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: Фильтрация изображения

Студент гр. 3331506/70401 Чернов Е.И.

Преподаватель Варлашин В.В.

« »_____2020 г.

Санкт-Петербург
2020

Задание

Реализовать *Closing process* со структурным элементом размера 3x3 и якорной точкой в центре. Структурный элемент изображен на рисунке 1.

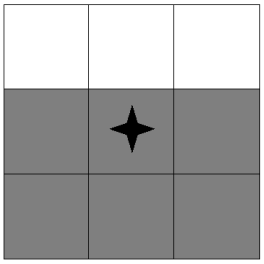

Рисунок 1 – Структурный элемент

Для обработки граничных условий должен быть применен *Border Reflect*.

Ход работы

Для реализации фильтра сначала выполняется операция дилатации и затем операция эрозии. Названные операции семантически применяется к объекту белого цвета. При этом структурный выполняет операции построчно слева на права сверху вниз.

В качества объекта фильтрации возьмем изображение, представленное на рисунке 2.

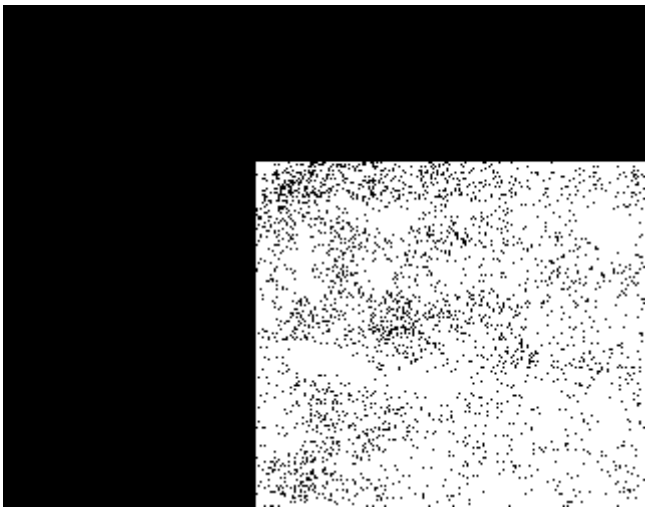

Рисунок 2 – Изображение для фильтрации

Класс *ClosingProcess*

Класс *ClosingProcess* является утилитным и содержит только статические методы и поля. Предоставленная реализация применима только к бинарным изображениям.

Класс содержит следующие открытые поля:

1. `const int OBJ = 255`

Поле содержит цвет обрабатываемого объекта на изображении (белый).

2. `const int BG = 0`

Поле содержит цвет фона изображения (черный).

3. `const int CORE[]`

Поле описывает структурный элемент.

4. `const int BORDER_WIDTH`

Поле задает размер рамки необходимой для морфологических операций.

Основные методы

1. `static int borderReflect(Mat& src, Mat& res)`

Позволяет получить копию исходного изображения с рамкой, созданной по правилу *Border Reflect.*

- Параметр *src* – исходное изображение.
- Параметр *res* – результат выполнения метода (исходное изображение с рамкой).

Исходный код метода представлен и ниже. Приватный метод `expandRow(…)` создает рамку изображения справа и слева. Приватный метод `copyRow(…)` позволяет создать рамку сверху и снизу.

```
int ClosingProcess::borderReflect(Mat& src, Mat& res)
{
	res = Mat(src.rows + (BORDER_WIDTH * 2), src.cols + (BORDER_WIDTH * 2), CV_8UC1);

	for (int row = 0; row < src.rows; row++)
	{
		expandRow(row, src, res);
	}

	for (int shift = 0; shift < BORDER_WIDTH; shift++)
	{
		int filledRow = BORDER_WIDTH + shift;
		int cleanRow = BORDER_WIDTH - 1 - shift;
		copyRow(filledRow, cleanRow, res);
	}

	for (int shift = 0; shift < BORDER_WIDTH; shift++)
	{
		int filledRow = res.rows - BORDER_WIDTH - 1 - shift;
		int cleanRow = res.rows - BORDER_WIDTH + shift;
		copyRow(filledRow, cleanRow, res);
	}
	return 0;
}
```

Для более показательного результата выполнения используем изображение, представленное на рисунке 3.

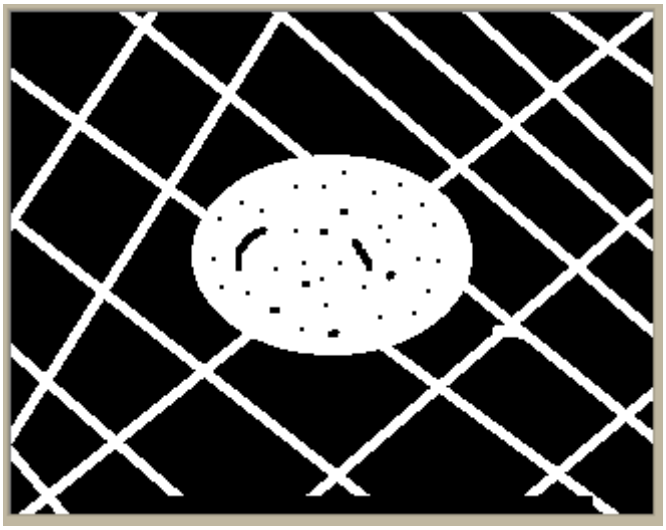

Рисунок 3 – Исходное изображение для примера работы `borderReflect(…)`

Результатом выполнения данного метода при параметре `BORDER_WIDTH` равным 20 будет служить изображение, представленное на рисунке 4.

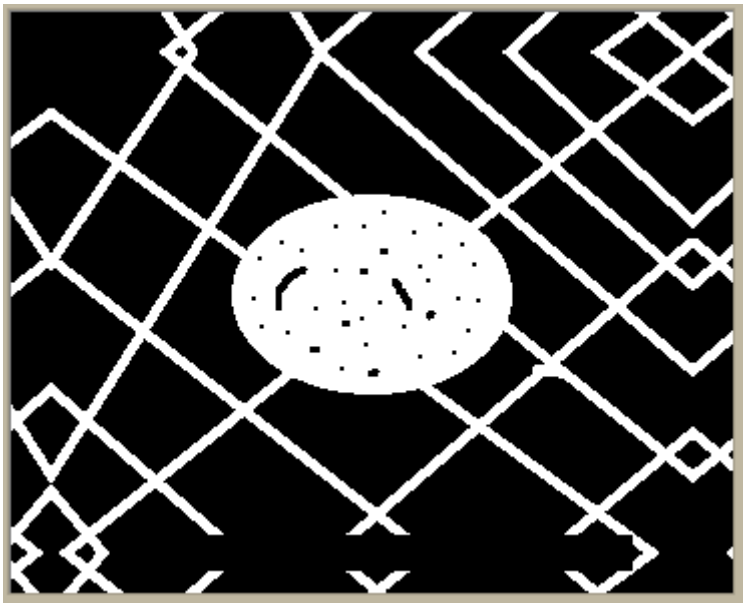

Рисунок 4 – Результат выполнения `borderReflect(…)`

2. `int erode(Mat& src, Mat& res)` и `int prErode(Mat& src, Mat& res)`

Позволяет произвести операцию эрозии исходного изображения,

$$\mathrm{res}(x, y) = \min(x', y'): \mathrm{element}(x', y') \neq 0\ \mathrm{src}(x+x', y+y')$$

- Параметр *src* – исходное изображение.
- Параметр *res* – результат выполнения метода (результат выполнения эрозии над исходным изображением).

Исходный код методов представлен и ниже.

```
int ClosingProcess::erode(Mat& src, Mat& res)
{
	Mat temp;
	src.copyTo(temp);
	borderReflect(src, temp);
	prErode(temp, res);
	return 0;
}
```

```
int ClosingProcess::prErode(Mat& src, Mat& res)
{
	res = Mat(src.rows - BORDER_WIDTH * 2, src.cols - BORDER_WIDTH * 2, CV_8UC1);

	for (int row = BORDER_WIDTH; row < src.rows - BORDER_WIDTH; row++)
	{
		for (int col = BORDER_WIDTH; col < src.cols - BORDER_WIDTH; col++)
		{
			int resRow = row - BORDER_WIDTH;
			int resCol = col - BORDER_WIDTH;
			int iCore = 0;
			bool flag = false;
			for (int i = row - 1; i <= row + 1; i++)
			{
				for (int j = col - 1; j <= col + 1; j++)
				{
					if (CORE[iCore] == OBJ && src.at<uchar>(i, j) != OBJ)
					{
						res.at<uchar>(resRow, resCol) = BG;
						flag = true;
						break;
					}
					iCore++;
				}
				if (flag == true) break;
			}

			if (flag == false)
			{
				res.at<uchar>(resRow, resCol) = OBJ;
			}
		}
	}

	return 0;
}
```

Результатом выполнения `erode(…)` будет служить изображение, представленное на рисунке 5.

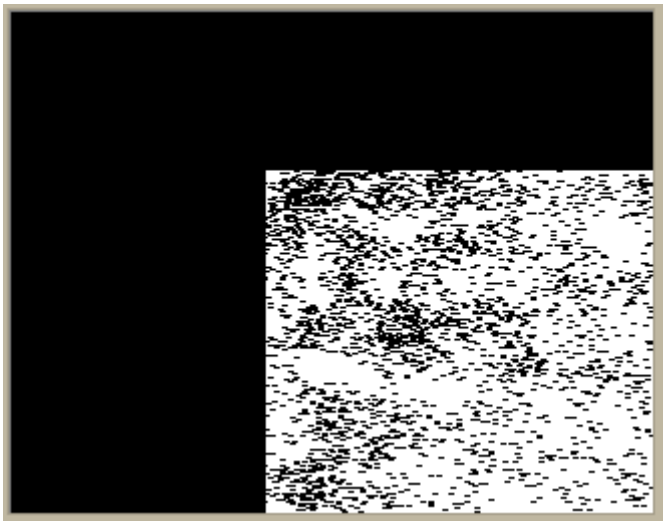

Рисунок 5 – Результат выполнения `erode(…)`

3. `int dilate(Mat& src, Mat& res)` и `prDilate(Mat& src, Mat& res)`

Позволяет произвести операцию дилатации исходного изображения,

$$\mathrm{res}(x, y) = \max(x', y'): \mathrm{element}(x', y') \neq 0\ \mathrm{src}(x+x', y+y')$$

- Параметр *src* – исходное изображение.
- Параметр *res* – результат выполнения метода (результат выполнения дилатации над исходным изображением).

Исходный код методов представлен и ниже.

```
int ClosingProcess::dilate(Mat& src, Mat& res)
{
	Mat temp;
	src.copyTo(temp);
	borderReflect(src, temp);
	prDilate(temp, res);
	return 0;
}


int ClosingProcess::prDilate(Mat& src, Mat& res)
{
	res = Mat(src.rows - BORDER_WIDTH * 2, src.cols - BORDER_WIDTH * 2, CV_8UC1);

	for (int row = BORDER_WIDTH; row < src.rows - BORDER_WIDTH; row++)
	{
		for (int col = BORDER_WIDTH; col < src.cols - BORDER_WIDTH; col++)
		{
			int resRow = row - BORDER_WIDTH;
			int resCol = col - BORDER_WIDTH;
			int iCore = 0;
			bool flag = false;
			for (int i = row - 1; i <= row + 1; i++)
			{
				for (int j = col - 1; j <= col + 1; j++)
				{
					if (CORE[iCore] == OBJ && src.at<uchar>(i, j) == OBJ)
					{
						res.at<uchar>(resRow, resCol) = OBJ;
						flag = true;
						break;
					}
					iCore++;
				}
				if (flag == true) break;
			}
			if (flag == false)
			{
				res.at<uchar>(resRow, resCol) = BG;
			}
		}
	}
	return 0;
}
```

Результатом выполнения `dilate(…)` будет служить изображение, представленное на рисунке 6.

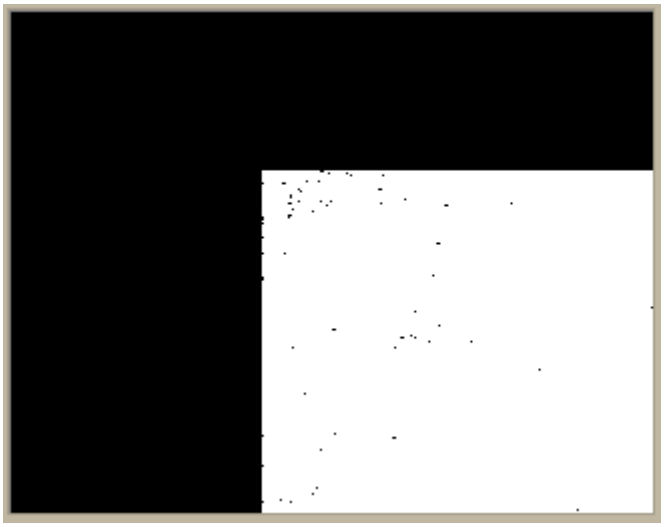

Рисунок 6 – Результат выполнения `dilate(…)`

4. `int useClosingFilter(Mat& src, Mat& res)`

Позволяет произвести фильтрацию *Closing process* исходного изображения.

- Параметр *src* – исходное изображение.
- Параметр *res* – результат выполнения метода (результат выполнения фильтрации *Closing process* над исходным изображением).

Исходный код метода представлен и ниже.

```
int ClosingProcess::useClosingFilter(Mat& src, Mat& res)
{
	Mat temp;
	dilate(src, temp);
	erode(temp, res);
	return 0;
}
```

Результатом однократного выполнения `useClosingFilter(…)` будет служить изображение, представленное на рисунке 7.

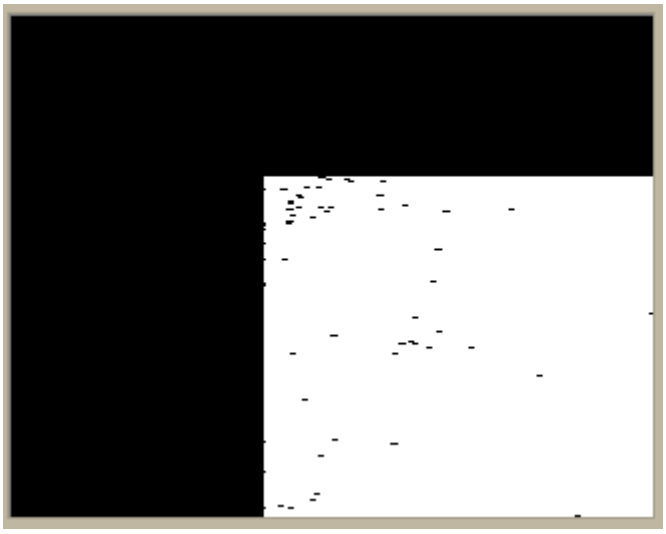

Рисунок 7 – Результат однократного выполнения `useClosingFilter(…)`

Результатом тридцатикратного выполнения `useClosingFilter(…)` будет служить изображение, представленное на рисунке 8.

Рисунок 8 – Результат тридцатикратного выполнения `useClosingFilter(…)`

5. `int testOcvFilter(Mat& src, Mat& dst)`

Метод осуществляет сравнение результатов выполнения фильтрации *Closing process* реализаций *OpenCV* и представленной в данной работе. Также осуществляет логирование времени выполнения и фактической разницы изображений обоих реализаций в стандартный поток вывода.

- Параметр *src* – исходное изображение.

- Параметр *dst* – результат выполнения фильтрации *Closing process* реализации *OpenCV*.

Исходный код метода представлен и ниже. Закрытый метод `compareImg(…)` сравнивает поданные на него изображения попиксельно.

```
int ClosingProcess::testOcvFilter(Mat& src, Mat& dst)
{
	// My
	Mat temp;
	int startMyTime = clock();
	useClosingFilter(src, temp);
	int endMyTime = clock();
	int resMyTime = endMyTime - startMyTime;

	// OpenCV
	uint8_t data[9] = { 255, 255, 255, 0, 0, 0, 0, 0, 0 };
	Mat kernel(3, 3, CV_8UC1, data);
	Mat ocvTemp;
	int startOcvTime = clock();
	morphologyEx(src, dst, MORPH_CLOSE, kernel, Point(1, 1), 1, BORDER_REFLECT);
	int endOcvTime = clock();
	int resOcvTime = endOcvTime - startOcvTime;

	compareImg(temp, dst);

	int res = resOcvTime - resMyTime;
	cout << "My time " << resMyTime << endl;
	cout << "OpenCV time " << resOcvTime << endl;
	cout << "DIFFERENT of time " << res << endl;
	return res;
}
```

Результатом выполнения `testOcvFilter(…)` в режиме Debug для предоставленного в данной работе изображения будет служить следующий текст:

```
Count of errors 0
My time 375
OpenCV time 4
DIFFERENT of time -371
```

Вывод

В ходе роботы была успешна реализована фильтрация *Closing Process* для соответствующего технического задания.

Также было произведено сравнение с реализацией *OpenCV*. При полном соответствии итоговых изображений были получены следующие средние значения времени выполнения фильтрации в режиме *Release*:

- Для приведенной реализации 8,1 мс
- Для реализации OpenCV 0,4 мс

Видно явное отставание приведенной в данной работе реализации, из чего следует вывод, что необходима оптимизация по времени.