AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ ELEKTRONIKI,
INFORMATYKI I TELEKOMUNIKACJI

KATEDRA TELEKOMUNIKACJI

Praca dyplomowa magisterska

Algorytm rozpoznawania gestów dłoni na podstawie sekwencji wideo
Algorithm for recognize hand gestures using video sequence

Autor: *Agnieszka Job*
Kierunek studiów: *Teleinformatyka*
Opiekun pracy: *dr inż. Jarosław Bułat*

Kraków, 2018

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Serdecznie dziękuję rodzinie, wszystkim przyjaciołom oraz znajomym, którzy pomogli w tworzeniu tej pracy, za pomoc w gromadzeniu bazy zdjęć trenujących.

Spis treści

1. Wstęp

Każdy komputer posiada interfejs, za pomocą którego wydawane są żądania i komunikaty od użytkownika. Najbardziej popularna jest myszka i klawiatura lub ekran dotykowy. Aby ta interakcja była bardziej naturalna dla człowieka zaczęto wprowadzać aplikacje, które pozwalają na komunikacje z komputerem za pomocą gestów lub głosu. Projekt ten w pewnym sensie wychodzi naprzeciw temu zadaniu. Algorytm rozpoznawania gestów na podstawie sekwencji wideo można wykorzystać do tego, żeby rozpoznane gesty były równoważne z pewnym zadaniem jakie ma wykonać urządzenie.

W tym projekcie interfejsem miedzy maszyną, a użytkownikiem jest kamera internetowa. Zarejestrowane przez nią obrazy przesyłane są do komputera, gdzie następnie są przetwarzane. Celem aplikacji jest wykrycie dłoni w ramce wideo, następnie rozpoznanie gestu. Do detekcji dłoni zastosowano algorytm kalibracji, który wyznacza wartość pikseli znajdujących się na ręce. Liczby te używane są jako wartości progowe oddzielające kolor skóry od koloru tła. Gdy dłoń zostanie odseparowana od otoczenia program zaczyna predykcje gestu. Klasyfikacja gestów przeprowadzana jest za pomocą techniki uczenia maszynowego. Posłużyła do tego maszyna wektorów nośnych SVM. Aby dwuwymiarową macierz przedstawić za pomocą jednowymiarowego wektora, wykorzystano metodę histogramu zorientowanych gradientów. Dzięki niej zostały zachowane wszystkie najważniejsze informacje potrzebne do wykrycia konkretnego znaku. Tak zbudowana aplikacja musi być w stanie wykryć gest w czasie rzeczywistym.

Niniejsza praca opisuje proces projektowania oraz wyniki jakie udało się osiągnąć dla algorytmu rozpoznawania gestów dłoni na podstawie sekwencji wideo. Ponadto zawiera opis teoretyczny zastosowanych metod w tym projekcie.

2. Cel i założenia projektu

Niniejsza praca opisuje proces tworzenia algorytmu rozpoznawania gestów na podstawie sekwencji wideo. Jej celem było zbudowanie oraz przetestowanie określonego ciągu czynności dążących do detekcji gestów wykonywanych dłonią. Potencjalnym zastosowaniem algorytmu jest automatyczne identyfikowanie języka migowego (np. tylko cyfr) albo automatyczne rozpoznawanie gestów sterujących urządzeniami.

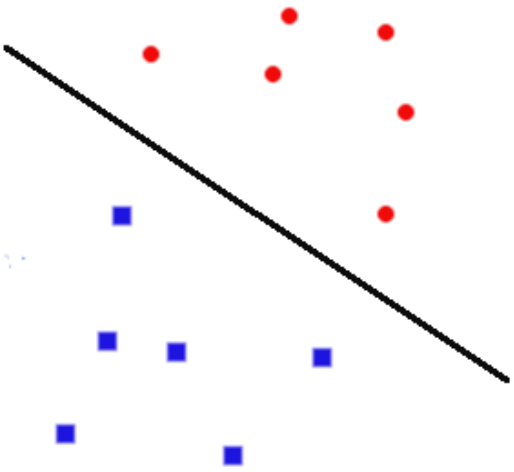
Zakresem tej pracy było znalezienie metody detekcji gestów, wykonanie oraz przetestowanie jej. Do tworzenia algorytmu można posłużyć się bibliotekami pomocniczymi takimi jak np. OpenCV, przy odpowiednim rozszerzeniu jej funkcjonalności oraz niezawodności. Początkowym zamysłem pracy było stworzenie aplikacji na system mobilny **Android** mającej na celu rozpoznawanie gestów cyfr od zera do dziewięciu. Na obrazie przechwytywanym przez tylną kamerę telefonu wykrywana miała być dłoń, a następnie gesty odpowiadające liczbom. Program w kolejnym etapie przepisywałby cyfry na numer telefonu, z którym kontaktowałby się lub zapisywał w liście kontaktów. Do tej idei aplikacji należało wymyślić sekwencje gestów odpowiadających cyfrom przedstawianym na dwóch dłoniach tak aby można było je przedstawić wyłącznie za pomocą jednej dłoni. W tym przypadku posłużono się istniejącą już metodą używaną przez Chińczyków. Do detekcji gestów zdecydowano się na zastosowanie techniki uczenia maszynowego. Dlatego użyto metodę wektorów nośnych SVM. Dla tak wybranej procedury jednym z etapów pracy było zgromadzenie bazy zdjęć mających na celu wytrenowanie klasyfikatora. Minimalną liczbą fotografii dla każdego gestu był zestaw 100 próbek wejściowych. Kolejnym krokiem było znalezienie efektywnej metody wydzielania dłoni od tła. Po zaimplementowaniu algorytmu dla systemu **Android**, okazało się że biblioteka ucznia maszynowego posiada błąd niepozwalający na jakąkolwiek detekcje gestów. Z tego powodu zdecydowano się na przepisanie programu dla systemu operacyjnego **Linux** w języku programowania **Python**, używając do przechwytywania obrazu wideo kamery internetowej.

Ostatnim etapem niniejszej pracy było przetestowanie oraz walidacja zastosowanej techniki. W szczególności sprawdzenie działania aplikacji z różnymi kamerami internetowymi w odmiennych warunkach oświetlenia.

3. Opis teoretyczny zagadnień

3.1. Metoda wektorów nośnych

Metoda wektorów nośnych (ang. Support Vector Machine) została opracowana przez Vapnika, Lernera oraz Czervonenkisa na początku lat sześćdziesiątych. Jednak doceniono ją dopiero w latach dziewięćdziesiątych XX wieku. Jest to system uczący się, który ma na celu podzielić zestaw wejściowych wektorów wzorcowych na dwie klasy za pomocą optymalnej hiperpłaszczyzny. Możemy powiedzieć, że zestaw wektorów jest optymalnie odseparowany jeżeli możemy oddzielić klasy bez żadnego błędu, a odległości miedzy najbliższymi wektorami do hiperpłaszczyzny są maksymalne[1]. SVM został zaimplementowany pierwotnie dla dwóch klas, które można oddzielić prostą. Z czasem implementacje rozszerzono o klasyfikacje wieloklasową. Dodatkowo, gdy klasy nie są liniowo separowalne stosowane są transformacje nieliniowych metod dyskryminacyjnych na model liniowy w wyższej przestrzeni wymiarowej. W takich przypadkach najczęściej stosowana jest radialna funkcja bazową (ang. radial basis function - RBF) z jądrem funkcji Gaussa.



Rys. 3.1. Dwie klasy danych wejściowych liniowo separowalnych

Na potrzeby tego dokumentu przedstawiony zostanie przykład dwóch populacji liniowo separowalnych. Załóżmy dwie klasy wektorów, które możemy podzielić co najmniej jedną prostą $G_i (i = 1, 2)$.

Jako n-elementową próbę uczącą zastosujemy $L_n = \{(x_1, y_1), ..., (x_n, y_n)\}$, gdzie x_j jest wektorem zawierającym wartości cech charakterystycznych danej próbki, a y_j mówi nam do której z klas wektor trenujący należy $(j = 1, 2, ..., n)$. Przyjmijmy, że

$$y_j = \begin{cases} +1 & \text{jeżeli } x_j \in G_1 \\ -1 & \text{jeżeli } x_j \in G_2. \end{cases} \tag{3.1}$$

Hiperpłaszczyzna separująca klasy ma postać

$$H : w \cdot x + b = 0. \tag{3.2}$$

W tym przypadku dane treningowe można podzielić na dwie klasy w sposób:

$$\begin{cases} w \cdot x_j + b > 1 & \text{jeżeli } y_j = 1 \\ w \cdot x_j + b \leq -1 & \text{jeżeli } y_j = -1, \end{cases} \tag{3.3}$$

gdzie $w \in R^n$ i $b \in R$. Otrzymując w ten sposób [1]:

$$y_j(w \cdot x_i + b) \geq 1 \quad \forall x_i \quad i = 1, 2, ..., n. \tag{3.4}$$

Inaczej mówiąc, punkty x_j leżą na zewnątrz dwóch równoległych hiperpłaszczyzn:

$$H_+ = \{x \in R^n \quad : w \cdot x + b = 1\} \tag{3.5}$$

$$H_- = \{x \in R^n \quad : w \cdot x + b = -1\} \tag{3.6}$$

Odległość między punktem x, a hiperpłaszczyzną H wynosi $d(w, b; x) = |w \cdot x + b| \div \|w\|$ [1]. Odnosząc się do zależności (3.3), minimalna odległość między jedną z dwóch klas, a hiperpłaszczyzną jest $1/\|w\|$. Idąc dalej w tym kierunku odległość między dwoma klasami wynosi

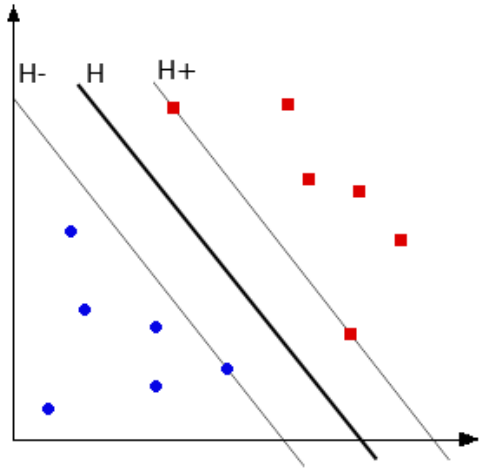
$$\triangle = 2/\|w\|. \tag{3.7}$$

Zadaniem SVM jest maksymalizacja odległości między hiperpłaszczyznami H_+ i H_-, którą można osiągnąć poprzez wyznaczenie wektora normalnego **w** o minimalnej normie przy zadanych ograniczeniach (3.3). Parametr b hiperpłaszczyzn (3.5) oraz (3.6) nie wpływa na wartość odległości $\triangle$. Jego wartość jest wyznaczana z (3.3) [2]. Ten nieliniowy problem optymalizacyjny możemy rozwiązać stosując na przykład metodę mnożników Lagrange'a [1]

$$L(w, b, \alpha) = \frac{1}{2} w \cdot w - \sum_{i=1}^{K} \alpha_i (y_i (w \cdot x_i + b) - 1), \tag{3.8}$$

gdzie $\alpha = (\alpha_1, \alpha_1, ..., \alpha_n)' \geq 0$ są mnożnikami Lagrange'a. W tym momencie ten problem minimalizacyjny może zostać przedstawiony jako problem dualny, zwany problemem programowania kwadratowego [1].

$$\frac{\partial L(w, b, \alpha)}{\partial w}|_{w=w_0} = w_0 - \sum_{i=1}^{K} \alpha_i y_i x_i = 0 \tag{3.9}$$



Rys. 3.2. Model SVM z przedstawionymi hiperpłaszczyznami separującymi

$$\frac{\partial L(w,b,\alpha)}{\partial b}|_{b=b_0} = \sum_{i=1}^{K} \alpha_i y_i = 0 \tag{3.10}$$

Aby sformułować problem w postaci dualnej (3.9) oraz (3.10) podstawia się do (3.8) otrzymując [1]

$$L(\alpha) = \sum_{i=1}^{K} \alpha_i - \frac{1}{2} \sum_{i=1}^{K} \sum_{j=1}^{K} \alpha_i \alpha_j y_i y_j x_i x_j. \tag{3.11}$$

Szukany jest więc punkt $\alpha \in R^n$ dla którego funkcja $L(\alpha)$ osiąga maksimum dla warunków [1]:

$$\sum_{i=1}^{K} \alpha_i y_i = 0, \quad \alpha_i \geq 0. \tag{3.12}$$

Rozwiązanie powyższego problemu związane jest z dodatkową zależnością opartą o twierdzenie Karusha-Kühna-Tuckera postaci:

$$\alpha_i [y_i (w x_i + b_0) - 1] = 0, \quad i = 1, 2, ..., n. \tag{3.13}$$

Dla wszystkich α_i większych od 0

$$y_i (w x_i + b_0) = 1. \tag{3.14}$$

Oznacza to, że wszystkie wektory danych wejściowych x_i, którym odpowiadają niezerowe mnożniki Lagrange'a leżą na hiperpłaszczyznach kanonicznych. Wektory te nazywane są wektorami nośnymi (ang. support vectors)[3]. Co za tym idzie hiperpłaszczyznę separującą można wyznaczyć poprzez [1]:

$$w_0 = \sum_{i=1}^{K} \alpha_i y_i x_i \tag{3.15}$$

$$b_0 = y_i - w_0 \cdot x_i, \tag{3.16}$$

gdzie x_i są wektorami wspierającymi, a $y_i \in \{-1, 1\}$. Wektory, które podlegają automatycznej klasyfikacji x_t są rozpoznawane za pomocą funkcji [1]:

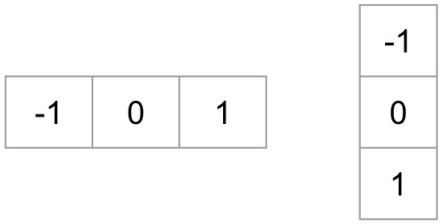
$$f(x) = sign(w_0 \cdot x_t + b_0). \tag{3.17}$$

Hiperpłaszczyzna separująca jest wyznaczana wyłącznie na podstawie wektorów nośnych, a nie na podstawie całego zestawu wektorów wejściowych. Wyżej przedstawiona metoda SVM jest ograniczona tylko do dwóch klas wektorów wejściowych, które można odseparować liniowa hiperpłaszczyzną. Jednak ten system uczący się jest też poszerzony o wieloklasowość oraz nieliniowe klasyfikatory opierające się na radialnej funkcji bazowej.

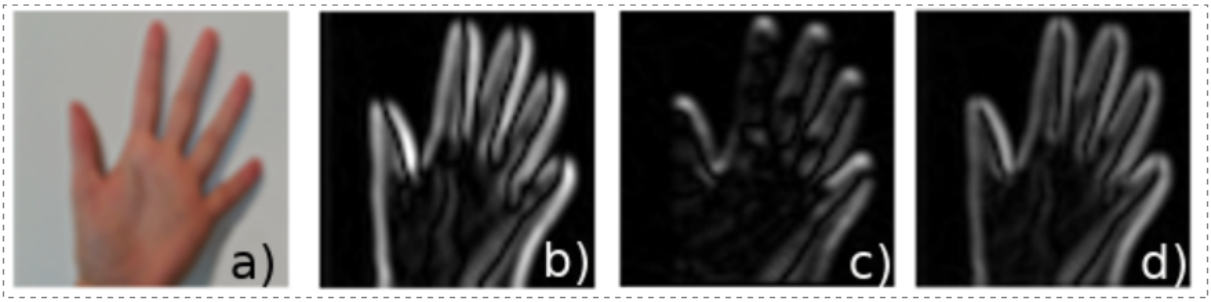
3.2. Histogramy zorientowanych gradientów - HOG

Do sklasyfikowania obiektu za pomocą SVM potrzebny jest wektor n-wymiarowy. Nasuwa się pytanie jak można zastosować metodę wektorów nośnych w przypadku obrazów, które są macierzami 2D, często zapisywane w formacie trój-kanałowym RGB. Odpowiedzą na nie jest histogram zorientowanych gradientów (ang. Histogam of Oriented Gradients). Metoda ta ma na celu z macierzy 2D stworzyć wektor zawierający najbardziej istotne informacje znajdujące się w obrazie. W przypadku rozpoznawania wzorców najważniejszymi cechami nie są kolory zawarte w obrazie, a obiekt który, nas interesuje równie dobrze może być przedstawiony w skali szarości. Idąc dalej, wystarczającą ilość informacji dostarczy sam zarys. W deskryptorze HOG rozkład (histogramy) kierunków gradientów (zorientowanych gradientów) używany jest jako cechy charakterystyczne [4]. Wartości gradientów są największe w obszarach znajdujących się wokół krawędzi i narożników, przez co niosą ze sobą więcej informacji o kształcie obrazu niż płaskie obszary.

Danymi wejściowymi dla HOG są obrazy w skali szarości o wymiarach n x m pikseli. Obraz dzielony jest na fragmenty przez gęstą siatkę małych komórek. Pierwszy etapem algorytmu jest wyznaczenie horyzontalnych oraz wertykalnych gradientów. Stosowana do tego jest jednowymiarowa wyśrodkowana, punktowa, dyskretna maska w obu kierunkach, poziomym i pionowym. Obraz filtrowany jest za pomocą następujących filtrów jądra:



Rys. 3.3. Filtry jądra [4]



Rys. 3.4. a) obraz oryginalny b) gradienty wertykalne c) gradienty horyzontalne d) zsumowane gradienty wertykalne i horyzontalne

Niech $G_i(i,j)$ oraz $G_j(i,j)$ oznaczają gradient danego piksela, a $I(i,j)$ oznacza wartość intensywności w położeniu (i,j), gdzie $i = (1,2,...,n)$ oraz $j = (1,2,...,m)$.

$$G_i(i,j) = I(i+1,j) - I(i-1,j) \tag{3.18}$$

$$G_j(i,j) = I(i,j+1) - I(i,j-1) \tag{3.19}$$

Wielkość oraz kierunek gradientu dla każdego piksela jest obliczana jako [5]:

$$M(i,j) = \sqrt{G_i^2(i,j) + G_j^2(i,j)} \tag{3.20}$$

$$\theta(i,j) = tan^{-1}(\frac{G_i(i,j)}{G_j(i,j)}) \tag{3.21}$$

Następnie przedział kątowy od $0°$ do $180°$ jest dzielony na 9 kontenerów każdy o przedziale $20°$. Każda komórka posiada własną grupę dziewięciu kontenerów, które będą przedstawiać histogram gradientów w danej komórce. Wielkości gradientów są dodawane do odpowiednich kontenerów zgodnie z jego wagą obliczoną na podstawie orientacji:

$$W = (a + 0,5) - b \cdot \frac{\theta(i,j)}{180°}, \tag{3.22}$$

gdzie a to numer kontenera, do którego należy dany gradient, a b to liczba wszystkich kontenerów. Jeżeli waga $W \geq 0$, wielkość gradientu jest umieszczana w odpowiednim kontenerze za pomocą:

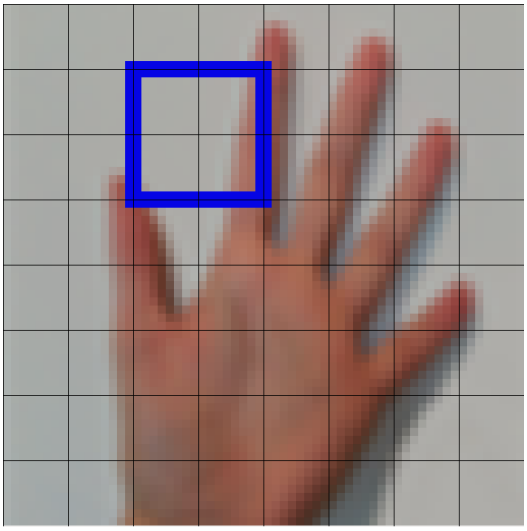
$$M_1 = W \cdot M(i,j) \tag{3.23}$$

$$M_2 = (1 - W) \cdot M(i,j) \tag{3.24}$$

Jeżeli $W < 0$ wielkość gradientu jest umieszczana w odpowiednim kontenerze za pomocą:

$$M_1 = (1 + W) \cdot M(i,j) \tag{3.25}$$

$$M_2 = |W| \cdot M(i,j) \tag{3.26}$$



Rys. 3.5. Obraz podzielony na gęstą siatkę komórek o wymiarach 8×8 pikseli, na niebiesko zaznaczono blok 4 komórek

W ten sposób tworzony jest histogram dziewięcioelementowy dla każdej komórki w obrazie. M_1 oraz M_2 umieszcza ważone wielkości gradientu w odpowiedni kontener. Wymiary komórek dla których liczony jest histogram zależy od tego w jakiej skali szukane są cechy. Nie są one sztywno określone przez algorytm. Kolejny krok ma na celu spowodować, aby histogram gradientów był odporny na zmiany intensywności światła. W tym celu komórki grupuję się w bloki, następnie każdy blok podlega normalizacji za pomocą zależności:

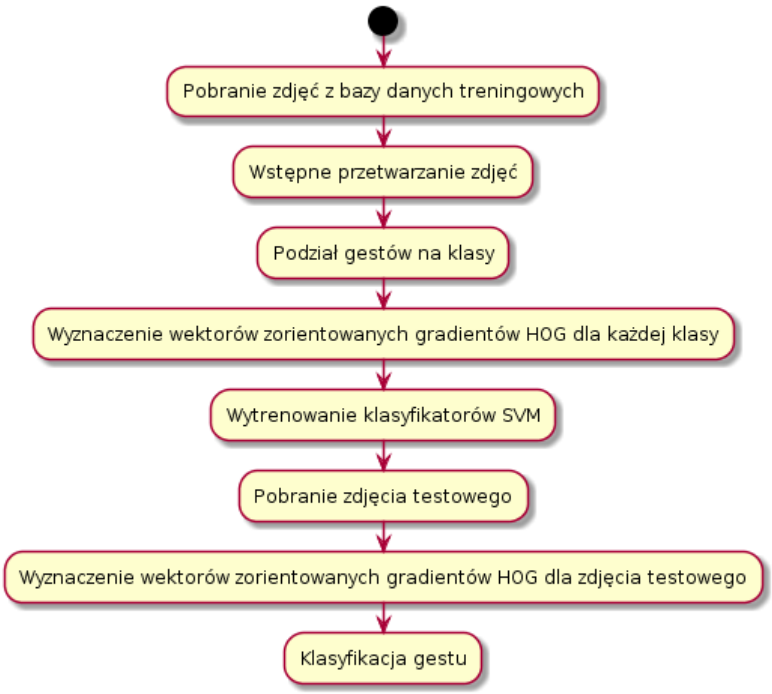
$$fv_k^n = \frac{fv_k}{\sqrt{\sum_{k=1}^{B} fv_k^2 + \varepsilon^2}} \tag{3.27}$$

Gdzie fv_k^n to znormalizowany wektor połączonych histogramów danego bloku, B to liczba kontenerów znajdujących się w bloku, a ε to bardzo mała stała pomocnicza zapobiegająca błędowi dzielenia przez zero.

Podając przykład obrazu o wymiarach 64×64 piksele. Dzielony jest on na komórki o wymiarze 8×8 pikseli, dla których jest wyliczany osobny dziewięcioelementowy histogram. Co daje 64 niezależnych komórek, a w każdej komórce 9 kontenerów histogramu. Dla nieprzetworzonego obrazu w komórce o tych wymiarach przechowywane jest 64 wartości pikseli, gdzie po pierwszych krokach algorytmu wymiar zredukowany jest do jednego dziewięcioelementowego histogramu. Następnie komórki grupowane są w bloki po 4 komórki z 50% nadmiarowością. Dla całego obrazu daje to 49 bloków. Ostateczny wymiar wektora zawierającego histogramy zorientowanych gradientów wynosi $49 \times 4 \times 9 = 1764$.

4. Implementacja

Proces tworzenia algorytmu do rozpoznawania gestów na podstawie sekwencji wideo został podzielony na dwa etapy. W pierwszym rozpoznawanie gestów zrobiono statycznie. Bazę zdjęć gestów pogrupowano na część treningową i testową, na podstawie której dobierano odpowiednie wielkości komórek i bloków dla histogramu zorientowanych gradientów oraz wartości parametrów dla algorytmu SVM. Po dobraniu atrybutów dających najlepsze wyniki, zastosowano je do dynamicznego rozpoznawania gestów. W tym przypadku gesty klasyfikowane były w czasie rzeczywistym. Baza treningowa pozostała ta sama, jednak do rozpoznawania stosowane były gesty pokazywane przez użytkownika przed kamerą. Zastosowany algorytm pokrywa oba etapy. Na schemacie przedstawiony jest model statyczny:



Rys. 4.1. Ogólna sekwencja algorytmu rozpoznawania gestów za pomocą programu statycznie rozpoznawanego gesty

4.1. Środowisko programowe

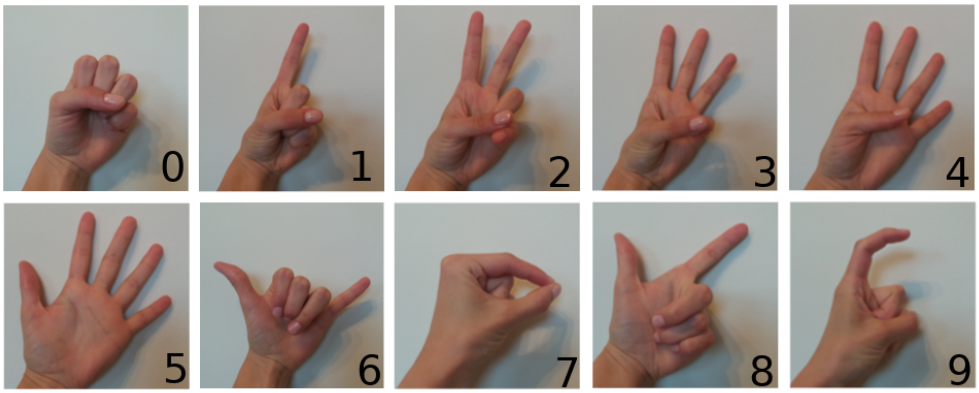
Pierwszy etap został zaimplementowany w języku programowania **Python** wersji 3.6. Język ten jest łatwy do nauki oraz w pełni funkcjonalny, zawierający pełną gamę bibliotek rozszerzających oraz optymalizujących jego działanie. Jest on interpretowanym, obiektowym, wysokopoziomowym językiem [6]. Do operacji na obrazach posłużyła biblioteka **OpenCV** wersji 3.4. Posiada ona wiele funkcji pozwalających na prace ze zdjęciami, filmami jak i na bardzo wydajną obróbkę klatek wideo w czasie rzeczywistym, która umożliwia ingerencję w film bez zmiany liczby kadrów wyświetlanych w ciągu sekundy. W tym projekcie posłużono się takimi funkcjami jak przekształcanie modeli przestrzeni barw z RGB na HSV, histogram zorientowanych gradientów lub metody, dzięki którym można było odseparować piksele za pomocą wartości progowych. Dodatkowym atutem tej biblioteki jest zaimplementowane uczenie maszynowe z klasą `cv::ml::TrainData` zawierająca przykładowe zbiory danych, opcjonalne zestawy odpowiedzi dla próbek [7]. Pakiet ten pozwala również na pełne wykorzystanie takich metod jak normalny klasyfikator Bayesa, k-najbliższych sąsiadów, maszyna wektorów nośnych, drzewa decyzyjne oraz sieci neuronowe. W tym projekcie zastosowano klasę `cv::ml::SVM`, jest ona oparta na oprogramowaniu **LIBSVM**, które w sposób optymalny implementuje maszynę wektorów nośnych pozwalając przy tym na:

- wpływ na różne parametry SVM,
- wydajną wieloklasową typizację,
- sprawdzanie poprawności dla wyboru modelu,
- estymację prawdopodobieństwa,
- zastosowanie różnych jąder,
- oraz wiele więcej [8].

Dodatkowo do operacji na liczbach i macierzach użyto biblioteki **Numpy**. Początkowo druga część była implementowana dla systemu operacyjnego **Android** z pomocą biblioteki **OpenCV**. W planach było stworzenie aplikacji, która za pomocą tylnej kamery będzie rejestrować gesty oznaczające cyfry, rozpoznawać je, a następnie wybierać rozpoznany numer telefonu. Dlatego bazą danych wejściowych są gesty przedstawiające liczby w zakresie od zera do dziewięciu przedstawione wyłącznie na jednej ręce. Niestety biblioteka dla tego systemu operacyjnego zawiera błąd niepozwalający na poprawną klasyfikacje za pomocą metody wektorów nośnych. Z tego powodu implementacja rozpoznawania gestów w czasie rzeczywistym została przepisana na język **Python** w systemie operacyjnym **Ubuntu** z zastosowaniem biblioteki **Tkinter** pozwalającej na zbudowanie interfejsu użytkownika.

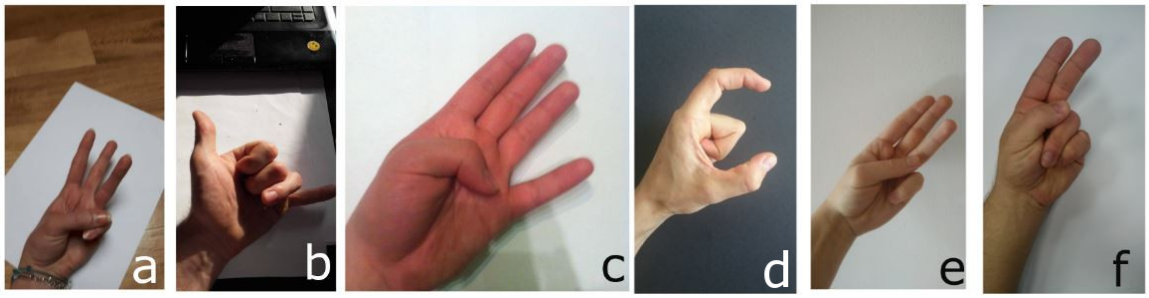
4.2. Analiza danych

Zestaw gestów, jaki został wybrany do tego projektu przedstawia cyfry w zakresie od zera do dziewięciu. Każdy z nich musi być na tyle oryginalny i odróżnialny, aby wytrenowane klasy dla maszyny wektorów nośnych były od siebie położone wystarczająco daleko, pozwalając na dokładniejszą predykcję. Znaki przedstawiane na dłoniach powinny być także łatwe do wykonania dla przeciętnego użytkownika. W przypadku trudnych do przedstawienia gestów napotyka się problem z zebraniem poprawnej wejściowej bazy trenującej. Ponadto podczas predykcji użytkownik może napotkać problem z wykonaniem danego znaku, co spowoduje błędne rozpoznanie. Gesty, na podstawie których wytrenowano maszynę wektorów nośnych przedstawiają cyfry z zakresu od zera do dziewięciu, używane na co dzień przez Chińczyków. Metoda ta jest wygodna dla użytkownika, ponieważ wymaga użycia wyłącznie jednej dłoni dla wszystkich znaków. Pozwalając na użycie drugiej dłoni w inny sposób, na przykład trzymać telefon z uruchomioną aplikacją rozpoznającą gesty. Na 4.2 przedstawiono znaki kolejno obrazujące numeracje.



Rys. 4.2. Chińskie gesty wraz z ich znaczeniem.

Baza danych wejściowych została zapełniona zdjęciami zgromadzonymi samodzielnie, nie pochodzi ona z źródeł zewnętrznych. Zdjęcia wykonano różnymi aparatami, w różnych rozdzielczościach przez różne osoby. Fotografie te wymagały wstępnej selekcji oraz obróbki. Spowodowane było to tym, że posiadały one niejednolite tło lub cienie deformujące kształt, co powodowało złe wytrenowanie klasyfikatora. Początkowo starano się, aby zdjęcia dłoni były robione na względnie jasnym tle. Jednak spostrzeżono, że o wiele lepsze w tym przypadku jest tło ciemne lub czarne. Dzięki takiej oprawie dłoni cienie nie stanowiły problemu przy selekcji koloru skóry. Początkowy zbiór wyniósł około 160 zdjęć na każdy gest, co dało w sumie bazę 1600 zdjęć. Wstępna kwalifikacja polegała na wyrzuceniu zdjęć, które nawet po obróbce nie nadawałyby się do poprawnego wytrenowania maszyny wektorów nośnych. Najczęściej zawierały one źle wykonane gesty lub gesty nie mieszczące się w kadrze. Na 4.3 przedstawiono kilka obrazów, które zostały wyeliminowane z bazy treningowej z powodu błędnie wykonanego gestu lub wymagały bardzo dużej ingerencji w ich wygląd lub rozdzielczość. Modyfikacje fotografii wymagane były żeby dostosować je w taki sposób, aby spełniały założenia wejściowe dla algorytmu histogramu



Rys. 4.3. Przykłady zdjęć nie nadających się do bazy treningowej.

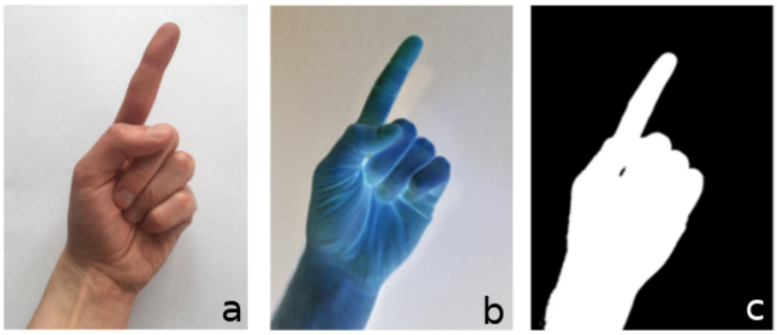
zorientowanych gradientów. Zdjęcie 4.3.a na samym początku wymagały odpowiedniego wykadrowania. Następną czynnością dla tego typu zdjęć było usunięcie cieni, gdyż po binaryzacji, ciemne ślady zostają rozpoznane jako część dłoni zniekształcając jej zarys. Fotografie 4.3.b wyeliminowano z bazy wejściowej, ponieważ wymagało zbyt dużo modyfikacji w tym kadrowania, usuwania cieni oraz zbędnych elementów tła. Dodatkowym powodem wyeliminowania tego przykładu był nie mieszczący się w kadrze palec. W przypadku 4.3.c należało zmienić rozdzielczość zdjęcia, starając się przy tym zachować poprawny kształt dłoni. Dodatkowo jak w przykładach powyżej wymagane było ręczne usunięcie cieni. Na ilustracji 4.3.d gest powinien przedstawiać cyfrę dziewięć, jednak wystający kciuk zniekształca widok, co w kolejnych etapach spowodowałoby złe wytrenowanie klasy dla tej liczby. Kolejne dwa gesty na pierwszy rzut oka wydają się bardzo dobrymi próbkami do wejściowego zestawu danych trenujących, mają odpowiednia jakość oraz nie mają problemu z cieniami mogącymi spowodować zniekształcenia. Problem jaki zaistniał w tych przypadkach to złączone palce w geście. Po zamianie kolorowych zdjęć na maski nie pozwala to na rozpoznanie, który de facto gest przedstawiają. Wyliczając z maski histogram zorientowanych gradientów obrazy te mają bardzo podobny rozkład gradientów jak gest przedstawiający cyfrę jeden. Spowoduje to podczas trenowania modelu niejasne nachodzące na siebie granice między klasami, w skutek czego predykcja staje się o wiele trudniejsza, ciągnąc za sobą osłabienie efektywności takiego algorytmu. Dodatkowo, niektóre fotografie przekształcano w taki sposób, żeby dłonie na nich były ułożone w tym samym kierunku oraz zajmowały więcej niż 70% powierzchni zdjęcia. W następnym kroku próbki zmodyfikowano w taki sposób, aby wszystkie posiadały tą samą rozdzielczość. Czynność ta miała na celu przygotować zdjęcia do poprawnego zastosowania algorytmu histogramu zorientowanych gradientów. Po pierwszej selekcji zbiór posiadał około 1350 zdjęć wszystkich gestów z poprawnie wykonanymi gestami, nadających się do przekształcenia na czarno białą maskę po uprzedniej ingerencji pozwalającej na lepsze wykrycie dłoni od tła. Następnym etapem analizy danych było usuniecie tła zawierającego cienie oraz niejednolitą fakturę, ze zdjęć należących do bazy danych i wykrycie dłoni na zdjęciu. W trudnych przypadkach należało to zrobić ręcznie dla reszty posłużył do tego skrypt w języku Python. Program zamieniał obraz w formacie trój kanałowym RGB na format przestrzeni barw HSV. Dla zdjęć z czarnym tłem stosowano uprzednio inwersje kolorów, za pomocą metody `cv2.bitwise_not()`. W kolejnym kroku obraz rozdzielano na trzy odrębne macierze jednokanałowe zawierające kolejno wartości odcienia (ang. Hue), nasycenia (ang. Saturation) oraz wartości mocy

światła białego (ang. Brightness) równoważne z słowem wartość (ang. Value). Spośród nich wybierana jest macierz zawierająca wartości nasycenia pikseli. Dawała ona najlepsze wyniki dla kolejnych operacji. Dla wyeliminowania szumu w macierzy, zastosowano trzy funkcje mające na celu rozmycie i ujednolicenie: `cv2.blur()`, `cv2.dilate()` oraz `cv2.erode()`. Funkcje te wyliczają średnią z wszystkich pikseli w obszarze jądra i zastępują tą wartością piksel centralny lub stosują operacje morfologiczne do usuwania białego szumu. Na podstawie takiej macierzy wyliczany jest histogram z wartości nasycenia. Z histogramu wyliczana jest wartość średnia, przedstawiająca wartość pikseli jaka występuje najczęściej w macierzy. Używana jest ona jako wartość progowa nasycenia znajdującego się w obrębie dłoni na zdjęciu. Funkcja pogującą z pakietu biblioteki OpenCV zwraca macierz, w której piksele barwy czarnej posiadają wartość nasycenia mniejszą niż progowa, a białe większą od niej. W ten sposób wyjściowym produktem jest obraz przedstawiający białą maskę gestu na czarnym tle. W przypadku tak uzyskanego zdjęcia nie potrzebna już jest wysoka rozdzielczość obrazu, najistotniejszy jest wyselekcjonowany kształt gestu. Dla tego ostatnim elementem programu było pomniejszenie zdjęcia do rozdzielczości 480×800 pikseli. Poniżej fragment kodu w języku Python przedstawiający skrypt, za pomocą którego wyznaczano maski do bazy wejściowej do wyliczenia histogramów zorientowanych gradientów:

```
import numpy as np
import cv2
N = 135
for j in range(1, N):
    kernel = np.ones((5,5),np.uint8)
    img = cv2.imread('test/%d.jpg' %j, cv2.IMREAD_COLOR)
    width, height = img.shape[:2]
    if j >= 58:
        img = cv2.bitwise_not(img)
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    h,s,v = cv2.split(hsv)
    s = cv2.blur(s, (5, 5))
    s = cv2.dilate(s, kernel, iterations=1)
    s = cv2.erode(s, kernel, iterations=1)
    averages = 0
    summary = 0
    mHist, bins = np.histogram(s, 255, [0, 255])
    for i in range(0,255):
        averages += mHist[i] * i
    summary = sum(mHist)
    average = averages / summary
    th, dst = cv2.threshold(s, average, 255, cv2.THRESH_BINARY)
    mask = cv2.resize(dst,(480, 800) , interpolation=cv2.INTER_AREA)
    cv2.imwrite('testmask/%d.jpg' %j, mask)
```

Rysunek 4.4 przedstawia trzy etapy przetwarzania obrazu. Wejściowe zdjęcie w formacie trój kanałowym RGB na 4.4.a. Kolejne przedstawia wyselekcjonowaną macierz wartości nasycenia z trój kanałowego formatu HSV. Porównując oba zdjęcia ewidentnie widać, że na macierzy z wartościami nasycenia

można w o wiele łatwiejszy sposób oddzielić dłoń od tła. 4.4.c przedstawia jedno kanałową macierz składająca się wyłącznie z dwóch wartości pikseli 0 lub 255. Wartość 255 określająca kolor biały, są to te piksele które posiadały wartość nasycenia większa niż już progowa. Kolor czarny określony wartością 0 reprezentuje wszystkie piksele posiadające wartości nasycenia mniejsze niż progowa. W ten sposób w masce kolorem białym wyznaczono miejsce, w którym znajduje się dłoń, a czarnym resztę.



Rys. 4.4. a) obraz w formacie RGB, b) obraz zawierający wyłącznie wartości nasycenia z formatu HSV, c) maska z wyznaczoną dłonią

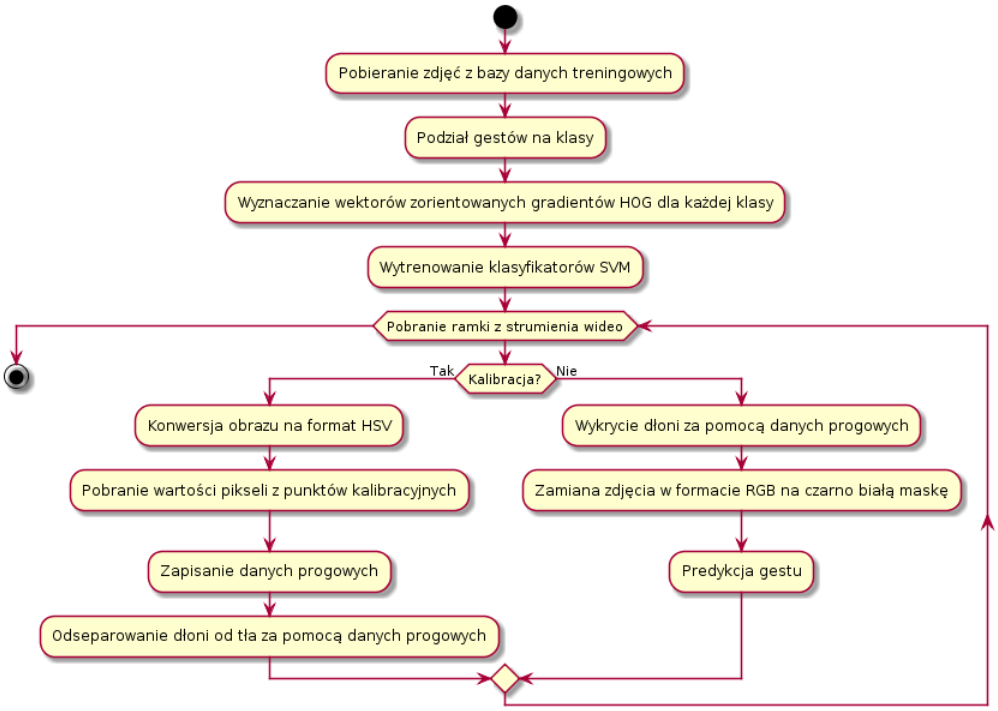
Ostatecznie bazę danych wejściowych do wyliczenia histogramów zorientowanych gradientów stanowi 1300 zdjęć. W zestawie wejściowym znajdują się czarno białe fotografie, o takiej samej rozdzielczości wynoszącej 800×480. Próbkami są maski z wykrytą dłonią przedstawioną w kolorze białym na czarnym tle. Wartości pikseli w macierzach wynoszą 0 lub 255.

4.3. Trenowanie klasyfikatorów i detekcja gestów

Przygotowaną bazę danych podzielono na dziesięć klas odpowiednio dla każdego gestu. Każda klasa treningowa zawierała 125 fotografii, z czego do celów predykcji statycznej pozostawiono 84 zdjęcia. Chcąc eksperymentalnie wypróbować różne metody predykcji oraz aby znaleźć odpowiednie wartości stałych dla algorytmów zastosowano skrypt napisany w języku Python. Posłużył on do wytrenowania oraz sprawdzenia działania maszyny wektorów nośnych z pomocą algorytmu histogramu zorientowanych gradientów. W każdej grupie odpowiadającej danemu gestowi obrazy zmniejszono do rozdzielczości 60×100 pikseli i zastosowano algorytm histogramu zorientowanych gradientów HOG. Najlepsze osiągnięcia odnotowano, gdy rozmiar komórki wyniósł 10×10 pikseli, rozmiar bloku to 40×40 pikseli oraz wielkość nadmiarowości wyniosła 20×20 pikseli. Dla każdej kategorii stworzono wektor zawierający wszystkie 125 histogramy zorientowanych gradientów. Następnie wektory te sklejono w macierz o dziesięciu kolumnach, każda reprezentująca odpowiednią klasę symbolu od 0 do 9. Kolejnym krokiem było wytrenowanie klasyfikatora. Do tego użyto klasy SVM z pakietu biblioteki OpenCV. W aplikacji zastosowano maszynę wektorów nośnych opartą o radialną funkcje bazową RBF z parametrami $C = 12,5$ oraz $\gamma = 0.3$. Wstępną predykcję przeprowadzono na aplikacji statycznej. W tym przypadku dziesięć klas wytrenowanych na podstawie 1250 zdjęć przetestowano na podstawie bazy danych posiadających

84 czarno białe maski. Dla parametrów dobranych jak powyżej precyzja detekcji wyniosła 94.62%, co można uznać za bardzo zadowalający wynik.

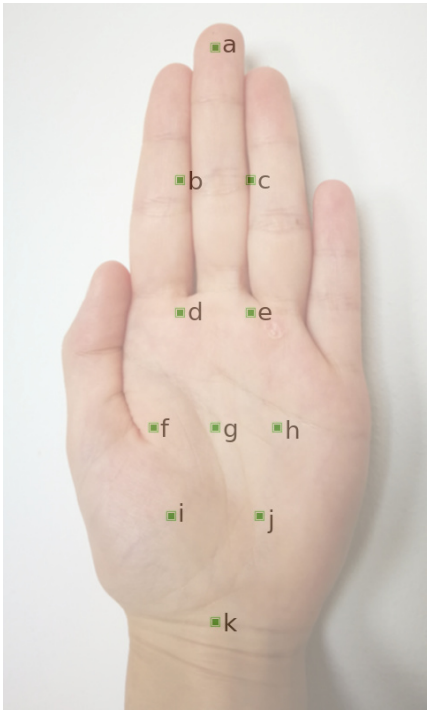
Docelowa aplikacja posłużyć ma do detekcji gestów w czasie rzeczywistym, a dane wejściowe, które podlegają predykcji przekazywane są od użytkownika za pomocą kamerki internetowej. W tym celu aplikacje statycznej detekcji gestów rozbudowano. Dodano do niej funkcjonalność kalibracji mającą na celu wykrycie koloru dłoni, obsługę przesyłania strumieniowego wideo oraz interfejs użytkownika. Na rysunku 4.5 znajduje się przepływ ostatecznej wersji aplikacji rozpoznającej gesty użytkownika w czasie rzeczywistym.



Rys. 4.5. Diagram procesu docelowej aplikacji z detekcją gestów w czasie rzeczywistym

Program od fazy startowej, czyli pobierania danych treningowych, aż do klasyfikatora SVM wygląda tak samo jak w przypadku programu statycznie wykrywającego gesty. Etapem zupełnie nowym jest kalibracja. Pierwszym jej etapem jest konwersją zdjęcia z formatu RGB na HSV. W przypadku, gdy aplikacja jest używana przez użytkownika, przestrzeń poza dłonią posiada różne barwy, a na tle ręki znajdują się często inne elementy , niż w przypadku zdjęć znajdujących się w bazie trenującej. Co więcej może się ono zmieniać w trakcie używania aplikacji. Użytkownik także może zmieniać położenie dłoni względem kamery. Powoduje to, że nie jest możliwe wykrycie dłoni tylko na podstawie wartości progowej równej najczęściej występujących wartości pikseli. Często też w tle znajdują się elementy, które nie pozwalają na poprawną detekcje dłoni na klatce wideo. Dlatego wartości kolorów znajdujące się na dłoni są pobierane podczas kalibracji za pomocą 11 punktów kalibracyjnych. Dodatkowo, aby jeszcze dokładniej określić wartości kolorów znajdujące się na dłoni, punkty kalibracyjne podzielono na 4 grupy,

w obrębie których wyliczane są wartości progowe mające na celu oddzielić kolory należące do dłoni od tła. Żeby kalibracja przebiegła poprawnie użytkownik musi umieścić dłoń w kadrze w taki sposób, aby dłoń pokrywała wszystkie punkty i równocześnie nie wychodziła poza niego.



Rys. 4.6. Rozmieszczenie punktów kalibracyjnych wraz z przykładem poprawnie ułożonej dłoni względem punktów

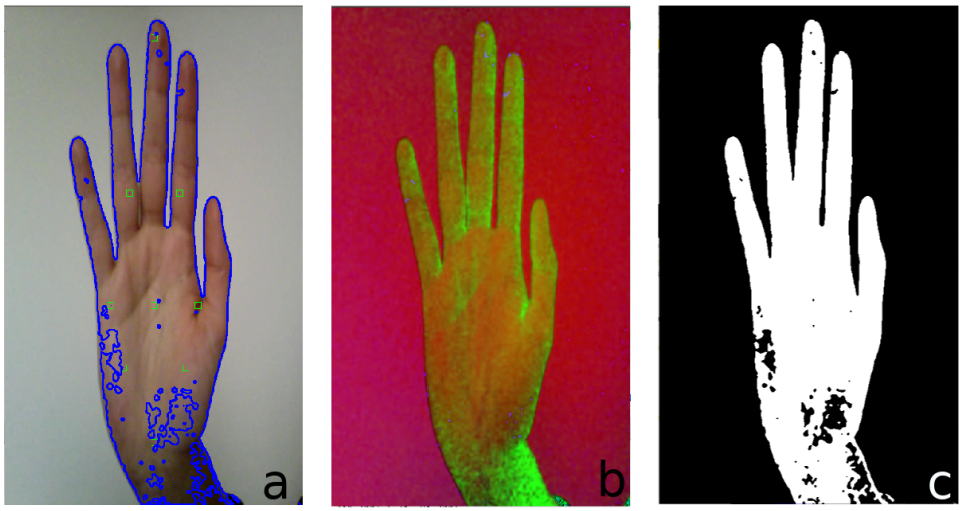
Na rysunku 4.6 kolorem zielonym zaznaczono położenie jedenastu punktów kalibracyjnych. W tych miejscach pobierane są wartości pikseli w formacie HSV. Punkty podzielono na cztery grupy w następujący sposób:

- Grupa I: punkty od a do c,
- Grupa II: punkty d oraz e,
- Grupa III: punkty od f do h,
- Grupa IV: punkty od i do k.

Następnie dla każdej grupy z osobna wybierane są największa oraz najmniejsza wartość parametrów odcienia H oraz nasycenia S. Natężenie światła białego V nie wpływała na jakość wykonywanej kalibracji, dlatego zaniechano przeprowadzania klasyfikacji pikseli na podstawie tej wartości. Wybrane wartości H oraz S podlegają dodatkowej korekcji pozwalającej na lepsza selekcję wartości pikseli. Wartości korekcyjne zostały wybrane osobno dla każdej grupy pikseli kalibrujących. W kolejnym kroku wykorzystywane one są w funkcji progującej. Poniżej przykładowa metoda skryptu w języku Python wybierająca wartości kalibracyjne dla danej grupy:

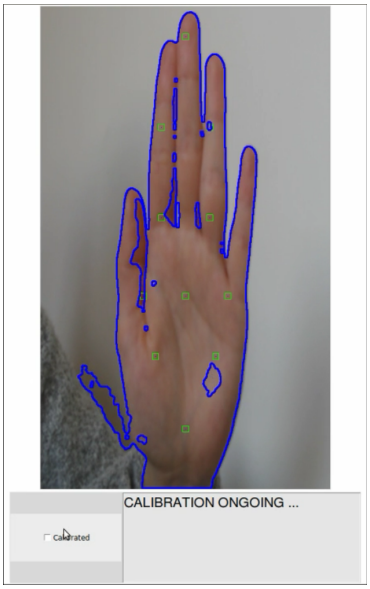
```
def calibrationOfTresholdEMN(self):
    valuesmn = []
    uppermn = [0, 0, 0]
    lowermn = [0, 0, 0]
    for i in range(0, 3):
        valuesmn.append(e[i])
        valuesmn.append(m[i])
        valuesmn.append(n[i])
        sorted(valuesmn)
        lowermn[i] = valuesmn[0]
        uppermn[i] = valuesmn[2]
        valuesmn.clear()
    self.avUpperTmn = np.asarray([uppermn[0] + 30, uppermn[1] + 30, 255])
    lowermn[0] = lowermn[0] - 30
    if (lowermn[0]) < 0:
        lowermn[0] = 0
    self.avLowerTmn = np.asarray([lowermn[0], lowermn[1] - 20, 0])
```

Algorytm ten zostaje obojętny wobec wartości mocy światła białego. W ten sposób wyznaczane są cztery biało-czarne maski kolejno dla każdej grupy. Kolorem białym są zaznaczane piksele posiadające wartości H oraz nasycenia S mieszczące się w wyznaczonym wcześniej przedziale, reszta pikseli barwiona jest na kolor czarny. Powstałe w ten sposób cztery maski oddzielające piksele barwy skóry od tła łączone są razem w jeden obraz za pomocą operacji bitowej OR. Na 4.7 przedstawiono proces kalibracji gdzie 4.7.a pokazuje efekt wyjściowy dla użytkownika, 4.7.b obraz w formacie HSV, z którego pobierane są wartości pikseli oraz 4.7.c połączone maski, na podstawie których rysowany jest niebieski obwód na 4.7.a.



Rys. 4.7. Proces kalibracji: a) widok wyjściowy dla użytkownika, b) obraz w formacie HSV, c) maska

Obraz połączonych masek jest ostatecznym wynikiem detekcji dłoni w ramce wideo. Na podstawie tej maski, na ramce wyjściowej, która nie podlegała żadnym modyfikacjom rysowany jest obwód wyznaczonego obszaru dłoni. Użyta do tego została funkcja z biblioteki OpenCV cv2.findContours(). Operacja ta pozwala użytkownikowi określić, czy układ dłoni na ekranie jest odpowiedni oraz czy cała dłoń została wykryta. Dzięki temu może on zdecydować w odpowiednim momencie kiedy zakończyć kalibrację. Na 4.8 przedstawiono okno interfejsu użytkownika podczas kalibracji. Na zielono widać punkty do pobierania wartości progowych. Ręka użytkownika jest ułożona w taki sposób, aby każdy punkt kalibracyjny znajdował się na niej. Wraz ze zmianą położenia ręki zmienia się niebieski obrys. Żeby kalibracja została przeprowadzona w jak najlepszy sposób niebieska obwódka powinna okrywać całą dłoń i nie pokrywać ewentualnych cieni znajdujących się na obrazie. W lewym dolnym rogu znajduje się miejsce, gdzie wyświetlany jest komunikat: “CALIBRATION ONGOING ...“ oznaczający, że aplikacja jest w stanie kalibracji.



Rys. 4.8. Okno interfejsu użytkownika podczas kalibracji

W momencie, gdy użytkownik uzna, że aplikacja została skalibrowana poprawnie i obszar zaznaczany przez program określa dłoń w poprawny sposób, powinien zaznaczyć pole określające stan aplikacji jako skalibrowany. Gdy zostanie to już zrobione aplikacja wchodzi w stan detekcji. Punkty kalibracyjne znikają z widoku. Wartości progowe H oraz S zostają zapisane z chwili, w której użytkownik zmienia stan aplikacji i na podstawie nich wykrywana jest dłoń. Następnie ramka przekształcana jest na czarno-białą maskę za pomocą funkcji pogującej, gdzie kolorem białym wyznaczona jest dłoń, a kolorem czarnym tło. Taka maska w kolejnym etapie jest zmniejszana do wymiarów 60×100 pikseli. Z tak przygotowanego zdjęcia wyliczany jest histogram zorientowanych gradientów, na podstawie którego algorytm wektorów nośnych dopasowuje go do jednej z wyuczonych wcześniej klas. Cyfra jaką program rozpoznał zwracana jest odbiorcy w interfejsie użytkownika. Na 4.9 przedstawiono okno interfejsu

użytkownika w momencie kiedy aplikacja jest w stanie predykcji. Jak widać nie ma już na obrazie widocznych punktów kalibracyjnych. Obrys dłoni nadal jest widoczny. Można zauważyć że, nawet podczas przemieszczania dłoni względem kamery oraz zmieniając rodzaj gestu zarys podąża za położeniem dłoni. Wynika to z wcześniej poprawnie wykonanej kalibracji. W prawym dolnym rogu okna wyznaczone jest odpowiednie miejsce, w którym aplikacja wypisuje cyfrę jaką udało jej się przydzielić do wyuczonej klasy.

Rys. 4.9. Okno interfejsu użytkownika w momencie klasyfikacji gestu

Aplikacja napisana w skryptowym języku Python za pomocą funkcji kalibracji pozwala na wykrycie koloru skóry dłoni na ramce wideo. Następnie używając wartości progowych określonych podczas kalibracji wyznaczane są maski zawierające rozpoznaną dłoń. Taka czarno biała maska podlega zamianie na wektor wartości histogramu zorientowanych gradientów, po czym zostaje dopasowana do wyuczonych wcześniej klas SVM.

5. Analiza wyników

Wyniki dla aplikacji statycznej oraz rozpoznającej gesty w czasie rzeczywistym należy opisać osobno. Program statyczny posłużył jako model testowy, w którym dobierano parametry histogramu zorientowanych gradientów oraz maszyny wektorów nośnych w taki sposób, aby predykcja zadziałała najskuteczniej. Pomimo że obie aplikacje ostatecznie używały tych samych parametrów wyniki dla nich różnią się. Zbiór danych wejściowych stanowi baza 1395 zdjęć, od 130 do 140 na każdy gest. Do trenowania maszyny wektorów nośnych zastosowano bazę 1300 zdjęć, po 130 zdjęć dla każdego gestu. Pozostałą część zdjęć użyto w aplikacji statycznie rozpoznającej gesty do dobrania parametrów oraz przetestowania algorytmu.

W programie statycznie rozpoznającym gesty dłoni dobierano wielkość komórki oraz bloku dla histogramu zorientowanych gradientów.

Rozmiar komórki	Rozmiar bloku	Rozmiar nadmiarowości	Sprawność klasyfikatora
10×10	20×20	10×10	20.48%
5×5	20×20	10×10	78.31%
10×10	40×40	20×20	93.98%

Tabela 5.1. Rozmiary bloków oraz nadmiarowości z odpowiadającymi im sprawnościami klasyfikatora

Dla obrazu o wymiarach 60×100 pikseli, gdy rozmiar bloku i nadmiarowości wynosiły odpowiednio 20×20 oraz 10×10 pikseli, sprawność klasyfikacji wyniosła 20.48%. Przy tych parametrach zmniejszając rozmiar komórki do 5×5 pikseli sprawność wyniosła 78.31%. Najlepszą sprawność predykcji otrzymano dla rozmiaru bloku 40×40, wielkości komórki 10×10 oraz nadmiarowości 20×20 pikseli. Dla testowej bazy wynoszącej 94 obrazy różnych gestów dokładność wyniosła 93.98%.Wobec tego ostatecznie parametry histogramu zorientowanych gradientów wyniosły:

- rozmiar bloku: 40×40 pikseli,
- rozmiar nadmiarowości: 20×20 pikseli,
- rozmiar komórki: 10×10 pikseli.

Powyższe badania zostały przeprowadzone dla stałych maszyny wektorów nośnych odpowiednio równających się:

– stała C: 15.5,

– γ: 0.50625.

Manipulacja parametrem C w tym przypadku nic nie zmieniała. Natomiast zmniejszając stopniowo parametr γ skuteczność poprawiała się. W momencie gdy γ wyniosła 0.3 dokładność osiągnęła 94.62%, dalsze zmniejszanie parametru nie wnosiło żadnych zmian. Tabela 5.2 przedstawia tablice pomyłek [9] dla klasyfikatora SVM.

		klasa rzeczywista									
klasa predykowana		**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
	0	9	0	0	0	0	0	0	0	0	0
	1	0	2	0	0	0	0	0	0	0	0
	2	0	0	11	2	1	0	0	0	0	0
	3	0	0	1	3	0	0	0	0	0	0
	4	0	0	0	4	6	0	0	0	0	0
	5	0	0	0	0	0	5	0	0	0	0
	6	0	0	0	0	0	0	14	0	0	0
	7	0	0	0	0	0	0	0	11	0	0
	8	0	0	0	0	0	0	0	0	11	0
	9	0	0	0	0	0	0	0	0	0	15

Tabela 5.2. Tablica pomyłek dla klasyfikatora SVM

Dodatkowo w celu porównania innych technik uczenia maszynowego, w programie statycznie rozpoznającym gesty zastosowano maszynę wektorów nośnych z liniowym jądrem oraz algorytm k-średnich. W obu wypadkach wyniki były co najmniej nie zadowalające. Maszyna wektorów nośnych z jądrem RBF dla tego projektu sprawdziła się najlepiej. W tabeli 5.3 przestawiono sprawność klasyfikatorów.

Algorytm	Sprawność klasyfikatora
SVM jądro RBF	94.98%
SVM jądro liniowe	73.64%
algorytm k-średnich	11.49%

Tabela 5.3. Wyniki sprawności poszczególnych klasyfikatorów

Dobrane powyżej parametry zostały zastosowane w aplikacji rozpoznającej gesty w czasie rzeczywistym. Pomimo skrupulatnie dobranych parametrów przynoszących bardzo dobre efekty dla aplikacji statycznej, predykcja dla programu działającego w czasie rzeczywistym jest o wiele gorsza. W dużej mierze wpływ na to ma przeprowadzenie poprawnej kalibracji. Do prawidłowej kalibracji aplikacji potrzebne jest bardzo dobre oświetlenie oraz kamerka internetowa o wysokiej rozdzielczości. Światło rzucane w

kierunku dłoni nie powinno być zbyt intensywne, gdyż zmienia ono kolor jaki jest odbierany przez kamerkę. Dodatkowo może to powodować pojawienie się cieni, które często są błędnie interpretowane przez klasyfikator. Co więcej w przypadku oddalania dłoni od obiektywu kamery przy zbyt intensywnym świetle ręka odbija światło w inny sposób niż podczas kalibracji, przez co przy predykcji jest ona gubiona przez program. Gdy oświetlenie jest zbyt słabe, piksele w obrębie dłoni nie posiadają na tyle odróżniającej się luminancji, aby została celnie odseparowane od tła. Kolejnym problemem przy zbyt słabym naświetleniu jest pojawiający się szum zniekształcający wydzielony obszar dłoni co prowadzi w następnych krokach do błędnej detekcji gestu. Kamera internetowa jaka została użyta w tym przypadku to kamera firmy Logitech model HD PRO WEBCAM C920, pozwalająca na przesyłanie strumieniowe wideo z jakością Full HD do 1920×1080 pikseli za pomocą kompresji $H.264$ z dodatkową funkcją automatycznej korekcji ekspozycji przy słabym oświetleniu. Podejmowano także próby pracy programu z kamerami o słabszych parametrach, jednak poprawna detekcja w tych przypadkach była praktycznie niemożliwa z powodu zaszumienia obrazu.

Rys. 5.1. Przykłady złej kalibracji lub detekcji: a) kamera złej jakości przy słabym oświetleniu, b) kamera złej jakości przy zbyt dużym naświetleniu, c) kamera firmy Logitech przy słabym oświetleniu

Na 5.1 przedstawiono ilustracje gdy aplikacja nie działa poprawnie lub nie ma możliwości poprawnego działania. Zdjęcie 5.1.a przedstawia przypadek, w którym używana jest kamera o jakości HD 1280×720 pikseli z niewystarczającym oświetleniem. Można na nim zauważyć że, że obraz jest bardzo zaszumiony. Aplikacja nie jest w stanie objąć całej dłoni. W środku dłoni znajdują się piksele o luminancji nieodpowiadającej pozostałym. Przez co, niebieski kontur nie jest jednolity. Przy takim oświetleniu poprawna kalibracja jest niemożliwa. Dlatego też, po takiej kalibracji poprawne sklasyfikowanie gestu jest praktycznie niemożliwe. W wyniku błędnej kalibracji, algorytm bardzo często podejmuje błędne

decyzje, nawet przy najprostszym geście żerożeprezentowanym za pomocą zaciśniętej dłoni. Na 5.1.b pokazuje kalibrację przeprowadzoną na tej samej kamerce internetowej co w przykładzie poprzednim. Jednak podczas kalibracji światło padające na dłoń jest zbyt intensywne. Można zauważyć, pomimo że wszystkie punkty kalibracyjne znajdują się na ręce aplikacja nie jest w stanie wyznaczyć jej poprawnego konturu. Spowodowane jest to tym, że w większości wartości pikseli przedstawiają kolor biały. Funkcje wyznaczające wartości progowe dla parametrów H i S po dodaniu wartości korekcyjnych, dobranych w warunkach z poprawną ilością światła, zwracają wartości progowe, dla których tło mieści się w tym przedziale. 5.1.c przedstawia aplikacje już w momencie predykcji. Kamera jaka została do tego użyta przesyła wideo w jakości Full HD. Widać znaczącą poprawę jakości oraz pokaźną redukcje szumu jaka znajdowała się na ramce wideo. Jednak pomimo tego klasyfikacja gestu nie powiodła się. Problemem w tym przypadku jest zbyt słabe światło i nie doświetlona dłoń. W środku dłoni i na bokach palców pojawiają się cienie nie pasujące do wcześniej wyznaczonych wartości progowych koloru oraz nasycenia. Wyznaczony w ten sposób kontur dłoni nie odpowiada w pełni wyuczonemu modelowi liczby pięć. Kciuk posiada zbyt duży cień przez co aplikacja rozpoznała go jako gest numer cztery.

Aby kalibracja została przeprowadzona w poprawny sposób, muszą zostać spełnione takie warunki jak:

- zastosowana kamerka internetowa o wysokiej jakości nagrywania (Full HD),
- światło powinno być rozproszone i skierowane w kierunku dłoni,
- odpowiednio doświetlona dłoń, bez cieni które mogą zniekształcić jej obwód,
- natężenie oraz rodzaj oświetlenia nie może zmieniać się podczas używania aplikacji.

Gdy to nastąpiło aplikacja pracująca w czasie rzeczywistym przyniosła słabsze lecz wciąż zadowalające rezultaty. Ogólna trafność dla wszystkich gestów wahała się w przedziale 70%. Jednak rozpatrując sprawność modelu wektorów nośnych dla poszczególnych klas, różniły się one między sobą. Największy problem aplikacja miała z odpowiednim rozpoznaniem gestu numer siedem. Zamiast siódemki program często rozpoznawał gest jako cyfrę zero lub dziewięć. Kolejnym problemem dla aplikacji były gesty przedstawiające cyfry od jeden do pięciu. Program czasami mieszał je między sobą. Miedzy nimi najczęściej program miał problem z gestem przedstawiającym cyfrę trzy. Często zamiast niej wykrywał cyfrę dwa lub cztery. Takie błędy spowodowane są tym, że te gesty są do siebie bardzo podobne lub użytkownik pokazuje je w nieodpowiedni sposób. Kolejnym powodem są pojawiające się cienie w kadrze często wychwytywane jako dodatkowy palec lub w przeciwną stronę zamazując widok któregoś z palcy. Inną przyczyną nierozpoznania niektórych gestów były złączone palce. W takim przypadku po zamianie ramki wideo na maskę, algorytm nie jest w stanie kreślić ile palców znajduje się w środku. Taki przypadek miał miejsce przy gestach przedstawiających liczby od dwóch do pięciu. Gdy użytkownik złączył palce podczas predykcji gesty były rozpoznawane jako znaki jeden lub zero. Natomiast najlepszą predykcje odnotowano dla gestu numer sześć i osiem. Są one na tyle charakterystyczne, że program radził sobie z ich klasyfikacją z skutecznością około 80%. Testowanie aplikacji rozpoznającej gesty w

czasie rzeczywistym przeprowadzano na pięciu użytkownikach, których zdjęcia rąk nie znalazły się w bazie trenującej oraz z powodu brakującej ilości osób pięć użytkowników, których zdjęcia dłoni znajdują się w bazie wejściowej maszyny wektorów nośnych.

Istnieją zaimplementowane inne algorytmy do rozpoznawania gestów. Niniejszy projekt porównano do dwóch już istniejących prac: *Real-Time Hand Gesture Recognition Using Finger Segmentation* [10] oraz *Hand Gesture Recognition Based on Dimensionality Reduction of Histogram of Oriented Gradients* [11]. W obu przypadkach posłużono się podobnymi metodami takimi jak histogram zorientowanych gradientów lub maszynę wektorów nośnych. Główny zamysł prac jest bardzo podobny, jednak różnią się one metodą implementacji oraz sposobami radzenia sobie z problemami selekcji dłoni od tła oraz detekcji rodzaju gestu. Najlepsze wyniki osiąga projekt *Real-Time Hand Gesture Recognition Using Finger Segmentation* [10]. Spowodowane jest to bardzo dobrze dopracowaną metodą lokalizacji dłoni, jak i bardziej dokładną metoda wykrywania gestu. W tym przypadku wykryta dłoń także zamieniana jest na biało-czarną maskę jednak wykryta ręka podlega segmentacji na każdy palec osobno oraz rdzeń dłoni. Projekt ten osiągnął sprawność $96,69\%$ dla statycznej bazy testującej zasilonej 1300 zdjęciami gestów. Kolejny projekt używa takiej samej metody rozpoznawania gestów jak niniejsza praca. Jednak rozpoznaje ona wyłącznie w sposób statyczny. Dla algorytmu HOG i klasyfikatora SVM sprawność sprawność wyniosła $93,84\%$. Porównując obie metody do niniejszej pracy, przyniosły one bardzo podobne rezultaty dla gestów wykrywanych statycznie.

Proces tworzenia aplikacji podzielony na dwa etapy pozwolił na dostosowanie odpowiednich parametrów dla algorytmów zastosowanych w tym projekcie. Pomimo to dla każdej z tych części wyniki różniły się. Spowodowane było to warunkami w jakich używane były aplikacje. Mimo wszystko algorytmy sprawdziły się w roli rozpoznawania gestów dłoni na podstawie sekwencji wideo.

6. Podsumowanie

Celem projektu magisterskiego było zaimplementowanie algorytmu rozpoznawania gestów dłoni na podstawie sekwencji wideo. Gesty jakie wykrywa aplikacja to znaki używane na co dzień w Chinach, przedstawiające cyfry w zakresie od zera do dziewięciu. Do ich predykcji zastosowano technikę uczenia maszynowego SVM. Dodatkowo użyto histogram zorientowanych gradientów, w celu ekstrakcji cechy obrazu, zapisanego następnie w jednowymiarowym wektorze. Wektory te następnie wykorzystywane są w SVM do trenowania klasyfikatora jak i podlegają predykcji. Bazę danych treningowych do wyuczenia algorytmu stanowi zestaw samodzielnie zebranych zdjęć gestów. Zestaw danych uczących zawiera po 130 zdjęć dla każdego gestu. Proces tworzenia algorytmu podzielono na trzy etapy. Pierwszy to zbieranie danych wejściowych oraz analiza ich. Kolejną częścią było zaimplementowanie skryptu w języku Python, który w sposób statyczny rozpoznaje rodzaj gestu na fotografii. Miało to na celu dobranie parametrów przynoszących jak najlepsze rezultaty dla algorytmów histogramu zorientowanych gradientów oraz maszyny wektorów nośnych. Po selekcji odpowiednich wartości zostały one zastosowane w docelowej aplikacji rozpoznającej gesty w czasie rzeczywistym. Skuteczność aplikacji statycznej wyniosła $93,98\%$, a docelowej około 70%. Program można rozwinąć o funkcję, która będzie używać w dany sposób sklasyfikowanego ciągu cyfr. Na przykład może ona posłużyć do wprowadzania hasła dostępu lub do wybierania numeru telefonu. Dodatkowo poprawienie jakości kalibracji udoskonaliłoby predykcje w przypadku aplikacji rozpoznającej gesty w czasie rzeczywistym.

Bibliografia

[1] Frank Y. Shih. *Image processing and pattern recognition*. Wydawnictwo Wiley.

[2] Włodzimierz Kwiatkowski. *Metody automatycznego rozpoznawania wzorców*. Warszawa: Wydawnictwo BEL Studio, 2007.

[3] Skorzybut Michał Krzyśko Mirosław Górecki Tomasz Wołyński Waldemar. *Systemy uczące się, rozpoznawanie wzorców, analiza skupień i redukcja wymiarowości*. Warszawa: Wydawnictwa Naukowo-Techniczne, 2009.

[4] Satya Mallick. „Histogram of Oriented Gradients". W: *Learn OpenCV* (2016).

[5] P. Patil B. Almeida N. Chettiar J. Babu. „Offline Signaure Recognition System using Histogram of Oriented Gradients". W: *International Conference on Advances in Computing, Communication and Control (ICAC3)* (2017). DOI: *10.1109/ICAC3.2017.8318766*.

[6] *Python 3.6.5 documentation*. Spraw. tech. www.docs.python.org/3.6/. Python Software Foundation, 2018.

[7] OpenCV. „OpenCV modules". W: *OpenCV* (23.01.2018).

[8] Chih-Chung Chang i Chih-Jen Lin. „LIBSVM: A Library for Support Vector Machines". W: *LIBSVM* (4.03.2013).

[9] Satya Mallick. „Ocena modelu, testowanie klasyfikatora". W: *Uniwersytet Śląski* (2016).

[10] Jianning Liang Jing Zhangand Yu-Bo Yuan Zhi-hua Chen Jung-Tae Kim. „Real-Time Hand Gesture Recognition Using Finger Segmentation". W: *The Scientific World Journal* (2014).

[11] Mahmoud I. Abdalla1 Rania A. Elsayed Mohammed S. Sayed. „Hand Gesture Recognition Based on Dimensionality Reduction of Histogram of Oriented Gradients". W: *Electronics and Communications Engineering, Zagazig University* (2017).