Relatório de projecto VCOM

Cláudio Monteiro - 201105084
João Soares - 201200740

05/11/2016

Especificações adicionais

Ao correr o programa indicar o nome da imagem a testar. São abertas duas janelas, uma com a imagem a testar e outra com a imagem do parque vazio. Para criar uma região de interesse clicar em dois pontos na mesma fila para criar um retângulo horizontal, o qual será analisado e onde os resultados serão apresentados.

Descrição do algoritmo proposto

Para demonstrar o máximo de conhecimento possível obtido na cadeira começou-se por aplicar *GaussianBlur* para diminuir o ruído da imagem seguido do *CannyThreshold* e logo a seguir *dilate.* Com o resultado do CannyThreshold foi possível calcular os contornos das linhas do parque vazio o que permitiu depois obter a distância entre as linhas de modo a calcular os tamanhos dos retângulos necessários para a detecção dos lugares de estacionamento. Como algoritmo de comparação foi utilizado o cálculo e comparação de histogramas.

Comentários relevantes acerca da eficácia dos métodos usados, descrevendo os problemas principais e soluções propostas

O grupo começou por utilizar MatchTemplate() mas deparou-se logo com grandes limitações porque para funcionar precisava de condições óptimas o que forçava o grupo a usar apenas um determinado tipo de imagens, enquanto que o algoritmo final permitiu utilizar fotografias em diferentes alturas do dia por exemplo.

O método utilizado não é de longe o mais eficaz, mas provavelmente um dos mais simples de aplicar e de obter resultados satisfatórios. Houveram dois problemas que se podem dizer principais, o primeiro foi no cálculo do threshold que apesar da aplicação de blur e dilate não foi possível eliminar a 100% algum do ruído que aparecia em certos casos e que apesar de acabar por dar resultados certos, por vezes colocava 2 retângulos num lugar. O segundo problema foi da parte do cálculo de histogramas, como não é um algoritmo complexo por vezes pode dar falsos positivos, apesar do grupo ter experimentado inúmeras vezes com a comparação.

A melhor solução para o projecto passa por utilizar um algoritmo mais complexo mas que permite melhores resultados. Não ser necessária uma fotografia do parque vazio também era uma melhoria bem vinda.

Estado da solução proposta e grau de cumprimento

No geral o grupo encontra-se satisfeito com a realização do trabalho que acabou por ir de encontro às expectativas iniciais.

Código com comentários em anexo.

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/opencv.hpp>
#include <iostream>
#include <string>
using namespace cv;
using namespace std;
string windowname = "Region of Interest";
Mat image, imageEmpty, regionOfInterest, regionOfInterestEmpty, greyroi, greyroiempty,
detected_edges;
vector<Point> roi, rect;
int nRois = 1;
int size[2];

// operation needed to sort the contours vector
struct contour_sorter // 'less' for contours
{
	bool operator ()(const vector<Point>& a, const vector<Point> & b)
	{
		Rect ra(boundingRect(a));
		Rect rb(boundingRect(b));
		// scale factor for y should be larger than img.width
		return ((ra.x) < (rb.x));
	}
};

/*
void matchTemplate(Point start, Point finish) {
	cv::Mat input = regionOfInterestEmpty;
	cv::Mat gray;
	cv::cvtColor(input, gray, CV_BGR2GRAY);

	cv::Mat templ = regionOfInterest(Rect(start, finish));

	cv::Mat img = input;
	cv::Mat result;
	/// Create the result matrix
	int result_cols = img.cols - templ.cols + 1;
	int result_rows = img.rows - templ.rows + 1;

	result.create(result_cols, result_rows, CV_32FC1);
```

```
	int match_method = TM_SQDIFF_NORMED;

	/// Do the Matching and Normalize
	matchTemplate(img, templ, result, match_method);
	//normalize(result, result, 0, 1, cv::NORM_MINMAX, -1, cv::Mat());

	/// Localizing the best match with minMaxLoc
	double minVal; double maxVal; cv::Point minLoc; cv::Point maxLoc;
	cv::Point matchLoc;

	minMaxLoc(result, &minVal, &maxVal, &minLoc, &maxLoc, cv::Mat());
	/// For SQDIFF and SQDIFF_NORMED, the best matches are lower values. For all the
other methods, the higher the better
	if (match_method == CV_TM_SQDIFF || match_method == CV_TM_SQDIFF_NORMED)
	{
		matchLoc = minLoc;

		if (minVal <= 0.05) {
			cv::rectangle(input, start, finish, CV_RGB(0, 255, 0), 2, 8, 0);
			cv::rectangle(result, start, finish, CV_RGB(0, 255, 0), 2, 8, 0);
		}
		else {
			cv::rectangle(input, start, finish, CV_RGB(255, 0, 0), 2, 8, 0);
			cv::rectangle(result, start, finish, CV_RGB(255, 0, 0), 2, 8, 0);
		}
	}
}
*/

// Histogram calculation similar to the openCV book
void hist(Point start, Point finish) {

	//Histogram calculation regarding the empty version of the park
	Mat input = regionOfInterestEmpty(Rect(start, finish));
	//cv::Mat input = regionOfInterest;
	Mat gray;
	cvtColor(input, gray, CV_BGR2GRAY);
	//namedWindow("Gray", 1);    imshow("Gray", gray);
	equalizeHist(gray, gray);


	// Set histogram bins count
	int bins = 256;
	int histSize[] = { bins };
	// Set ranges for histogram bins
	float lranges[] = { 0, 255 };
	const float* ranges[] = { lranges };
	// create matrix for histogram

	int channels[] = { 0 };

	// create matrix for histogram visualization
	int const hist_height = 256;
	Mat3b hist_image = Mat3b::zeros(hist_height, bins);
```

```
	// Calculate histogram
	MatND hist;
	calcHist(&gray, 1, channels, Mat(), hist, 1, histSize, ranges, true, false);
	normalize(hist, hist, 0, 1, NORM_MINMAX, -1, Mat());

	//Histogram calculation regarding the full version of the park
	cv::Mat templ = regionOfInterest(Rect(start, finish));
	cv::Mat grayroi;
	cv::cvtColor(templ, grayroi, CV_BGR2GRAY);
	//namedWindow("Gray ROI", 1);    imshow("Gray ROI", grayroi);
	equalizeHist(grayroi, grayroi);
	MatND histroi;
	calcHist(&grayroi, 1, channels, Mat(), histroi, 1, histSize, ranges, true, false);
	normalize(histroi, histroi, 0, 1, NORM_MINMAX, -1, Mat());

	// Histograms comparison
	double matchvalue = compareHist(histroi, hist, CV_COMP_CHISQR);


	// Comparision between histograms and further paiting of rectangles
	if (matchvalue < 15.50) {
		cv::rectangle(regionOfInterestEmpty, start, finish, CV_RGB(0, 255, 0), 2, 8,
0);
		cv::rectangle(regionOfInterest, start, finish, CV_RGB(0, 255, 0), 2, 8, 0);
	}
	else {
		cv::rectangle(regionOfInterestEmpty, start, finish, CV_RGB(255, 0, 0), 2, 8,
0);
		cv::rectangle(regionOfInterest, start, finish, CV_RGB(255, 0, 0), 2, 8, 0);
	}

}

// Callback function to allow the user to use the left mouse button
void CallBackFunction(int event, int x, int y, int flags, void* point)
{

	if (event == EVENT_LBUTTONDOWN)
	{
		Point p;
		// feedback to the position
		cout << "Left button of the mouse is clicked - position (" << x << ", " << y
<< ")" << endl;
		p.x = x;
		p.y = y;
		//save point on the region of interess vector
		roi.push_back(p);
	}

}
// Main function that calculates Canny, Contours, park lines and calls for the histogram
calculation
void detect_edges(Mat &imageToDetect) {
	// Blur to remove noise from the image
```

```
	GaussianBlur(greyroiempty, imageToDetect, Size(3, 3), 3);
	Mat ignore;
	double otsu_thresh_val = threshold(imageToDetect, ignore, 0, 255, CV_THRESH_BINARY
| CV_THRESH_OTSU);
	double high_thresh_val = otsu_thresh_val, lower_thresh_val = otsu_thresh_val * 0.5;
	// Canny com valores definidos pelo otsu
stackoverflow.com/questions/4292249/automatic-calculation-of-low-and-high-thresholds-for-th
e-canny-operation-in-open
	Canny(imageToDetect, imageToDetect, lower_thresh_val, high_thresh_val);
	dilate(imageToDetect, imageToDetect, MORPH_RECT);

	//Contour calculation and draw similar to the one from the books
	vector<vector<Point> > contours;
	vector<Vec4i> hierarchy;
	findContours(imageToDetect, contours, hierarchy, CV_RETR_CCOMP,
CV_CHAIN_APPROX_NONE, Point(0, 0));

	/// Draw contours
	RNG rng(12345);
	Mat drawing = Mat::zeros(imageToDetect.size(), CV_8UC3);
	for (int i = 0; i < contours.size(); i++)
	{
		Scalar color = Scalar(rng.uniform(0, 255), rng.uniform(0, 255),
rng.uniform(0, 255));
		drawContours(drawing, contours, i, color, 2, 8, hierarchy, 0, Point());
	}
	// sorting of the contours vector to allow better line calculation
	std::sort(contours.begin(), contours.end(), contour_sorter());

	//Verification for the first point followed by the histogram calculation, condition
needed to avoid array out of bounds
	if (contours[0][0].x - 5 >= 0) {
		hist(Point(3, 0), Point(contours[0][0].x - 5, imageToDetect.size().height /
2));
	}
	else {
		hist(Point(3, 0), Point(0, imageToDetect.size().height / 2));
	}


	// same as above but this time with a for cycle.
	for (int i = 0; i < contours.size() - 1; i++)
	{
		if ((contours[i + 1][0].x - contours[i][0].x) <= 15)
			continue;
		else {
			if (contours[i + 1][0].x - 3 >= 0) {
				hist(Point(contours[i][0].x + 3, 0), Point(contours[i +
1][0].x - 3, imageToDetect.size().height / 2));
			}
			else {
				hist(Point(contours[i][0].x + 3, 0), Point(0,
imageToDetect.size().height / 2));
			}
```

```
                }
        }

        // Last call for the histogram function
        hist(Point(contours[contours.size() - 1][0].x + 3, 0),
Point(imageToDetect.size().width - 3, imageToDetect.size().height / 2));

        //print of the contours graph
        namedWindow("Contours", CV_WINDOW_AUTOSIZE);
        imshow("Contours", drawing);

}

int main(int argc, char** argv)
{
        // Input field for the name of the picture
        Point p;
        string imagename;
        cout << "Please enter the name of the file: ";
        getline(cin, imagename);
        cout << "The value you entered is " << imagename << "\n\n";
        image = imread(imagename, CV_LOAD_IMAGE_COLOR); // Read the file

        // While cycle until the user provides a proper name for a file.
        while (!image.data) // Check for invalid input
        {
                cout << "Could not open or find the image" << std::endl;
                cout << "Please enter the name of the file: ";
                getline(cin, imagename);
                cout << "The value you entered is " << imagename;
                image = imread(imagename, CV_LOAD_IMAGE_COLOR); // Read the file
        }
        //definition of the empty parking lot and regular parking lot windows
        imageEmpty = imread("pls.jpg");
        namedWindow("Parking Lot", WINDOW_AUTOSIZE);// Create a window for display.
        namedWindow("Empty Parking Lot", WINDOW_AUTOSIZE);// Create a window for display.
        setMouseCallback("Parking Lot", CallBackFunction, NULL);
        imshow("Parking Lot", image); // Show our image inside it.
        imshow("Empty Parking Lot", imageEmpty); // Show our image inside it.
        cout << "Select two points to create the region of interest and press SPACE" <<
endl;

        // while to allow selection of multiple ROI
        while (nRois <= 5) {
                // wait for "SPACE"
                while (cv::waitKey(1) != 32);

                if (nRois > 1) {
                        destroyWindow("Contours");
                        destroyWindow("ROI CANNY");
                        destroyWindow("REGION OF INTEREST EMPTY");
                        destroyWindow("REGION OF INTEREST");
                        regionOfInterest.release();
                        regionOfInterestEmpty.release();
```

```
                greyroi.release();
                greyroiempty.release();
                detected_edges.release();
            }
            // ROI from both the empty and normal parking lot
            regionOfInterest = image(Rect(roi[0].x, roi[0].y, roi[1].x - roi[0].x,
(roi[1].y - roi[0].y) ));
            regionOfInterestEmpty = imageEmpty(Rect(roi[0].x, roi[0].y, roi[1].x -
roi[0].x, (roi[1].y - roi[0].y) ));
            cvtColor(regionOfInterestEmpty, greyroiempty, COLOR_RGB2GRAY);



            roi.clear();

            //caling of the main function
            detect_edges(detected_edges);

            // Window of both region of interest
            //namedWindow("ROI CANNY", WINDOW_AUTOSIZE);
            namedWindow("REGION OF INTEREST EMPTY", WINDOW_AUTOSIZE);
            namedWindow("REGION OF INTEREST", WINDOW_AUTOSIZE);

            //imshow("ROI CANNY", detected_edges); // Show our image inside it.
            imshow("REGION OF INTEREST EMPTY", regionOfInterestEmpty);
            imshow("REGION OF INTEREST", regionOfInterest); // Show our image inside it.

            cout << "ROI created\n\n";

            nRois++;

        }



        waitKey(0); // Wait for a keystroke in the window

        return 0;
}
```