ЦЕЛЬ

Ознакомиться с программированием периферийных устройств на примере ввода данных с Web-камеры с использованием библиотеки OpenCV.

ОПИСАНИЕ РАБОТЫ

Вывод видеопотока с камеры на экран

Подключим библиотеку `#include <opencv2/videoio.hpp>`, чтобы использовать `VideoCapture capture(0)` для создания потока ввода видеоданных с камеры номер 0.

Теперь необходимо считать кадр из видеопотока. Для этого используется класс Mat, который хранит в себе данные изображения (по сути, представляет собой массив, который используется для хранения данных изображений в оттенках серого или цветных изображений, векторных полей и т. д.). С помощью метода .read можем считать нужный нам кадр в класс Mat, а с помощью imshow вывести на экран.

```
while (true) {
    Mat frame;
    capture.read(frame); //считываем кадр с видеопотока

    imshow("monkey", frame); //выводим этот кадр в отдельное окно
    }
}
```

Для остановки программы используем кнопку Esc, номер которой равен 27. Если для каждого кадра будем ожидать нажатие этой кнопки в течение 33 мс, то получим частоту показа 1 кадр в 33 мс, то есть 30 кадров в секунду.

В итоге получаем такую программу, которая выводит в отдельное окно видео.

```
#include <opencv2/videoio.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/core.hpp>

using namespace std;
using namespace cv;

int main() {

    VideoCapture capture(0);

    if (!capture.isOpened()) return 0;

    while (true) {
        Mat frame;
        capture.read(frame);

        imshow("monkey", frame);

        char c = waitKey(33);
        if (c == 27) {
            break;
        }
```

```
    }

    capture.release();
    destroyWindow("monkey");

    return 0;

}
```

Преобразование изображения

Теперь стоит задача научиться преобразовывать изображение.

```
Попробуем сделать изображение более контрастным за счет
frame.convertTo(frame, -1, 4, 0);
```

Где первое значение представляет собой изображение, в которое выведем измененное изображение, второе тип выводимого изображения (если отрицательное, то тип такой же), третье произведение всех пикселей указанное количество раз, четвертое величина, на которую увеличится каждый пиксель.

Также уменьшим яркость изображения frame.convertTo(frame, -1, 1, -50).

Инвертируем цвета в получившемся изображении

```
for (int y = 0; y < frame.cols; y++) {
    for (int x = 0; x < frame.rows; x++) {
        for (int channel = 0; channel < 2; channel++) {
            uchar color = frame.at<Vec3b>(x, y)[channel];
            frame.at<Vec3b>(x, y)[channel] = (uchar)255 - 2*color;
        }
    }
}
```

В итоге имеем

```
#include <opencv2/videoio.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/core.hpp>

using namespace std;
using namespace cv;

int main() {

    VideoCapture capture(0);

    while (true) {
        Mat frame;
        capture.read(frame); //считываем кадр с видеопотока

        for (int y = 0; y < frame.cols; y++) {
            for (int x = 0; x < frame.rows; x++) {
```

```
                for (int channel = 0; channel < 2; channel++) {
                    uchar color = frame.at<Vec3b>(x, y)[channel];
                    frame.at<Vec3b>(x, y)[channel] = (uchar)255 - 2*color;
                }
            }
        }

        frame.convertTo(frame, -1, 4, 0);
        frame.convertTo(frame, -1, 1, -50);

        imshow("monkey", frame); //выводим этот кадр в отдельное окно

        char c = waitKey(33);
        if (c == 27) {
            break;
        }
    }

    capture.release();
    destroyWindow("monkey");

    return 0;

}
```

Измерение времени

Количество кадров в секунду

1) Для начала узнаем, сколько кадров в секунду обрабатывает наша программа. Для этого заведем счетчик кадров `frames_counter` и измерим `total_time` с помощью суммирования времени всех итераций.

```
int frames_counter = 0;

    while (true) {
        frames_counter++;
        auto iterStart = chrono::system_clock::now();

        /*.....обработка кадра.....*/
        auto iterEnd = chrono::system_clock::now();

        totalTime += chrono::duration_cast<std::chrono::milliseconds>(iterEnd
- iterStart).count();
    }

    double fps = (double)frames * 1000 / totalTime;
    cout << "FPS: " << fps << endl;
```

2) Для подсчета времени, затрачиваемого процессором на ввод, обработку и вывод видеоданных, заведем переменные start, `input`, `convertion`, `output`, `waiting` и `end`, где `start` считывает время начала итерации, `input` конец считывания кадра,

`convertion` конец преобразование кадра, `output` конец отправки изображения в отображаемое окно, `waiting` конец показа отображения, `end` время конца итерации.

Далее, суммируя разности всех этих параметров, получим полное количество времени, затрачиваемого на каждую операцию.

Листинг программы

```
#include <opencv2/videoio.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/core.hpp>
#include <iostream>
#include <chrono>

using namespace std;
using namespace cv;

int main() {

    VideoCapture capture(0);

    int frames_counter = 0;
    int total_input = 0;
    int total_convertion = 0;
    int total_output = 0;
    int total_time = 0;
    int total_waiting = 0;

    while (true) {
        frames_counter++;
        auto start = chrono::system_clock::now();

        Mat frame;
        capture.read(frame);

        auto input = chrono::system_clock::now();
        for (int y = 0; y < frame.cols; y++) {
            for (int x = 0; x < frame.rows; x++) {
                for (int channel = 0; channel < 2; channel++) {
                    uchar color = frame.at<Vec3b>(x, y)[channel];
                    frame.at<Vec3b>(x, y)[channel] = (uchar)255 - 2*color;
                }
            }
        }

        frame.convertTo(frame, -1, 4, 0);
        frame.convertTo(frame, -1, 1, -50);

        auto convertion = chrono::system_clock::now();

        imshow("monkey", frame);

        auto output = chrono::system_clock::now();

        char c = waitKey(1);

        auto waiting = chrono::system_clock::now(); //конец вейт кея
```

```
        if (c == 27) {
            break;
        }

        auto end = chrono::system_clock::now();

        total_input += chrono::duration_cast<std::chrono::milliseconds>(input
- start).count();
        total_convertion +=
chrono::duration_cast<std::chrono::milliseconds>(convertion - input).count();
        total_output +=
chrono::duration_cast<std::chrono::milliseconds>(output -
convertion).count();
        total_waiting +=
chrono::duration_cast<std::chrono::milliseconds>(waiting - output).count();
        total_time += chrono::duration_cast<std::chrono::milliseconds>(end -
start).count();
    }

    capture.release();
    destroyWindow("monkey");

    double fps = (double)frames_counter * 1000 / total_time;
    double inp = (double)total_input * 100 / total_time;
    double cnv = (double)total_convertion * 100 / total_time;
    double out = (double)total_output * 100 / total_time;
    double wt = (double)total_waiting * 100 / total_time;

    cout << "total time: " << total_time << endl;
    cout << "FPS: " << fps << endl;
    cout << "input: " << total_input << endl;
    cout << "convertion: " << total_convertion << endl;
    cout << "output: " << total_output << endl;
    cout << "waiting: " << total_waiting << endl;
    cout << "time for input: " << inp << "%" << endl;
    cout << "time for converting: " << cnv << "%" << endl;
    cout << "time for output: " << out << "%" << endl;
    cout << "time for waiting: " << wt << "%" << endl;
    return 0;

}
```

Проанализируем вывод программы. Попробуем запускать ее примерно на десять секунд.

total time: 12134 FPS: 13.3509 input: 501 convertion: 5947 output: 51 waiting: 5390 time for input: 4.12889% time for converting: 49.011% time for output: 0.420307% time for waiting: 44.4206%	total time: 10558 FPS: 13.4495 input: 504 convertion: 5083 output: 52 waiting: 4714 time for input: 4.77363% time for converting: 48.1436% time for output: 0.492518% time for waiting: 44.6486%
total time: 10288 FPS: 13.0249 input: 503 convertion: 5083 output: 46 waiting: 4455 time for input: 4.88919% time for converting: 49.4071% time for output: 0.447123% time for waiting: 43.3029%	

Во время первых двух измерений объекты перед камерой двигались, во время второго же окружающее камеру было статичным. Как можем заметить, это особо не повлияло на полученные нами данные.

Однако возникает проблема с тем, что мы ожидали 30 FPS, когда получили 13-13,5.

Чтобы решить это, попробуем изменить значение waitKey(), и вместо 33 поставим 10.

```
total time: 13018
FPS: 20.3564
input: 503
convertion: 9343
output: 34
waiting: 2754
time for input: 3.86388%
time for converting: 71.7699%
time for output: 0.261177%
time for waiting: 21.1553%
```

Отображаемая картинка стала более плавной, однако FPS все еще низкое.

Видим, что много времени уходит и на преобразование изображения. Попробуем убрать, например, два вызова метода convertTo и оставим только инвертирование исходного изображения.

```
total time: 10825
FPS: 29.6536
input: 3776
convertion: 3144
output: 57
waiting: 3343
time for input: 34.8822%
time for converting: 29.0439%
time for output: 0.526559%
time for waiting: 30.8822%
```

Значение FPS наконец приблизилось к ожидаемому результату.

Например, если убрать преобразование изображения совсем и вернуть waitKey значение 33, увидим

```
total time: 10589
FPS: 27.1036
input: 560
convertion: 0
output: 146
waiting: 9574
time for input: 5.28851%
time for converting: 0%
time for output: 1.37879%
time for waiting: 90.4146%
```

Что так же приближает значение FPS к ожидаемому.

Могу сделать вывод, что на FPS очень влияет время преобразования изображение, а также время показа этого преобразованного изображения.

По результатам измерений видим, что большая часть времени уходит на преобразование изображения и на его показ. Так же вывод его на эркан не занимает почти ничего, когда получение самого изображения тоже почти не имеет никакого веса, кроме случая, когда кадр почти не преобразовывается и значение waitKey тоже маленькое.

Если попробуем настроить программу так, чтобы она обрабатывала одинаковое количество кадров за раз, получим такой результат.

```
total time: 7596
FPS: 13.1648
input: 501
convertion: 3566
output: 38
waiting: 3346
time for input: 6.59558%
time for converting: 46.9458%
time for output: 0.500263%
time for waiting: 44.0495%
```

```
total time: 7625
FPS: 13.1148
input: 503
convertion: 3583
output: 46
waiting: 3348
time for input: 6.59672%
time for converting: 46.9902%
time for output: 0.603279%
time for waiting: 43.9082%
```

total time: 7944 FPS: 12.5881 input: 501 convertion: 3903 output: 35 waiting: 3345 time for input: 6.30665% time for converting: 49.1314% time for output: 0.440584% time for waiting: 42.1073%	total time: 7902 FPS: 12.655 input: 516 convertion: 3841 output: 32 waiting: 3350 time for input: 6.52999% time for converting: 48.6079% time for output: 0.404961% time for waiting: 42.3943%

Пусть процентное соотношение почти не меняется, изменение времени на обработку заметно меняется.

ЗАКЛЮЧЕНИЕ

С помощью библиотеки OpenCV мне удалось считать в Web-камеры моего ноутбука изображение, преобразовать его и вывести в отдельное окно.

Также я смогла оценить долю времени, затрачиваемого процессором на каждом шаге программы, и узнала какие операции при работе с периферийными устройствами могут занять много времени, а какие выполняются почти мгновенно.

Приложение 1. Листинг программы

```
#include <opencv2/videoio.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/core.hpp>
#include <iostream>
#include <chrono>

using namespace std;
using namespace cv;

int main() {

    VideoCapture capture(0);

    int frames_counter = 0;
    int total_input = 0;
    int total_convertion = 0;
    int total_output = 0;
    int total_time = 0;
    int total_waiting = 0;

    while (true) {
        frames_counter++;
        auto start = chrono::system_clock::now();

        Mat frame;
        capture.read(frame);

        auto input = chrono::system_clock::now();

        for (int y = 0; y < frame.cols; y++) {
            for (int x = 0; x < frame.rows; x++) {
                for (int channel = 0; channel < 2; channel++) {
                    uchar color = frame.at<Vec3b>(x, y)[channel];
                    frame.at<Vec3b>(x, y)[channel] = (uchar)255 - 2*color;
                }
            }
        }

        frame.convertTo(frame, -1, 4, 0);
        frame.convertTo(frame, -1, 1, -50);

        auto convertion = chrono::system_clock::now();

        imshow("monkey", frame);

        auto output = chrono::system_clock::now();

        char c = waitKey(1);

        auto waiting = chrono::system_clock::now();

        if (c == 27) {
            break;
        }

        auto end = chrono::system_clock::now();

        total_input += chrono::duration_cast<std::chrono::milliseconds>(input
- start).count();
        total_convertion +=
```

```
chrono::duration_cast<std::chrono::milliseconds>(convertion - input).count();
        total_output +=
chrono::duration_cast<std::chrono::milliseconds>(output -
convertion).count();
        total_waiting +=
chrono::duration_cast<std::chrono::milliseconds>(waiting - output).count();
        total_time += chrono::duration_cast<std::chrono::milliseconds>(end -
start).count();
    }

    capture.release();
    destroyWindow("monkey");

    double fps = (double)frames_counter * 1000 / total_time;
    double inp = (double)total_input * 100 / total_time;
    double cnv = (double)total_convertion * 100 / total_time;
    double out = (double)total_output * 100 / total_time;
    double wt = (double)total_waiting * 100 / total_time;

    cout << "total time: " << total_time << endl;
    cout << "FPS: " << fps << endl;
    cout << "input: " << total_input << endl;
    cout << "convertion: " << total_convertion << endl;
    cout << "output: " << total_output << endl;
    cout << "waiting: " << total_waiting << endl;
    cout << "time for input: " << inp << "%" << endl;
    cout << "time for converting: " << cnv << "%" << endl;
    cout << "time for output: " << out << "%" << endl;
    cout << "time for waiting: " << wt << "%" << endl;
    return 0;

}
```