Санкт-Петербургский политехнический университет Петра Великого
Институт машиностроения, материалов и транспорта
Высшая школа автоматизации и робототехники

Отчёт

по лабораторной работе №2

Дисциплина: Техническое зрение

Тема: Реализация метода Оцу с использованием библиотеки OpenCV

Студент гр. 3331506/70401 Шкабара Я. А.

Преподаватель Варлашин В. В.

« »_____2020 г.

Санкт-Петербург

2020

Оглавление

Математическое описание

Метод Оцу – алгоритм вычисления порога бинаризации для полутонового изображения.

Алгоритм позволяет разделить пиксели на два класса(«полезные» и «фоновые»), рассчитывая такой порог, чтобы внутриклассовая дисперсия была минимальной.

Формулы, необходимые для расчета внутриклассовой дисперсии σ_w^2, представлены на рисунке 1.

$$\sigma_w^2(t) = W_b(t)\sigma_b^2(t) + W_f(t)\sigma_f^2(t)$$

$$W_b(t) = \sum_{i=1}^{t} P(i) \quad \& \quad W_f(t) = \sum_{i=t+1}^{I} P(i)$$

$$\mu_b(t) = \sum_{i=1}^{t} \frac{iP(i)}{W_b(t)} \quad \& \quad \mu_f(t) = \sum_{i=t+1}^{I} \frac{iP(i)}{W_f(t)}$$

$$\sigma_b^2(t) = \sum_{i=1}^{t} [i - \mu_b(t)]^2 \frac{P(i)}{W_b(t)} \quad \& \quad \sigma_f^2(t) = \sum_{i=t+1}^{I} [i - \mu_f(t)]^2 \frac{P(i)}{W_f(t)}$$

Рисунок 1 – Расчет σ_w^2

В таблице 1 представлены обозначения, используемые на рисунке 1. Величины, имеющие индекс f относятся к «полезным» пикселям, а имеющие индекс b - к фоновым пикселям.

Таблица 1 – Обозначения в формулах

Обозначение	Расшифровка
W	Вероятность того, что случайно выбранный пиксель относится к выбранному классу
μ	Среднее значение интенсивности пикселей в классе
σ^2	Дисперсия
t	Порог бинаризации
$P(i)$	Вероятность того, что случайно выбранный пиксель имеет интенсивность i

Реализация алгоритма на C++

Ниже представлен класс *OtsuMethod*, реализующий метод Оцу.

```
class OtsuMethod
{
public:
	OtsuMethod();
	~OtsuMethod();
	int setImage(Mat im);
	Mat getImage();
	int binarization();

private:
	Mat image;
	int threshold = 0;
	std::vector<int> histogram;
	int sizeOfImage;
	double variance;
	int sumOfPixels = 0;

	void createHistogram();
	double withinClassVariance(int th);
};
```

Реализация методов класса:

setImage – передача изображения в алгоритм для обработки:

```
int OtsuMethod::setImage(Mat im)
{
	if (im.empty()) return (-1);

	image = im.clone();
	sizeOfImage = image.rows * image.cols;

	return 0;
}
```

getImage – получение обработанного изображения:

```
Mat OtsuMethod::getImage()
{
	return image;
}
```

binarization – бинаризация изображения:

```
int OtsuMethod::binarization()
{
	createHistogram();
	variance = withinClassVariance(0);
	for (int th = 1; th < 256; th++)
	{
		double newVariance = withinClassVariance(th);
		if (newVariance <= variance)
		{
			variance = newVariance;
			threshold = th;
		}
	}

	for (int i = 0; i < image.rows; i++)
		for (int j = 0; j < image.cols; j++)
			if (image.at<uint8_t>(i, j) > threshold)
image.at<uint8_t>(i, j) = 255;
			else image.at<uint8_t>(i, j) = 0;

	return 0;
}
```

createHistogram – создание гистограммы изображения:

```
void OtsuMethod::createHistogram()
{
	std::vector<int> hist;
	hist.resize(256);
	int pixelSum=0;

	for (int i = 0; i < image.rows; i++)
		for (int j = 0; j < image.cols; j++)
		{
			hist[image.at<uint8_t>(i, j)]++;
			pixelSum += image.at<uint8_t>(i, j);
		}

	histogram = hist;
	sumOfPixels = pixelSum;
	threshold = 0;
}
```

withinClassVariance – определение внутриклассовой дисперсии

```
double OtsuMethod::withinClassVariance(int th)
{
	double weight_bg = 0, weight_fg = 0;
	double average_bg = 0, average_fg = 0;
	double variance_bg = 0, variance_fg = 0, newVariance = 0;
	int pixelsUnderTh = 0, pixelsAboveTh = 0; //Under <=
	int sum = 0;

	for (int i = 0; i < th; i++)
	{
		pixelsUnderTh += histogram[i];
		sum += i * histogram[i];
	}

	weight_bg = static_cast<double>(pixelsUnderTh) / sizeOfImage;
	if (pixelsUnderTh != 0) average_bg = static_cast<double>(sum) /
pixelsUnderTh;

	for (int i = 0; i <= th; i++)
	{
		variance_bg += ((i - average_bg) * (i - average_bg) *
histogram[i]);
	}
	if (pixelsUnderTh != 0) variance_bg = variance_bg / pixelsUnderTh;


	pixelsAboveTh = sizeOfImage - pixelsUnderTh;
	sum = sumOfPixels - sum;
	weight_fg = 1-weight_bg;
	if (pixelsAboveTh != 0) average_fg = static_cast<double>(sum) /
pixelsAboveTh;

	for (int i = (th+1); i < 256; i++)
	{
		variance_fg += ((i - average_fg) * (i - average_fg) *
histogram[i]);
	}
	if (pixelsAboveTh != 0) variance_fg = variance_fg / pixelsAboveTh;

	newVariance = weight_bg * variance_bg + weight_fg * variance_fg;

	return newVariance;
}
```

Сравнение со встроенной функцией OpenCV для метода Оцу

Правильность определения порога бинаризации: для этого бинаризуем изображение с помощью реализованного алгоритма Оцу и с помощью встроенной функции OpenCV. Затем найдем абсолютную разницу между изображениями и сложим разности для каждого пикселя.

```
int main()
{
	Mat image = imread("C:/Users/anana/Pictures/voron.bmp");
	cvtColor(image, image, COLOR_RGB2GRAY);

	auto begin = std::chrono::steady_clock::now();

	OtsuMethod otsu;
	otsu.setImage(image);
	otsu.binarization();
	Mat binarizedImage = otsu.getImage();

	auto end = std::chrono::steady_clock::now();
	auto myOtsu =
std::chrono::duration_cast<std::chrono::milliseconds>(end - begin);

	imshow("binarizedImage", binarizedImage);
	while (waitKey(50) != 'b');

	Mat check, diff;
	check = image.clone();

	begin = std::chrono::steady_clock::now();

	cv::threshold(check, check, 0, 255, THRESH_OTSU);

	end = std::chrono::steady_clock::now();
	auto openCVOtsu =
std::chrono::duration_cast<std::chrono::milliseconds>(end - begin);

	absdiff(binarizedImage, check, diff);
	int error = 0;
	for (int i = 0; i < check.rows; i++)
		for (int j = 0; j < check.cols; j++)
			error += diff.at<uint8_t>(i, j);

	imshow("check", check);
	while (waitKey(50) != 'b');

	return 0;}
```

Для измерения времени работы алгоритма используется библиотека <chrono>.

В результате выполнения куска кода, находящегося выше, получены следующие результаты:

error	0
myOtsu	1014 milliseconds
openCVOtsu	7 milliseconds

Таким образом, реализованный алгоритм работает корректно, но на его выполнение требуется во много раз больше времени.

Пример работы программы: