

Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH



Instytut Informatyki

Praca dyplomowa inżynierska

na kierunku Informatyka
w specjalności Inżynieria systemów informatycznych

Zastosowanie analizy obrazu twarzy do sterowania aplikacją na
urządzeniach z systemem Android

Adamski Maciej

Numer albumu 300184

promotor
prof. dr hab. inż. Przemysław Rokita

WARSZAWA 2022

Zastosowanie analizy obrazu twarzy do sterowania aplikacją na urządzeniach z systemem Android

Streszczenie.

Celem pracy inżynierskiej było stworzenie aplikacji na urządzenia z systemem *Android*, która przy pomocy analizy twarzy będzie reagowała na gesty użytkownika takie jak mruganie czy ruch gałkami ocznymi.

Został przygotowany zestaw algorytmów do detekcji poszczególnych fragmentów twarzy, które następnie były testowane pod kątem skuteczności detekcji i złożoności czasowej. Do implementacji wybranych metod przetwarzania obrazu zostały użyte biblioteki *OpenCV* oraz *Dlib*. Najlepsze rozwiązania z poszczególnych grup zostały wykorzystane w finalnym projekcie oprogramowania. Proces detekcji twarzy odbywa się przy pomocy algorytmu opartego na *Histogramach zorientowanych gradientów*, natomiast znaczniki są wykrywane dzięki metodzie *Kazemi. Punkty charakterystyczne twarzy* pozwalają określić położenie na zdjęciu oczu, a także stwierdzić czy są one zamknięte przy użyciu współczynnika *Eye Aspect Ratio*. Wykorzystywane jest *progowanie na podstawie dystrybunaty* celem wyznaczenia środka źrenicy.

Końcowym efektem pracy dyplomowej jest prosta aplikacja, która prezentuje działanie analizy obrazu oraz reaguje na ruch oczami w poziomie czy mruganie użytkownika. Wykrycie takich gestów przedstawione było w zrozumiałej formie przez wyświetlanie komunikatów oraz przesuwanie obrazów w prostej galerii zdjęć.

Praca pomogła zapoznać się z wytwarzaniem oprogramowania użytkowego na systemy *Android*. Przyniosła również dużo wiedzy z zakresu przetwarzania obrazu oraz sieci neuronowych i uczenia maszynowego. Pozwoliła też poznać dwie popularne biblioteki z dziedziny widzenia komputerowego.

Słowa kluczowe: Cyfrowe przetwarzanie i analiza obrazów, Detekcja twarzy, Śledzenie oczu, Programowanie aplikacji mobilnych, Kontrola aplikacji z wykorzystaniem obrazów

Facial image based application control on Android devices

Abstract.

The main goal of this thesis was to create the application for the devices that run the Android operating system, which with the aid of face analysis, would be able to react to the user gestures such as blinking or eyeball movement.

To detect specific parts of the face there was prepared a set of algorithms. Each of them was further tested in terms of detection effectiveness and time complexity. The chosen image processing methods were implemented with the usage of *OpenCV* and *Dlib* libraries. The best solutions from each group were used in a final software project. The face detection process is performed with the aid of an algorithm based on a *Histogram of Oriented Gradients*, whereas the facemarks are being detected with the *Kazemi* method. The face pointers allow determining the eyes' location on the photo as well as ascertain whether they are closed with the *Eye Aspect Ratio* coefficient. There is used the *thresholding by cumulative distribution function* in order to find the center of the pupil.

The final result of this thesis is a simple application that presents the working of the image analysis, reacts to horizontal eyeballs movement, and to blinking. Those gestures detection were shown to the user in a readable way as a notification and also in the form of moving images in the simple photo gallery.

This work was a helpful introduction to application development on Android devices. It also brought a significant amount of knowledge related to image processing, neural networks, and machine learning. Moreover, it allowed getting to know the two popular libraries from the domain of computer vision.

Keywords: Digital image processing and analysis, Face detection, Eye gaze tracking, Mobile application development, Image based application



.....
miejscowość i data

.....
imię i nazwisko studenta

.....
numer albumu

.....
kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanego z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

.....
czytelny podpis studenta

Spis treści

1. Wstęp	11
1.1. Cel pracy	11
1.2. Motywacja pracy	11
1.3. Etapy pracy	12
2. Stos technologiczny	13
2.1. System operacyjny Android	13
2.2. Biblioteki	13
2.2.1. OpenCV	13
2.2.2. OpenCV-contrib	13
2.2.3. Dlib	13
2.2.4. CameraX	13
2.3. Języki programowania	14
2.4. JNI	14
3. Sposób badania algorytmów	15
3.1. Zbiór danych	15
3.2. Urządzenie do testów	15
4. Detekcja twarzy	16
4.1. Algorytmy detekcji twarzy	16
4.1.1. Klasyfikator kaskadowy	16
4.1.1.1. Haar	16
4.1.1.2. Local binary patterns	16
4.1.2. Histogram zorientowanych gradientów	17
4.1.3. CNN + MMOD	18
4.1.4. Głębokie sieci neuronowe	18
4.2. Filtrowanie zwracanych obszarów twarzy	18
5. Porównanie algorytmów detekcji twarzy	20
5.1. Odrzucenie algorytmu CNN+MMOD	20
5.2. Testowanie na statycznych zdjęciach	20
5.2.1. Oczekiwany wynik	21
5.2.2. Warunki testowania	21
5.2.3. Badanie skuteczności detekcji	21
5.2.4. Badanie szybkości detekcji	24
5.2.5. Wpływ rozdzielczości na szybkość algorytm <i>DNN Caffe</i>	25
5.2.6. Precyzja detekcji algorytmu <i>DNN Caffe</i> zależnie od sposobu filtracji .	26
5.3. Testowanie na obrazie z kamery na żywo	26
5.3.1. Badanie skuteczności detekcji	27
5.3.2. Badanie szybkość detekcji	27

5.4. Wybór algorytmu detekcji twarzy	27
6. Detekcja znaczników twarzy	29
6.1. Algorytmy detekcji znaczników twarzy	29
6.1.1. Local binary features	29
6.1.2. Kazemi	29
7. Porównanie algorytmów detekcji znaczników twarzy	30
7.1. Testowanie na statycznych zdjęciach	30
7.1.1. Usunięcie części zdjęć ze zbioru danych	30
7.1.2. Badanie skuteczności detekcji	30
7.1.3. Badanie szybkości detekcji	31
7.2. Testowanie na obrazie z kamery na żywo	32
7.2.1. Badanie skuteczność detekcji	32
7.2.2. Badanie szybkość detekcji	32
7.3. Wybór algorytmu detekcji znaczników twarzy	33
8. EAR - Eye Aspect Ratio	34
8.1. Wyznaczanie współczynnika EAR	34
8.2. Zasada działania EAR w kontekście mrugania i określenia czy oko jest otwarte/zamknięte	34
8.3. Ustalenie progu EAR	35
8.3.1. EAR dla statycznych zdjęcia	35
8.3.2. EAR na obrazie z kamery	36
8.4. Wnioski	37
9. Detekcja oczu	38
9.1. Algorytmy detekcji oczu	38
9.1.1. Klasyfikator kaskadowy Haar	38
9.1.2. Znaczniki twarzy wokół oczu	38
9.2. Filtrowanie wyników detekcji oczu metodą Haar	39
9.3. Obcięcie obszaru twarzy dla detekcji oczu metodą Haar	39
9.4. Dostosowanie rozmiaru zwracanego obszaru oczu dla znaczników twarzy	41
10. Porównanie algorytmów detekcji oczu	42
10.1. Testowanie na statycznych zdjęciach	42
10.1.1. Oczekiwany wynik	42
10.1.2. Warunki testowania	42
10.1.3. Badanie skuteczności detekcji	43
10.1.4. Badanie szybkości detekcji	44
10.2. Testowanie na obrazie z kamery na żywo	45
10.2.1. Badanie skuteczność detekcji	45
10.2.2. Badanie szybkość detekcji	45
10.3. Wybór algorytmu detekcji oczu	46

11.Detekcja źrenic	47
11.1.Algorytmy detekcji źrenic	47
11.1.1.Algorytm CDF (Cumulative Distribution Function)	47
11.1.1.1.Kroki algorytmu	47
11.1.1.2.Wynik kolejnych kroków algorytmu	48
11.1.2.Algorytm PF (Projection Function)	48
11.1.2.1.Funkcja projekcji	49
11.1.2.2.Kroki algorytmu	50
11.1.2.3.Wynik kolejnych kroków algorytmu	50
11.1.3.Algorytm EA (Edge Analysis)	51
11.1.3.1.Kroki algorytmu	51
11.1.3.2.Wynik kolejnych kroków algorytmu	51
12.Porównanie detekcji źrenic	52
12.1.Testowanie na statycznych zdjęciach	52
12.1.1.Oczekiwany wynik	52
12.1.2.Zbierane dane	52
12.1.3.Wybór funkcji projekcji	52
12.1.4.Badanie skuteczności detekcji	53
12.1.5.Badanie szybkości detekcji	54
12.2.Testowanie na obrazie z kamery	55
12.2.1.Badanie skuteczność detekcji	55
12.3.Wybór algorytmu detekcji źrenic	56
13.Detekcja mrugania	57
13.1.Algorytm detekcji mrugania	57
13.2.Testowania detekcji mrugania na obrazie na żywo z kamery	57
14.Detekcja ruchu gałek ocznych	59
14.1.Algorytm detekcji ruchu oczu	59
14.2.Testowania detekcji ruchu oczu na obrazie na żywo z kamery	59
15.Architektura	61
15.1.Wzorce projektowe	61
15.1.1.Wstrzykiwanie zależności	61
15.1.1.1.Przykład użycia	62
15.1.1.2.Wykorzystanie w projekcie	62
15.1.2.Obserwator	63
15.1.2.1.Wykorzystanie w projekcie	63
16.Podsumowanie	64
16.1.Wykonane prace	64
16.2.Zdobyta wiedza i wyciągnięte wnioski	64
16.3.Możliwość rozwoju i dalszych badań	65

Bibliografia	67
Spis rysunków	70
Spis tabel	71

1. Wstęp

1.1. Cel pracy

Celem niniejszej pracy dyplomowej było zaprojektowanie i implementacja aplikacji na urządzenia z systemem Android, która wykorzystując analizę twarzy miała reagować na gesty użytkownika. By to osiągnąć należało zaimplementować i przetestować wybrane metody przetwarzania obrazów do detekcji twarzy i wybranych jej fragmentów. Następnie, na podstawie wyników testów wybrać najlepsze, które miały zostać wykorzystane do sterowania prostą aplikacją w systemie Android prezentującą działanie detektorów i ich przykładowe zastosowanie.

1.2. Motywacja pracy

W dzisiejszych czasach coraz większą popularność zyskuje wirtualna rzeczywistość, bezdotykowa obsługa urządzeń czy dostosowanie interfejsu aplikacji do osób niepełnosprawnych. Rozwiązania niegdyś znane wyłącznie z filmów, które wydawały się niemożliwe do stworzenia nie są już pieśnią przeszłości.

Dla ludzi o różnych nieprawidłowościach ruchowych kończyn górnych użytkowanie w codziennym życiu takich urządzeń jak komputer czy telefon prawdopodobnie jest problematyczne. Zapewnienie im możliwości sterowania interfejsami tych narzędzi za pomocą gestów np. twarzy lub komendami głosowymi może być pożądaną przez nich alternatywą.

W dziedzinie sterowania głosem istnieją bardzo dobrze rozwinięte systemy takie jak Siri, Alexa czy asystent Google. Powszechnie stosowane są w urządzeniach przenośnych i systemach inteligentnego domu. Natomiast analiza obrazu i gestów nie jest już tak szeroko rozpowszechniona, a również niesie duże możliwości w aspekcie bezdotykowego sterowania urządzeniami.

Zostało stworzonych już wiele gier wykorzystujących wirtualną rzeczywistość, w której kamera podąża za ruchami głowy gracza, a interakcja ze światem gry wykonywana jest przez gesty rękami. Konstruktory takich rozwiązań cały czas dążą do jeszcze większej immersyjności. Wykorzystanie emocji czy analizy twarzy gracza pozwoliłaby mu się mocniej zanurzyć w świat wirtualny, a granica między rzeczywistością jeszcze bardziej by się zacierała.

Czytanie artykułów czy przeglądanie galerii zdjęć na urządzeniu mobilnym wymaga ciągłych ruchów palcem po ekranie. Byłoby wielkim uproszczeniem jakby tekst pisany przesuwał się automatycznie wraz ze zmianą położenia oczu w czasie czytania kolejnych linii.

1.3. Etapy pracy

Pierwszym etapem pracy nad projektem był przegląd znanych algorytmów oraz bibliotek mając na uwadze następujące zastosowania:

- detekcja twarzy
- detekcja oczu
- detekcja żrenic
- detekcja punktów charakterystycznych twarzy

Wybór rozwiązań był nie tylko zależny od ich zalet typowo funkcjonalnych (skuteczność, szybkość itp.), ale także od tego jak problematyczna jest integracja z docelowym środowiskiem, jakości dokumentacji czy popularności w społeczności programistów.

Następnie należało przetestować i porównać ze sobą algorytmy w odpowiednio podzielnych grupach. Zbierane były dane o ich skuteczności oraz szybkości działania. Badania były prowadzone z użyciem zbioru zdjęć, a także na podstawie obrazu na żywo z kamery. Drugi przypadek miał bardzo istotny wpływ na końcowy dobór algorytmów, ponieważ projekt z założenia miał działać w czasie rzeczywistym na urządzeniach mobilnych. Na podstawie tak przeprowadzonych testów z każdej grupy wybierana była jedna z metod, która później była wykorzystywana w finalnej wersji projektu.

Następnym etapem było stworzenie aplikacji wykorzystującej przygotowany zestaw algorytmów. Demonstruje ona zarówno ich przykładowe zastosowanie w kontekście sterowania interfejsem programu, ale także prezentuje na żywo analizę twarzy oraz pozwala wykonać opisane badania na zestawie zdjęć.

Końcowo, celem weryfikacji prawidłowego działania projektu przetestowano gotową aplikację w różnych warunkach codziennego użytkowania urządzenia.

2. Stos technologiczny

2.1. System operacyjny Android

Docelowym systemem operacyjnym, na który przygotowywany był projekt jest Android. Opiera się on na jądrze Linux przystosowanym do urządzeń mobilnych. Aktualnie stosowany jest też w wielu innych urządzeniach elektronicznych takich jak telewizory, systemy audi czy nawet komputery pokładowe w samochodach.

2.2. Biblioteki

2.2.1. OpenCV

Otwarta biblioteka widzenia komputerowego (ang. Open Source Computer Vision Library) [1] - jedna z najbardziej rozpowszechnionych bibliotek służąca do przetwarzania obrazu i widzenia komputerowego. Napisana z użyciem języka C/C++ co zapewnia jej szybkie działanie i możliwość wykorzystania niskopoziomowych mechanizmów. Z rozwiązań tego pakietu można korzystać na wielu systemach operacyjnych, a dzięki wiązaniom do popularnych języków, kod może być pisany również w Javie czy Pythonie. Dzięki zastosowaniu takich mechanizmów sprzętowych jak CUDA [2] część obliczeń jest przenoszona z jednostek arytmetycznych na akceleratory graficzne.

2.2.2. OpenCV-contrib

Zbiorcza nazwa dla dodatkowych modułów [3] biblioteki OpenCV. Nie są one zawarte w wersji stabilnej API głównej pakietu. Jednak są to rozwiązania, które po szeregu testów i pewnym okresie przejściowym mogą zostać włączone do podstawowego modułu. Znajdują się tam rozwiązania do rozpoznawania twarzy [4], sieci konwolucyjnych czy detekcji obiektów.

2.2.3. Dlib

Wieloplatformowy zestaw narzędzi [5] napisany z wykorzystaniem języka C++. Początkowo głównym obszarem zastosowań było uczenie maszynowe, lecz z czasem zaczęto rozwijać także sektor sieciowy, przetwarzania obrazów czy operacje numeryczne. Rozpowszechniona jest na licencji otwartego oprogramowania i stale rozwijana.

2.2.4. CameraX

Jeden z modułów [6] tzw. *Android Jetpack* [7]. Biblioteka pozwalająca na łatwą obsługę kamer zamontowanych w urządzeniu, na którym uruchamiany jest program. Pozwala na przechwytywanie na żywo obrazu, a następnie jego analizę klatka po klatce oraz zapisywanie uzyskanego nagrania wideo i zdjęć.

2.3. Języki programowania

W projekcie wykorzystywane są dwa języki programowania:

- Java - większość projektu napisane jest w tym języku
- C++ - wykorzystywany do wołania metod biblioteki Dlib przy pomocy JNI (rozdz. 2.4)

Dodatkowo, szablony aplikacji Android napisane są z użyciem języka znaczników XML.

2.4. JNI

Natywny interfejs programistyczny dla języka Java (ang. Java Native Interface) [8] - jest to interfejs pozwalający uruchamiać natywne programy i biblioteki w innym języku i przeznaczone na inne systemy w wirtualnej maszynie Java. W projekcie pracy dyplomowej pozwala to wykorzystać bibliotekę Dlib, która nie posiada gotowych wiązań do Javy. W tym przypadku kod wywołujący metody tego pakietu jest napisany w języku C++, który następnie wywoływany jest przez obiekty natywne w maszynie wirtualnej.

3. Sposób badania algorytmów

Algorytmy były badane i porównywane przy pomocy statycznych zdjęć z przygotowanego zbioru danych, jak i wykorzystując obraz na żywo z przedniej kamery urządzenia.

3.1. Zbiór danych

Na potrzeby badań i porównania algorytmów przygotowany został zbiór danych składający się z 80 zdjęć zawierających twarze.

Źródła zdjęć:

- 50 zdjęć wybranych z datasetu *Young and Old Images Dataset* [9]
- 30 zdjęć wykonanych przez autora pracy dyplomowej

Został on przygotowany w taki sposób, żeby zawierał różnicowane zdjęcia pod względem takimi jak: jakość obrazu, oświetlenie, powierzchnia zajmowana przez twarz, kolorystyka, płeć, kolor skóry, częściowe zakrycie twarzy, okulary czy odwrócona w bok głowa.

Wszystkie 80 zdjęć zostało opisanych ręcznie przez autora projektu na potrzeby badań, w szczególności: obszar twarzy, oczu czy środek źrenic.

3.2. Urządzenie do testów

Wszystkie testy zostały przeprowadzone na urządzeniu Huawei P30 Pro w normalnym trybie wydajności i bez włączonego oszczędzania energii.

4. Detekcja twarzy

W przetwarzaniu cyfrowym obrazu stosujemy technikę od ogółu do szczegółu. Przykładowo chcąc uzyskać barwę nadwodzia najpierw musimy wykryć samochód itp. W pracy dyplomowej by uzyskać dane dotyczące oczy czy znaczników potrzebujemy najpierw informacji o tym czy twarz występuje na zdjęciu i ewentualnie gdzie się ona na nim znajduje. Dlatego pierwszym etapem przetwarzania na potrzeby projektu jest detekcja ludzkiej twarzy. Z dostarczonego obrazu uzyskujemy informację o jej położeniu, a w następnych etapach operować tylko na wycinku zdjęcia.

4.1. Algorytmy detekcji twarzy

Na potrzeby projektu zostało zaimplementowane użycie pięciu algorytmów, których celem jest wykrycie twarzy na zdjęciu. Krótki opis poszczególnych metod znajduje się w następnych podrozdziałach.

4.1.1. Klasifikator kaskadowy

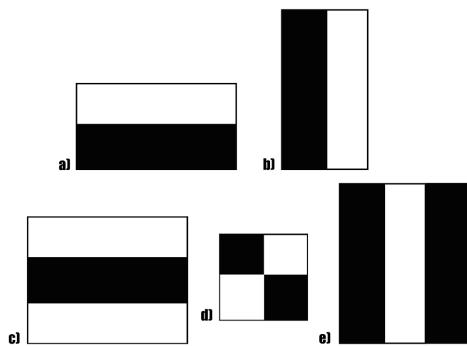
Cascade Classifier to jedno z podejść do zadania klasyfikacji obiektów. Kaskadowość tego rozwiązania przejawia się tym, że składa się on z łańcucha mniejszych klasifikatorów. Z danych wejściowych jednego może korzystać następnym jako dodatkowe źródło informacji i użyć je do własnej klasyfikacji. Z tego powodu kolejne elementy są bardziej zaawansowane i operują na większym zestawie danych. Dzięki swojej kaskadowej naturze modele takie mogą być lepiej trenowane i dawać lepsze rezultaty niż klasifikatory typu monolity.

Do ładowania i przetwarzania kaskadowych klasifikatorów w projekcie używany jest moduł *CascadeClassifier* [10] biblioteki *OpenCV*.

4.1.1.1. Haar Jednym z najbardziej znanych modeli klasyfikacji kaskadowej jest *Haar*, który został opisany po raz pierwszy w 2001 roku [11]. Może on być używany do klasyfikacji różnych obiektów, ale autorzy skupiali się głównie na detekcji twarzy. Algorytm [12] [13] [14] bazuje na podzieleniu zdjęcia na regiony i wykorzystaniu w każdej z nich pięciu cech krawędzi (patrz *rysunek 4.1*). Algorytm porównując jasność pikseli w białej i czarnej części stwierdza czy istnieją krawędzie lub linie. Cechy składające się tylko z dwóch regionów odpowiadają za wykrycie pionowych i poziomych krawędzi. Zestaw trzech za wykrycie linii. Natomiast kwadratowa cecha za zmiany przekątne. W dzisiejszych czasach *Haar* nie jest już tak często stosowany jak jeszcze parę lat temu.

Na potrzeby pracy dyplomowej Wykorzystywany jest model *Haarcascade Frontalface Default* [15] autorstwa Rainera Lienharta.

4.1.1.2. Local binary patterns Metoda ta porównuje piksele z ośmioma swoimi najbliższymi sąsiadami w ustalonej kolejności. Jeśli jasność głównego piksela jest większa



Rysunek 4.1. Obliczane cechy w modelu Haar. Źródło: [12]

niż porównywanego to na odpowiedniej pozycji 8-bitowej liczby wstawia 1, inaczej 0. Następnie z uzyskanych w ten sposób liczb tworzy histogram używany jako deskryptor cech. Takie dane mogą być użyte do uczenia maszynowego. [16]

Jest to metoda cechująca się wysoką szybkością działania i z tego powodu stosowana w systemach z ograniczonymi zasobami sprzętowymi. Niestety kosztem efektywności.

W projekcie stosowany jest model *LBP Cascade Frontalface* [17].

4.1.2. Histogram zorientowanych gradientów

Metoda *HOG* (*Histograms of Oriented Gradients*) [18] została opracowana kilkanaście lat temu przez Navneet Dalal i Bill Triggs celem detekcji ludzkiego ciała. Aktualnie, mimo upływu lat, jest wciąż szeroko wykorzystywana do klasyfikacji obrazów czy wykrywania twarzy.

Uzyskanie histogramu HOG składa się z kilku etapów. Metoda [19] [20] [21] ta bazuje na obliczeniu gradientów poziomych i pionowych. Możliwe jest to poprzez filtrowanie za pomocą odpowiedniego jądra lub przez operator Sobela [22]. Dla tak wyodrębnionych gradientów oblicza się ich długość i kierunek (kąt). Następnie dzielimy zdjęcie na obszary o wielkości 8×8 . Dla każdego regionu tworzymy jednowymiarowy wektor o 9 komórkach, w których będzie zapisany histogram HOG. Pola wektora odzwierciedlają kierunek gradientu i odpowiadają kolejnym wielokrotnościom kąta $\angle 20^\circ$. Wypełniamy go dodając do pól odpowiadającym danemu kątowi wartość gradientu kolejnych pikseli. Jeśli kierunek znajduje się pomiędzy dwoma kątami to wartość dzieli się zależnie od różnicy między dwiema komórkami. Celem wyeliminowania wpływu jasności i oświetlenia przeprowadza się normalizację wartości. Gdy obliczy się już histogram dla każdego regionu, łączy się je w wektor deskryptora cech HOG. Tak uzyskany wektor możemy wykorzystać jako dane uczące algorytmów klasyfikujących. W przypadku metody HOG często wykorzystuje się maszynę wektorów nośnych (SVM) [23].

Metoda *HOG* wykorzystywana w projekcie zaimplementowana jest w bibliotece *dlib*, która uczona była z użyciem liniowego SVM.

4.1.3. CNN + MMOD

Konwolucyjne sieci neuronowe (CNN) uczą się jakie cechy obrazu pozwalają sklasyfikować widoczne na nim obiekty. Za pomocą operacji splotowych, nakładając odpowiednie filtry są wstanie je uwypuklić i uzyskać istotne informacje. To właśnie w warstwie konwolucyjnej używane są odpowiednie jądra przekształceń. Sieć przez trening sama dobiera optymalne filtry oraz ich wartości. Dodatkowo występują warstwy próbkowania (downsampling), których celem jest zmniejszenie wielkości obrazu przez pominięcie części pikseli. Pomaga to uprościć sieć, ale kosztem utraty pewnej ilości informacji. Czasem zamiast pomijać piksele brane są wartości uśredniane lub maksymalne z pewnego sąsiedztwa. [24]

W bibliotece *dlib* sieć CNN często jest zestawiona z metodą *Max-Margin Object Detection (MMMOD)* [25]. Służy ona do optymalizacji i zwiększenia prędkości detekcji obiektów.

Taka implementacja CNN+MMOD dostępna w *dlib* stosowana jest w pracy dyplomowej.

4.1.4. Głębokie sieci neuronowe

Głębokie sieci neuronowe (DNN) różnią się od klasycznych tym, że mają większą liczbę warstw ukrytych. Taki algorytm tworzy plamki o ustalonej wielkości ze zdjęć wejściowych, a następnie przepuszcza je przez kolejne warstwy sieci celem wykrycia pożądanych obiektów. Na wyjściu podaje prawdopodobieństwo, że na obrazie znajduje się interesujący nas element.

W projekcie wykorzystywany do tego jest jeden z modułów biblioteki *OpenCV* zawierający implementację DNN [26]

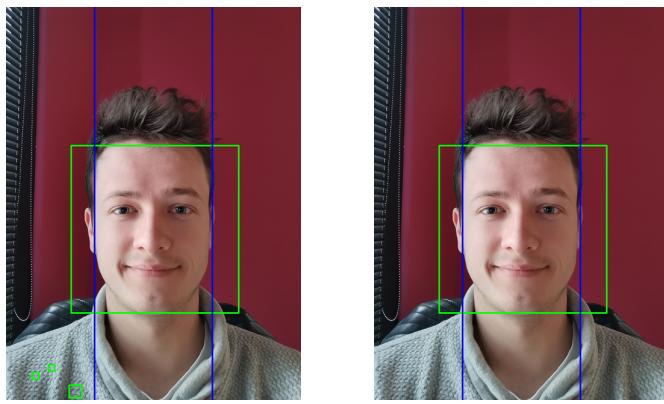
Jednym z modeli dostępnych do detekcji twarzy przy pomocy głębokich sieci neuronowych są modele Caffe (*Convolutional Architecture for Fast Feature Embedding*) [27]. W projekcie używany jest wzorzec *caffe res10_300x300_ssd_iter_140000_fp16* [28].

4.2. Filtrowanie zwracanych obszarów twarzy

Użyte algorytmy mogą dawać w wyniku błędnie określone obszary twarzy. Z tego względu zwraca tablica obszarów poddawana jest filtrowaniu.

Proces ten składa się z następujących etapów:

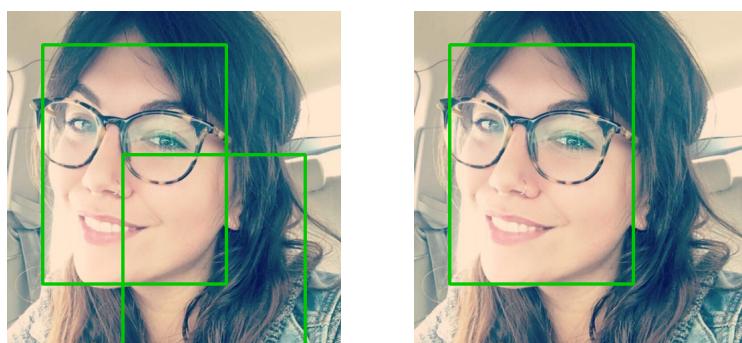
- Na początku odrzucane są obszary, których środek znajduje się poza ustalonym pionowym obszarem (przyjętem przedział [0.25, 0.75] szerokości). Wynika to z założeń, że osoba używająca urządzenia mobilnego, korzysta z niego patrząc na wprost, a nie z boku. Natomiast odchylenie od pionu to indywidualne preferencje - dlatego nie określony jest poziomy obszaru. (patrz *Rysunek 4.2*)
- Kolejnym etapem jest odrzucenie tych detekcji, które wychodzą zbyt daleko poza zdjęcie. Jeśli którykolwiek z boków prostokąta wystaje pionowo/poziomo o odległość



(a) Przed filtrowaniem zależnym od położenia
 (b) Po filtrowaniu

Rysunek 4.2. Działanie filtrowania detekcji twarzy w oparciu o położenie twarzy w centralnej części zdjęcia.

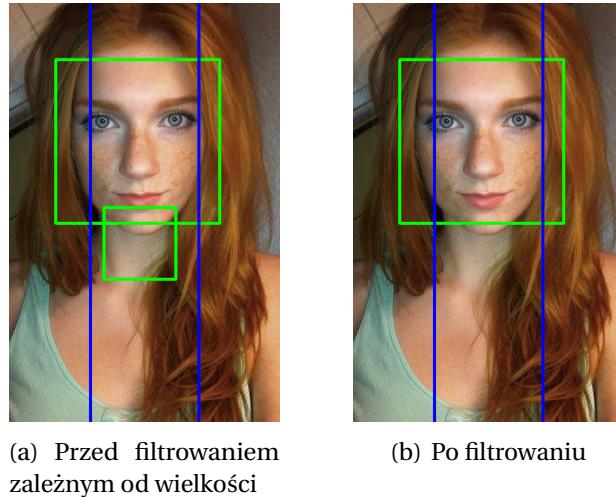
większą niż 10% odpowiednio wysokości/szerokości to zostaje odrzucony. (patrz rys. 4.3)



(a) Przed filtrowaniem zależnym od wystawania poza obraz
 (b) Po filtrowaniu

Rysunek 4.3. Działanie filtrowania detekcji twarzy w oparciu o odległość wykrytego obszaru poza zdjęcie.

- Z pozostałych obszarów wybierany jest ten, który zajmuje największą powierzchnię. Taki wybór umotywowany jest własnymi obserwacjami autora o zachowaniu się algorytmów detekcji twarzy oraz tym, że głowa użytkownika telefonu na obrazie z kamery przedniej zajmuje większą część płaszczyzny, ponieważ korzystając z urządzenia nie trzymamy go bardzo daleko od siebie. (patrz Rysunek 4.4)



Rysunek 4.4. Działanie filtrowania detekcji twarzy w oparciu o wielkość wykrytego obszaru. Źródło zdj.: [29]

5. Porównanie algorytmów detekcji twarzy

W rozdziale 4.*Detekcja twarzy* przedstawionych zostało kilka metod, które zostały zaimplementowane w projekcie. Ze względu, że dla prawidłowego i akceptowalnego działania aplikacji potrzebna jest odpowiednia skuteczność i szybkość detekcji twarzy, wszystkie metody zostały przetestowane i porównane (z wyjątkiem CNN MMOD - patrz rozdz. 5.1). Na podstawie wyników została wybrana jedna, która używana jest w finalnej części projektu.

5.1. Odrzucenie algorytmu CNN+MMOD

Ze względu na bardzo wolne działanie algorytmu dlib opartego na konwolucyjnych sieciach neuronowych MMOD został on wykluczony z testów. Czas przetwarzania jednego zdjęcia 500x500 wynosił kilka sekund, co całkowicie uniemożliwia działanie aplikacji w czasie rzeczywistym. Bardzo niska prędkość detekcji prawdopodobnie wynika z niemożności skorzystania z obliczeń na karcie graficznej na urządzeniach mobilnych. Metoda ta jest bardzo szybka gdy wykorzystuje do działania takie architektury jak CUDA [2], natomiast dużo gorzej radzi sobie z obliczeniami wykonywanymi na procesorach CPU.

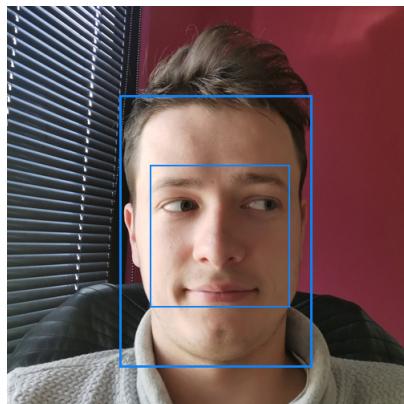
5.2. Testowanie na statycznych zdjęciach

Pierwszy etap testowania algorytmów detekcji twarzy bazował na statycznych zdjęciach z głównego zbioru danych (patrz rozdz. 3.1.*Zbiór danych*). Pozwoliło to przetestować na jednakowych danych wszystkie metody pod względem ich skuteczności wykrywania docelowych obszarów w różnych warunkach oraz uzyskać miarodajne wyniki.

5.2.1. Oczekiwany wynik

Każde zdjęcie z przygotowanego zbioru zostało opisane przez dwa prostokąty między którymi powinna się znaleźć wykryta przez algorytm twarz. Obszar ten został dobrany w następujący sposób:

- Wewnętrzna część obejmuje minimalny obszar, na którym znajdują się brwi, oczy, nos i usta.
- W zewnętrznym prostokącie powinna znaleźć się cała twarz. Powiększony jest on o pewną tolerancję.



Rysunek 5.1. Oczekiwany obszar detekcji twarzy.

5.2.2. Warunki testowania

Dla każdego algorytmu zostały przeprowadzone testy na zestawach obrazów o następujących rozdzielczościach i przestrzeniach barw:

- 300x300 RGB
- 500x500 RGB
- 300x300 skala szarości
- 500x500 skala szarości

W przypadku *DNN Caffe* nie było możliwe przeprowadzenie badań dla zdjęć w skali szarości, ponieważ wymaga on obrazu z trzema kanałami barw.

Testy wykonywane były w trybie *wydaniowym* (ang. *release*), ponieważ w trybie *debugowania* algorytm *Dlib HOG* działał nawet 180 razy wolniej. W przypadku pozostałych algorytmów tryb budowania nie miał większego wpływu na prędkość obliczeń, ale żeby wyniki były jak najbardziej miarodajne to każdy test musiały być przeprowadzony w tych samych warunkach.

5.2.3. Badanie skuteczności detekcji

W tym teście zostały zebrane i porównane następujące dane:

- **Prawidłowe detekcje** - suma perfekcyjnych i częściowo dobrych detekcji

5. Porównanie algorytmów detekcji twarzy

- **Perfekcyjne detekcje** - jeśli wykryty obszar w pełni znajduje się pomiędzy oczekiwany prostokątami
- **Częściowo dobre detekcje** - jeśli są krawędzie, które znajdują się poza oczekiwany obszarem, ale w zadowalającej odległości (patrz niżej - *Uwaga 1.*)
- **Z 3 na 4 krawędzie perfekcyjne detekcje** - jeśli tylko jedna krawędź znajduje się poza oczekiwany obszarem w zadowalającej odległości (patrz niżej - *Uwaga 2.*)
- **Złe detekcje** - jeśli twarz nie została wykryta lub wskazany obszar jest niezadowalający
- **Twarze niewykryte** - jeśli całkowicie nie udało się wykryć twarzy (patrz niżej - *Uwaga 3.*)

Uwaga 1. Obszar uznawany jest za częściowo dobry jeśli żadna krawędź nie jest oddalona o więcej niż $1.2x$ i maksymalnie jedna oddalona jest o długość z przedziału $[1.1x, 1.2x]$. Odległość x to szerokość lub wysokość (zależnie od krawędzi) maksymalnego oczekiwanej obszaru twarzy.

Uwaga 2. Obszar zaliczany jest do grupy 3/4 perfekcyjnych detekcji, jeśli 3 krawędzie znajdują się w oczekiwany obszarze, a czwarta odchylona od normy w przedziale $[1.0x, 1.2x]$.

Uwaga 3. Dodatkowy podział złej detekcji na niewykryte twarze wynikał z faktu, że metody oparte o *Cascading Classifier* na wyjściu podają obszar kwadratowy i przy rozcięgnietej lub pochylonej twarzy boczne obszary mogą być bardzo oddalone od oczekiwanej wartości, ale dalej skutecznej wykrywać twarz.

Tabela 5.1. Skuteczność algorytmów detekcji twarzy dla obrazów RGB ze zbioru danych

	Prawidłowe detekcje	Perfekcyjne detekcje	Częściowo dobre detekcje	3/4 krawędzie perfekcyjne	Złe detekcje	Niewykryte twarze
Haar Cascade 500x500	69	4	65	32	11	9
Haar Cascade 300x300	68	5	63	33	12	9
LBP Cascade 500x500	58	4	54	29	22	22
LBP Cascade 300x300	61	4	57	34	19	17
DNN Caffe 500x500	80	68	12	11	0	0
DNN Caffe 300x300	80	62	18	18	0	0
Dlib HOG 500x500	78	31	47	36	2	2
Dlib HOG 300x300	76	24	52	33	4	4

Metoda *Haar Cascade* pozwoliła uzyskać średnio $\sim 69/80$ (86%) dobrych detekcji. Jest to relatywnie przeciętny wynik. Na taki rezultat składa się kilka problemów tej metody. Nie radzi sobie ona dobrze z częściowo zakrytymi twarzami lub gdy głowa jest pochylona w bok. Kolejnymi czynnikiem wpływającym negatywnie na detekcję jest światło - problem z wykrywaniem występuje gdy zdjęcie jest zbyt jasne, twarz oświetlona lub źródło

Tabela 5.2. Skuteczność algorytmów detekcji twarzy dla obrazów w skali szarości ze zbioru danych

	Prawidłowe detekcje	Perfekcyjne detekcje	Częściowo dobre detekcje	3/4 krawędzie perfekcyjne	Złe detekcje	Niewykryte twarze
Haar Cascade 500x500	69	4	65	31	11	9
Haar Cascade 300x300	67	4	63	34	13	9
LBP Cascade 500x500	60	5	55	30	20	17
LBP Cascade 300x300	60	4	56	31	20	17
DNN Caffe 500x500	nd.	nd.	nd.	nd.	nd.	nd.
DNN Caffe 300x300	nd.	nd.	nd.	nd.	nd.	nd.
Dlib HOG 500x500	75	27	48	35	5	5
Dlib HOG 300x300	72	23	49	31	7	7

światła świeci prosto w obiektyw. Zaletą tej metody jest mała ilość zwróconych przez nią dodatkowych, błędnych obszarów, które musiały zostać odfiltrowane.

Klasyfikator kaskadowy bazując na modelu *LBP* miał najgorsze wyniki detekcji twarzy, na poziomie ~ 60/80 (75%). Jednak co zwraca uwagę to fakt, że bardzo duży odsetek twarzy nie został w ogóle wykryty. Występują tu te same problemy co w *Haar Cascade*, ale dodatkowo algorytm nie radzi sobie gdy twarz zajmuje dużą część zdjęcia.

Najlepszy wynik detekcji uzyskał bezdyskusyjnie *DNN Caffe*. Fakt, że w każdym z dwóch testów wykrył on 100% twarzy jest warty odnotowania. Co więcej, perfekcyjne detekcje były na poziomie ~ 65/80 (81,25%). Nie występują tu problemy takie jak w poprzednich algorytmach. Radzi sobie on dobrze nawet w złych warunkach oświetleniowych. Częściowe zakrycie twarzy nie wpływa w znacznym stopniu na detekcję. Wykrywa on dobrze zarówno pochylone jak i odwrócone twarze. Jedynym negatywnym zjawiskiem, które udało się zaobserwować w tej metodzie to zwracanie wielu dodatkowych obszarów, które są błędne. Zastosowanie filtrowania pozwoliło jednak odrzucić wszystkie błędne wskazania.

Bardzo dobre wyniki detekcji uzyskał również algorytm *Dlib HOG*, w szczególności w przypadku zdjęć RGB 500x500 - jego skuteczność była na poziomie 78/80 (97,5%). Zaletą tej metody jest zwracanie tylko jednego wykrytego obszaru - na żadnym z 80 zdjęć nie zwrócił ani jednego dodatkowego miejsca, które uznał za twarz. Nie wykrył on twarzy gdy była ona w połowie zakryta. Zakładając jednak, że aplikacja będzie analizować twarz użytkownika korzystającego z telefonu można przyjąć, że będzie ona w wystarczającym stopniu widoczna. W przeciwieństwie do pozostałych metod, które zwracają obszar całej twarzy, ta wykrywa częściowo obcięty rejon - np. pomijając czoło. Nie jest to w ogólności wadą, ponieważ te elementy nie są wykorzystywane w projekcie pracy dyplomowej.

Różnica w procencie perfekcyjnych detekcji pomiędzy *DNN Caffe*, a pozostałymi wynika z rodzaju obszarów zwracanych przez te algorytmy. Metoda oparta na głębokich sieciach neuronowych zwraca prostokąt o dowolnym stosunku boków, natomiast reszta

5. Porównanie algorytmów detekcji twarzy

zwraca kwadrat. Dzięki temu *DNN* lepiej dopasowuje się do kształtu twarzy niż *Cascading Classifier* i *HOG*.

Zmiana przestrzeni barw nie wpłynęła istotnie na rezultaty poszczególnych algorytmów. Jedynie zauważalne było obniżenie skuteczności detekcji w skali szarości w porównaniu do RGB dla metody opartej na *histogramie gradientów zorientowanych Dlib*.

5.2.4. Badanie szybkości detekcji

W tym teście zostały zebrane i porównane następujące dane:

- **Całkowity czas przetwarzania** - suma czasów wszystkich 20 iteracji, całkowity czas testu.
- **Średni czas przetwarzania pojedynczej iteracji** - uśredniony czas pojedynczej iteracji
- **Średni czas przetwarzania jednego zdjęcia** - uśredniony czas przetwarzania pojedynczego zdjęcia

Uwaga 1. Celem miarodajnego wyniku czasu przetwarzania każdy test został przeprowadzony 20 razy.

Tabela 5.3. Czas przetwarzania algorytmów detekcji twarzy dla obrazów RGB ze zbioru danych

	Całkowity czas przetwarzania	Średni czas przetwarzania pojedynczej iteracji	Średni czas przetwarzania pojedynczego zdjęcia
Haar Cascade 500x500	116,72 s	5,836 s	0,072 s
Haar Cascade 300x300	46,24 s	2,312 s	0,028 s
LBP Cascade 500x500	63,27 s	3,163 s	0,039 s
LBP Cascade 300x300	21,25 s	1,062 s	0,013 s
DNN Caffe 500x500	106,17	5,308	0,066 s
DNN Caffe 300x300	103,56 s	5,178 s	0,064 s
Dlib HOG 500x500	71,10 s	3,555 s	0,044 s
Dlib HOG 300x300	26,08 s	1,304 s	0,016 s

Najszybszy okazał się algorytm operujący na histogramach gradientowych. Niewiele wolniej przetwarzał algorytm kaskadowy *LBP*. *DNN Caffe* dla zdjęć 500x500 był porównywalnie szybki jak *Haar Cascade*, natomiast już w przypadku 300x300 około 2.5 razy wolniejszy.

Istotnym i wartym odnotowania jest fakt, że algorytm *DNN* przetwarzał prawie tak samo szybko obie rozdzielcości zdjęć. Z tego powodu sformułowano i zbadano tezę, że dla tej metody wielkość obrazu nie ma wpływu na szybkość przetwarzania (patrz rozdz. 5.2.5.).

Tabela 5.4. Czas przetwarzania algorytmów detekcji twarzy dla obrazów w skali szarości ze zbioru danych

	Całkowy czas przetwarzania	Średni czas przetwarzania pojedynczej iteracji	Średni czas przetwarzania pojedynczego zdjęcia
Haar Cascade 500x500	116,52 s	5,826 s	0,072 s
Haar Cascade 300x300	45,93 s	2,296 s	0,028 s
LBP Cascade 500x500	62,34 s	3,117 s	0,038 s
LBP Cascade 300x300	21,64 s	1,082 s	0,013 s
DNN Caffe 500x500	nd.	nd.	nd.
DNN Caffe 300x300	nd.	nd.	nd.
Dlib HOG 500x500	58,60 s	2,930 s	0,036 s
Dlib HOG 300x300	21,88 s	1,094 s	0,013 s

Na szybkość detekcji algorytmu *HOG* niewątpliwie miało wpływ użycie go w języku C++ mimo narzutu związanego z wywoływaniem go przez interfejs JNI (patrz rozdz. 2.4).

Zmiana detekcji z trójkanałowej RGB na skalę szarości w przypadku *Haar* i *LBP* nie skróciła czasu detekcji. W przypadku metody z biblioteki *Dlib* algorytm przetwarzał te zdjęcia ~ 15 – 20% krócej niż w wersji kolorowej.

5.2.5. Wpływ rozdzielczości na szybkość algorytm *DNN Caffe*

Ze względu na wyniki szybkości poszczególnych metod z poprzedniego rozdziału sformułowano tezę, że rozdzielcość nie ma wpływu na złożoność obliczeniową algorytmu opartego o głębokie sieci neuronowe. Celem poparcia tezy przeprowadzono dodatkowe badania na czterech wielkościach zdjęć:

- 300x300
- 500x500
- 1000x1000
- 2000x2000

Tabela 5.5. Wpływ rozdzielczości zdjęcia na detekcję DNN

	Prawidłowe detekcje	Perfekcyjne detekcje	Częściowo dobre detekcje	Średni czas przetwarzania pojedynczej iteracji	Średni czas przetwarzania pojedynczego zdjęcia
300x300	80	62	18	5,148 s	0,064 s
500x500	80	68	12	5,239 s	0,065 s
1000x1000	80	67	13	5,29 s	0,066 s
2000x2000	80	65	15	4,988 s	0,062 s

5. Porównanie algorytmów detekcji twarzy

Test ten potwierdza postawioną wcześniej tezę, że wielkość zdjęcia nie ma wpływu na szybkość działania algorytmu *DNN Caffe*. Przetwarzanie każdej rozdzielczości trwało porównywalną ilość czasu, a różnice zapewne były skutkiem obciążenia urządzenia w danej chwili i są pomijalne.

Takie działanie tej metody wynika prawdopodobnie z faktu, że tworzy ona plamki o ustalonej wielkości niezależnie od rozdzielczości zdjęcia wejściowego. W zaimplementowanym algorytmie jest to rozmiar 300x300. Dzięki temu ostatecznie zawsze przetwarzana jest taka sama ilość danych, co powinno skutkować w przybliżeniu stałym czasem obliczeń.

5.2.6. Precyza detekcji algorytmu *DNN Caffe* zależnie od sposobu filtracji

Metoda oparta na głębokich sieciach neuronowych na wyjściu zwraca wiele obszarów wraz ze wskaźnikiem pewności detekcji. Im większy współczynnik tym w teorii większa szansa, że jest to obiekt, który chcemy wykryć.

Z tego powodu porównano autorskie filtrowanie (patrz rozdz. 4.2) i wybór detekcji z największym procentem pewności.

Tabela 5.6. Wynik porównania sposobów filtrowania detekcji twarzy na zbiorze danych

	Prawidłowe detekcje	Perfekcyjne detekcje	Częściowo dobre detekcje	3/4 krawędzie perfekcyjne	Złe detekcje	Niewykryte twarze
Autorskie filtrowanie 300x300	80	62	18	18	0	0
Autorskie filtrowanie 500x500	80	68	12	11	0	0
Najwyższy współczynnik pewności 300x300	76	58	18	18	4	4
Najwyższy współczynnik pewności 500x500	76	64	12	11	4	4

Wyniki w tabeli 3.6 pokazują, że zaproponowana przez autora sekwencja filtrowania wykrytych obszarów daje na zbiorze danych lepsze rezultaty niż wybór najwyższego współczynnika pewności. Autorskie rozwiązanie wykryło wszystkie twarze, natomiast wybór oparty na procencie pewności nie poradził sobie z 4 zdjęciami.

5.3. Testowanie na obrazie z kamery na żywo

W teście opartym na statycznych zdjęciach najlepsze okazały się algorytmy *DNN* i *HOG*, a dodatkowo ten drugi był również najszybszy. Z tego względu w próbie wykorzystującej obraz na żywo badane były wyłącznie te dwa algorytmy, a pozostałe odrzucone.

Obraz przechwytywano w domyślnej rozdzielczości dla modułu CameraX - 640x480.

Oba algorytmy zostały przetestowane w czterech różnych warunkach:

- 1. w zwykłych, domowych warunkach oświetleniowych
- 2. w ciemnym miejscu
- 3. z intensywnym oświetleniem zza osoby
- 4. z intensywnym oświetleniem zza urządzenia

Zostały zebrane informacje o procencie klatek z wykrytą twarzą oraz chwilową i średnią ilość klatek na sekundę.

Podczas każdego testu pobrano i przetworzono 210 klatek, ale pierwsze i ostatnie 5 nie było branych pod uwagę przy wynikach. Związane było to z faktem inicjalizacji algorytmów oraz ręcznego wyłączenia aplikacji przez co nie przenosiły one w pełni wartościowych informacji.

5.3.1. Badanie skuteczności detekcji

Sprawdzenie skuteczności algorytmów polegało na zebraniu informacji na ilu klatkach z obrazu na żywo udało się wykryć twarz.

Tabela 5.7. Skuteczność algorytmów detekcji twarzy dla obrazu na żywo z kamery

Warunki	1.	2.	3.	4.
DNN rgb	200	200	200	200
HOG rgb	200	200	200	200
HOG sk. szaro.	200	200	200	200

Jak widać oba algorytmy wykrywały w każdej odebranej klatce twarz. Potwierdzają to zarówno logi jak i subiektywna ocena podglądu obrazu na żywo.

5.3.2. Badanie szybkość detekcji

Szybkość detekcji była porównana za pomocą średniej ilości klatek na sekundę. Jako, że w statystykach nie jest uwzględniany tylko czas detekcji, a działanie całej aplikacji to występują tu pewne narzuty czasowe związane z wyświetlaniem obrazu i rysowaniem na nich wykrytego obszaru celem podglądu testowanych parametrów na żywo. Jednak opóźnienie to występuje w każdym algorytmie i można uznać je za takie same.

Tabela 5.8. Szybkość algorytmów detekcji twarzy dla obrazu na żywo z kamery [klatki/s]

Warunki	1.	2.	3.	4.	Średnia:
DNN rgb	14,004	14,298	14,175	14,270	14,186
HOG rgb	16,488	16,224	16,337	16,400	16,362
HOG sk. szaro.	19,405	19,546	19,367	19,796	19,528

Podobnie jak w testach na statystycznych zdjęciach algorytm *HOG* okazał się szybszy od *DNN Caffe*. W przypadku histogramu gradientów w oparciu o zdjęcie w skali szarości ilość uzyskanych klatek na sekundę była większa aż o 37%, a w trójkanałowej przestrzeni barw o 15%.

5.4. Wybór algorytmu detekcji twarzy

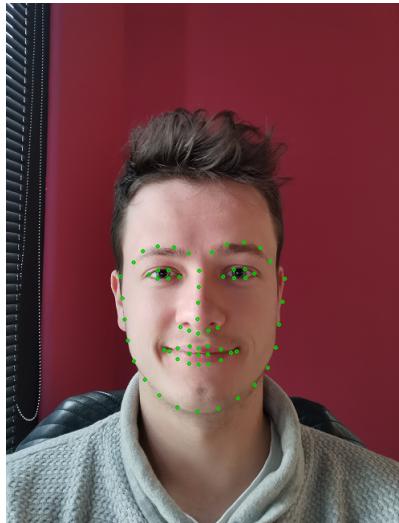
Na podstawie wyników zdecydowano się w ramach projektu na wykorzystanie algorytmu *dlib HOG* do detekcji twarzy. W testach skuteczności okazał się on prawie tak samo

5. Porównanie algorytmów detekcji twarzy

skuteczny jak *DNN Caffe*, ale zdecydowanie od niego szybszy, co w przypadku urządzeń mobilnych jest ważnym czynnikiem.

6. Detekcja znaczników twarzy

Facemarks (lub *Face landmarks*) to punkty nakładane na twarz wokół interesujących obszarów - takich jak oczy, nos czy usta. Pozwalają określić położenie, rozmiar czy kształt tych obiektów. Mogą być również użyte do predykcji czy mamy zamknięte/otwarte oczy (patrz rozdz. 8) lub czy się uśmiechamy.



Rysunek 6.1. Przykład zdjęcia z naniesionymi punktami charakterystycznymi twarzy

6.1. Algorytmy detekcji znaczników twarzy

6.1.1. Local binary features

Metoda oparta o histogram LBF [30]. W projekcie została użyta implementacja tego algorytmu z modułu *face* [4] zawartego w dodatkach do biblioteki OpenCV (patrz rozdz. 2.2.2). Do jej działania wykorzystywany jest gotowy model [31], który był trenowany na datasetie *HELEN* [32].

6.1.2. Kazemi

Kolejnym algorytmem służącym do estymacji znaczników jest Kazemi [33], który wykorzystuje drzewa regresyjne. Jest on zaimplementowany w bibliotece *dlib*. Gotowy model był trenowany na podstawie datasetu *iBUG 300-W face landmark* [34].

7. Porównanie algorytmów detekcji znaczników twarzy

Podobnie jak w przypadku detekcji twarzy przeprowadzone zostały testy algorytmów wykrywania facemarków, przedstawionych w rozdziale 6. Ze względu na specyfikę nanoszenia punktów charakterystycznych oraz ich ilość, trudno jest określić dokładność działania korzystając z matematycznych i liczbowych form wyrazu. Z tego powodu ocena jakości obu algorytmów to subiektywna opinia autora na podstawie obserwacji znaczników obszaru oczu i ust. Duża uwaga podczas opiniowania poświęcona została dokładności odwzorowaniu punktów w przypadku przykniętych lub całkiem zamkniętych oczu, ponieważ ma to istotny wpływ na inne aspekty pracy dyplomowej. Natomiast złożoność czasowa jest już mierzalna i została wyrażona liczbowo.

7.1. Testowanie na statycznych zdjęciach

Oba algorytmy zostały przetestowane na statycznych zdjęciach z ze zbioru danych w zakresie skuteczności i szybkości działania.

7.1.1. Usunięcie części zdjęć ze zbioru danych

Ze względu na wybór algorytmu *HOG* do detekcji twarzy konieczne okazało się odrzucenie 2 z 80 przygotowanych zdjęć, ponieważ metodzie tej nie udało się wykryć na nich twarzy (patrz rozdz. 5.2.3).

7.1.2. Badanie skuteczności detekcji

Podczas testu zostały zebrane następujące dane:

- **Prawidłowe detekcje** - pokrycie twarzy znacznikami, które uznane zostały za dobre
- **Złe detekcje** - pozostałe, które nie zostały uznane za dobre
- **Detekcje lepsze niż drugiego algorytmu** - który z dwóch algorytmów poradził sobie lepiej w danym przypadku testowym.

Zebrane dane są całkowicie subiektywnym odczuciem i inne osoby mogą mieć odmienną opinię oraz wyniki.

Oba algorytmy dawały taki sam rezultat zarówno w skali szarości jak i w trzy kanałowym zestawie barw, dlatego tabela wynikowa została uproszczona przez usunięcie takiego podziału.

Tabela 7.1. Skuteczność algorytmów detekcji znaczników twarzy na zbiorze danych

	Prawidłowe detekcje	Złe detekcje	Detekcje lepsze niż drugiego algorytmu
LBF	35	42	16
Kazemi	66	12	62

Zebrane dane pokazują jasno, że model oparty na metodzie *Kazemi* dał zdecydowanie lepsze wyniki niż drugi badany algorytm. W 62 przypadkach testowych pokrycie twarzy facemarkami było subiektywnie dokładniejsze niż w metodzie *LBF*. Tylko 12 z 78 detekcji uznałem za błędne. Można przyjąć, że jest to wynik co najmniej poprawny.

Kazemi dobrze radził sobie z obróconymi i pochylonymi twarzami, natomiast *LBF* w takich przypadkach okazywał się mocno niedokładny i nakładał znaczniki w sposób podobny jak dla twarzy ustawionych pionowo. Oba algorytmy miały problem w przypadku gdy cień padał na obszar oczu, wtedy znaczniki w tych rejonach były odchylone od prawidłowych pozycji. Metoda *LBF* miała problem w przypadku osób z ciemniejszymi odcieniami skóry, a także gdy proporcje twarzy były rozcięgnięte. Gdy osoba na zdjęciu nosiła okulary nie wpływało to znacząco na dokładność algorytmu *Kazemi* w przeciwieństwie do drugiej metody, która osiągała wtedy złe rezultaty. Rozwiążanie *LBF* natomiast radziło sobie lepiej jeśli twarz i oczy były częściowo zasłonięte. W obu przypadkach trudne i intensywne warunki oświetleniowe wpływały negatywnie na dokładność odwzorowania punktów charakterystycznych.

Metoda *LBF* myliła się na wielu zdjęciach bardzo mocno, a punkty były rozłożone chaotycznie i losowo. W tych przypadkach trudno oszacować powód, ale jest to fakt praktycznie dyskwalifikujący to rozwiązanie. Ma to odwzorowanie również w tabeli, gdzie mniej niż połowa wyników została uznanych za dobre.

7.1.3. Badanie szybkości detekcji

W tym teście zostały zebrane i porównane następujące dane:

- **Całkowity czas przetwarzania** - suma czasów detekcji znaczników dla wszystkich 20 iteracji
- **Średni czas przetwarzania pojedynczej iteracji** - uśredniony czas detekcji znaczników dla pojedynczej iteracji
- **Średni czas przetwarzania jednego zdjęcia** - uśredniony czas detekcji znaczników dla pojedynczego zdjęcia

Tabela 7.2. Czas przetwarzania algorytmów detekcji znaczników twarzy na zbiorze danych

	Całkowity czas przetwarzania	Średni czas przetwarzania pojedynczej iteracji	Średni czas przetwarzania pojedynczego zdjęcia
Kazemi RGB	5,717 s	0,285 s	0,00366 s
LBF RGB	6,545 s	0,327 s	0,00419 s
Kazemi sk. szaro.	5,472 s	0,273 s	0,00351 s
LBF sk. szaro.	6,084 s	0,304 s	0,00391 s

Algorytm *Kazemi* z użyciem biblioteki dlib i języka C++ okazał się szybszy o ponad 10% od odpowiednika w postaci *LBF*. Obie metody uzyskały lepszy czas w teście opartym

7. Porównanie algorytmów detekcji znaczników twarzy

na obrazach w skali szarości niż w RGB o kilka procent. Istotnym faktem jest, że oba algorytmy potrzebują mało czasu (rzad wielkości $10^{-3} s$) na przetworzenie pojedynczego zdjęcia, dzięki czemu w warunkach czasu rzeczywistego nie powodują znacznego spadku klatek na sekundę.

7.2. Testowanie na obrazie z kamery na żywo

Kolejnym etapem testowania detekcji punktów charakterystycznych twarzy było wykorzystanie obrazu z kamery na żywo. Warunki przeprowadzenia eksperymentu zostały określone w rozdz. 5.3.

7.2.1. Badanie skuteczność detekcji

Ze względu na opisaną wyżej trudność matematycznego wyrażenia skuteczności nawiązania znaczników, wyniki porównania zostały przedstawione w formie opisowej i są subiektywnym odczuciem autora na podstawie obserwacji działania algorytmu na żywo.

Algorytm Kazemi bez zarzutu poradził sobie w trzech scenariuszach. Natomiast w jednym - przy mocnym oświetleniu padającym na obiektyw i na twarz - występowało wtedy chwilowe niedokładne dopasowanie. Poza tym problemem radził sobie on bardzo dobrze. Ruchy twarzy nie przeszkadzały w prawidłowym ułożeniu znaczników. Punkty były bardzo stabilne, a podczas sztywnego położenia twarzy nie występowały ich drgania.

Gorsze wyniki uzyskała metoda LBF. Podobnie jak Kazemi miał pewne problemy podczas scenariusza opartego na mocnym oświetleniu. Występowało ciągłe drwanie punktów, nawet podczas sztywnego położenia twarzy. Metoda ta oznaczała twarz jako szerszą niż w rzeczywistości była. Podczas ruchów twarzy algorytm gubił prawidłowe położenie punktów.

W obu przypadkach potwierdzają się problemy z testów na statycznych zdjęciach, gdzie intensywne oświetlenie negatywnie wpływało na odwzorowanie punktów charakterystycznych.

7.2.2. Badanie szybkość detekcji

Test był przeprowadzony używając detekcji twarzy *HOG*, do którego dostarczano obraz w przestrzeni barw RGB.

Tabela 7.3. Szybkość algorytmów detekcji znaczników twarzy dla obrazu na żywo z kamery [klatki/s]

Warunki	1.	2.	3.	4.	Średnia:
LBF RGB	16,076	15,960	15,820	16,079	15,983
LBF sk. szaro.	15,959	16,016	15,829	15,793	15,899
Kazemi RGB	15,887	15,941	16,077	16,171	16,019
Kazemi sk. szaro.	15,747	16,146	15,866	16,096	15,963

Mniejsza ilość klatek w przypadku testów w skali szarości prawdopodobnie jest związana z dodatkowym narzutem czasowym w postaci konwersji obrazu z trójkanałowej barwy na jednokanałową.

Oba algorytmy uzyskały bardzo zbliżone wyniki. Niewiele szybsza okazała się jednak metoda Kazemi. Rezultat jest porównywalny z testem przeprowadzonym na statycznych zdjęciach.

7.3. Wybór algorytmu detekcji znaczników twarzy

Kazemi okazał się przede wszystkim dużo skuteczniejszym i stabilnym algorytmem niż detekcja znaczników oparta na *Local Binary Features*. Dodatkowo jest on około 10% szybszy. Wszystkie testy wskazują na wyższość *Kazemi* i z tych powodów jest on używany w dalszej części pracy dyplomowej i projektu, jako główny algorytm określenia położenia punktów charakterystycznych.

8. EAR - Eye Aspect Ratio

Metoda polegająca na obliczeniu *EAR* [35] [36], czyli stosunek otwarcia oczu - wysokość do szerokości widocznej części gałki ocznej. Wykorzystuje się tu znaczniki twarzy (patrz rozdz. 6) naniesione dookoła oczu.

8.1. Wyznaczanie współczynnika EAR

Zależnie od ilości punktów wokół oka będzie różny wzór obliczania *EAR*.

Dla 6 punktów:

$$EAR = \frac{dist(L_0, L_1) + dist(L_2, L_4)}{2 * dist(L_3, L_5)} \quad (1)$$

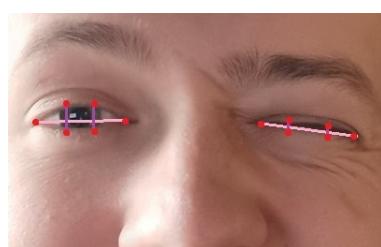
Natomiast, dla 4 punktów:

$$EAR = \frac{dist(L_0, L_2)}{dist(L_1, L_3)} \quad (2)$$

Gdzie L_x to kolejne landmarki dokoła oczu, a *dist* to odległość między dwoma punktami (odległość euklidesowa).

8.2. Zasada działania EAR w kontekście mrugania i określenia czy oko jest otwarłe/zamknięte

W teorii otwarte oczy będą miały większy wymiar liczbowy *EAR*, niż oczy zamknięte. Na rysunku 8.1 widać, że oko otwarte ma większe odległości między punktami pionowymi niż w przypadku oka zamkniętego. Dzięki takim różnicom możemy wykryć spadek wskaźnika *EAR* poniżej pewnego ustalonego poziomu, oznaczający zamknięcie oka, natomiast wzrost, otwarcie oka. Całkowicie obserwując zmiany np. za pomocą pochodnej jesteśmy w stanie stwierdzić mrugnięcie.



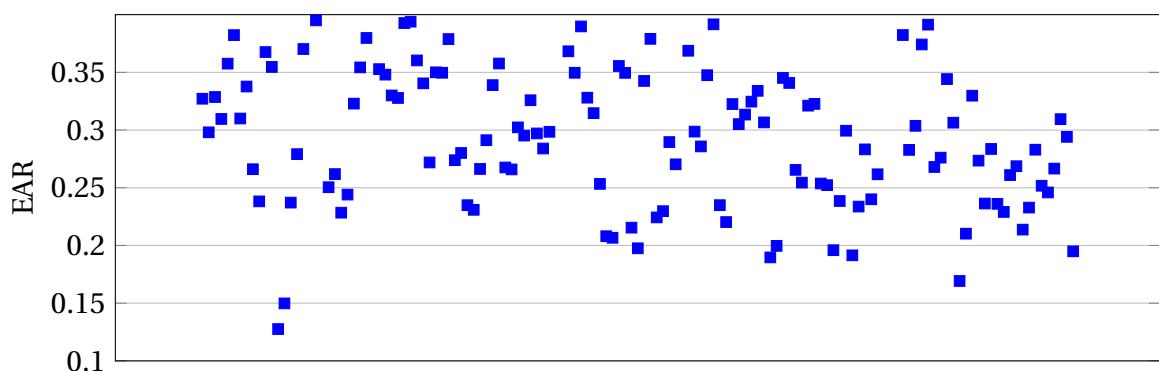
Rysunek 8.1. Teoretyczny rozmieszczenie znaczników wokół oczu wraz z naniesionymi połączeniami do obliczenia *EAR*

8.3. Ustalenie progu EAR

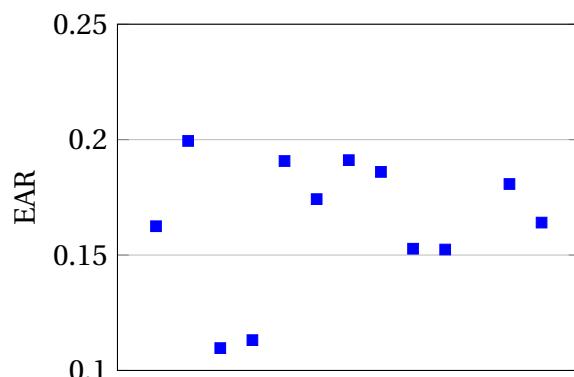
Do skutecznego działania metod opartych na współczynniku EAR należało wyznaczyć progów jego wartości poniżej, którego oko klasyfikowane jest jako zamknięte. W tym celu na potrzeby pracy dyplomowej zostały obliczone i zebrane wartości EAR wykorzystując statyczne zdjęcia ze zbioru danych oraz na podstawie obrazu z kamery urządzenia.

8.3.1. EAR dla statycznych zdjęć

Na podstawie używanego przygotowanego zbioru zdjęć został obliczony EAR dla wszystkich oczu i podzielony na dwie grupy: oczy otwarte (rys. 8.2) i oczy zamknięte (rys. 8.3).



Rysunek 8.2. Rozkład wartości współczynnika EAR oczu otwartych na zdjęciach ze zbioru danych



Rysunek 8.3. Rozkład wartości współczynnika EAR oczu zamkniętych na zdjęciach ze zbioru danych

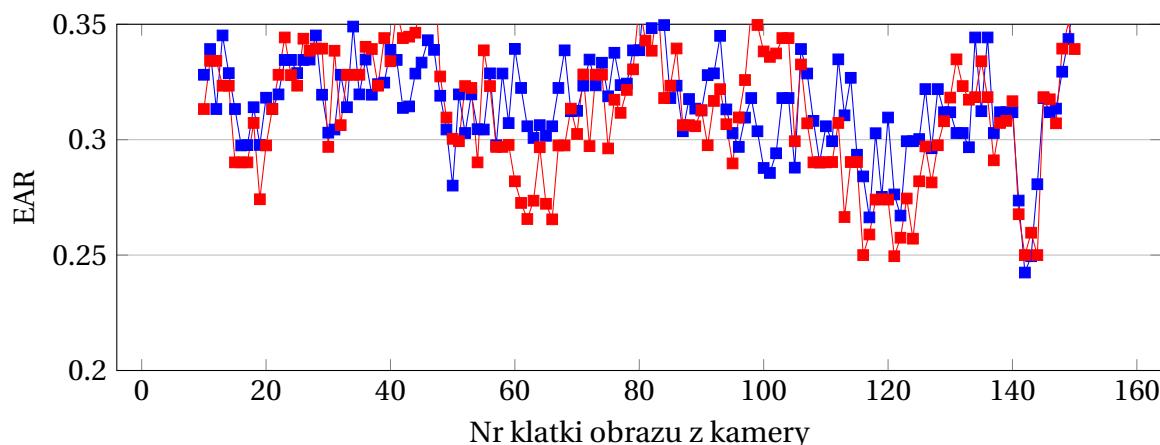
W przypadku oczu otwartych widać, że poza kilkoma wyjątkami wskaźnik EAR znajduje się powyżej 0.20, a czasem przekraczając nawet 0.35. Przypadek gdzie wartość jest poniżej 0.15 wynika najprawdopodobniej z faktu, że oczy były mocno przymknięte i ze względu na perspektywę rozciągnięte horyzontalnie. Odchylenie takie traktowane jest jako przypadek skrajny i nie brany pod uwagę w końcowym doborze współczynnika.

8. EAR - Eye Aspect Ratio

Na wykresie dla oczu zamkniętych ponownie próg zdaje się być na poziomie 0.20, a większość wskaźników osiąga wartość w przedziale [0.15, 0.20].

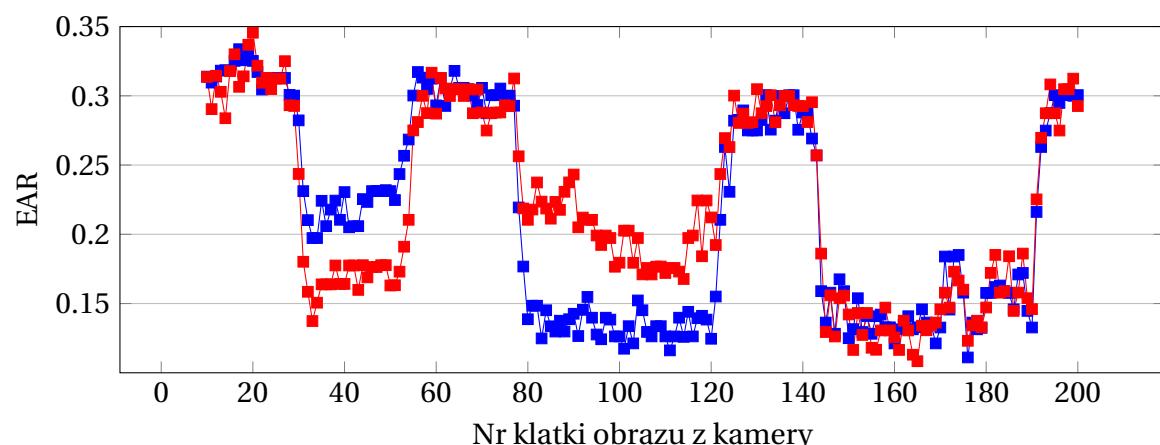
8.3.2. EAR na obrazie z kamery

Na obrazie z kamery urządzenia przeprowadzone zostały trzy testy wskaźnika EAR: oczy otwarte (rys. 8.4), oczy chwilowo zamknięte (rys. 8.5) oraz mrugnięcie (rys. 8.6). Na wykresach czerwony kolor oznacza EAR dla prawego oka, natomiast niebieski dla lewego.



Rysunek 8.4. Wartość współczynnika EAR dla oczu otwartych na obrazie z kamery

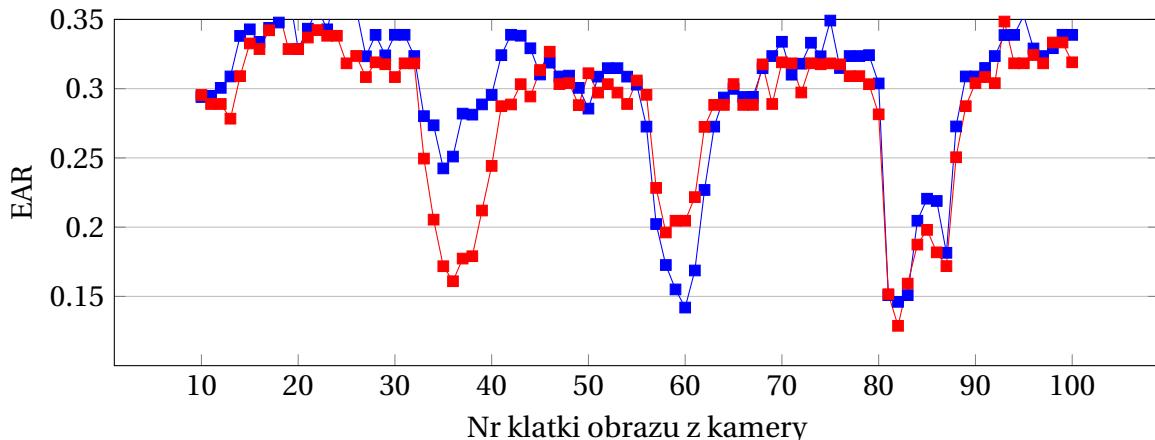
W przypadku gdy oczy były przez cały czas otwarte. Liczbowy wymiar EAR nie spadał poniżej 0.25, a przez większość czasu trzymał się nawet powyżej 0.30.



Rysunek 8.5. Wartość współczynnika EAR dla oczu czasowo zamkniętych na obrazie z kamery

W drugim teście oczy były czasowo zamknięte. Najpierw w przedziale klatek [25, 60] zamknięte było tylko oko prawe, następnie w przedziale [80, 120] tylko lewe, a na końcu w okresie [140, 190] oba oczy jednocześnie. Zmiana EAR jest tutaj mocno wyraźna, a w tych trzech przedziałach miał on wartość między 0.10, a 0.20. Dodatkowo na podstawie tych

danych trzeba również odnotować spadek EAR dla obu oczu nawet w przypadku zamknięcia tylko jednego z nich. Może to wynikać z faktu, że człowiek zamykając jedno przyjmuje lekko również drugie. Ale przyczyną może być też sam algorytm znaczników, w którym punkty dla różnych oczu mogą być ze sobą skorelowane.



Rysunek 8.6. Wartość współczynnika EAR podczas mrugania na obrazie z kamery

W ostatniej wariacji testu na obrazie z kamery występowały mrugnięcia: w przedziale [30, 40] prawego oka, [55, 65] lewego, a w przedziale [80, 90] oboma oczami na raz. Szybka zmiana wymiaru EAR jest w tych chwilach bardzo wyraźna i łatwa do wykrycia. W szczytowych chwilach mrugnięcia wartość spadała prawie do poziomu 0.15. Ponownie występuje to zmniejszenie EAR dla obu oczu nawet w przypadku mrugnięcia wyłącznie jednym.

8.4. Wnioski

Zmiany wartości EAR na obrazie z kamery są bardzo wyraźne i łatwe do wykrycia. Dzięki temu metoda ta nadaje się do detekcji mrugania i tego czy oczy są zamknięte.

Na podstawie testów progowy poziom EAR dla oka zamkniętego ustalony został na ≤ 0.19 . Powyżej tej wartości oko traktowane jest jako otwarte.

Jednak problem ze zmianą EAR dla obu oczu nawet w przypadku zamykania tylko jednego z nich sprawia, że w finalnej wersji pracy dyplomowej odnotowany jest wyłącznie fakt mrugnięcia bez podziału na to którym okiem została wykonana czynność.

9. Detekcja oczu

W tej pracy dyplomowej dużą rolę odgrywają oczy i ich wpływ na sterowanie aplikacją. Z tego powodu musiała zostać określona i zaimplementowana skuteczna metoda ich detekcji.

9.1. Algorytmy detekcji oczu

Na potrzeby tego etapu zostały zastosowane dwie metody opierające się na algorytmach opisanych wcześniej - klasyfikatory kaskadowe oraz znacznikach twarzy.

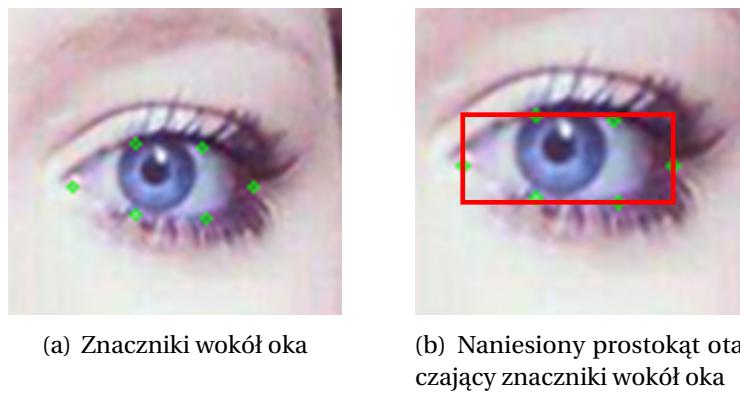
9.1.1. Klasyfikator kaskadowy Haar

Metody detekcji oparte na klasyfikatorach kaskadowych zostały opisane w rozdz. 4.1.1 przy okazji wykrywania twarzy. Do detekcji oczu z użyciem tego algorytmu został wykorzystany model autorstwa Shameem Hameed [37].

9.1.2. Znaczniki twarzy wokół oczu

Opisane w rozdz. 6 algorytmy facemarków nanoszą punkty charakterystyczne na obraz twarzy. Znajdują się one m.in. wokół oczu. Fakt ten można wykorzystać do detekcji ich obszaru. Celem uzyskania takiego wyniku należy wyznaczyć prostokąt, który będzie otaczał wszystkie sześć znaczników danego oka. [38]

Przykład takiego działania widoczny jest na rysunku 9.1.



Rysunek 9.1. Wykorzystanie znaczników twarzy do detekcji obszaru oczu

Wykorzystywany jest tu też wskaźnik *EAR* (patrz rozdz. 8) do wykrycia czy oko jest zamknięte czy otwarte.

Skuteczność detekcji oczu przy pomocy tej metody zależy w dużym stopniu od dokładności algorytmu odwzorowującego punkty charakterystyczne twarzy. Ewentualne niedoskonałości można niwelować zwiększając wielkość prostokąta o pewną tolerancję z danej strony. Taki współczynnik w projekcie został testowo ustalony, a przebieg badań i wyników został opisany w rozdz. 9.4.

9.2. Filtrowanie wyników detekcji oczu metodą Haar

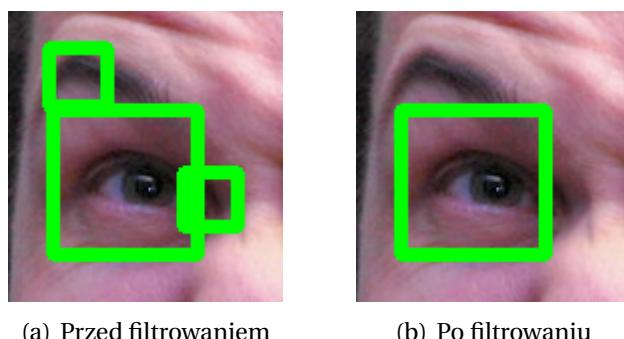
Ze względu, że algorytmu Haar może zwracać większą ilość prawdopodobnych obszarów, w których spodziewa się on wykryć pożądany obiekt, wprowadzone zostało filtrowanie wyników detekcji oczu.

Algorytm filtrowania składa się z dwóch etapów:

- Podzielenie wykrytych obszarów na dwie grupy - na lewą i prawą stronę twarzy
- W obu grupach wybranie największego obszaru

Dodatkowo pozwoliło to na łatwe zidentyfikowanie który obszar to które oko i ich posortowanie.

Przykładowy rezultat takiego filtrowania pokazany jest na *rysunku 9.2*.



Rysunek 9.2. Efekt filtrowania obszarów detekcji oczu

9.3. Obcięcie obszaru twarzy dla detekcji oczu metodą Haar

Dla metody opartej na Haar zdecydowano się dodatkowo zawęzić płaszczyznę przeszukiwań o pewną część twarzy, celem uzyskania wyników lepszych niż bez takiego zmniejszenia.

Ustalenie jaki obszar da najlepszy rezultat odbyło się przez testowe sprawdzanie kombinacji trzech parametrów oznaczających jaka część wykrytej twarzy zostaje obcięta z poszczególnej strony. Mogą one przyjmować wartości w następujących przedziałach:

- Góra: [0.0, 0.2]
- Dół: [0.2, 0.6]
- Boki: [0.0, 0.2]

Każdy parametr mógł osiągać wartości będące wielokrotnością 0.05 w poszczególnych przedziałach.

Ze względu na dużą liczbę kombinacji (225) nie zostaną podane wyniki, a jedynie najlepsza kombinacja:

- Góra: 0.05
- Dół: 0.3

9. Detekcja oczu

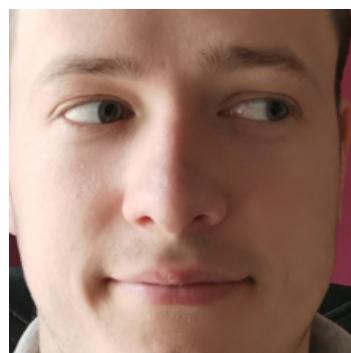
- Boki: 0.0

Algorytm Haar uzyskał lepszą skuteczność detekcji wykorzystując dodatkowe obcięcie niż bez niego - tabela 9.1.

Tabela 9.1. Skuteczność algorytmu detekcji oczu Haar z dodatkowym obcięciem i bez

	Prawidłowe detekcje	Perfekcyjne detekcje	Częściowo dobre detekcje	Złe detekcje	Niewykryte oczy otwarte	Niewykryte oczy zamknięte
Haar bez obcięcia	124	85	39	34	8	32
Haar z obcięciem	133	95	38	23	9	11

Przykład takiego obcięcia z wybranymi parametrami widoczny jest na *rysunku 9.3*.



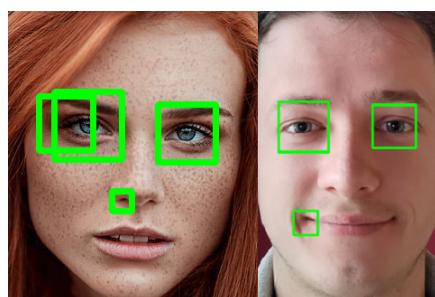
(a) Wykryty obszar twarzy



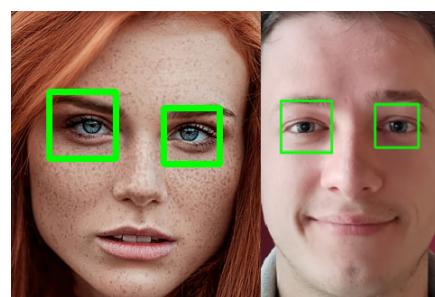
(b) Obszar po obcięciu

Rysunek 9.3. Obcięcie obszaru detekcji oczu zgodnie z dobranymi wcześniej parametrami

Wprowadzenie takiej modyfikacji pozwoliło odrzucić część błędnie ustalanych obszarów, szczególnie w dolnej części twarzy. Przykład poprawionej detekcji oczu dzięki temu zabiegowi widoczny jest na rysunku 9.4



(a) Wykrywanie oczu bez dodatkowego obcięcia obszaru



(b) Wykrywanie oczu z dodatkowym obcięciem obszaru

Rysunek 9.4. Odrzucenie błędnych rezultatów detekcji oczu po dodatkowym obcięciu obszaru.
Źródło pierwszego zdj.:[39]

9.4. Dostosowanie rozmiaru zwracanego obszaru oczu dla znaczników twarzy

Dla metody opartej o punkty charakterystyczne twarzy zdecydowano się dodać pewną tolerancję do obszaru oczu wynikającego jedynie z połączenia tych punktów.

Ustalenie jakie zwiększenie zwracanego regionu da najlepsze rezultaty detekcji oczu odbyło się przez testowe sprawdzenie kombinacji trzech parametrów, które oznaczały wielkość tolerancji z danej strony. W trakcie testu przyjmowały one następujące wartości:

- Góra: z przedziału [0.0, 1.0], będące wielokrotnością 0.1
- Dół: z przedziału [0.0, 1.0], będące wielokrotnością 0.1
- Boki: z przedziału [0.0, 0.5], będące wielokrotnością 0.05

Ze względu na dużą ilość kombinacji (1331) nie zostaną podane wyniki, a jedynie wybrana najlepsza kombinacja:

- Góra: 0.7
- Dół: 0.5
- Boki: 0.2

Tabela 9.2. Skuteczność algorytmu detekcji oczu wykorzystując znaczniki twarzy z dodatkowym zwiększeniem obszaru i bez na zbiorze danych

	Prawidłowe detekcje	Perfekcyjne detekcje	Częściowo dobre detekcje	Złe detekcje	Niewykryte oczy otwarte	Niewykryte oczy zamknięte
Znaczniki oczu bez zwiększenia obszaru	142	17	125	14	6	6
Znaczniki oczu ze zwiększeniem obszaru	144	142	2	12	6	6

Zmiana wielkości zwracanego obszaru oczu nie zwiększył znacznie liczbę dobrych detekcji. Natomiast największy zysk widoczny jest w perfekcyjnych detekcjach. Dzięki takiemu dostosowaniu udało się osiągnąć prawie 100% wskaźnik perfekcyjnych względem prawidłowych detekcji. Pozwoli to uzyskać lepiej wykryty obszar oczu, co może mieć przełożenie na skuteczność detekcji źrenic.

10. Porównanie algorytmów detekcji oczu

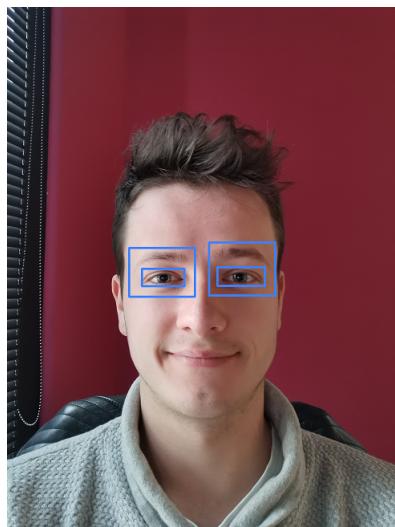
10.1. Testowanie na statycznych zdjęciach

Testowanie detekcji oczu wykorzystując statyczne zdjęcia odbywa się na obrazach z przygotowanego zbioru (odrzucone 2, dla których wybrany algorytm nie wykrył prawidłowo twarzy). Znajduje się na nich w sumie 166 oczu do wykrycia, w tym 14 zamkniętych.

10.1.1. Oczekiwany wynik

Podobnie jak w przypadku testowania detekcji twarzy na statycznych zdjęciach, tak w przypadku wykrywania oczu akceptowalny obszar został opisany dwoma prostokątami - minimalnym i maksymalnym. Przykład takiego oznaczenia widoczny jest na rys. 10.1. Oczekiwany obszar został dobrany w następujący sposób:

- prostokąt wewnętrzny obejmuje jedynie widoczną część gałki ocznej
- prostokąt zewnętrzny jest powiększony na boki i w dół o pewną odległość od gałki, a od góry zawiera w sobie również brwi.



Rysunek 10.1. Przybliżony obszar oczu, który jest oczekiwany wynikiem algorytmu

10.1.2. Warunki testowania

Oba algorytmy zostały przetestowane na wykrytych przez algorytm HOG twarzach ze zdjęć 500x500. W tym teście odrzucona została rozdzielcość 300x300, ze względu, że na części z nich osoba jest już bardzo oddalona, co skutkuje bardzo małym rozmiarem oczu. W przypadku korzystania z przedniej kamery telefonu nie będą występowały takie odległości między twarzą, a urządzeniem, żeby oczy zajmowały tak małą powierzchnię. Z tego powodu takie warunki nie przyniosłyby znaczących i wartościowych w kontekście pracy dyplomowej wyników.

Testy zostały przeprowadzone w przestrzeni barw RGB oraz w skali szarości.

10.1.3. Badanie skuteczności detekcji

Na początku porównywana była skuteczność zaprezentowanych metod. Chociaż w pewien sposób procent detekcji był już przedstawiony podczas dostrajania algorytmów, to tutaj zostały skonfrontowane ze sobą oba algorytmy.

W przypadku metody opartej o znaczniki twarzy, niewykryte oczy zarówno otwarte jak i zamknięte mogą oznaczać źle określenie położenia lub złą klasyfikację do jednej z tych grup. Przykładowo oko otwarte mające *EAR* mniejszy niż ustalony próg klasyfikowane było jako zamknięte, co skutkowało oznaczeniem jako błędna detekcja.

Tabela 10.1. Skuteczność algorytmów detekcji oczu na zbiorze danych

	Prawidłowe detekcje	Perfekcyjne detekcje	Częściowo dobre detekcje	Złe detekcje	Niewykryte oczy otwarte	Niewykryte oczy zamknięte
Znaczniki oczu RGB	144	142	2	12	6	6
Znaczniki oczu sk. szaro.	145	140	5	11	1	6
Haar RGB	133	95	38	23	9	11
Haar sk. szaro.	130	95	35	26	11	7

Najlepsze wyniki uzyskała metoda oparta na punktach charakterystycznych twarzy w trójkanałowej przestrzeni barw. Udało się jej wykryć średnio 87% wszystkich oczu. A w przypadku przestrzeni RGB aż 86% z nich perfekcyjnie. Ta druga statystyka jest dużo lepsza niż w przypadku metod Haar które uzyskały raptem 57%. Algorytm wykorzystujący znaczniki był skuteczniejszy od drugiej metody o około 10% jeśli weźmie się pod uwagę wykryte oczy.

Detekcja oczu przy pomocy znaczników i EAR zwracała bardzo dobrze dopasowany rejon oczu, bez zbyt dużego narzutu w postaci niepotrzebnych obszarów twarzy. Negatywne detekcje wynikały najczęściej ze zmrużonych oczu przez co współczynnik EAR sygnalizował, że oczy są zamknięte. Jednak większość zwróconych obszarów była prawidłowa. Wyniki z rozdziału 7.2.1 sugerują również, że w ciężkich warunkach oświetleniowych detekcja oczu może nie dawać zadowalających wyników. Nie można zapominać o fakcie, że wpływ na skuteczność tej metody ma przede wszystkim dokładność odwzorowania znaczników wokół oczu. W przypadku błędnej ich alokacji nie jest możliwe wykrycie prawidłowego obszaru oczu.

Największy problem klasyfikatorom kaskadowym używanym do wykrywania oczu sprawiły zdjęcia, na których osoba miała przekrzywioną pod kątem względem pionu głowę. Wynika to zapewne z samej natury algorytmu, ponieważ taki sam problem występował w przypadku wykorzystania jej do detekcji twarzy. Metoda Haar lepiej radziła sobie z jasnym oświetleniem niż ta oparta na znacznikach. Cienie padające na obszar oczu również nie wpływały na obniżenie skuteczności detekcji. Miała ona problem z wykryciem oczu w przypadku gdy były one częściowo zasłonięte np. włosami. Jeśli osoba na zdjęciu nosiła okulary to w większości przypadków detekcja była na dobrym poziomie. Problem

pojawiał się w momencie gdy na szkłach występuowały refleksy świetlne. Metodzie tej udało się prawidłowo wykryć większą liczbę przymkniętych oczu niż w przypadku drugiej metody. Wynikało to z faktu, że klasyfikatory kaskadowe zwracała również lokalizację oczu całkowicie zamknięte. W ramach projektu jest to skutek co najmniej niepożądany, z tego względu w wynikach uznawany był za błędny.

Ważną różnicą w działaniu obu algorytmów jest kształt i rozmiar oznaczanego obszaru. Metoda oparta o klasyfikatory zwraca kwadratowy region, zawierający dużo większą część twarzy niż same oczy, np. brwi. Natomiast punkty charakterystyczne tworzą obszar o kształcie prostokąta o dowolnym stosunku boków i zawierają głównie samą gałkę oczną. Zmniejsza to ilość punktów, które mogą przeszkadzać w innych aspektach analizy twarzy np. podczas detekcji źrenic.

Porównując przestrzenie barw ciekawym wynikiem jest uzyskanie większej liczby dobrych detekcji przez metodę opartą o znaczniki w przypadku obrazów w skali szarości, ale mniej perfekcyjnych niż dla RGB. Natomiast, algorytm Haar lepiej poradził sobie w przypadku trójkanałowego zestawu barw, ale wykrywał on więcej zamkniętych oczu jako otwarte przez co w tej statystyce wypadł gorzej niż w skali szarości.

10.1.4. Badanie szybkości detekcji

Dla detekcji oczu z użyciem znaczników twarzy zostały przedstawione dwa wyniki czasowe - jeden uwzględniający czas wykrycia punktów charakterystycznych, a drugi bez. Związane jest to z faktem, że mimo iż do wykorzystania tej metody niezbędna jest detekcja facemarków, to etap taki i tak będzie wykonany, ponieważ punkty te są używane np. do stwierdzenia mrugnięcia.

W przypadku czasów zawierających algorytm wykrywania punktów charakterystycznych test był przeprowadzony dla dwóch przestrzeni barw. Natomiast, na szybkość przeobrażenia facemarków w obszar oka nie wpływa liczba kanałów, ponieważ nie operuje on na pikselach, tylko na zwróconych punktach kartezjańskich. Z tego powodu przedstawiony jest tylko jeden wynik.

Wyniki bezdyskusyjnie pokazują dużo większą szybkość detekcji oczu z użyciem znaczników. Nawet biorąc pod uwagę czas wykrycia punktów charakterystycznych metoda ta była ponad 4 razy szybsza od algorytmu opartego na klasyfikatorach kaskadowych Haar.

Tworzenia obszarów z facemarków osiągnęło czas rzędu 10^{-5} s, co w porównaniu do pozostałych algorytmów wykorzystywanych w projekcie jest praktycznie natychmiastowe. Tak bliski零 czas przetwarzania sprawia, że nie ma on żadnego wpływu na ilość klatek na sekundę podczas odbierania obrazu na żywo z kamery.

Przestrzeń barw nie miała znaczącego wpływu na czas detekcji, a wyniki w przypadku trójkanałowych kolorów były porównywalne do skali szarości.

Tabela 10.2. Czas przetwarzania algorytmów detekcji oczu na zbiorze danych

	Całkowity czas przetwarzania	Średni czas przetwarzania pojedynczej iteracji	Średni czas przetwarzania pojedynczego zdjęcia
Znaczniki oczu RGB (z detekcją punktów)	5,638 s	0,281 s	0,00361 s
Znaczniki oczu sk. szaro. (z detekcją punktów)	5,803 s	0,290 s	0,00372 s
Znaczniki oczu (bez detekcją punktów)	0,024 s	0,00122 s	0,00000157 s
Haar RGB	26,219 s	1,310 s	0,0168 s
Haar sk. szaro.	25,830 s	1,291 s	0,0165 s

10.2. Testowanie na obrazie z kamery na żywo

Detekcja oczu została przetestowana również na obrazie z kamery na żywo. Warunki przeprowadzenia eksperymentu zostały opisane w rozdz. 5.3.

10.2.1. Badanie skuteczność detekcji

Obserwując na żywo detekcje oczu autor projektu był w stanie wysnuć kilka wniosków na temat badanych algorytmów.

Przede wszystkim oba działały bardzo stabilnie. Nie występowało nieuzasadnione grubienie obszaru oczu podczas statycznego położenia głowy. W metodzie Haar występował problem przy mocnym skręceniu głowy w bok. Zwracany był wtedy często błędny i bardzo powiększony obszar dalszego oka. Natomiast detekcja oczu na podstawie znaczników miała problemu z mocno przymkniętymi oczami. Wskaźnik EAR sygnalizował wtedy je jako zamknięte przez co algorytm nie zwracał ich regionu.

Subiektywnie oceniąc obie metody dają zadowalające i wystarczające rezultaty z perspektywy pracy dyplomowej, ale ze względu na stabilniejsze działanie metody wykorzystującej znaczniki podczas ruchów głową uznany został za lepsze rozwiązanie w tym przypadku.

10.2.2. Badanie szybkość detekcji

Ze względu na praktycznie zerowy obciążenie podczas przekształcania znaczników w obszar twarzy (patrz rozdz. 10.1.4), wyniki dla facemarków zostały skopiowane z rezultatów uzyskanych przez algorytm Kazemi podczas porównania metod nakładania punktów charakterystycznych na obraz z kamery na żywo w rozdz. 7.2.2.

Ponownie detekcja twarzy odbywała się przy zastosowaniu algorytmu *HOG* dostarczając obraz RGB.

Tabela 10.3. Szybkość algorytmów detekcji oczu dla obrazu na żywo z kamery [klatki/s]

Warunki	1.	2.	3.	4.	Średnia:
Haar RGB	11,698	13,600	12,290	12,942	12,632
Haar sk. szaro.	11,956	12,403	12,333	12,472	12,290
Znaczniki RGB	15,887	15,941	16,077	16,171	16,019
Znaczniki sk. szaro.	15,747	16,146	15,866	16,096	15,963

Wyniki przeprowadzonych badań przedstawiają znaczną przewagę szybkościową metody opartej na znacznikach. Klasyfikatory kaskadowe uzyskały około 23% mniejszą liczbę klatek na sekundę, co jest znaczącym obniżeniem wydajności.

10.3. Wybór algorytmu detekcji oczu

Porównanie dwóch algorytmów detekcji oczu bezdyskusyjnie pokazało wyższość metody opartej na znacznikach nad klasyfikatorem kaskadowym Haar. Punkty charakterystyczne uzyskały lepszy procentowo wynik prawidłowych detekcji, a dodatkowo w teście na statycznych zdjęciach prawie wszystkie były perfekcyjne. Próby szybkościowe jednoznacznie wskazały na facemarki, które były 4 razy szybsze od Haar, a jeśli brać pod uwagę jedynie czas tworzenia obszarów to ponad tysiąckrotnie i ich przetwarzanie jest niezauważalne w perspektywie pozostałych detekcji.

Na podstawie przeprowadzonych testów zostały wybrane znaczniki twarzy do detekcji oczu i to ten algorytm jest wykorzystywany w projekcie i pracy dyplomowej.

11. Detekcja źrenic

W pracy dyplomowej występuje sterowanie aplikacją za pomocą ruch gałek ocznych, w szczególności analizując położenie źrenic. Mając wykryty obszar oczu (patrz rozdz. 9) metodami klasycznymi przetwarzania obrazów określone położenie środka źrenicy.

W projekcie zaimplementowane zostały trzy algorytmy do testów i wybrany jeden, który dawał najlepsze rezultaty.

11.1. Algorytmy detekcji źrenic

11.1.1. Algorytm CDF (Cumulative Distribution Function)

Algorytm zaimplementowany na podstawie dwóch artykułów o detekcji źrenic [40] [41]. Opiera się w głównej mierze na progowaniu za pomocą dystrybuanty. Następnie analizuje najciemniejszy obszar wyznaczany jest środek źrenicy.

11.1.1.1. Kroki algorytmu

- Za pomocą progowania z użyciem dystrybuanty tworzony jest obraz binarny.

$$CDF(r) = \sum_{w=0}^r p(w) \quad (3)$$

Gdzie $p(w)$ to prawdopodobieństwo znalezienia punktu o jasności równej w - określone przy pomocy dystrybuanty.

$$I'(x, y) = \begin{cases} 255, & CDF(I(x, y)) < a \\ 0, & wpp \end{cases} \quad (4)$$

Gdzie I to jasność piksela, natomiast a to ustalony próg

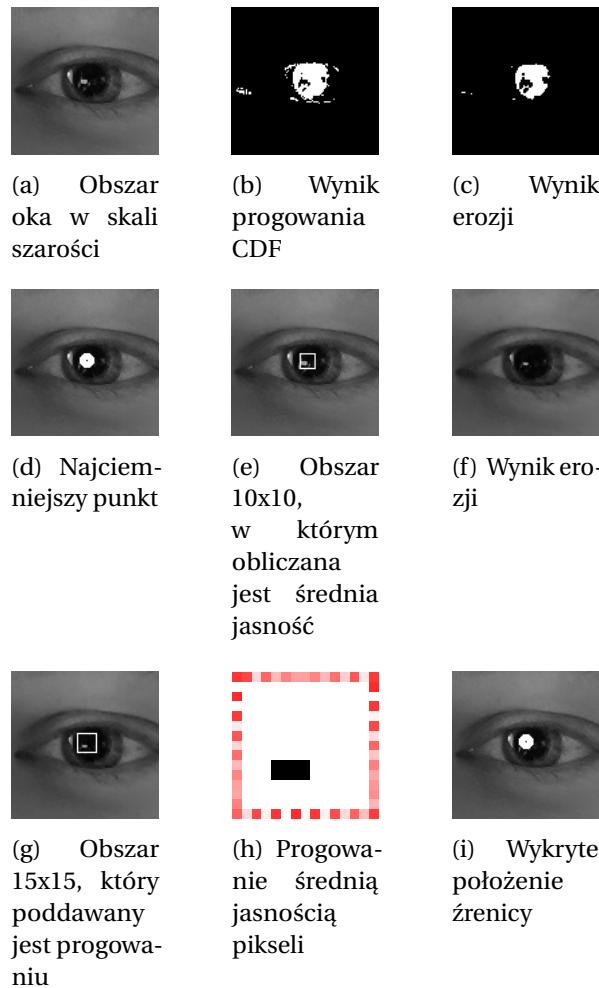
- Na uzyskany obraz binarny nakładana jest operacja morfologiczna erozji (filtr minimałny), celem usunięcia pojedynczych ciemnych pikseli
- Następnie znajdowany jest najciemniejszy piksel na oryginalnym obrazie wśród tych, które mają wartość 255 (są białe) na obrazie binarnym
- Obliczana jest średnia jasność pikseli w kwadracie 10×10 wokół wybranego najciemniejszego punktu
- Nakładana jest erozja na obszarze 15×15 wokół wybranego punktu
- Na tym obszarze stosowane jest progowanie na podstawie wartości

$$I'(x, y) = \begin{cases} 255, & I(x, y) < I_{AVG} \\ 0, & wp.p. \end{cases} \quad (5)$$

Gdzie I_{AVG} to średnia jasność obszaru obliczona wcześniej

- Szukanym punktem źrenicy jest środek ciężkości białych punktów na uzyskanym binarnym obrazie

11.1.1.2. Wynik kolejnych kroków algorytmu Na rys. 11.1 przedstawiony jest rezultat kolejnych etapów wykrywania źrenic przy pomocy metody CDF na przykładowym zdjęciu oka.



Rysunek 11.1. Kolejne etapy wykrywania źrenic metodą CDF

11.1.2. Algorytm PF (Projection Function)

Algorytm po raz pierwszy opisany w artykule zatytułowanym *Projection Functions for Eye Detection* [42] z 2004 roku. Jest oparty na rzutowaniu jasności pikseli na składowe poziome i pionowe. Do obliczenia tych wartości wykorzystuje się funkcje projekcji, która może przyjmować różne postaci - z czego trzy zostały opisane w tej pracy dyplomowej. Największe różnice wartości oznaczają szybkie zmiany jasności, które mogą być konturami oka. [41]

11.1.2.1. Funkcja projekcji Funkcja projekcji służąca do rzutowania jasności rzędów i kolumn może przyjmować różne formy. Najczęściej stosowana jest funkcja całkowa, jednak ze względu na swoje niedociągnięcia i kłopoty z wykrywaniem wariancji powstały także inne algorytmy. Trzy różne funkcje opisane są poniżej.

Funkcja całkowa Wylicza się za jej pomocą średnią jasność pikseli w danym rzędzie lub kolumnie. W teorii jest to całka:

$$IPF_h(y) = \frac{1}{x_b - x_a} \int_{x_a}^{x_b} I(x, y) dx \quad (6)$$

$$IPF_v(x) = \frac{1}{y_b - y_a} \int_{y_a}^{y_b} I(x, y) dy \quad (7)$$

Ale w praktyce degeneruje się do funkcji sumy:

$$IPF_h(y) = \frac{1}{x_b - x_a} \sum_{x=x_a}^{x_b} I(x, y) \quad (8)$$

$$IPF_v(x) = \frac{1}{y_b - y_a} \sum_{y=y_a}^{y_b} I(x, y) \quad (9)$$

Funkcja wariancji Średnia różnica między jasnością danego piksela, a wyliczonego IPF danego rzędu lub kolumny

$$VPF_h(y) = \frac{1}{x_b - x_a} \sum_{x=x_a}^{x_b} (I(x, y) - IPF_h(y)) \quad (10)$$

$$VPF_v(x) = \frac{1}{y_b - y_a} \sum_{y=y_a}^{y_b} (I(x, y) - IPF_v(x)) \quad (11)$$

Chociaż we wskazanym wyżej artykule funkcja ta występuje w przedstawionej formie to spotykane są również postacie z różnicą kwadratową oraz z pierwiastkami z obliczonej wartości.

Funkcja ogólna Parametryzowana suma wyliczonych wartości IPF i VPF .

$$GPF_h(y) = (1 - \alpha) * IPF_h(y) + \alpha * VPF_h(y) \quad (12)$$

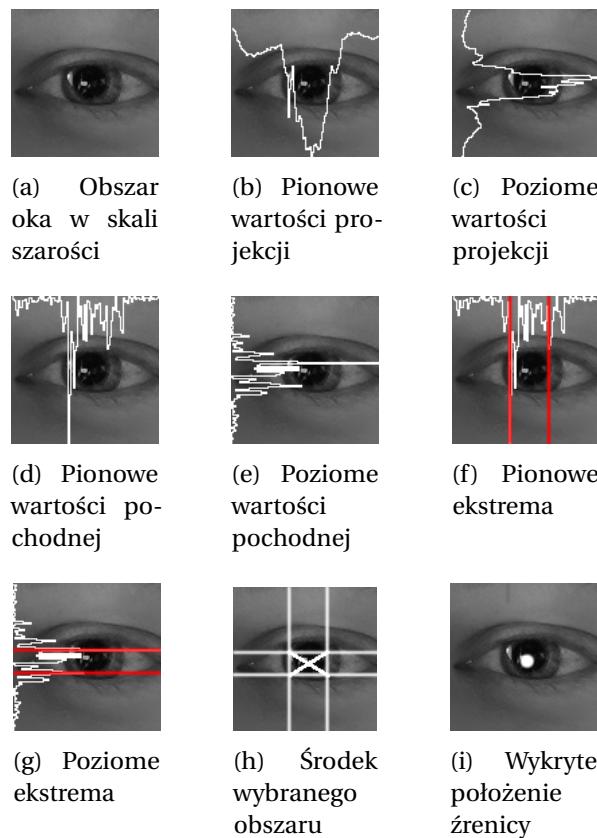
$$GPF_v(x) = (1 - \alpha) * IPF_v(x) + \alpha * VPF_v(x) \quad (13)$$

Autorzy tej metody zalecają parametr $\alpha = 0.6$ jako dający najlepsze rezultaty.

11.1.2.2. Kroki algorytmu

- Dla każdego rzędu i kolumny obliczana jest wartość funkcji projekcji
- Dla wyliczonych funkcji projekcji określana jest wartość pochodnej w każdym rzędzie i kolumnie
- Wybierane są dwa ekstrema dla poziomego i pionowego wymiaru
- Przecięcie wybranych kolumn i rzędów tworzy prostokąt, którego średnica jest poszukiwany punkt na źrenicy

11.1.2.3. Wynik kolejnych kroków algorytmu Na rys. 11.2 przedstawiony jest rezultat kolejnych etapów wykrywania źrenic przy pomocy metody PF na przykładowym zdjęciu oka.



Rysunek 11.2. Kolejne etapy wykrywania źrenic metodą PF

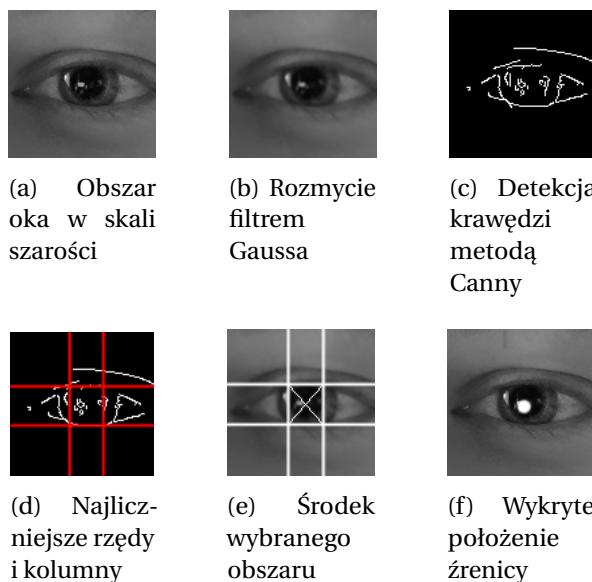
11.1.3. Algorytm EA (Edge Analysis)

Algorytm detekcji źrenic [41] opierający się na wykryciu i analizie krawędzi. w teorii krawędzie najbardziej pionowe i poziome na zdjęciu powinny należeć do tęczówki i źrenicy.

11.1.3.1. Kroki algorytmu

- Dla obszaru oka w skali szarości nakładany jest filtr rozmywający, np. Gaussa. Pozwala to pozbyć się drobnych szumów i wygładzić obraz.
- Wykrywane są krawędzie i tworzony obraz binarny. Można zastosować np. algorytm Canny, który jest jedną z najpopularniejszych metod detekcji krawędzi
- Wybierane są dwa rzędy i dwie kolumny z największą liczbą punktów o wartości 255 (białe piksele na obrazie binarnym)
- Przecięcie wybranych rzędów i kolumn tworzy prostokąt, którego środek jest poszukiwany punkt na źrenicy

11.1.3.2. Wynik kolejnych kroków algorytmu Na rys. 11.3 przedstawiony jest rezultat kolejnych etapów wykrywania źrenic przy pomocy metody EA na przykładowym zdjęciu oka.



Rysunek 11.3. Kolejne etapy wykrywania źrenic metodą EA

12. Porównanie detekcji źrenic

12.1. Testowanie na statycznych zdjęciach

12.1.1. Oczekiwany wynik

12.1.2. Zbierane dane

Podczas testów zbierane były następujące dane:

- **W obszarze tęczówki** - ilość wykrytych punktów znajdujących się wewnątrz tęczówki oka
- **Poza obszarem tęczówki** - ilość wykrytych punktów znajdujących się poza tęczówką oka
- **Średni błąd** - średni błąd ze wszystkich detekcji (patrz Uwaga 1.)
- **Błąd <= 0.1** - ilość detekcji z błędem nie większym niż 10%
- **Błąd <= 0.05** - ilość detekcji z błędem nie większym niż 5%

Uwaga 1. Błąd dla danego zdjęcia obliczany był następującym wzorem

$$b(P_1, P_2) = \frac{dist(P_1, P_2)}{hypot(w, h)} * 100\% \quad (14)$$

Gdzie:

- **P₁** - wykryty punkt
- **P₂** - oczekiwany środek źrenicy
- **dist** - odległość między punktami (euklidesowa)
- **hypot** - przeciwprostokątna dla podanych przyprostokątnych (w przypadku regionu oka jest to przekątna)
- **w, h** - szerokość i wysokość regionu oka

12.1.3. Wybór funkcji projekcji

Ze względu, że metoda PF może wykorzystywać różne funkcje projekcji do swojego działania konieczne było wyznaczenie i wybranie najlepszej opcji. Badanie skuteczności przeprowadzone było dla następującego zestawu funkcji:

- Całkowa
- Wariancja kwadratowa
- Wariancja pierwiastkowa
- Wariancja liniowa
- Ogólna z wariancją kwadratową
- Ogólna z wariancją pierwiastkową
- Ogólna z wariancją liniową

W przypadku funkcji ogólnej dodatkowo zostało przeprowadzone przeszukiwanie celem ustalenia najlepszego współczynnika α dla poszczególnych opcji. Badana była wielkość tego parametru w przedziale $[0.01, 0.99]$ ze skokiem 0.01 (wykluczono wartości 0.00 oraz 1.00 bo odpowiadają one odpowiednio funkcji całkowej i funkcji wariancji).

Finalnie najlepsze w poszczególnych wariantach okazały się następujące wartości współczynnika α :

- Ogólna z wariancją kwadratową: 0.99
- Ogólna z wariancją pierwiastkową: 0.01
- Ogólna z wariancją liniową: 0.31

W przypadku funkcji ogólnej z wariancją kwadratową najlepszy rezultat dał parametr $\alpha = 0.99$, co w przybliżeniu degeneruje ją do wariancji kwadratowej. Natomiast w przypadku pierwiastkowej ($\alpha = 0.01$) do funkcji całkowej. Potwierdzają to również wyniki w tabeli 12.1, gdzie opisane przypadki osiągają takie same lub zbliżone rezultaty.

Tabela 12.1. Skuteczność algorytmu PF zależnie od funkcji projekcji na zbiorze danych

	W obszarze tęczówki	Poza obszarem tęczówki	Średni błąd	Błąd <= 0.1	Błąd <= 0.5
Całkowa	85	51	14,35%	53	26
Wariancji, kwadratowa	98	38	13,51%	60	27
Wariancji, pierwiastkowa	79	57	14,66%	47	29
Wariancji, liniowa	51	85	23,12%	24	4
Ogólna, kwadratowa	98	38	13,51%	60	27
Ogólna, pierwiastkowa	85	51	14,27%	53	26
Ogólna, liniowa	86	50	14,17%	53	26

Najlepszą skuteczność wykazała funkcja wariancji kwadratowej oraz ogólna kwadratowa. Wykryły ona najwięcej punktów wewnątrz tęczówki - 72% oraz uzyskały najmniejszy średni błąd - 13.51%. Ze względu na opisaną wyżej degradację, której przyczyną jest współczynnik α z tej pary wybrana została funkcja wariancji, która musi wykonać mniej obliczeń w czasie swojego działania. Dlatego ta wersja funkcji była wykorzystywana w dalszej części badań.

12.1.4. Badanie skuteczności detekcji

Tabela 12.2. Skuteczność algorytmów detekcji źrenic na zbiorze danych

	W obszarze tęczówki	Poza obszarem tęczówki	Średni błąd	Błąd <= 0.1	Błąd <= 0.5
CDF	113	23	10,72%	87	54
PF	98	38	13,51%	60	27
EA	68	68	19,68%	42	14

W przypadku progowania przy pomocy dystrybuanty na prawidłową detekcję wpływ miała jakość wycinka zdjęcia podawanego na wejście metody. W przypadku gdy duża była jego ostrość, a oko zajmowało subiektywnie duży obszar to algorytm radził sobie bardzo dobrze. Metoda uzyskiwała dobre wyniki zarówno w przypadku gdy tęczówka oka była ciemna, jak i jasna. Negatywny wpływ na skuteczność detekcji miało występowanie na wycinku zdjęcia brwi, a w szczególności gdy były one w ciemnym odcieniu lub nałożony był na nie mocny makijaż. Podobnie intensywna barwa rzęs skutkowała obniżeniem dokładności wskazywania lokalizacji źrenicy. Lepsze rezultaty metoda uzyskiwała gdy na wycinku znajdowała się tylko i wyłącznie gałka oczna. W przypadku gdy źrenica miała jasny odcień wynikający np. z dużego natężenia padającego światła to uzyskiwane rezultaty nie były zadowalające. Jeśli algorytmowi udało się wskazać punkt należący do tęczówki, to w większości przypadków wynikiem był jej środek oraz źrenica. W ogólności, skuteczność tej metody opierała się w dużej mierze na występowaniu ciemnych punktów innych niż tęczówka i źrenica.

Metoda projekcji lepsze rezultaty detekcji wykazywała na obszarach oczu o małym kontraście i ostrości. Gorsza jakość miała pozytywny wpływ na jej działanie. Częstym zjawiskiem było wskazanie tylko jednej składowej poprawnie - poziomej lub pionowej. Algorytm ten w części przypadków wskazywał rzęsy zamiast źrenicy. Prawidłowe detekcji występowały przy bardzo wyraźnych przejściach między białą częścią oka, a tęczówką. Jeśli region źrenicy był ciemniejszy niż reszta obszaru oka to detekcja uzyskiwała lepsze rezultaty. Występowanie ramek okularów przeszkadzały w prawidłowym wskazaniu szukanego punktu.

Analiza krawędzi całkowicie nie radziła sobie z detekcją w przypadku gdy występowało dużo wyraźnych konturów innych niż tęczówka lub źrenica. Dobre rezultaty osiągane były jeśli obszar oka był jasny i naświetlony. W przeciwieństwie do projekcji metoda ta lepiej radziła sobie przy wycinkach dobrej jakości i o dużej ostrości. Porównując ten algorytm do dwóch pozostałych jego zachowanie wydawało się dużo bardziej losowe, ponieważ w wielu przypadkach trudno było wskazać powody zwróconej złej lokalizacji. Wyniki procentowe jak i subiektywne odczucia wskazują jednoznacznie, że metoda oparta na analizie krawędzi radziła sobie najgorzej.

12.1.5. Badanie szybkości detekcji

Najszybszym okazał się algorytm oparty na funkcji projekcji. Przetworzenie jednego wycinka oka zajęło mu średnio $2.0 * 10^{-5}$ s. Dwukrotnie dłużej trwało wykonanie metody z użyciem progowania opartego o dystrybuantę. Najwolniej natomiast działała analiza krawędzi uzyskując czas $5.6 * 10^{-5}$.

Wyniki zdają się oddawać naturę poszczególnych metod, ponieważ projekcja wymaga maksymalnie dwóch przejść całej macierzy obrazu. Natomiast algorytm oparty na analizie krawędzi wykorzystuje filtry oraz *Canny*, dlatego ma największą złożoność czasową.

Wszystkie metody są bardzo szybkie (rzad wielkości 10^{-5} s) i nawet najwolniejsza

Tabela 12.3. Czas przetwarzania algorytmów detekcji źrenic na zbiorze danych

	Całkowy czas przetwarzania	Średni czas przetwarzania pojedynczej iteracji	Średni czas przetwarzania pojedynczego zdjęcia
PF	0,272 s	0,00272 s	0,00002001 s
EA	0,765 s	0,00765 s	0,00005627 s
CDF	0,637 s	0,00637 s	0,00004684 s

z nich mogłyby być wykonana prawie 18 tyś. razy na sekundę. Przekłada się to na zerowe obciążenie całej aplikacji, która wykrywając jedynie twarz osiąga raptem 16 klatek na sekundę dla barw RGB (patrz rozdz. 5.3.2). z tego powodu podczas wyboru najlepszego algorytmu detekcji źrenic nie była brana pod uwagę szybkość poszczególnych metod, a jedynie ich skuteczność.

12.2. Testowanie na obrazie z kamery

Przetestowanie skuteczności detekcji na obrazie z kamery i przedstawienie ich w postaci liczbowej byłoby bardzo trudne. z tego powodu algorytm został przebadany poprzez obserwacje zachowania detekcji na podglądzie na żywo. Badania były przeprowadzone w takich samych warunkach jak pozostałe elementy systemu. Wyniki są subiektywnym odczuciem autora projektu i przedstawione w formie opisowej.

Ze względu na wyniki szybkościowe detekcji źrenic (patrz rozdz. 12.1.5), które jednoznacznie wykazały marginalny czas przetwarzania wszystkich algorytmów, testy czasowe na obrazie z kamery na żywo zostały uznane za niepotrzebne i pominięte.

12.2.1. Badanie skuteczność detekcji

Podobnie jak w testach przeprowadzonych na statycznych zdjęciach CDF uzyskał subiektywnie najlepsze rezultaty. We wszystkich scenariuszach sprawdził się wystarczająco. Nawet w momencie gdy nie wykrywał dobrze środka źrenicy to wskazania znajdowały się w obszarze tęczówki. Pracował stabilnie zarówno w centralnym położeniu oka jak i zwróconym w bok.

Algorytm PF radził sobie dobrze tylko połowicznie. Najlepsze rezultaty osiągnął w przypadku światła padającego zza użytkownika. Wtedy zarówno centralne jak i boczne położenie był prawidłowo wykrywane. w pozostałych badaniach dobre wyniki osiągał gdy oczy był skierowane w bok, a w przypadku spojrzenia wprost przed siebie lokalizacja podawana była chaotycznie. Test w ciemnym pomieszczeniu potwierdził, że metoda dobrze radzi sobie przy małym kontraście i jasności.

Analiza krawędzi w przypadku obrazu na żywo w ogóle się nie sprawdziła. w żadnych warunkach detekcja nie była wystarczająca na potrzeby pracy dyplomowej. Przez większość czasu zwracaną lokalizacją były rogi obszaru oczu.

12.3. Wybór algorytmu detekcji źrenic

Najskuteczniejsza okazała się metoda oparta na progowaniu z użyciem dystrybuanty. w teście na zbiorze danych 83% wykrytych punktów znajdowało się wewnątrz tęczówki. Subiektywnie, najlepsze i wystarczające rezultaty uzyskała również w badaniu obrazu na żywo z kamery.

Szybkość wszystkich algorytmów stała na tak wysokim i marginalnym poziomie, że nie była brana jako czynnik wpływający na wybór algorytmu.

Z powodu wykrywalności źrenic rozwiązanie CDF zostało wybrane jako podstawowe i używany w projekcie.

13. Detekcja mrugania

Celem stwierdzenia czy wystąpiło mrugniecie zostało zaimplementowane proste rozwiązanie polegające na analizie określonej ilości klatek wstecz. Wykorzystywany jest fakt czy w danej klatce oczy były zamknięte czy otwarte i zmianę tego stanu system informuje o ewentualnym wystąpieniu zdarzenia.

13.1. Algorytm detekcji mrugania

1. Utworzenie tablicy T przechowującej a wartości typu prawda/fałsz oznaczających czy oczy były zamknięte czy otwarte w a ostatnich klatkach obrazu.
2. Usunięcie najstarszej wartości z tablicy T jeśli jest pełna
3. Dodanie do tablicy T nowej flagi zamkniętych oczu w danej klatce
4. Analiza zawartości tabeli T . Jeśli wszystkie pola mają taką samą wartość to ustalenie stanu S_i na tę wartość i porównanie z poprzednim stanem S_{i-1} :
 - Jeśli S_i jest inny niż S_{i-1} i stan S_i oznacza oczy zamknięte to wystąpiło mrugnięcie
5. Powrót do punktu 3

Uwaga 1. Parametr a oznacza ile klatek z rzędu musi występować dany stan oka by uznać go za wiarygodny (służy odrzuceniu pojedynczych błędnych wskazań i szumów).

Uwaga 2. Stan o wartości prawda oznacza oczy zamknięte.

13.2. Testowania detekcji mrugania na obrazie na żywo z kamery

Przeprowadzone zostały testy detekcji ruchu oczu na obrazie na żywo. w każdych warunkach zostały wykonane trzy testy powtórzone dwukrotnie:

- mrugnięcie lewym okiem
- mrugnięcie prawym okiem
- mrugnięcie oboma oczami na raz

Poszczególne czynności w ramach jednego testu wykonane były 10 razy.

Tabela 13.1. Skuteczność wykrywania mrugania na obrazie na żywo z kamery

Warunki	1.	2.	3.	4.
Lewym okiem	9	10	10	9
Prawym okiem	10	10	8.5	7
Oboma oczami	10	10	10	10

Skuteczność detekcji mrugania okazała się bardzo dobra, lepsza niż oczekiwana. w przypadku testu gdy czynność została wykonana oboma oczami na raz została wykryta ona za każdym razem. w pozostałych przypadkach osiągała wynik na poziomie

13. Detekcja mrugania

średnio 92% skuteczności. Takie rezultaty wydają się być całkowicie wystarczające i zadowalające z perspektywy pracy dyplomowej.

14. Detekcja ruchu gałek ocznych

Do detekcji ruchu gałek ocznych wykorzystywany jest identyczny mechanizm jak w przypadku wykrywania mrugania. Analizowane są położenia żrenic w określonej ilości klatek obrazu, a na ich podstawie zgłoszana jest ewentualna zmiana położenia oczu, a w konsekwencji ich ruch.

14.1. Algorytm detekcji ruchu oczu

1. Utworzenie tablicy T przechowującej a wartości oznaczających położenie oka w a ostatnich klatkach obrazu.
2. Usunięcie najstarszej wartości z tablicy T jeśli jest pełna
3. Dodanie do tablicy T nowego położenia oka danej klatce
4. Analiza zawartości tabeli T i porównanie z poprzednim stanem S_{i-1} :
 - a) Jeśli wszystkie pola mają taką samą wartość to ustalenie stanu S_i na tę wartość.
Inaczej skok do punktu 6
 - b) Jeśli stan S_i jest inny niż S_{i-1} to:
 - i. Jeśli stan S_{i-1} oznaczał oczy zamknięte to nastąpiło ich otwarcie
 - ii. Jeśli stan S_{i-1} oznaczał położenie otwartych oczu to S_i oznacza ich nowe położenie
 - iii. Jeśli stan S_i oznacza oczy zamknięte to nastąpiło ich zamknięcie
5. Powrót do punktu 3

Uwaga 1. Parametr a oznacza ile klatek z rzędu musi występować dany stan oka by uznać go za wiarygodny (służy odrzuceniu pojedynczych błędnych wskazań i szumów).

Uwaga 2. Stan może przyjmować wartości: lewo, środek, prawo oraz zamknięte. Pierwsze trzy oznaczają położenie tęczówki i żrenicy (kierunek patrzenia użytkownika).

14.2. Testowania detekcji ruchu oczu na obrazie na żywo z kamery

Sposób badania był identyczny jak w przypadku detekcji mrugania (patrz rozdz. 13.2). Testowane były zdarzenia ruchu oczu w lewo, w prawo i na przemian.

Tabela 14.1. Skuteczność wykrywania ruchu oczu na obrazie na żywo z kamery

Warunki	1.	2.	3.	4.
W lewo	10	10	10	8.5
W prawo	9.5	9.5	10	9
Na przemian	10	10	10	9.5

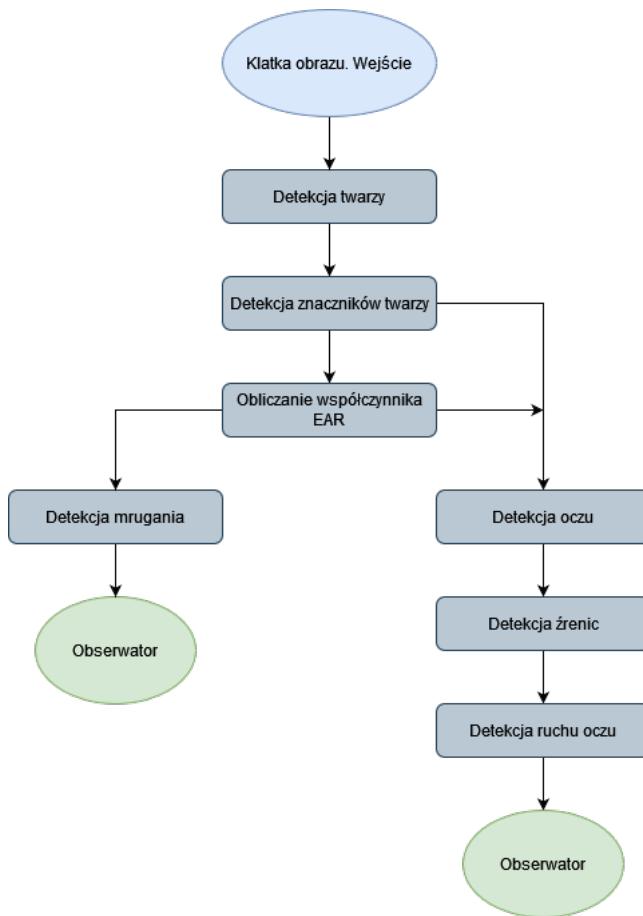
W każdym teście wykrytych zostało co najmniej 8 na 10 zdarzeń. Aż w 14 przypadkach osiągając 100% wykrywalności. Sumarycznie wykrywalność uplasowała się na poziomie 95%.

14. Detekcja ruchu gałek ocznych

mie średnio 96,6%. Podobnie jak w detekcji mrugania rezultat ten wydaj się całkowicie adekwatny na potrzeby pracy dyplomowej i projektu aplikacji.

15. Architektura

Przepływ danych między kolejnymi detektorami wykorzystywanymi w aplikacji został przedstawiony na diagramie 15.1.



Rysunek 15.1. Diagram przepływu detekcji i danych

15.1. Wzorce projektowe

W ramach implementacji projektu zostały zastosowane dwa istotne dla działania aplikacji wzorce projektowe: wstrzykiwania zależności oraz obserwatora.

15.1.1. Wstrzykiwanie zależności

W myśl tego rozwiązania komponenty klasy nie są tworzone bezpośrednio w niej, lecz poza nią, a następnie za pomocą metody (ang. setter injection) lub konstruktora (ang. constructor injection) są *wstrzykiwane* i przypisywane do właściwości tej klasy.

Zastosowanie tego wzorca pozwala zmniejszyć sprzężenia między klasami, co jest dobrą praktyką programowania.

15.1.1.1. Przykład użycia Istnieje klasa detektora twarzy, która podczas swojego działania wykorzystuje algorytm detekcji, którego instancja jest przechowywana w polu tej klasy.

Przykładowo kod tej klasy gdy instancja algorytmu detekcji tworzona jest w jej konstruktorze może wyglądać tak:

Listing 1. Przykład klasy bez użycia wzorca wstrzykiwania zależności

```
1 class FaceDetector {  
2     private final FaceDetectionAlgorithm detectionAlgorithm;  
3  
4     public FaceDetector() {  
5         detectionAlgorithm = new FaceDetectionAlgorithm();  
6     }  
7 }
```

Natomiast, kod w przypadku gdy wykorzystany jest wzorzec wstrzykiwania zależności, a instancja algorytmu detekcji tworzona jest poza nią może wyglądać następująco:

Listing 2. Przykład klasy z użyciem wzorca wstrzykiwania zależności

```
1 class FaceDetector {  
2     private final FaceDetectionAlgorithm detectionAlgorithm;  
3  
4     public FaceDetector(FaceDetectionAlgorithm detectionAlgorithm) {  
5         this.detectionAlgorithm = detectionAlgorithm;  
6     }  
7 }
```

15.1.1.2. Wykorzystanie w projekcie W projekcie pracy dyplomowej wzorzec wstrzykiwania zależności występuje w następujących detektorach:

- Twarz
- Znaczników
- Oczu
- Źrenic

W wypadku tych klas wstrzykiwane są do nich wybrane metody detekcji poszczególnych części twarzy. Dzięki zastosowaniu takiego rozwiązania istnieje łatwa możliwość zmiany algorytmu bez potrzeby ingerowania i dostosowania całego systemu, ponieważ implementują one interfejs z metodą, którą wywołuje dany detektor.

15.1.2. Obserwator

Polega na połączeniu klas w strukturę obserwowany-obserwatorzy. Ten pierwszy przechowując odwołania do wszystkich obiektów może je poinformować o zmianach własnego stanu.

Zaletą stosowania tego rozwiązania są *luźne powiązania* między obiektem obserwowanym, a obserwatorem. Dodatkowo pozwala on na dynamiczne zmienianie zależności między tymi obiektami w czasie wykonywania programu.

15.1.2.1. Wykorzystanie w projekcie W stworzonej aplikacji wzorzec obserwatora został zastosowany w przypadku wykrywania detekcji ruchu oczu i mrugania. Gdy zostanie stwierdzone wystąpienie któregoś z tych zdarzeń odpowiednia klasa wysyła informację o tym do obserwatorów. Mogą one wtedy odpowiednio zareagować na te zmiany. Pozwala to na łatwe dodawanie nowych komponentów, które swoje działanie uzależniają od mrugania i ruszania oczami przez użytkownika.

16. Podsumowanie

16.1. Wykonane prace

Na początku wykonano przegląd dostępnych rozwiązań mających na celu detekcję twarzy i jej punktów charakterystycznych, oczu oraz źrenic. Część z nich dostępna była w bibliotekach OpenCV (wraz z dodatkowymi modułami) oraz Dlib. Wykorzystując krzyżową komplikację udało się je dostosować do użytku w projekcie.

Wybrane algorytmy zostały przetestowane pod kątem skuteczności oraz złożoności czasowej. Badania były prowadzone zarówno przetwarzając statyczne zdjęcia, jak i kolejne klatki obrazu na żywo z kamery. Dla każdego zdjęcia zostały przygotowane i opisane ręcznie takie informacje jak: położenie twarzy czy oczu, co pozwoliło na powtarzalne wyniki liczbowe testów. Na podstawie przeprowadzonych badań wybierano najlepsze metody na potrzeby projektu. Okazały się nimi:

- do detekcji twarzy - Histogram zorientowanych gradientów + maszyna wektorów nośnych
- do detekcji facemarków - Kazemi
- do detekcji oczu - Eye Aspect Ratio
- do detekcji źrenic - CDF

Niektóre z metod były optymalizowane i dostrajane z użyciem autorskich rozwiązań, które pozwoliły na uzyskanie lepszych rezultatów niż surowe implementacje. Zaproponowane zmiany wpływały głównie na skuteczność detekcji.

Finalnie, udało się zrealizować założony cel pracy tworząc aplikację na urządzenia z systemem Android, która analizując twarz użytkownika reaguje w zadany sposób na jego gesty. Celem demonstracyjnym przygotowano dwa scenariusze - informowanie użytkownika, że udało się prawidłowo wykryć jego mrugnięcie oraz prostą galerię zdjęć, w której kolejne obrazy możemy przeglądać poruszając oczami w odpowiednią stronę.

16.2. Zdobyta wiedza i wyciągnięte wnioski

Czas poświęcony na prace dyplomową pozwolił poznać wiele aspektów zarówno z dziedziny przetwarzania obrazów, jak i tworzenia oprogramowania na systemy Android.

Przystępując do pracy dyplomowej autor nie miał nigdy wcześniej styczności z wytwarzaniem oprogramowania na ten system operacyjny, co wiązało się z koniecznością pozyskania wiedzy całkowicie od podstaw w tym segmencie. Chociaż zdobyte podczas prac nad projektem umiejętności pozwalają już na pisanie aplikacji na ten system operacyjny, to ilość wiedzy, która jest jeszcze do zdobycia jest ogromna.

Integracja wybranych bibliotek ze środowiskiem Android obciążona była wieloma problemami związanymi z krzyżową komplikacją z ich języków natywnych do Javy. Dodatkowo wymagały one poznania i wykorzystania komponentu JNI, który pozwolił wprowadzić elementy języka C++ w projekcie. Praca nad projektem wymagała również poznania

i zaznajomienia się z możliwościami oferowanymi przez wybrane biblioteki oraz z ich działaniem. Niewątpliwie są to bardzo popularne rozwiązania na rynku, więc zdobyta wiedza może być wykorzystana podczas dalszego rozwoju autora w dziedzinie przetwarzania obrazu.

Mimo, że finalnie udało się uzyskać wystarczające wyniki czasowe na testowanym urządzeniu to można wysnuć tezę, że urządzenia mobilne z systemem Android nie są jeszcze w pełni gotową platformą na analizę obrazu w czasie rzeczywistym. Wykorzystywany sprzęt w czasie rozpoczęcia prac był jednym z najnowocześniejszych i najwydajniejszych modeli dostępnych na rynku. A tego powodu można przypuszczać, że na przeciętnym urządzeniu ilość osiąganych klatek na sekundę mogłaby nie być wystarczająca. Dodatkowo bardzo dużym problemem jest brak kompatybilności niektórych rozwiązań z kartami graficznymi urządzeń mobilnych, czego konsekwencją jest wykonywanie algorytmów przez jednostki obliczeniowe. Z tego powodu niemożliwe było wykorzystanie np. sieci konwolucyjnych, które na kartach graficznych działają z bardzo dużą częstotliwością. Jednakże rynek urządzeń mobilnych rozwija się bardzo szybko i w najbliższej przyszłości wydajne modele będą standardem, co pozwoli na powszechnie stosowanie przetwarzania obrazu na żywo. Możliwe też, że wykonanie takich algorytmów będzie przeniesione do zyskującej ogromną popularność chmury, a wtedy mocne urządzenia mobilne nie będą nawet warunkiem koniecznym.

16.3. Możliwość rozwoju i dalszych badań

Aplikacja i projekt pozostawia możliwość dalszego rozwoju. Zarówno przez implementację porównanie kolejnych algorytmów do detekcji opisanych już elementów, jak i przez dodawanie analizy innych fragmentów twarzy i nie tylko.

Inne metody mogą okazać się zarówno bardziej skuteczne, jak również i szybsze. Możliwe jest, że w zależności od sytuacji różne rozwiązania dadzą znacznie odmienne wyniki. Z tego względu można pokusić się o dostosowanie wyboru algorytmu zależnie od warunków, celem osiągnięcia jak najlepszych rezultatów. Dzięki zaproponowanej architekturze oraz zastosowaniu wzorca projektowego *wstrzykiwanie zależności* zmiana jednego algorytmu danej detekcji na inny jest bardzo prosta, co pozwoli na szybkie wykorzystanie nowych rozwiązań.

Detekcję źrenic można spróbować oprzeć na sieciach neuronowych i uczeniu maszynowym zamiast na klasycznych metodach przetwarzania obrazu. Może się okazać, że kosztem wydajności zyskamy większą skuteczność.

Jednym z pomysłów na nowe elementy może być wykrywanie uśmiechu i emocji użytkownika. Aplikacja reagowałaby na zadowolenie czy zmiany humoru. W dobie powszechnych mediów społecznościowych mogłoby to być wykorzystane celem automatycznego opiniowania zdjęć, na które użytkownik patrzy w zależności od tego czy się uśmiecha lub czy jest radosny.

Bibliografia

- [1] OpenCV. “Home: OpenCV”. (), adr.: <https://opencv.org/> (term. wiz. 30.10.2021).
- [2] WikiPedia. “nVidia CUDA”. (2021), adr.: <https://pl.wikipedia.org/wiki/CUDA> (term. wiz. 31.10.2021).
- [3] OpenCV. “OpenCV-contrib”. (), adr.: https://github.com/opencv/opencv_contrib (term. wiz. 24.10.2021).
- [4] ——, (), adr.: https://github.com/opencv/opencv_contrib/tree/master/modules/face (term. wiz. 12.11.2021).
- [5] D. E. King. “Dlib C++ Library”. (2021), adr.: <http://dlib.net/> (term. wiz. 01.03.2022).
- [6] Google. “CameraX overview”. (2021), adr.: <https://developer.android.com/training/camerax> (term. wiz. 30.08.2021).
- [7] *Android Jetpack*. (term. wiz. 01.03.2022).
- [8] Oracle. “Java Native Interface”. (2021), adr.: <https://docs.oracle.com/javase/8/docs/technotes/guides/jni/> (term. wiz. 01.02.2022).
- [9] A. Yanamandra. “Young and Old Images Dataset”. (2019), adr.: <https://www.kaggle.com/abhishekryana/young2old-dataset> (term. wiz. 18.10.2021).
- [10] OpenCV. “Class CascadeClassifier”. (), adr.: <https://docs.opencv.org/3.4.15/javadoc/org/opencv/objdetect/CascadeClassifier.html> (term. wiz. 30.10.2021).
- [11] P. Viola i M. Jones, “Rapid object detection using a boosted cascade of simple features”, w *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, t. 1, 2001, s. I–I. DOI: 10.1109/CVPR.2001.990517.
- [12] G. S. Behera. “Face Detection with Haar Cascade”. (2020), adr.: <https://towardsdatascience.com/face-detection-with-haar-cascade-727f68dafd08> (term. wiz. 30.10.2021).
- [13] A. Rosebrock. “OpenCV Haar Cascades”. (2021), adr.: <https://www.pyimagesearch.com/2021/04/12/opencv-haar-cascades/> (term. wiz. 25.09.2021).
- [14] A. Obukhov, “Chapter 33 - Haar Classifiers for Object Detection with CUDA”, w *GPU Computing Gems Emerald Edition*, ser. Applications of GPU Computing Series, W.-m. W. Hwu, red., Boston: Morgan Kaufmann, 2011, s. 517–544, ISBN: 978-0-12-384988-5. DOI: <https://doi.org/10.1016/B978-0-12-384988-5.00033-4>. adr.: <https://www.sciencedirect.com/science/article/pii/B9780123849885000334>.
- [15] R. Lienhart. “Haarcascade Frontalface Default”. (), adr.: https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_default.xml (term. wiz. 20.09.2021).
- [16] J. E. C. Cruz, E. H. Shiguemori i L. N. F. Guimarães, “A comparison of Haar-like, LBP and HOG approaches to concrete and asphalt runway detection in high resolution imagery”, 2016.

16. Bibliografia

- [17] "LBP cascade frontalface model". (), adr.: https://github.com/opencv/opencv/blob/master/data/lbpcascades/lbpcascade_frontalface.xml (term. wiz. 01.10.2021).
- [18] N. Dalal i B. Triggs, "Histograms of oriented gradients for human detection", w *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, t. 1, 2005, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177.
- [19] ichi.pro. "Delikatne wprowadzenie do histogramu zorientowanych gradientów". (), adr.: <https://ichi.pro/pl/delikatne-wprowadzenie-do-histogramu-zorientowanych-gradientow-279201309061802> (term. wiz. 31.10.2021).
- [20] S. Mallick. "Histogram of Oriented Gradients explained using OpenCV". (2016), adr.: <https://learnopencv.com/histogram-of-oriented-gradients/> (term. wiz. 31.10.2021).
- [21] M. Fabien. "A full guide to face detection". (2019), adr.: <https://maelfabien.github.io/tutorials/face-detection/#4-which-one-to-choose-> (term. wiz. 31.10.2021).
- [22] M. S. Nixon i A. S. Aguado, *Feature Extraction & Image Processing for Computer Vision*, 3 wyd. Academic Press, 2013, ISBN: 0123965497.
- [23] R. Gandhi. "Support Vector Machine — Introduction to Machine Learning Algorithms". (2018), adr.: <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47> (term. wiz. 31.10.2021).
- [24] M. Mamczur. "Jak działają konwolucyjne sieci neuronowe (CNN)". (), adr.: <https://mirosławmamczur.pl/jak-działają-konwolucyjne-sieci-neuronowe-cnn/> (term. wiz. 20.10.2021).
- [25] D. E. King, "Max-Margin Object Detection", *CoRR*, t. abs/1502.00046, 2015. arXiv: 1502.00046. adr.: <http://arxiv.org/abs/1502.00046>.
- [26] openCV. "OpenCV: Deep Neural Network module". (2021), adr.: https://docs.opencv.org/3.4.15/d6/d0f/group_dnn.html (term. wiz. 10.09.2021).
- [27] Y. Jia, E. Shelhamer, J. Donahue i in., "Caffe: Convolutional Architecture for Fast Feature Embedding", *MM 2014 - Proceedings of the 2014 ACM Conference on Multimedia*, czer. 2014. DOI: 10.1145/2647868.2654889.
- [28] S. Mallick. "Face Detection comparison - models". (), adr.: <https://github.com/spmallick/learnopencv/blob/master/FaceDetectionComparison/models/> (term. wiz. 28.10.2021).
- [29] L. P. LLC. "Always Bet on Red". (), adr.: <https://pl.pinterest.com/pin/169025792249522554/> (term. wiz. 20.09.2021).
- [30] S. Ren, X. Cao, Y. Wei i J. Sun, "Face Alignment at 3000 FPS via Regressing Local Binary Features", w *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, s. 1685–1692. DOI: 10.1109/CVPR.2014.218.
- [31] L. Kurnianggoro. (), adr.: <https://github.com/kurnianggoro/GSOC2017/blob/master/data/lbfmodel.yaml> (term. wiz. 12.11.2021).

- [32] V. Le. "HELEN dataset". (2012), adr.: <http://www.ifp.illinois.edu/~vuongle2/helen/> (term. wiz. 12.11.2021).
- [33] V. Kazemi i J. Sullivan, "One millisecond face alignment with an ensemble of regression trees", w *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, s. 1867–1874. DOI: 10.1109/CVPR.2014.241.
- [34] C. Sagonas, E. Antonakos, G. Tzimiropoulos, S. Zafeiriou i M. Pantic, "300 Faces In-The-Wild Challenge: database and results", *Image and vision computing*, t. 47, s. 3–18, mar. 2016, ISSN: 0262-8856. DOI: 10.1016/j.imavis.2016.01.002.
- [35] N. Kamarudin, N. A. Jumadi, L. M. Ng i in., "Implementation of Haar Cascade Classifier and Eye Aspect Ratio for Driver Drowsiness Detection Using Raspberry Pi", *Universal Journal of Electrical and Electronic Engineering*, t. 6, s. 67–75, grud. 2019. DOI: 10.13189/ujeee.2019.061609.
- [36] A. Rosebrock. "Eye blink detection with OpenCV, Python, and dlib". (2017), adr.: <https://www.pyimagesearch.com/2017/04/24/eye-blink-detection-opencv-python-dlib/> (term. wiz. 13.09.2021).
- [37] S. Hameed. "Haar eye detection model". (), adr.: https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_eye.xml (term. wiz. 14.11.2021).
- [38] A. Rosebrock. "Detect eyes, nose, lips, and jaw with dlib, OpenCV, and Python". (2017), adr.: <https://www.pyimagesearch.com/2017/04/10/detect-eyes-nose-lips-jaw-dlib-opencv-python/> (term. wiz. 14.11.2021).
- [39] Nikolaj. "2018". (2013), adr.: https://www.zastavki.com/eng/Girls/Beautyful_Girls/wallpaper-126539.htm (term. wiz. 20.09.2021).
- [40] M. Asadifard i J. Shanbezadeh, "Automatic Adaptive Center of Pupil Detection Using Face Detection and CDF Analysis", w *Proceedings of the International MultiConference of Engineers and Computer Scientists 2010 Vol I*, IMCES, Hong Kong, 2010, March 17–19.
- [41] M. Cieśla i P. Kozioł, *Eye Pupil Location Using Webcam*, Wydział Fizyki, Astronomii i Informatyki Stosowanej, Uniwersytet Jagielloński, ul. Reymonta 4, 30-059, Kraków, Polska, 2012.
- [42] Z.-H. Zhou i X. Geng, "Projection functions for eye detection", *Pattern Recognition*, t. 37, s. 1049–1056, maj 2004. DOI: 10.1016/j.patcog.2003.09.006.

Spis rysunków

4.1	Obliczane cechy w modelu Haar. Źródło: [12]	17
4.2	Działanie filtrowania detekcji twarzy w oparciu o położenie twarzy w centralnej części zdjęcia.	19
4.3	Działanie filtrowania detekcji twarzy w oparciu o odległość wykrytego obszaru poza zdjęcie.	19
4.4	Działanie filtrowania detekcji twarzy w oparciu o wielkość wykrytego obszaru. Źródło zdj.: [29]	20
5.1	Oczekiwany obszar detekcji twarzy.	21
6.1	Przykład zdjęcia z naniesionymi punktami charakterystycznymi twarzy	29
8.1	Teoretyczny rozmieszczenie znaczniki wokół oczu wraz z naniesionymi połączeniami do obliczenia EAR	34
8.2	Rozkład wartości współczynnika EAR oczu otwartych na zdjęciach ze zbioru danych	35
8.3	Rozkład wartości współczynnika EAR oczu zamkniętych na zdjęciach ze zbioru danych	35
8.4	Wartość współczynnika EAR dla oczu otwartych na obrazie z kamery	36
8.5	Wartość współczynnika EAR dla oczu czasowo zamkniętych na obrazie z kamery	36
8.6	Wartość współczynnika EAR podczas mrugania na obrazie z kamery	37
9.1	Wykorzystanie znaczników twarzy do detekcji obszaru oczu	38
9.2	Efekt filtrowania obszarów detekcji oczu	39
9.3	Obcięcie obszaru detekcji oczu zgodnie z dobranymi wcześniej parametrami .	40
9.4	Odrzucenie błędnych rezultatów detekcji oczu po dodatkowym obcięciu obszaru. Źródło pierwszego zdj.: [39]	40
10.1	Przybliżony obszar oczu, który jest oczekiwany wynikiem algorytmu	42
11.1	Kolejne etapy wykrywania źrenic metodą CDF	48
11.2	Kolejne etapy wykrywania źrenic metodą PF	50
11.3	Kolejne etapy wykrywania źrenic metodą EA	51
15.1	Diagram przepływu detekcji i danych	61

Spis tabel

5.1	Skuteczność algorytmów detekcji twarzy dla obrazów RGB ze zbioru danych .	22
5.2	Skuteczność algorytmów detekcji twarzy dla obrazów w skali szarości ze zbioru danych	23
5.3	Czas przetwarzania algorytmów detekcji twarzy dla obrazów RGB ze zbioru danych	24

5.4 Czas przetwarzania algorytmów detekcji twarzy dla obrazów w skali szarości ze zbioru danych	25
5.5 Wpływ rozdzielczości zdjęcia na detekcję DNN	25
5.6 Wynik porównania sposobów filtrowania detekcji twarzy na zbiorze danych .	26
5.7 Skuteczność algorytmów detekcji twarzy dla obrazu na żywo z kamery	27
5.8 Szybkość algorytmów detekcji twarzy dla obrazu na żywo z kamery [klatki/s] .	27
7.1 Skuteczność algorytmów detekcji znaczników twarzy na zbiorze danych	30
7.2 Czas przetwarzania algorytmów detekcji znaczników twarzy na zbiorze danych	31
7.3 Szybkość algorytmów detekcji znaczników twarzy dla obrazu na żywo z kamery [klatki/s]	32
9.1 Skuteczność algorytmu detekcji oczu Haar z dodatkowym obcięciem i bez . .	40
9.2 Skuteczność algorytmu detekcji oczu wykorzystując znaczniki twarzy z dodatkowym zwiększeniem obszaru i bez na zbiorze danych	41
10.1 Skuteczność algorytmów detekcji oczu na zbiorze danych	43
10.2 Czas przetwarzania algorytmów detekcji oczu na zbiorze danych	45
10.3 Szybkość algorytmów detekcji oczu dla obrazu na żywo z kamery [klatki/s] .	46
12.1 Skuteczność algorytmu PF zależnie od funkcji projekcji na zbiorze danych .	53
12.2 Skuteczność algorytmów detekcji źrenic na zbiorze danych	53
12.3 Czas przetwarzania algorytmów detekcji źrenic na zbiorze danych	55
13.1 Skuteczność wykrywania mrugania na obrazie na żywo z kamery	57
14.1 Skuteczność wykrywania ruchu oczu na obrazie na żywo z kamery	59