PRZETWARZANIE OBRAZÓW Z WYKORZYSTANIEM URZĄDZEŃ MOBILNYCH NA PRZYKŁADZIE SYSTEMU ROZPOZNAWANIA TABLIC REJESTRACYJNYCH

IMAGE PROCESSING WITH THE USE OF MOBILE DEVICES ON THE BASIS OF CAR LICENSE PLATES RECOGNITION SYSTEM

Radosław Stelmach MSc, *Technichal University Of Lodz*

Abstract

This study presents the advanced features of mobile devices in the field of image processing with the example application used to recognize Polish license plates. The author presents the different transformation operations performed on images - from geometric transformations, by point operations, filters and morphological operations. All these transformations are performed by own implementation of appropriate image processing algorithms, as well as a highly developed OpenCV library, which has recently been transferred to mobile devices on the Android platform. The application being the result of the work of image processing algorithms is able to separate license plate from the vehicle image, and then select the individual characters forming this plate. In the last stage, separate character images are processed by the Kohonen neural network, which aims to transform the graphic sign letters in its text equivalent.

Streszczenie

Niniejsze opracowanie przedstawia zaawansowane możliwości nowoczesnych urządzeń przenośnych w dziedzinie przetwarzania i obróbki obrazów na przykładzie aplikacji służącej do rozpoznawania polskich tablic rejestracyjnych. Autor przedstawia różne operacje przekształceń wykonywanych na obrazach – począwszy od przekształceń geometrycznych, poprzez punktowe, filtry oraz operacje morfologiczne. Wszystkie te przekształcenia wykonywane są za pomocą własnej implementacji odpowiednich algorytmów przetwarzania obrazów, jak również za pomocą bardzo rozwiniętej biblioteki OpenCV, która w ostatnim czasie została przeniesiona na urządzenia mobilne na platformie Android. Aplikacja będąca efektem prac z algorytmami przetwarzania obrazów jest w stanie wydzielić z obrazu tablicę rejestracyjną pojazdu, a następnie wyselekcjonować poszczególne znaki tworzące tablicę. W ostatnim etapie, wydzielone obrazy znaków przetwarzane są przez sieć neuronową Kohonena, która ma za zadanie przekształcić znak graficzny litery w jej tekstowy odpowiednik.

1. Urządzenia mobilne

Urządzenia mobilne są nieodłącznym elementem dzisiejszego życia. Wraz z rozwojem technologii informatycznych ich rozwój zarówno pod kątem sprzętowym jak i programistycznym należy ocenić jako ekstremalnie szybki. Możemy to zaobserwować poprzez szybką analizę rynku – największą grupę urządzeń przenośnych stanowią obecnie smartfony, czyli urządzenia służące jednocześnie do wykonywania połączeń telefonicznych oraz zastosowania do celów biznesowych, a także rozrywkowych. Producenci prześcigają się w parametrach technicznych swoich produktów, obecnie możemy spotkać urządzenia z czterordzeniowymi procesorami, gdzie każdy rdzeń taktowany jest zegarem ponad 1,5GHz oraz wysoką ilością pamięci RAM (ponad 2GB). Z punktu widzenia liczb - takich parametrów technicznych nie powstydziłby się nawet obecnie produkowany komputer stacjonarny, którego zapotrzebowanie na energię jest wielokrotnie większe od tego, jakie zapewnić może bateria urządzenia przenośnego.

Również peryferia, w które wyposażane są smartfony rozwijają się bardzo szybko. Szczególny rozwój zaobserwować można w aparatach cyfrowych będących dzisiaj standardowym wyposażeniem urządzenia przenośnego. Obecnie najlepsze posiadają matryce o rozdzielczości 4128 x 3096 pikseli (około 13MPx) oraz zaawansowaną optykę umożliwiającą poprawę jakości wykonywanego zdjęcia.

Rosnące parametry urządzeń mobilnych idą w parze z rozwojem systemów operacyjnych, które kładą coraz większy nacisk na prostotę i szybkość powstawania finalnego oprogramowania tworzonego

przez firmy trzecie. Jeszcze do niedawna tak popularny system operacyjny SymbianOS wymagał bardzo dobrej znajomości technicznych danego urządzenia, a poziom skomplikowania jego API był na tyle duży, że do wytworzenia nawet najprostszej aplikacji konieczne było poświęcenie wielu godzin na naukę architektury. Obecnie dwa najpopularniejsze systemy – Apple iOS oraz Google Android dzięki bogatemu wachlarzowi narzędzi programistycznych oraz mnogości poradników i samouczków pozwalają wytworzyć prostą aplikację w bardzo krótkim czasie.

Wykorzystany jako podstawa dla opisywanej w artykule aplikacji LPrecoA system operacyjny Android daje programiście jeszcze jedno bardzo pomocne narzędzie - możliwość pisania kodu w języku Java, co dzięki prostej składni oraz bogactwu gotowych bibliotek, znacznie skraca czas potrzebny do wytworzenia gotowego oprogramowania. Gdyby jednak zachodziła konieczność skorzystania z natywnych bibliotek napisanych w języku C lub C++ – interfejs JNI, który zostanie omówiony później, umożliwia odwołanie się do nich bezpośrednio z języka Java.

2. Przetwarzanie obrazów

Przetwarzanie komputerowych obrazów graficznych jest dynamicznie rozwijającą się dziedziną informatyki. Dzięki temu, jesteśmy w stanie analizować z coraz większą dokładnością dane graficzne w postaci matrycy punktów. Nikogo już nie dziwi rozpoznawanie w ułamek sekundy obrazu twarzy, kciuka, jak również innych elementów otaczającego nas świata – jak np. samochodów, a dokładnie jego marki, modelu oraz tablicy rejestracyjnej.

2.1 Ogólny schemat przetwarzania obrazów

Pełen proces przetwarzania obrazu można podzielić na kilka podstawowych etapów [1]:

1. Akwizycja – jest procesem pobrania obrazu z urządzenia.
2. Dyskretyzacja obrazu – przetworzenie sygnału analogowego na sygnał cyfrowy.
3. Przekształcenia na obrazie, na które składają się przekształcenia:
   a. Geometryczne
   b. Punktowe
   c. Morfologiczne
   d. Filtry
4. Analiza obrazu
5. Zapis obrazu wynikowego lub danych pośrednich uzyskanych z jednego z powyższych etapów.

Dwa pierwsze etapy opisane powyżej wymagają skorzystania z peryferiów – aparatu lub kamery podłączonego do komputera. Im lepsza jakość tych urządzeń, tym dokładność (wierność odwzorowania kolorów, geometrii obrazu, prawidłowe doświetlenie sceny) rośnie. Jednak poprawa jakości najczęściej wiąże się ze zwiększeniem ilości pikseli przechwytywanych przez matrycę kamery, co prowadzi z kolei do spowolnienia działania pozostałych trzech etapów, gdyż ilość informacji rośnie kwadratowo w stosunku do rozmiarów pozyskiwanego obrazu.

2.2 Przetwarzanie obrazów na urządzeniach mobilnych

O ile analiza obrazów na komputerach stacjonarnych wydaje się być oczywista i większość implementacji algorytmów przetwarzania obrazów powstaje wyłącznie z myślą o nich, należy dostrzec jak wysoki potencjał drzemie w nowoczesnych urządzeniach mobilnych. Podejście takie wydaje się być po części zrozumiałe – do niedawna smartfony kojarzyły się z urządzeniami prężnie rozwijającymi się, jednak o bardzo małych zasobach sprzętowych (wolny procesor, mała ilość pamięci RAM, słabe peryferia). Jednak jak to zostało opisane w rozdziale pierwszym, dzisiaj pod nazwą smartfon kryje się praktycznie w pełni funkcjonalny komputer, który najczęściej wyposażony jest w aparat cyfrowy o rozdzielczości matrycy minimum 5MPx. Taka rozdzielczość pozwala uzyskać bardzo zadowalające wyniki w kontekście przetwarzania obrazów.

Największym problemem przy przetwarzaniu obrazów na urządzeniach mobilnych jest brak natywnych bibliotek służących do tego celu. Najczęściej dostępne są jedynie gotowe aplikacje, które jednak bardziej skupiają się na poprawianiu efektów pamiątkowych zdjęć uwiecznionych smartfonem niż udostępnieniu interfejsu do wykonywania pojedynczych (atomowych) operacji na obrazach. Dlatego też, bardzo często deweloperzy piszą własne implementacje podstawowych funkcji służących do wykonywania przekształceń na obrazach. W niniejszej publikacji omówiona zostanie własna implementacja takiej biblioteki, której zasób funkcji umożliwi zlokalizowanie tablicy rejestracyjnej na obrazie, jej wydzielenie oraz wykrycie poszczególnych znaków.

3. Biblioteka do przetwarzania obrazów w systemie Android

Przygotowana biblioteka jest w zasadzie klasą w języku Java, której dołączenie do projektu umożliwia wykonanie następujących operacji dla celów przetwarzania obrazów:

- Dilate (LPImg img, byte[][] kernel)
- Erode (LPImg img, byte[][] kernel)
- Open (LPImg img, byte[][] kernel)
- Close (LPImg img, byte[][] kernel)
- Threshold (LPImg img, byte threshold)
- Histogram (LPImg img)

- CountThresholdValue (LPImg img, byte[] hist)
- AddImages (LPimg img1, LPImg img2)
- SubtractImages (LPimg img1, LPImg img2)
- Crop (LPImg img, int x, int y, int w, int h)
- Resize (LPImg img, int w, int h)
- Rotate (LPImg img, angle)

Pierwsze cztery metody wykonują operacje morfologiczne na obrazie wejściowym (img) z ustalonym elementem strukturalnym (podawanym jako dwuwymiarowa tablica) – kernel.

Metoda Threshold pozwala poddać obraz binaryzacji, czyli operacji redukcji ilości kolorów do dwóch (najczęściej biały i czarny) z określonym progiem (wartością graniczną koloru uznawanego za biały lub czarny). Wprowadzona została również metoda pomocnicza obliczająca statystycznie najkorzystniejszy próg binaryzacji (CountThresholdValue), wykorzystująca jako argument histogram obrazu, który może zostać wyliczony za pomocą metody Histogram.

Kolejne dwie metody wykonują odpowiednio dodawanie i odejmowanie od siebie wartości poszczególnych pikseli obrazów, sprawdzając wcześniej, czy obrazy te są tych samych rozmiarów.

Metoda Crop pozwala wyciąć z obrazu podobraz o rozmiarach w, h zaczynając w punkcie x, y.

Metoda Resize skaluje obraz do nowych rozmiarów w, h; natomiast metoda Rotate pozwala obrócić obraz o zadany kąt (angle).

Wyjaśnienia wymaga również struktura obrazu wejściowego, który jest obiektem klasy LPImg. Klasa ta zawiera w sobie dwuwymiarową tablicę typu byte, przechowującą odcienie szarości pikseli oraz informację o szerokości i wysokości obrazu. Posiada również odpowiednie metody ustawiające i pobierające wartość konkretnego piksela oraz rozmiarów obrazu. Dodatkowo zaimplementowana jest jedna statyczna metoda FromBitmap, która konwertuje plik wejściowy zapisany jako obiekt klasy Bitmap na obiekt klasy LPImg. Ograniczenie przestrzeni kolorów do odcieni szarości okazuje się wystarczające dla celów prawidłowego wykonywania przekształceń opisanych powyżej.

4. Biblioteka OpenCV w systemie Android

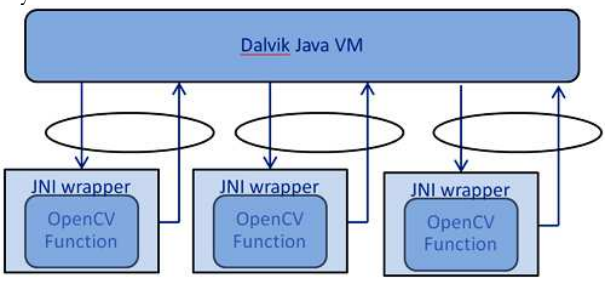
Projekt OpenCV (Open Computer Vision) został założony w 1999 roku przez firmę Intel w celu opracowania biblioteki do zaawansowanych operacji graficznych takich jak Real Time RayTracing oraz przetwarzanie obrazów 3D. Od roku 2012 jest otwartym projektem zrzeszającym bardzo dużą grupę deweloperów stale poprawiających i przyspieszających kod biblioteki. Natywnie, biblioteka ta pracuje na komputerach klasy PC, a jej kod napisany jest przy użyciu języka C oraz C++. Jednak jej stały rozwój oraz duża popularność skłoniła programistów do przenoszenia jej na inne języki programowania, takie jak Python, Java, C#, a także na inne platformy sprzętowe, do których zaliczyć można Symbian, iOS oraz Android.

Aby możliwe było wykorzystanie biblioteki OpenCV w systemie Android, należy skorzystać w projektu OpenCV4Android [2], który dostarcza skompilowanej i zoptymalizowanej pod kątem urządzeń mobilnych wersji OpenCV.

OpenCV4Android umożliwia pisanie kodu w dwóch języka obsługiwanych przez platformę Android. Najprostszym posunięciem jest wykorzystanie OpenCV Java API, czyli specjalnych klas [3] w języku Java zapewniających bezpośrednie wywołanie natywnych metod biblioteki napisanych w języku C++. Obecnie większość natywnych metod posiada już swoje odpowiedniki w języku Java. Najważniejsze dwie klasy do przekształceń graficznych, które prawie w całości zostały przeniesione to: *org.opencv.core* oraz *org.opencv.imgproc*.

W przypadku, gdy oczekiwana przez programistę metoda nie została jeszcze przeniesiona na język Java, konieczne jest napisanie kodu w języku C++ oraz wywołanie go z poziomu języka Java za pomocą interfejsu JNI.

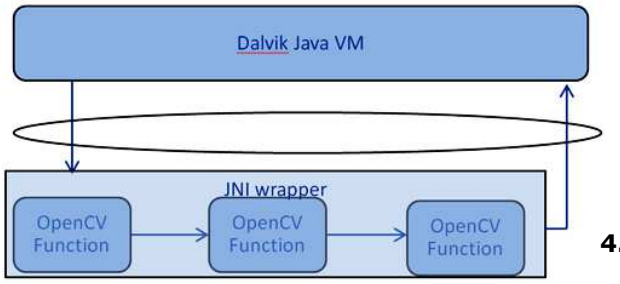
Obie opisane powyżej metody mają swoje wady i zalety. Tak naprawdę obie korzystają z interfejsu JNI, czyli wykorzystują natywny kod biblioteki OpenCV, co wiąże się z pewnym opóźnieniem, które wprowadza JNI. W przypadku pierwszej metody jednak należy liczyć się ze zwiększonym czasem wykonania, gdyż opóźnienie występuje przy każdym wywołaniu pojedynczej metody z biblioteki i to zarówno przy rozpoczęciu jak i zakończeniu wywołania:



Rys.1. Wywołanie pojedynczych instrukcji za pomocą interfejsu JNI.

Fig.1. Many instructions call with JNI interface.

W drugim przypadku, możliwe jest przeniesienie całego kodu przetwarzania obrazu do pojedynczej metody w języku C++ wywołującej w dalszej kolejności natywne metody OpenCV. Takie rozwiązanie wprowadza opóźnienie tylko raz, tzn. podczas rozpoczęcia i zakończenia wywołania JNI:



Rys.2. Wywołanie jednej instrukcji za pomocą interfejsu JNI.

Fig.2. One instruction call with JNI interface.

Ponieważ najczęściej przetwarzanie obrazu następuje sekwencyjnie klatka po klatce, każdorazowe wywołanie kilku metod OpenCV w języku Java wprowadzać może dość duże opóźnienia. Dlatego przy skomplikowanych przekształceniach, zaleca się zapisanie całego kodu procesu przetwarzania w jednej metodzie C++ tak aby zniwelować opóźnienia wprowadzane przez uruchamianie interfejsu JNI.

Większość metod opisanych w rozdziale 3 niniejszej publikacji posiada swoje odpowiedniki w OpenCV i umieszczone są natywnie w przestrzeni nazw cv jako metody: dilate, erode, threshold, calcHist, addWeighted, absDiff, resize. Operacje open oraz close wykonywane są za pomocą odpowiedniego połączenia operacji dilate oraz erode, crop uzyskiwane jest za pomocą zdefiniowania nowej przestrzeni ROI (Region Of Interest) i zapisania kopii obrazu. Operacje resize oraz rotate są trochę bardziej skomplikowane i dokładnie opisane w [4] .

4. Aplikacja LPRecoA

Aplikacja LPRecoA (License Plate Recognition for Android) została napisana w dwóch celach. Po pierwsze jest bardzo dobrą podstawą do testowania możliwości urządzeń przenośnych w kontekście przetwarzania obrazów. Złożoność operacji morfologicznych wykonywanych podczas analizy obrazu jest na tyle duże, że zasoby którymi dysponuje urządzenie są wykorzystywane praktycznie w całości. Dzięki temu sprawdzenie każdej nowszej biblioteki może potwierdzić wprowadzenie ulepszeń na etapie jej projektowania. Drugim celem powstania aplikacji było opracowanie dostatecznie dobrych a jednocześnie prostych algorytmów do wykonania procesu rozpoznania polskiej tablicy rejestracyjnej. Przy jej projektowaniu przyjęto założenie, że wykrywane będą tylko tablice nowego typu (czarne litery na białym tle) w kształcie prostokątnym (bez tablic kwadratowych, rzadko spotykanych)

Proces rozpoznawania tablicy rejestracyjnej można podzielić na cztery podstawowe etapy:

1. Akwizycja obrazu z aparatu wbudowanego w urządzenie
2. Wydzielenie obrazu tablicy rejestracyjnej z pobranego obrazu
3. Wydzielenie pojedynczych znaków (liter i cyfr) z obrazu tablicy rejestracyjnej
4. Rozpoznanie wydzielonych znaków.

4.1 Akwizycja obrazu

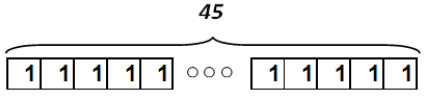
Akwizycja obrazu odbywa się za pomocą wbudowanego w urządzenie mobilne aparatu. W systemie Android możliwe jest pozyskiwanie pojedynczego zdjęcia lub ich sekwencji w czasie rzeczywistym (szybkość przetwarzania zależy od zasobów sprzętowych konkretnego urządzenia). W przypadku aplikacji LPRecoA wykorzystywana jest druga opcja, czyli pozyskiwanie sekwencji wideo, która na bieżąco jest analizowana. Następnie w celu przyspieszenia procesu przetworzenia obrazu, redukowana jest ilość kolorów do odcieni szarości.

Rys.3. Akwizycja obrazu.

Fig.3. Image acquisition.

4.2 Wydzielenie obrazu tablicy rejestracyjnej z pobranego obrazu

Wydzielenie obrazu z tablicy jest procesem złożonym. W pierwszej kolejności obraz skalowany jest do rozdzielczości 640 x 480 pikseli w celu przyspieszenia całej operacji. Kolejnym krokiem jest przeprowadzenie na obrazie operacji Bottom-Hat, która jest złożeniem dwóch innych, tj. najpierw wykonywana jest morfologiczna operacja zamknięcia, a następnie od jej wyniku odejmowany jest pierwotny obraz. Element strukturalny dobrany został doświadczalnie i wygląda następująco:



Wynikiem operacji jest obraz:

Rys.4. Obraz wejściowy po operacji BottomHat.

Fig.4. Input image after BottomHat operation.

Na powyższym obrazie można zauważyć, że mocno uwidocznione zostały kontury znaków na tablicy rejestracyjnej. Aby możliwe było dalsze przeprowadzenie segmentacji obrazu, należy wykonać operację binaryzacji:

Rys.5. Obraz po operacji binaryzacji.
Fig.5. Image after binarization operation.

Ostatnim etapem jest wykonanie dwóch operacji morfologicznych – zamknięcia z elementem strukturalnym o wymiarze 1 x 32, a następnie otwarcia z elementem strukturalnym 1 x 4. Końcowy efekt przedstawia poniższy obraz:

Rys.6. Możliwe położenia tablicy rejestracyjnej.
Fig.6. License plates possible positions.

Na powyższym obrazie uwidocznione zostały obszary, które mogą zawierać tablicę. Obraz taki poddawany jest następnie procesowi etykietowania jednorodnych obszarów. W powyższym wypadku wykrytych zostanie pięć jednorodnych obszarów, jednak tylko jeden z nich może być tablicą rejestracyjną. Za prawdopodobną tablicę uznawany jest obszar, którego szerokość jest przynajmniej 2,5 razy większa od wysokości.

Po wyznaczeniu najbardziej prawdopodobnego obszaru zawierającego tablicę pobierane są współrzędne jego rogów. Następnie z oryginalnego obrazu w podstawowej rozdzielczości wycinany jest obraz tablicy rejestracyjnej na podstawie pobranych współrzędnych odpowiednio przeskalowanych. Na powyższym przykładzie uzyskujemy następujący efekt:

Rys.7. Obraz wyciętej tablicy.
Fig.7. Cropped image of license plate.

Po ukończeniu tego etapu, obraz tablicy rejestracyjnej przekazywany jest do kolejnego modułu aplikacji, który odpowiada za wydzielenie z niej znaków.

4.3 Wydzielenie znaków z tablicy rejestracyjnej.

Aby wydzielić znaki z tablicy rejestracyjnej, konieczne jest skorzystanie z podobnych algorytmów, które wykorzystane zostały do jej zlokalizowania, szczególnie algorytm BottomHat. Problematyczne w tym przypadku okazuje się dobranie odpowiedniego elementu strukturalnego, który powinien być bardzo duży, aby pokryć obraz całej tablicy. Jednak liniowe ułożenie znaków w tablicy daje możliwość zastosowania elementu strukturalnego o wymiarze 1 x X. Aby zredukować rozmiar X, podzielono w pionie tablicę na kilka równych kawałków (empirycznie dobrana wartość to 9 elementów). Następnie w każdym elemencie wybierane jest maksimum wartości koloru (uproszczona wersja algorytmu BottomHat). Ostatecznie wykonywana jest operacja różnicy obrazu wynikowego i wejściowego oraz binaryzacja.

Po tych operacjach wynikowo otrzymywany jest obraz:

Rys.8. Wynikowy obraz po operacji wyszukiwania znaków na tablicy rejestracyjnej.
Fig.8. Final image of finding characters operations in license plate.

Ostatnim krokiem w tym etapie jest wykonanie indeksowania tożsamych obszarów, które układają się w znaki identyfikujące tablicę rejestracyjną, a następnie pobranie pozycji rogów zindeksowanych znaków i wycięcie ich z wynikowego obrazu.

4.4 Rozpoznanie znaków.

Niniejsza praca skupia się wyłącznie na etapach przetwarzania obrazów w celu zlokalizowania tablicy oraz wydzielenia znaków, dlatego etap rozpoznania znaków opisany zostanie skrótowo.

Do prawidłowego rozpoznania znaków użyto sieci neuronowej Kohonena, która składa się ze 160 neuronów w warstwie wejściowej oraz 35 neuronów w warstwie wyjściowej. Ilość neuronów w warstwie wejściowej dobrana została na podstawie ilości pikseli, na które zostaje podzielony dany znak wycięty z tablicy (wcześniej następuje przeskalowanie znaków do odpowiednich rozmiarów), natomiast rozmiar warstwy wyjściowej równy jest wszystkim możliwym znakom, które mogą zostać umieszczone na tablicy rejestracyjnej.

Wszystkie wycięte znaki z tablicy rejestracyjnej podawane są do sieci neuronowej w kolejności od lewej do prawej strony. Wynik przetwarzania wyświetlany jest użytkownikowi aplikacji.

5. Podsumowanie prac

Aplikacja LPRecoA została napisana w celu sprawdzenia możliwości nowoczesnych urządzeń mobilnych pod kątem przetwarzania obrazów. Do realizacji pracy wykorzystane zostały dwie biblioteki do przetwarzania obrazów – własna implementacja oraz profesjonalna biblioteka OpenCV. Obie implementacje poradziły sobie z powierzonym zadaniem, choć maksymalna prędkość przetwarzania obrazu wynosiła zaledwie cztery klatki na sekundę, niezależnie od wybranej biblioteki. Można więc na tej podstawie wnioskować, że pomimo użycia biblioteki OpenCV napisanej w języku C++, która teoretycznie powinna przynieść lepsze rezultaty (pod względem prędkości), kod własnej implementacji biblioteki napisanej w języku Java (a przez to wykorzystywania maszyny Java) nie powoduje spadku wydajności.

Szybkość przetwarzania obrazów jest dosyć niska. Wpływa na to użycie algorytmów, które wymagają wysokich zasobów urządzenia, na którym następuje przetwarzanie, a zapewnienie takich na urządzeniu mobilnym jest trudnym zadaniem. Oczywiście wartości liczbowe, które podają producenci często są bardzo wysokie, jednak w praktyce okazuje się, że pod wpływem wysokiego zapotrzebowania na moc obliczeniową, nie są one w stanie sprostać stawianym im wymaganiom.

Literatura

1. Ryszard Tadeusiewicz: *Komputerowa analiza i przetwarzanie obrazów*, Wydawnictwo Fundacji Postępu Telekomunikacji, Kraków 1997
2. Andrey Pavlenko: *OpenCV for Android*, Materiały Konferencyjne EECV 2012, 7-12 października 2012, Florencja
3. Dokumentacja dołączona do biblioteki OpenCV4Android, zamieszczona na stronie http://docs.opencv.org/doc/tutorials/introduction/android_binary_package/O4A_SDK.html
4. Dokumentacja dołączona do biblioteki OpenCV4Android, zamieszczona na stronie http://docs.opencv.org/modules/imgproc/doc/geometric_transformations.html

Adres służbowy:

Mgr inż. Radosław Stelmach
Katedra Mikroelektroniki i Technik Informatycznych, Politechnika Łódzka
ul. Wólczańska 221/223
90-924 Łódź
tel. 509 101 743
fax (042) 636 03 27

email: rstelmach@dmcs.pl