Laboratorium 11

Wykrywanie krawędzi

Jeżeli rzucicie okiem na projekt 2 zobaczycie, że brakuje nam już tylko wiedzy o tym jak wykonać jeden najważniejszy jego podpunkt (góra dwa). W związku z tym, abyście nie musieli przy wigilijnym stole myśleć za dużo o programowaniu i Waszych wspaniałych prowadzących, ci niesamowici ludzie spróbują Wam teraz trochę pomóc. Szkoda tylko, że ta pomoc to nie zrobienie za Was projektu, a kolejna lista na laboratorium…

Na początku pracy z tą listą pozbawimy Was jednak troszeczkę wolnej woli. Będziecie pracować na obrazku pochodzącym od prowadzącego. Możecie go chapnąć tutaj. Potem oczywiście będziecie mogli używać Waszych obrazków satanistycznych kotków (;-P), Emmy Watson (^^), wymęczonych owieczek (:-D), czy co tam jeszcze znajdziecie w otchłani Internetu.

Zacznijmy standardowo…

```
Mat before = imread("coins.jpg");
Mat after;
```

Jeżeli ktoś mnie zapyta co to robi, to nie ręczę za siebie…

Teraz użyjemy najważniejszej funkcji z punktu widzenia przeznaczenia tej listy. Funkcja *Canny* jest jedną z najbardziej znanych metod służących wykrywaniu krawędzi na obrazie. Nie mamy jednak ich współrzędnych, a jedynie widok (no chyba, że ręcznie obrobimy zmienną *after* – powodzenia).

```
Canny(before, after, 0, 50, 5);
```

Na dzisiejszych zajęciach, raz na jakiś czas, będziemy robili sobie pit-stopy w kodzie aby zobaczyć etapami, co już udało nam się zepsuć… poprawić w naszym rysunku.

```
imshow("rysun", after);
waitKey(0);
```

Zaiste, w chwili tej widzimy, że funkcja *Canny* działa. Shocking… Teraz udowodnimy sobie jak cenną wiedzę posiedliśmy na wcześniejszych zajęciach. Wszak jak „Słowacki wielkim poetą był”, tak „programowanie w C++ ważnym kursem jest”. Have you met *dilate* function?

```
int change_size = 1;
Mat change = getStructuringElement(MORPH_CROSS,
    Size(2*change_size + 1, 2*change_size + 1),
    Point(change_size, change_size));
dilate(after, after, change, Point(-1, -1), 2, 1, 1);
```

Czemu 1? Tak wyszło… To jednak pokazuje nam jak istotnym elementem obróbki obrazu jest dobór odpowiednich parametrów w trakcie korzystania z odpowiednich funkcji (użycie dwa razy wyrazu „odpowiednich" było celowe). Sprawdźmy jeszcze raz czy obraz nie zrobił się przypadkiem różowy.

```
imshow("rysun", after);
waitKey(0);
```

Jednak nie. To dobrze. To co teraz zrobiliśmy to oczywiście nic innego jak rozszerzenie wszystkich znalezionych wcześniej krawędzi, co w efekcie dało nam białą plamę w miejscu każdej monety. Dzięki temu pozostawiliśmy na obrazie jedynie krawędzie związane z samymi brzegami monet, czyli to o co nam chodziło. Serio. Tak planowaliśmy.

Teraz zróbmy coś co sprawi, że poczujemy prawdziwy powiew świeżości porównywalny z bieganiem w samej szlafmycy… po łące… o wiosennym poranku.

```
vector<vector<Point>> contours;
findContours(after.clone(), contours, CV_RETR_TREE,
    CV_CHAIN_APPROX_SIMPLE, Point(0, 0));
```

W ten sposób uzyskaliśmy zmienną *contours*, która przechowuje punkty położone na konturach wykrytych na obrazie jaki do tej pory udało nam się uzyskać (*after*). Ciekawymi elementami funkcji *findContours* są stałe (np. *CV_RETR_TREE*) mówiące o metodzie wykrywania „punktów na konturach" obrazu **binarnego**. O tych stałych możecie poczytać sobie tutaj (powinniście).

Ostatnim etapem naszej pracy jest narysowanie konturów znalezionych funkcją *findContours*. Początkujący użytkownik OpenCV zastanawiałby się co za chory kod musiałby teraz napisać aby narysować wszystkie te punkty w odpowiednich miejscach na obrazie, w odpowiednim kolorze… A w jaki sposób obliczyć pole powierzchni wytyczonych przez te kontury? O rany… Parafrazując jednak znaną grupę kabaretową: „Ale nieeeee, ale nieeeee…". OpenCV dostarcza nam już bowiem odpowiednie narzędzia do wykonania tych zadań.

O tym co się dzieje w kodzie poniżej… sobie poczytajcie w necie… Pytanie: Po co zastosowaliśmy warunek *if*?

```
Mat drawing = before.clone();
for (int i = 0; i < contours.size(); i++)
{
    if(contours[i].size() < 200 && contours[i].size() > 50)
    {
        drawContours(drawing, contours, i, Scalar(0, 0, 250), 2, 8);
        cout<<contourArea(contours[i])<<endl;
    }
}
```

Po wszystkim nie zapomnijcie o naszym pit-stopie.

```
imshow("rysun", drawing);
waitKey(0);
return 0;
```

Zadania do domu (tak, na święta, buahahahaha…):

- Dodajcie suwak do obrazu i spróbujcie sprawić aby dopasowanie konturów było wykonywane na bieżąco, po zmianie suwakiem jakiegoś parametru przetwarzania obrazu (np. zmiennej *change_size* w funkcji *dilate*).

Hint:

Do wykonania projektu może, ale nie musi, przydać Wam się funkcja przetwarzająca obraz na obraz w skali szarości.

```
Mat gray;
cvtColor(img, gray, CV_BGR2GRAY);
```

Wesołych świąt i szczęśliwego nowego roku !!!

(postarajcie się wrócić na zajęcia po sylwestrze… w skończonej liczbie kawałków)