

# **Sprawozdanie z laboratorium: Komunikacja człowiek - komputer**

**Sprawozdanie I: Przetwarzanie obrazu – aplikacja**

14 listopada 2017

**Prowadzący:** mgr inż. Paweł Liskowski

**Autorzy:** Mateusz Urbaniak 127345  
Kajetan Zimniak 127229

Zajęcia wtorkowe, 8:00.

# 1 Wstęp

Celem projektu było znalezienie i zaimplementowanie najlepszej metody wykrywania i rozpoznawania cyfr drukowanych ze zdjęcia. Wśród wielu możliwych dróg stworzenia programu, na bieżąco wybieraliśmy najlepsze, naszym zdaniem, funkcje i algorytmy, aby osiągnąć pożądany efekt. Podział prac wynikał z kolejnych etapów projektu: implementacja algorytmu szukającego krawędzi obiektów, segmentacja obrazu, dopasowanie do wzorca, testowanie i modyfikacja programu.

## 2 Opis algorytmu i wykorzystanych filtrów, funkcji i metod

W pierwszej kolejności za pomocą funkcji *io.ImageCollection()* ładujemy obrazy do programu, które następnie, w kolejności, poddawane są przekształceniu na odcienie szarości (*color.rgb2grey()*), w celu uzyskania dwuwymiarowej tablicy, na której dalej pracujemy. Funkcje *morphology.closing()* i *morphology.opening()* służą nam do usunięcia podstawowych zanieczyszczeń (noise) obrazu wejściowego. *Np.percentile()* i *exposure.rescale\_intensity()* służą do zmiany intensywności. W tym momencie jesteśmy gotowi do użycia *measure.find\_contours()* w celu znalezienia konturów, a dokładniej ich współrzędnych. Nie wystarczy to jednak do zakończenia pierwszej części programu. Na przykład dla cyfry „6”, funkcja wykryje nam dwa

kontury, ponieważ znak składa się z dwóch kształtów. Przy użyciu minimalnych i maksymalnych współrzędnych  $x$ ,  $y$ , usuwamy wewnętrzne części znaków, które nie pomogą nam w dalszej detekcji. W tym momencie jesteśmy gotowi do segmentacji, polegającej na wycięciu odpowiednich części macierzy obrazu. Jest to operacja nieskomplikowana, jednak potrzebna do użycia funkcji *match\_template()*. Zwraca ona liczbę przedstawiającą dokładność pokrycia, znalezienia szukanego elementu na obrazie. W naszym przypadku konieczne było początkowe wycięcie każdego znaku, aby zachować kolejność odczytywanych cyfr, ponieważ każdy znak sprawdzany jest w pętli, z każdym przygotowanym szablonem cyfr od 0 do 9.

W trakcie eksperymentów zdecydowaliśmy się dodać kolejną funkcjonalność. Za pomocą funkcji *draw.polygon\_perimeter()* i *cv2.putText()* zaznaczana jest wykryta cyfra i wypisywana odczytana przez program wartość. Całość zapisywana jest w pdfie. Sprawia to, iż wynik programu przedstawiany jest w bardzo prosty i przejrzysty sposób w jednym miejscu, mimo wielu obrazków wejściowych.

### 3 Eksperymenty

Program przystosowany jest do rozpoznania każdego rodzaju czcionki. W eksperymentach skupiliśmy się na czcionce Arial, po to, by dokładniej zbadać wpływ rozmiaru, koloru i tła zdjęcia.

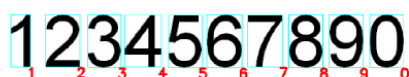
Na początku pracowaliśmy nad tym, aby program ze stu-procentową skutecznością rozpoznawał cyfry ze zrzutu ekranu (*rys.1 i 2*), na białym tle. Problemem okazały się różnice w rozmiarze zdjęcia i szablonu. Łatwo je jednak wyeliminowaliśmy skalując obrazy.

*Rysunek 1:*



1234567890

*Rysunek 2:*



1234567890

Po osiągnięciu sukcesu z powyżej przytoczoną sytuacją, zajęliśmy się badaniem przykładów nieco bardziej złożonych, mianowicie mieszanki tekstu i cyfr (na przykład rozpoznawanie peselu wśród innych danych *Rys. 3, 4*)

*Rysunek 3, obraz wejściowy:*



Imię i Nazwisko: **Kajetan Zimniak**

Pesel: **96030207656**

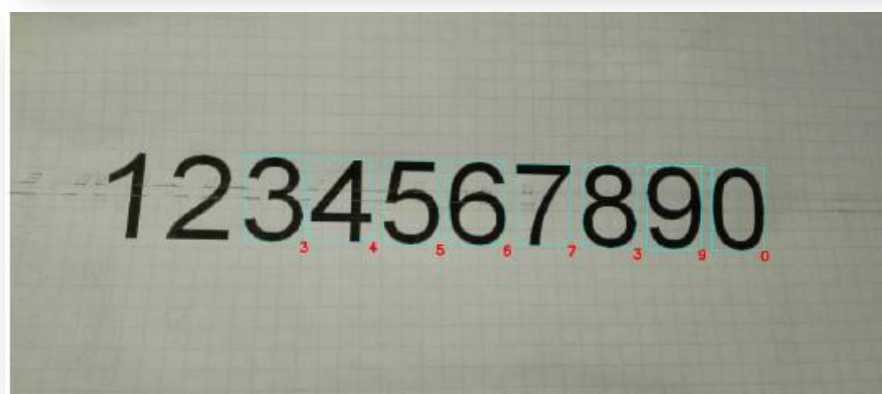
Rysunek 4 obraz wyjściowy:

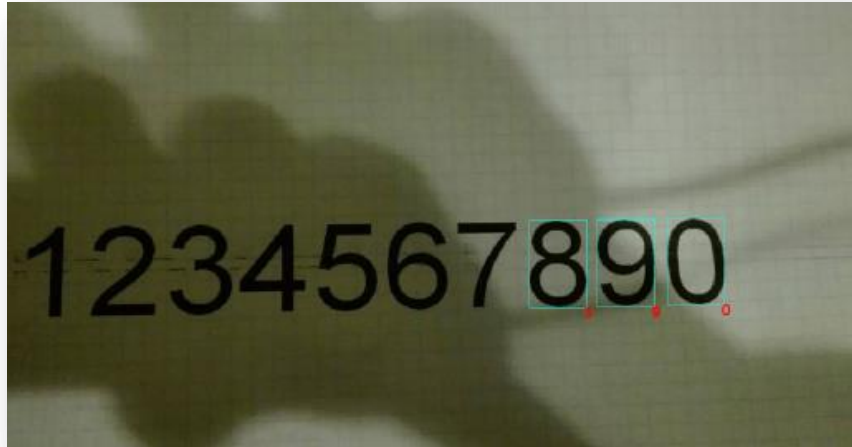
Kajetan Zimniak

Pesel: 960302007655

Następnym etapem naszych testów było sprawdzanie skuteczności na zdjęciach zrobionych telefonem. Celowo używaliśmy papieru z niejednorodnym tłem, oraz oświetlenia zakłócającego czytelność. Poniżej kilka przykładów. Można zaobserwować, że w większości sytuacji program działał bez zarzutu. Problemy pojawiały się tylko w miejscach bardzo zacienionych. Co było do przewidzenia w związku z niedoskonałością metod, jakie mogliśmy używać na tym poziomie.

Rysunek 5, 6, 7:





## 4 Podsumowanie

Po wielu eksperymentach i wyborze odpowiednich metod, udało nam się zrealizować założony cel. Stworzyliśmy program wykrywający i rozpoznawający cyfry, nie tylko na zrzucie ekranu, ale również te zrobione aparatem w zwykłym telefonie. Jest to program, który już na tym poziomie zaawansowania może przydać się w wielu miejscach i z łatwością można przerobić go wg potrzeb określonego celu. W ramach laboratorium wykonaliśmy szablony tylko jednej czcionki, jednak mając na uwadze potrzeby praktyczne, program dostosowaliśmy do łatwego dodania każdej innej, nie zajmie to więcej niż kilkadziesiąt sekund na każdą z rodzaju czcionki.

Ważnym wnioskiem z projektu może być to, iż przetwarzanie obrazu i wyciąganie z niego konkretnego, potrzebnego elementu, a tym bardziej rozpoznawanie znaków w sposób bezbłędny (w przypadku nr pesel każda pomyłka równa się z bezsensownością

programu) nie jest łatwe i wymaga dużej wiedzy i wysiłki. Należy mieć zawsze na uwadze takie elementy jak światło, wielkość, ostrość, różnicę barw.

## **Źródło**

[1] <http://scikit-image.org>

[2] <http://matplotlib.org>

[3] [docs.opencv.org](http://docs.opencv.org)