**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;

1. 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;  
  
}

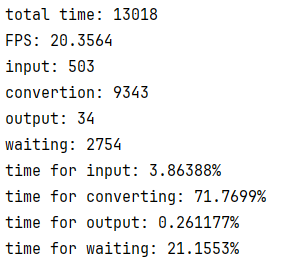
Проанализируем вывод программы. Попробуем запускать ее примерно на десять секунд.

|  |  |
| --- | --- |
|  |  |
|  |  |

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.

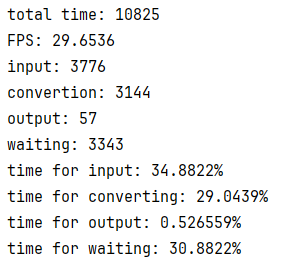
Однако возникает проблема с тем, что мы ожидали 30 FPS, когда получили 13-13,5.

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



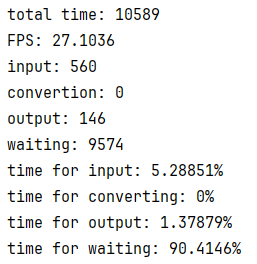
Отображаемая картинка стала более плавной, однако FPS все еще низкое.

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



Значение FPS наконец приблизилось к ожидаемому результату.

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



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.

|  |  |
| --- | --- |
|  |  |
|  |  |

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;  
  
}