiff AND edges eroded", img.morph);

    // инвертируем изображение

    bitwise\_not(img.morph, img.morph);

    imshow("absdiff AND edges eroded and inverted", img.morph);

    // создаем окна

    namedWindow("Segmented Image", WINDOW\_AUTOSIZE);

    // создаем ползунки

    createTrackbar("Min Length", "Segmented Image", &img.minLength, 500, onMinLengthChange, &img);

    createTrackbar("Threshold", "Segmented Image", &img.thresh, 500, onThresholdChange, &img);

    waitKey(0);

}

#endif MOVEMENT\_ENERGY\_H

1. Описание работы

Исходными данными являются два кадра видеопоследовательности с движущейся машиной и «статичным» фоном. Сначала был получен препарат межкадровой разности.





Рис. 1. Пара кадров (верхний ряд) и препарат межкадровой разности (внизу).

Далее с помощью фильтра Кэнни был получен контурный препарат.

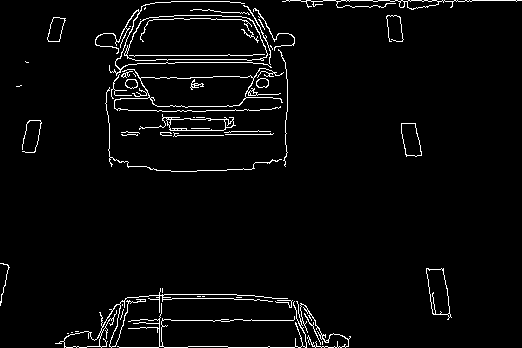


Рис. 2. Контурный препарат (фильтр Кэнни).

Далее после объединения препаратов по логическому «И» было получено следующее изображение.

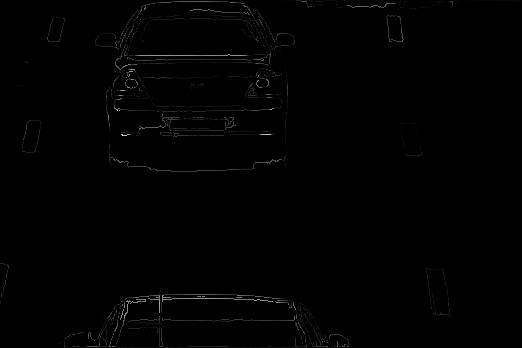


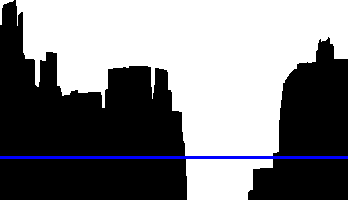
Рис. 3. Объединение препаратов по логическому «И».

Далее применяем морфологическое закрытие с размером ядра 31\*31 для формирования больших блоков вокруг объекта.



Рис. 4. Инвертированный морфологический препарат.

Затем выделяем проекции – совокупность строк (столбцов), у которых число ненулевых пикселей превышает некий порог и которые образуют на гистограмме непрерывный отрезок. Проекции, длина которых меньше заданной величины (определяемой априорным знанием о предполагаемых размерах объектов), отбрасывают, чтобы исключить изолированные помехи. На базе оставшихся проекций формируют стробы. В нашем случае минимальная длина отрезка проекции и порог были эмпирически подобраны как 100 и 100.



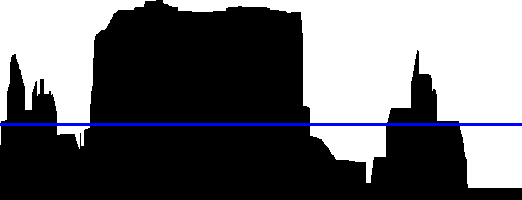


Рис. 5. Проекции морфологического препарата с отсечкой по порогам.



Рис. 6. Наложение строба на объект.

**Вывод:** в ходе работы был реализован алгоритм сегментации движущегося объекта с помощью оценки энергии движения. Также в ходе работы был изучен и реализован алгоритм проекций. Выяснилось, что если с помощью подбора порогов в алгоритме проекций попытаться выделить больше объектов – то проекции объектов начинают перекрывают друг друга, и из-за этого не удаётся установить центр тяжести сегментируемого объекта, что приводит к ошибке сегментации.