

A G H

AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Praca dyplomowa

Animacja awatara na podstawie wyrazu twarzy
Animation of an avatar based on face expression

Autor: *Anna Gnoińska*
Kierunek studiów: *Informatyka*
Opiekun pracy: *dr inż. Piotr Szwed*

Kraków, 2022

Serdecznie dziękuję promotorowi dr inż. Piotrowi Szwedowi za ogromną pomoc i wielkie wsparcie w trakcie tworzenia niniejszej pracy dyplomowej. Wyrażam również ogromną wdzięczność w kierunku bliskich mi osób, za wszystkie słowa zachęty w pojawiających się chwilach zwątpienia.

Spis treści

1. Wstęp.....	7
1.1. Wprowadzenie.....	7
1.2. Cel pracy.....	8
1.3. Założenia projektu.....	8
1.4. Struktura pracy	9
2. Przedstawienie powiązanych algorytmów	11
2.1. Wykrywanie twarzy	11
2.1.1. Schemat działania	13
2.1.2. Klasyfikator kaskadowy z użyciem cech Haara.....	13
2.1.3. Klasyfikator SVM z użyciem deskryptora HOG	16
2.2. Identyfikacja punktów charakterystycznych	19
2.3. Triangulacja Delaunaya.....	20
3. Przegląd istniejących zastosowań.....	23
3.1. Technika deepfake	23
3.2. Dostępne aplikacje mobilne	24
3.2.1. Wombo.ai	24
3.2.2. Anyface: face animation	25
3.2.3. MotionPortrait.....	25
3.3. Wnioski	25
4. Wykorzystane narzędzia i technologie	27
4.1. Python.....	27
4.2. Biblioteki	29
4.2.1. OpenCV	29
4.2.2. imutils	29
4.2.3. dlib	30
4.2.4. scikit-image.....	30

4.3. Flask	30
4.4. Pycharm.....	31
5. Algorytm animacji awatara	33
5.1. Struktura programu.....	33
5.2. Wykrycie twarzy oraz punktów charakterystycznych.....	34
5.3. Triangulacja zbioru punktów.....	36
5.4. Transformacja trójkątów.....	37
5.5. Nałożenie maski	40
5.6. Aplikacja webowa	43
6. Ewaluacja i rezultaty	45
6.1. Sposób testowania	45
6.2. Walidacja z użyciem aplikacji.....	46
6.3. Walidacja na zbiorze danych.....	49
6.4. Wnioski	51
7. Podsumowanie	53
7.1. Weryfikacja początkowych założeń.....	53
7.2. Możliwy rozwój projektu	54

1. Wstęp

1.1. Wprowadzenie

Dzisiejszy świat niezwykle szybko się rozwija. Ciężko wyobrazić sobie wykonywanie codziennych czynności bez udogodnień technicznych, które otaczają nas co dnia. Jeszcze kilkanaście lat temu codzienność przeciętnej osoby wyglądała zupełnie inaczej. Okres ostatnich 30 lat przyniósł wiele zmian, bez których teraz nie wyobrażamy sobie normalnego funkcjonowania.

W obecnym czasie, z dnia na dzień, na rynek wprowadzane są nowe urządzenia i oprogramowanie mające na celu polepszenie komfortu naszego życia. Ludziom zależy na ciągłym usprawnianiu technologii, aby zautomatyzować pewne czynności i móc zaoszczędzić swój cenny czas. Wielu naukowców poświęca się pracy w celu odkrycia przełomowych rozwiązań.

Nie da się ukryć, że w ostatnich latach coraz więcej mówi się o ogromnym potencjale sztucznej inteligencji (ang. Artificial Intelligence), która zaczyna odgrywać znaczącą rolę w światowym rozwoju technicznym. Według źródeł wiele gałęzi przemysłu decyduje się na wprowadzanie systemów tzw. wąskiej sztucznej inteligencji. W tym momencie mamy z nimi styczność na każdym kroku [1].

Systemy te charakteryzują się wyuczoną umiejętnością wykonywania określonego zadania. Rodzaj inteligencji, który reprezentują, najczęściej stosuje się w rozpoznawaniu mowy, rekomendowaniu produktów, czy też wykrywaniu elementów na obrazie. Ostatnie z wymienionych zastosowań jest bezpośrednio związane z tematyką poruszaną w owej pracy dyplomowej.

Wykrywanie, rozpoznawanie i przetwarzanie obrazów (ang. Computer Vision) to dziedziny sztucznej inteligencji, które wykorzystują uczenie maszynowe, aby umożliwić komputerom po prawną interpretację i zrozumienie otaczającego nas świata - tak jak robią to ludzie. Poprzez wykorzystanie obrazów cyfrowych i odpowiednich modeli tworzone oprogramowanie potrafi dokładnie identyfikować, klasyfikować obiekty, a następnie na podstawie otrzymanych rezultatów wykonywać zdefiniowane akcje [2].

Jednym z wielu istniejących systemów bazujących na przetwarzaniu obrazów jest oprogramowanie rozpoznawania twarzy. Wiele osób korzysta z niego każdego dnia chociażby w celu odblokowania telefonu, komputera lub autoryzacji transakcji w banku. To narzędzie ma też szereg innych zastosowań, jest pewnego rodzaju bazą dla bardziej rozbudowanych programów. W

niniejszej pracy dyplomowej wykrywanie twarzy, będące podstawą wspomnianego wyżej algorytmu rozpoznawania twarzy, odegrało znaczącą rolę. Jest to jeden z etapów algorytmu animacji awatara.

Niewątpliwie szybki rozwój technologiczny, z którym aktualnie mamy do czynienia, oraz jego ukierunkowanie na jak najefektywniejsze wykorzystanie sztucznej inteligencji sprawiły, że temat pracy inżynierskiej, który zdecydowałem się realizować dotyczy właśnie tych zagadnień. W połączeniu z moim zainteresowaniem grafiką komputerową oraz przetwarzaniem obrazów powstała chęć zaimplementowania algorytmu animacji awatara.

Na rynku dostępne są programy, które implementują takowe algorytmy. Powstaje coraz więcej aplikacji oferujących animacje zdjęcia na podstawie filmu wideo. Niektóre portale społecznościowe udostępniają specjalne filtry, nakładki na zdjęcia działające na podobnej zasadzie. Jednak większość dostępnych programów nie oferuje wglądu w kod źródłowy, są to rozwiązania komercyjne, gdzie ta dostępność jest mocno ograniczona.

Motywacją do stworzenia własnego programu jest chęć rozwoju i poszerzenia swojej wiedzy w tej tematyce oraz próba odwzorowania działania istniejących rozwiązań, które nie są dostępne w formie otwartego oprogramowania.

1.2. Cel pracy

Celem pracy dyplomowej było opracowanie algorytmu animacji awatara. Metoda opiera się na identyfikacji punktów charakterystycznych (ang. facial landmarks) oznaczających położenie łuków brwiowych, powiek, oczu, nosa i warg. Na podstawie wyznaczonych punktów i ich przemieszczeń są obliczane przemieszczenia analogicznych punktów awatara, według których jego obraz jest poddany lokalnym transformacjom.

Algorytm został przedstawiony z wykorzystaniem nieskomplikowanej aplikacji webowej, która posłużyła jako narzędzie walidacji.

1.3. Założenia projektu

Założono, że projekt stworzony na potrzeby tejże pracy powinien spełniać określone wymagania, zgodne z poniższymi punktami:

- działanie algorytmu powinno opierać się na analizie punktów charakterystycznych, na podstawie której nastąpią lokalne transformacje obrazu awatara,
- identyfikacja punktów charakterystycznych powinna zostać zrealizowana z użyciem biblioteki face_recognition lub dlib,

- metodę należy przetestować na zarejestrowanych obrazach lub gotowych zbiorach danych,
- w celu walidacji algorytmu należy stworzyć prostą aplikację webową.

1.4. Struktura pracy

Niniejsza praca została podzielona na sześć rozdziałów. W pierwszym z nich zawarte zostało wprowadzenie, cel pracy oraz krótki opis założeń projektowych.

Drugi rozdział ma na celu przedstawienie pojęć istotnych dla tematu pracy. Ponadto są w nim omówione istniejące algorytmy ściśle powiązane z implementowanym rozwiązaniem.

W trzecim rozdziale znajduje się przegląd współczesnych zastosowań i przykłady dziedzin wykorzystujących algorytmy zbliżone do zaimplementowanego w tejże pracy.

W czwartym rozdziale zostały wymienione i krótko opisane narzędzia oraz technologie wykorzystane w trakcie tworzenia projektu.

Piąty rozdział dotyczy kwestii implementacji programu. Składa się on z sześciu podrozdziałów. Pierwszy z nich przedstawia ogólny schemat działania algorytmu, aby pokróćce zaznajomić odbiorcę z jego strukturą. Kolejne części zawierają szczegóły każdego z etapów animacji awatara. Natomiast ostatnia sekcja poświęcona jest aplikacji webowej mającej posłużyć jako sposób walidacji programu.

Szósty rozdział zawiera opis części ewaluacyjnej. Przedstawiono w nim efekty testów algorytmu pod kątem użyteczności programu oraz osiąganych rezultatów.

W siódmym rozdziale zawarto podsumowanie ogólne pracy, wnioski wyciągnięte podczas tworzenia projektu. Dodatkowo zamieszczono sekcję przedstawiającą możliwe ścieżki rozwoju stworzonego modelu.

2. Przedstawienie powiązanych algorytmów

Algorytm, którego implementacja jest celem tejże pracy inżynierskiej można podzielić na kilka istotniejszych etapów. Kluczową kwestią jest moment wykrycia twarzy na obrazie, identyfikacja punktów charakterystycznych oraz zastosowanie triangulacji. Właśnie ze względu na powyższe fakty, w kolejnych podrozdziałach opisane zostaną wymienione algorytmy, ich działanie oraz zastosowania.

2.1. Wykrywanie twarzy

Poprzez pojęcie wykrywania twarzy (ang. face detection) [3] rozumie się opartą na sztucznej inteligencji technologię identyfikującą ludzkie twarze na obrazie cyfrowym. Nawiązując do wspomnianych wyżej faktów, owa technika używana jest w wielu rozbieżnych dziedzinach. Nie zawsze posługujemy się nią świadomie, w celu łatwego dostępu do różnego rodzaju sprzętu elektronicznego. Przedstawiana technologia otacza nas wszędzie.

Często wykorzystuje się ją w monitoringu wideo, aby zapewnić bezpieczeństwo osobiste jak i narodowe. Co za tym idzie, odpowiednie służby polegają na niej w momencie egzekwowania prawa lub identyfikacji osób w grupie. W dziedzinie marketingu zaczęto korzystać z personalizacji reklam dla danego użytkownika. Poprzez integrację kamery internetowej z telewizorem, można gromadzić informacje o osobie, lokalizując ją za pomocą detekcji twarzy. Zbiera się dane dotyczące rasy, płci oraz przedziału wiekowego, aby następnie na tej podstawie dostosować wyświetlane reklamy. W nowo powstających aparatach fotograficznych wykrywanie twarzy używa się podczas automatycznego ustawiania ostrości. Nowoczesne urządzenia przez namierzenie uśmiechu dobierają odpowiedni moment zrobienia zdjęcia, przez co odbywa się to bez manualnego użycia przycisków [4].

Na przestrzeni lat metody służące wykrywaniu twarzy bardzo się rozwinęły. Na początku używano podstawowych technik przetwarzania obrazów, następnie oparto je na uczeniu maszynowym. Aktualnie ważną rolę w efektywności tejże techniki odgrywają sieci neuronowe, których zastosowanie zdecydowanie przyspiesza działanie programów implementujących ową metodę.

Istnieje wiele różnych propozycji algorytmów służących wykrywaniu twarzy. W celu uporządkowania i pogrupowania metod stosuje się klasyfikację zawierającą cztery podstawowe techniki wykrywania twarzy na obrazie. Dany podział został zaproponowany w 2002 roku przez Ming-Hsuan Yanga (Rys. 2.1) i obowiązuje do dziś.



Rys. 2.1. Techniki detekcji twarzy według Younga

Charakteryzację każdej z grup [5] można przedstawić w następujący sposób:

- metody bazujące na wiedzy - wykorzystujące ludzkie obeznanie w temacie elementów jakie zawiera przeciętna twarz,
- techniki używające niezmienników - algorytmy skupiają się na zlokalizowaniu cech strukturalnych twarzy, które są niezmienne ze względu na kąt, oświetlenie, pozycję twarzy,
- metody bazujące na poszukiwaniu wzorców - dzięki wielu zgromadzonym wzorom, odnośnie wyglądu twarzy lub jej konkretnych elementów, poszukuje się korelacji między schematem a obrazem wejściowym,
- metody bazujące na obrazie - model trenowany jest zestawem obrazów, które zawierają różnorodne ludzkie twarze, przedstawiane w zmiennych warunkach.

Zagadnienie poruszane w tym rozdziale jest bardzo rozległe, a rozwiązania i algorytmy istniejące na rynku bardzo różnorodne. Większość powstałych implementacji łączy najlepsze cechy z kilku metod, przez co nie ma możliwości przypisania ich do konkretnej klasyfikacji. Ze względu na szeroki zakres tej tematyki oraz dużą ilość istniejących rozwiązań w dalszej

części zostaną opisane dwa najpopularniejsze algorytmy, których implementacje są udostępniane przez biblioteki programistyczne.

2.1.1. Schemat działania

Dla omawianych poniżej algorytmów proces wykrywania twarzy odbywa się dwuetapowo. Początkowym etapem jest wyszkolenie klasyfikatora, którego zadaniem będzie wykrycie twarzy na obrazie.

Następnym jest uruchomienie detektora skanującego cały obraz w celu lokalizacji istotnych cech takich jak oczy, usta, nos czy brwi. Ich rozpoznanie możliwe jest przez zgromadzone informacje znajdujące się w wytrenowanym wcześniej modelu.

Większość algorytmów uzależnia swoją efektywność od wielkości danych, na których został przeszkolony klasyfikator. Trenowanie na dużych zbiorach danych poprawia zdolność algorytmu do określenia czy na danym obrazie znajduje się twarz [3].

2.1.2. Klasyfikator kaskadowy z użyciem cech Haara

Paul Viola i Michael Jones w 2001 roku zaproponowali technikę wykrywania obiektów opartą na działaniu klasyfikatora kaskad Haara (ang. Haar Feature-based Cascade Classifier) [6]. Pomimo wysokiej konkurencji ze strony sieci neuronowych, algorytm cieszy się ogromną popularnością po dziś dzień dzisiejszy. Jego zastosowanie okazało się być przełomem w dziedzinie wykrywania twarzy.

Działanie tej techniki łączy ze sobą poniższe koncepcje:

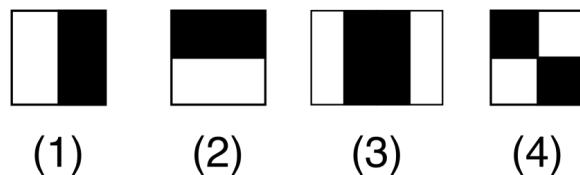
- poszukiwanie i wybór najdokładniejszych cech Haara,
- utworzenie zintegrowanych obrazów w celu szybkiego znalezienia danej cechy,
- użycie metody klasyfikacji Adaboost,
- zastosowanie klasyfikatora kaskadowego.

Podejście to jest oparte o wytrenowanie klasyfikatora kaskadowego na wielu pozytywnych i negatywnych obrazach w skali szarości. Przez pierwszy rodzaj rozumie się zdjęcia zawierające twarze, natomiast jako próbki negatywne określa się zdjęcia, na których nie znajduje się twarz ludzka. Według zaleceń autorów algorytmu obrazy powinny mieć wymiary 24×24 pikseli.

Pierwszym krokiem działania opisywanego modelu jest wydobycie ze wszystkich próbek odpowiednich cech Haara. Cechy te dotyczą zmian wartości kontrastu pomiędzy prostokątnymi grupami pikseli. W tym celu wykorzystuje się tak zwane funkcje Haara.

Są to kombinacje prostokątów o takich samych wymiarach (ciemnych i jasnych), pozwalające na detekcję konkretnych elementów. Funkcje dzielimy na trzy grupy, ze względu na ilość

prostokątów (2, 3, 4) tworzących daną cechę. Na Rys. 2.2 przedstawiono funkcje używane w tym algorytmie, wykorzystywane do wykrywania krawędzi na obrazie (1, 2) oraz prostych (3) i skośnych linii (4).



Rys. 2.2. Przykładowe funkcje Haara [7]

Każda funkcja Haara przemierza piksel po pikselu dany obraz. Dla każdego regionu (ciemnego jak i jasnego), który obejmuje owa cecha, sumowana jest wartość pikseli znajdujących się w tych obszarach. Następnie obliczana zostaje różnica między tymi sumami. Na podstawie tej różnicy, wyszukuje się lokalizację na obrazie, dla której dana cecha jest najbardziej odpowiednia.

Przykładowo, na Rys. 2.3 użyto funkcji złożonej z trzech prostokątów, o rozmiarze mającym na celu wykrycie oczu. Możliwe jest to poprzez fakt, iż obszar oczu jest ciemniejszy niż obszar nosa znajdujący się pomiędzy nimi. Cechy Haara są pewnego rodzaju wzorcem, dla którego należy znaleźć najlepsze dopasowanie na obrazie.

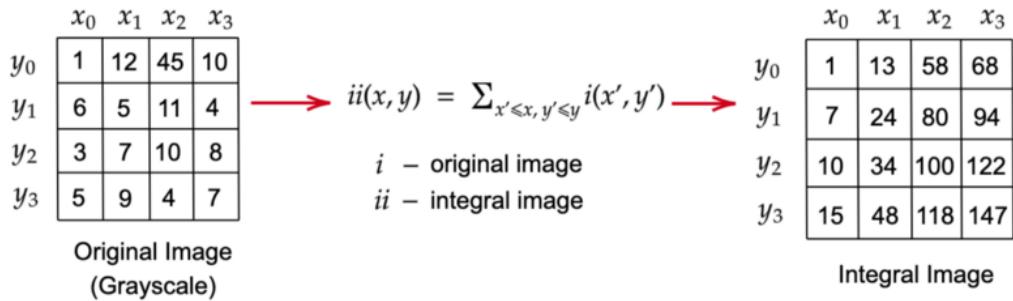


Rys. 2.3. Zastosowanie funkcji Haara do wykrycia oczu i obszaru nosa [6]

Próba dopasowania cech Haara dla każdego piksela zawartego na obrazie, we wszystkich możliwych rozmiarach, wymaga wykonania niewyobrażalnej liczby działań. Rozwiążaniem jest stworzenie zintegrowanego obrazu, dzięki czemu złożoność obliczeniowa programu staje się dużo korzystniejsza.

Obrazem integralnym nazywamy reprezentację obrazu, w którym dana wartość (x, y) równa jest sumie pikseli znajdujących się powyżej i na lewo od analizowanej lokalizacji (Rys. 2.4). Dana reprezentacja pozwala na przyspieszenie działań. Jest to skuteczny sposób obliczenia sumy wartości pikseli dla prostokątnego podzbioru rozważanego obrazu.

Kolejnym krokiem jest wybór cech, które są istotne dla konkretnych obszarów. Ten efekt można osiągnąć z pomocą algorytmu Adaboost (ang. Adaptive Boosting). Jest to algorytm

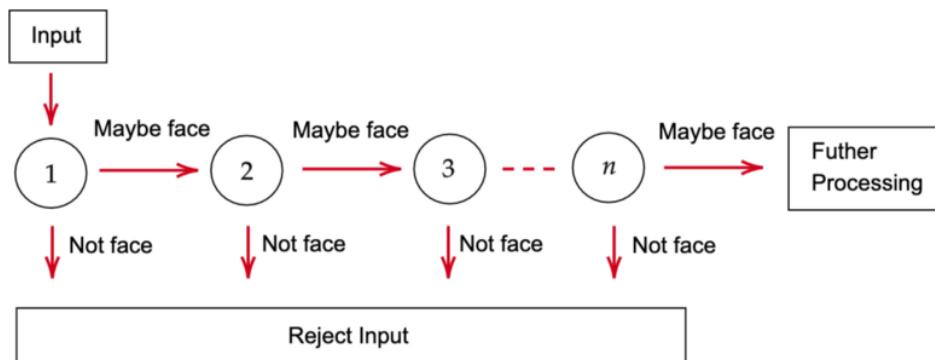


Rys. 2.4. Konwersja na obraz zintegrowany [8]

uczenia maszynowego służący do wyboru najlepszych funkcji z całego ich zbioru. Owa technika wzmacniająca, poprzez zastosowanie wszystkich możliwych cech na każdym obrazie treningowym osobno, wybiera te o najniższym błędzie dla danej iteracji. Algorytm trenuje silny klasyfikator na podstawie liniowej kombinacji słabych klasyfikatorów.

Po wyborze najlepszych cech spośród wszystkich możliwości pozostaje etap zastosowania klasyfikatora kaskadowego. Z definicji działa on wielostopniowo. Cechy wybrane jako kluczowe zostają podzielone na grupy, gdzie każda z nich odpowiada jednemu etapowi działania klasyfikatora. Pierwsze etapy zawierają niewiele funkcji, ale wybrane zostają te najbardziej pewne i charakterystyczne.

Funkcje są nakładane na każde okno o zadanych wymiarach wyodrębnionych na obrazie. W przypadku negatywnego rezultatu, tzn. niedopasowania jednej z cech sprawdzanych na konkretnym etapie, analizowane okno jest od razu odrzucane. Nie rozważa się dla niego funkcji zawartych w dalszych etapach. Jeśli natomiast okno przejdzie pomyślnie wszystkie etapy działania klasyfikatora kaskadowego, zostaje zaklasyfikowane jako obszar twarzy (Rys. 2.5).



Rys. 2.5. Klasyfikator kaskadowy [8]

W badaniu przeprowadzonym przez autorów algorytmu wybrano 6000 funkcji, które podzielono na 38 etapów klasyfikacji. Liczba cech w każdym z nich nie jest proporcjonalna, wynosi ona kolejno 1, 10, 25, 25, 50 funkcji dla pięciu pierwszych etapów. W początkowych etapach eliminujemy okna, w których nie ma elementów charakterystycznych twarzy. Oszczędza to zbędnych obliczeń i analiz.

Opisany w tym rozdziale algorytm jest wykorzystywany nie tylko do wykrywania twarzy na obrazach, ale także innego rodzaju obiektów, takich jak zwierzęta, tablice rejestracyjne czy całe ludzkie sylwetki. Cechuje go bardzo szybki czas działania. Jego dokładność nie jest aż tak wysoka, ze względu na podatność na fałszywe wykrycia. Popularność algorytmu, który został opisany w tym rozdziale, zdaje się dalej trwać, pomimo odkrycia wielu konkurencyjnych rozwiązań działających na podobnej zasadzie.

2.1.3. Klasyfikator SVM z użyciem deskryptora HOG

W 2005 roku pojawiła się kolejna przełomowa propozycja metody wykrywania obiektów, w tym twarzy. Jej autorami są Navneet Dalal i Bill Triggs, którzy wykazali, że do trenowania modelu można wykorzystać deskryptor HOG (ang. Histograms of Oriented Gradients) oraz klasyfikator SVM (ang. Support Vector Machine) [9].

Deskryptory wyodrębniają z obrazu przydatne informacje pomijając zbędne dane, pomagają zlokalizować konkretny, interesujący nas obiekt. W przypadku deskryptorów HOG ekstrakcja cech odbywa się za pomocą histogramu gradientów, używanych jako cechy analizowanych obrazów. Jako gradienty określamy pola wektorowe wskazujące kierunek, w którym można obserwować znaczącą zmianę w intensywności koloru.

W celu stworzenia deskryptora HOG dla obrazu należy wykonać kilka istotnych kroków [10]. Pierwszym z nich jest przygotowanie obrazu w odcieniach szarości, dla którego chcemy obliczyć histogram gradientów.

Według autorów algorytmu rozmiar zdjęcia powinno się zredukować do wymiarów 128×64 pikseli. Takie wymiary zostały wybrane ze względu na początkowe przeznaczenie algorytmu. Miał on służyć do detekcji pieszych. Po sukcesie i otrzymaniu dobrych wyników, skupiono się na detekcji twarzy. W tym przypadku również zdecydowano się na wybór zdjęć o owych wymiarach.

Dla każdego piksela znajdującego się na obrazie należy obliczyć gradient pionowy i poziomy. Najprostszym sposobem jest filtracja obrazu przez jądra zamieszczone na Rys. 2.6.

Aby znaleźć wielkość i kierunek każdego gradientu w danym bloku używa się określonych wzorów:

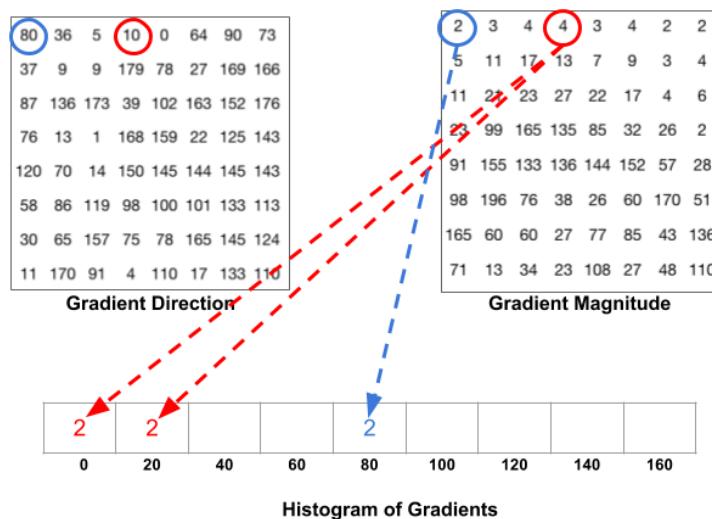
$$g = \sqrt{g_x^2 + g_y^2}, \theta = \arctan \frac{g_y}{g_x}$$



Rys. 2.6. Jądra służące do filtracji obrazu w celu obliczenia gradientów [10]

Kolejnym krokiem jest podział obrazu na jednakowe siatki o wymiarach 8×8 . Każdą komórkę da się przedstawić za pomocą 128 liczb. Pojedynczy fragment składa się z 64 pikseli, z każdym związane są dwie ważne wartości - wielkość i kierunek gradientu ($64 \times 2 = 128$).

Aby skompresować dane dla każdej siatki, należy stworzyć histogram z podziałem na dziewięć oddzielnych pojemników. Każdy z nich odpowiada kątom z zakresu 0-160 z 20-stopniowym przyrostem. Przyporządkowując każdy piksel do jednego z pojemników uwzględnia się kierunek gradientu (jego kąt) oraz wielkość, która go charakteryzuje. Pojemnik jest wybierany ze względu na kąt, natomiast wpisywana do niego wartość to wielkość przypisana do danego gradientu. Jeżeli piksel leży w połowie odległości między dwoma pojemnikami, jego wartość dzieli się między oba pola. Omawiana sytuacja dotyczy piksela oznaczonego na czerwono na Rys. 2.7.



Rys. 2.7. Schemat tworzenia histogramu dla komórki o wymiarach 8×8 [10]

Ważną kwestią jest rozważenie przypadku, gdy analizowany kąt posiada miarę większą niż 160 stopni (maksymalna wartość kąta w pojemniku). W takim przypadku wartość gradientu takiego piksela jest dzielona proporcjonalnie między dwa pojemniki, pierwszy i ostatni.

Po wykonaniu opisanego procesu dla wszystkich pikseli można zobaczyć w jaki sposób rozkładają się wartości w zależności od kierunku gradientu charakteryzującego każdy z nich. Taka reprezentacja zapewnia odporność na szum, który istnieje między gradientami na samym początku.

Gradienty obrazu są wrażliwe na oświetlenie. W przypadku zmiany jego natężenia, ich wielkość diametralnie się zmienia. Aby uodpornić deskryptor na jego wpływ stosuje się normalizację histogramu.

Bloki przekształca się w wektory elementów, a następnie dla każdego stosuje się normalizację. Blok o wymiarach 16×16 posiada 4 histogramy, gdzie każdy da się przedstawić jako wektor 9×1 , co w sumie daje wektor 36×1 . Normalizacja następuje dla pierwszego okna, po czym jest ono przesuwane o 8 pikseli, gdzie znaleziony wektor jest obliczany ponownie. Wszystko powtarza się do momentu przebycia wszystkich pozycji.

Każdy z omawianych bloków reprezentowany jest przez wektor 36×1 , co jest równoznaczne z 36 wyodrębnionymi cechami. Istnieje 105 pozycji takich bloków (7 poziomych, 15 pionowych). Sumarycznie, dla naszego zdjęcia, otrzymujemy aż 3780 cech.

Powyżej zostało opisane działanie deskryptora HOG na pojedynczym obrazie. Jest on podstawą rozpatrywanego algorytmu, którego celem jest stworzenie programu wykrywającego twarze. Aby osiągnąć takową użyteczność, należy połączyć działanie deskryptora z klasyfikatorem SVM.

Klasyfikator SVM pozwala na analizę danych, rozpoznanie wzorców i ich klasyfikację. Do jego wytrenowania używa się próbek pozytywnych i negatywnych, tak jak w przypadku przygotowywania klasyfikatora kaskadowego.

Z każdego zdjęcia, za pomocą deskryptora, należy wyciągnąć cechy HOG. Na ich podstawie trenuje się klasyfikator SVM. Uczy się on różnych możliwych cech twarzy, które potem próbuje zlokalizować na nowym zdjęciu. Po fazie uczenia, klasyfikator pozwala określić do jakiej klasy należą przetwarzane dane. W naszym przypadku rozróżni on fragment zawierający twarz i ten, na którym się ona nie znajduje.

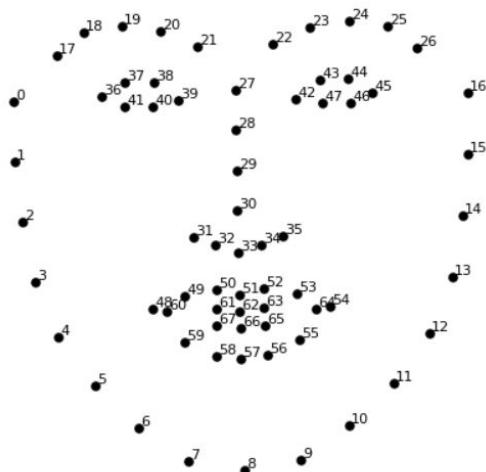
Analizowana w tej sekcji metoda stała się konkurencją dla algorytmu opisywanego we wcześniejszym podrozdziale. Działa ona dokładniej niż algorytm oparty na klasyfikatorze kaskadowym. Minusem jest wykrywanie twarzy tylko w przypadku widoku z przodu, program nie działa poprawnie w momencie jej rotacji i zmian kąta. Mimo wszystko opracowane rozwiązanie zdaje się być proste i efektywne.

2.2. Identyfikacja punktów charakterystycznych

Opisana wcześniej technika wykrywania twarzy jest początkowym etapem działania rozwiązania, które zostanie przedstawione w niniejszej sekcji. Mowa tutaj o algorytmie detekcji punktów charakterystycznych twarzy (ang. facial landmarks), poprzez które rozumie się elementy pomagające zidentyfikować jej najistotniejsze cechy. Zazwyczaj są to punkty utożsamiane z linią szczęki, ust, nosa, oczu i brwi [11].

Schemat wykrywania owych elementów działa dwuetapowo. Pierwszym krokiem jest lokalizacja twarzy. Metoda, która zostanie do tego wykorzystana nie jest istotna. Ważne jest uzyskanie ograniczonego obszaru, aby następnie można było rozpoczęć wykrywanie kluczowych struktur.

Tak jak w innych przypadkach, istnieje wiele różnych propozycji rozwiązania tego problemu. Algorytm, który aktualnie cieszy się dużą popularnością został zaproponowany w 2014 roku, a jego działanie opiera się na wykorzystaniu drzew regresji. Jego autorami są Vahid Kazemi oraz Josephine Sullivan [12].



Rys. 2.8. Wizualizacja 68 punktów charakterystycznych twarzy [13]

Działanie tej techniki można opisać w następujący sposób:

1. Jako zbiór testowy wybiera się taki zestaw danych, który składa się z obrazów z oznaczonymi ręcznie punktami charakterystycznymi twarzy.
2. Następnie model zostaje szkolony za pomocą drzew regresji na podstawie samej intensywności pikseli i próby dopasowania punktów w odpowiednie miejsca.
3. Korzysta się z obliczania prawdopodobieństwa odległości między parami pikseli.

4. Celem treningu jest optymalizacja funkcji błędu i dokonanie selekcji odpowiednich cech w oparciu o dane zawarte w zbiorach.

Rezultatem powyższych działań jest model zdolny do zlokalizowania charakterystycznych obszarów twarzy. W artykule, w którym pierwszy raz zaproponowano to rozwiązanie, testowano algorytm na zbiorze danych umożliwiającym wykrycie aż 192 istotnych punktów. Popularne biblioteki, w tym dlib, często udostępniają wariant wytrenowany na innym zbiorze danych. Zazwyczaj są to obrazy, na których oznaczono 68 kluczowych pozycji punktów (Rys. 2.8).

Zaletą przedstawionego algorytmu jest niezwykle szybki czas działania oraz dobra dokładność wykrywania elementów, nawet w przypadku twarzy ukazanych z profilu. Plusem jest także możliwość użycia dowolnych danych treningowych, czego efektem jest wyszkolenie własnego detektora punktów nie tylko twarzy, ale też innych niestandardowych elementów.

Technika, która została omówiona w tym rozdziale znajduje swoje zastosowania w wielu dziedzinach życia. Często jest ona uzupełnieniem wykrywania twarzy, w celu upewnienia się co do tożsamości danych osób. Jest to przydatny system w analizowaniu emocji ludzi, pozwala osobom z autyzmem bądź innymi chorobami lepiej zrozumieć otaczający ich świat. W przypadku oprogramowania czytającego z ruchu warg również analizuje się zmiany lokalizacji ważnych punktów charakterystycznych [4].

2.3. Triangulacja Delaunaya

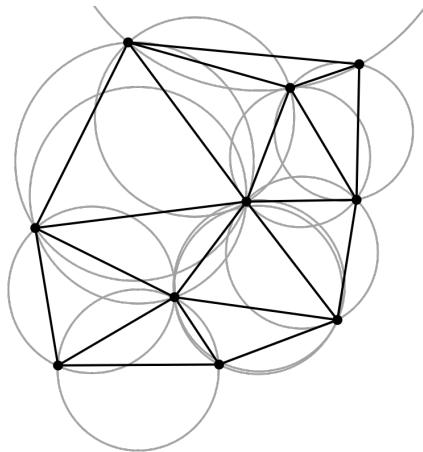
Poprzez pojęcie triangulacji, w kontekście matematycznym, można rozumieć podział figury geometrycznej na trójkąty, bądź też czworościany (określone jako sympleksy) w taki sposób, aby część wspólna dwóch sąsiadujących trójkątów (czworościanów) była ich wspólną ścianą, wierzchołkiem, bokiem, trójkątem lub zbiorem pustym [14].

Istnieje wiele różnych rodzajów triangulacji. W przypadku rozważanego w tej pracy algorytmu wykorzystano triangulacje zbioru punktów w przestrzeni 2D, do której należy między innymi triangulacja Delaunaya (lub Delone), której nazwa pochodzi od nazwiska autora tejże koncepcji Borysa Delaunaya.

Triangulacja Delone [15] rozumiana jest jako triangulacja T przestrzeni R^{n+1} , którą definiuje się w poniższy sposób. Jako T określa się podział przestrzeni R^{n+1} na $(n + 1)$ trójkątów, z punktami jako wierzchołki, spełniających określone warunki:

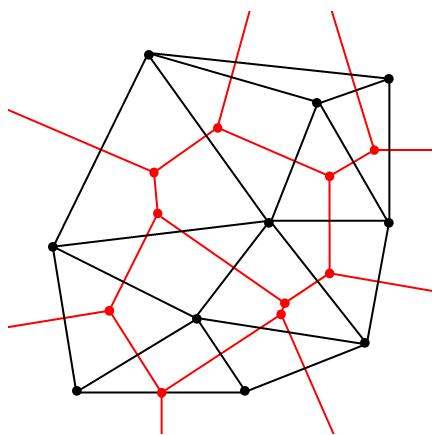
1. Każde dwa trójkąty należące do zbioru T posiadają wspólną ścianę albo nie są ze sobą połączone w żaden sposób.
2. Każdy z ograniczonych zbiorów w przestrzeni R^{n+1} ma część wspólną z ograniczoną liczbą trójkątów ze zbioru T .

3. Opisując kulę na dowolnym trójkącie ze zbioru T nie natkniemy się na sytuację, gdy wnętrze kuli będzie zawierało wierzchołki z pozostałych trójkątów zawartych w zbiorze T .



Rys. 2.9. Przykładowa triangulacja Delone dla zbioru punktów [16]

Przedstawioną powyżej koncepcję można uogólnić do przestrzeni o większych wymiarach. Wspominając o triangulacji Delone warto także wspomnieć o diagramach Voronoia, które są ściśle powiązane z owym pojęciem. Diagram, dla zestawu punktów, dzieli przestrzeń w taki sposób, że linie podziału znajdują się w równej odległości od punktów sąsiadujących.



Rys. 2.10. Graf przedstawiający komórki Vornoia i krawędzie triangulacji [16]

Ważną właściwością triangulacji Delone jest fakt, iż wraz z diagramem Voronoia tworzy graf dualny. Te definicje są powiązane, więc znając triangulację Delone dla zbioru punktów możemy w łatwy sposób obliczyć diagram Voronoia. Dwa trójkąty mające wspólną krawędź w triangulacji umożliwiają odnalezienie krawędzi w diagramie Voronoia.

Centra okrągów wyznaczonych przez owe trójkąty po połączeniu tworzą daną krawędź. Rys. 2.10 przedstawia krawędzie triangulacji (czarne linie) oraz utworzone na podstawie triangulacji komórki Voronoia (czerwone linie).

Istotną właściwością tej techniki jest fakt, że trójkąty powstałe w wyniku triangulacji nie mają kątów o dużych miarach, co zapewnia przejrzystość wygenerowanych figur. W przypadku zastosowania triangulacji na zbiorze punktów można uniknąć chaosu i nierównomierności.

Triangulacja ma swoje zastosowania w wielu dziedzinach. Wykorzystuje się ją w informatyce, grafice komputerowej czy też geodezji. Technika umożliwia tworzenie skomplikowanych figur, wypełnianie obszarów, wyznaczanie linii przecięcia.

Istnieje wiele różnych algorytmów przeznaczonych do znajdowania triangulacji Delone dla zbioru punktów. Przedstawienie ich w niniejszej pracy dyplomowej nie jest jednak kluczowe. Większość języków programowania dostarcza odpowiednie biblioteki, które posiadają funkcje implementujące takowe algorytmy.

3. Przegląd istniejących zastosowań

Niemal każdy z nas na co dzień używa smartfona i korzysta z różnego rodzaju aplikacji, których celem jest komunikacja z ludźmi, zarządzanie finansami, tworzenie notatek, dokumentów bądź też zapewnienie rozrywki użytkownikowi. W czasach, w których media społecznościowe odgrywają znaczącą rolę popularne stało się stosowanie filtrów modyfikujących twarz, sylwetkę lub dodających do zdjęć efekty specjalne.

Zgłębiając zasoby internetowe łatwo natrafić na narzędzia, które stosują rozwiązania związane do implementowanego w niniejszej pracy algorytmu.

W tym rozdziale zostanie opisana technika deepfake, która z dnia na dzień zyskuje na popularności. Zostaną również przedstawione przykładowe aplikacje mobilne implementujące ową technikę i oparte na wykorzystaniu sztucznej inteligencji w przetwarzaniu obrazów.

3.1. Technika deepfake

Technologia deepfake umożliwia przedstawienie dowolnej osoby jako uczestnika danego filmu, czy też postaci znajdującej się na konkretnym zdjęciu. Jej celem jest iluzja zamiany wypowiedzi jednej osoby, na wypowiedź innej, to samo dotyczy ruchów ciała. Istnieje także możliwość spreparowania dźwięku, przez co mamy wrażenie, że słyszmy słowa wypowiadane przez znajomą nam osobę. Tymczasem jest to fikcja uzyskana za pomocą sztucznej inteligencji. Owa technika to symulacja rzeczywistości, często wykorzystywana we współczesnym kinie w przypadku generowania komputerowych scenografii.

Kiedy podczas kręcenia siódmej części filmu Szybcy i Wściekli zmarł Paul Walker, kierowcy produkcji musieli zmierzyć się z niemałym na tamte czasy wyzwaniem, symulując sceny z jego udziałem. W dzisiejszych czasach technologia deepfake niezwykle się rozwinęła i stała się bardzo popularna, każda osoba ma możliwość wygenerowania filmu będącego deepfake'iem w kilka minut.

Słowo deepfake to tak naprawdę połączenie dwóch angielskich wyrazów. Nawiązuje ono do technologii uczenia głębokiego (ang. deep learning), z którą owa technika jest ściśle związana, oraz do słowa *fake* oznaczającego coś nieprawdziwego, udającego inną rzecz.

Działanie algorytmu deepfake często oparte jest na wykorzystaniu autoenkoderów lub sieci GAN (ang. Generative Adversarial Network). Według wielu osób wspomniane wyżej sieci mogą być związane z ogromnym rozwojem tejże technologii. Podobizny, które zostają wygenerowane poprzez ich zastosowanie wydają się być prawie nieodróżnialne od rzeczywistych twarzy [17].

Stworzenie filmu określonego jako deepfake musi zostać poprzedzone zastosowaniem odpowiednich procedur. W przypadku autoenkoderów pierwszym krokiem jest przepuszczenie zdjęcia źródłowego jak i docelowego przez enkoder, który uczy się podobieństw między dwiema twarzami. Następnie dane są kompresowane, wyciągane są wspólne cechy obydwu zdjęć. Kolejnym etapem jest wytrenowanie dwóch dekoderów, które zostaną użyte do odzyskania twarzy każdej z osób. Ostatnim krokiem jest podmiana twarzy między dekoderami. Dekoder ma za zadanie zrekonstruować twarz drugiej osoby na podstawie orientacji pierwszego obrazu.

Sieci GAN działają inaczej. Na początku znajdują się trenowane poprzez kilkugodzinne analizowanie rzeczywistego filmu wideo. Dzięki temu są w stanie nauczyć się jak wygląda twarz osoby pod różnymi kątami, w różnym świetle. Następnie łączy się wytrenowaną sieć z technikami grafiki komputerowej, nakłada się kopię osoby na danego aktora. Wszystko odbywa się poprzez mapowanie odpowiednich punktów charakterystycznych twarzy.

Technologia deepfake niesie ze sobą niemałe zagrożenie. Poziom zaawansowania sprawia, że czasami ciężko na pierwszy rzut oka wykryć film, w którym została zastosowana. Na dzień dzisiejszy dzięki dokładniejszej analizie takowego wideo jesteśmy w stanie wykryć, czy stworzony film jest fikcją. Jednak według wielu przypuszczeń, niewykluczone, że wraz z rozwojem sztucznej inteligencji przestanie to być możliwe.

3.2. Dostępne aplikacje mobilne

3.2.1. Wombo.ai

Wombo.ai [18] to bardzo popularna, darmowa aplikacja mobilna, która została stworzona w celu rozrywkowym. Jest ona dostępna zarówno na smartfony jak i tablety z systemem operacyjnym Android i IOS, co zdecydowanie jest jej walorem. Reprezentowaną przez nią ideą jest tworzenie zabawnych filmików, na których ożywiane są wgrane wcześniej fotografie. Zaletą tej aplikacji jest jej prostota i łatwość w obsłudze.

Wszystko odbywa się w kilku krokach, należy zrobić zdjęcie swojej twarzy albo wgrać takowe z galerii, następnie wybrać utwór muzyczny na podstawie którego otrzymujemy krótkie nagranie ze stworzoną animacją. Filmiki zostają wygenerowane z użyciem technologii deepfake, przez co są bardzo realistyczne. Jej działanie powiązane jest z wykorzystaniem sztucznej inteligencji do synchronizowania ruchu.

3.2.2. Anyface: face animation

Anyface [19] to aplikacja mająca na celu, podobnie jak poprzedniczka, zapewnienie rozrywki użytkownikowi. Jest ona dostępna w wersji mobilnej tylko na Androïda.

W odróżnieniu od wspomnianego wyżej programu to narzędzie posiada dużo więcej funkcji, przez co może sprawiać wrażenie bardziej atrakcyjnego dla użytkownika. Poza animacją zdjęcia na podstawie wybranej frazy istnieje możliwość dodania własnego nagrania, z którego użyciem zostanie wygenerowana animacja. Narzędzie posiada także moduł edycji i udoskonalania zdjęć, odbiorca jest w stanie dodawać filtry, efekty specjalne i inne obiekty, które nadadzą animacji unikalny charakter.

3.2.3. MotionPortrait

MotionPortrait [20] to bardzo podstawowa aplikacja mobilna dostępna zarówno na Androïda jak i system IOS. Kategoria, w której można ją znaleźć to rozrywka. Posiada opcję zrobienia własnego zdjęcia, dodania zdjęcia z galerii lub wyboru udostępnionej fotografii. Po wykonaniu tego etapu mamy możliwość animacji naszej podobizny poprzez wybór zdefiniowanych wyrazów twarzy.

Dzięki połączeniu z mikrofonem w przypadku wypowiadania jakichkolwiek słów nasza twarz jest animowana. Animacja dotyczy tutaj tylko ust, dodatkowo awatar mruga regularnie oczami. Efekt, który otrzymujemy nie jest zbyt satysfakcjonujący. Na zdjęcie możemy też nałożyć różne filtry, dodać efekty. Ostatecznie istnieje możliwość nagrania krótkiego wideo, które potem możemy zapisać.

3.3. Wnioski

Na rynku dostępna jest duża liczba aplikacji podobnych do programów opisanych powyżej. Niektóre z nich oferują bardzo podstawowe funkcje, a efekty osiągane przez te narzędzia nie wydają się być spektakularne. Istnieją jednak rozwiązania, które odstają od innych swoją dokładnością działania i efektowną realizacją.

Pewne jest, że użytkownicy chętnie korzystają z owych aplikacji. Przeglądając media społecznościowe na każdym kroku możemy się natknąć na filmiki stworzone z użyciem techniki deepfake. Co więcej, narzędzia oparte na tych technologiach niezwykle szybko się rozwijają i zaczynają zaskakiwać swoimi opcjami.

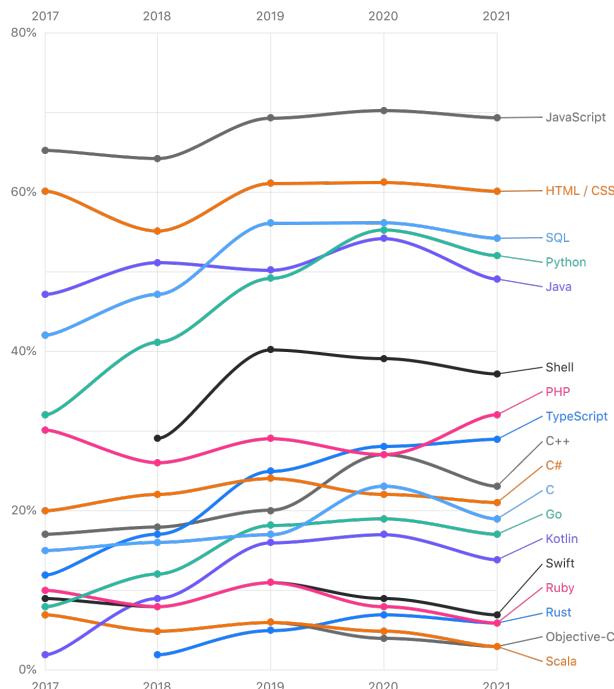
Powyższe fakty sprawiają, że stworzenie algorytmu animacji awatara w ramach tejże pracy inżynierskiej wydaje się być sensowne. Są to narzędzia, które wykorzystuje się nie tylko w celu rozrywkowym, ale także w dziedzinie kinematografii.

4. Wykorzystane narzędzia i technologie

W niniejszym rozdziale zostaną zawarte krótkie opisy narzędzi oraz technologii wykorzystanych w trakcie implementacji algorytmu będącego tematem owej pracy inżynierskiej. Zostańie również przedstawiony język programowania, w którym stworzono projekt, oraz charakteryzujące go zalety przeważające o jego wyborze.

4.1. Python

Język programowania Python [21] powstał we wczesnych latach dziewięćdziesiątych. Guido van Rossum jest uważany za głównego twórcę tego języka, jednak wkład w jego rozwój miały także inne osoby.



Rys. 4.1. Popularność języków programowania na przestrzeni lat [22]

Popularność narzędzia wzrosła diametralnie w momencie wydania wersji 2.0 w roku 2000 i rośnie aż do teraz. Aktualnie znajduje się on w czołówce najczęściej wykorzystywanych języków (Rys. 4.1). Jego interpretery dostępne są dla wielu systemów operacyjnych, obsługuje większość używanych w dzisiejszych czasach platform, takich jak Windows, Linux, AIX, iOS i inne.

Projekt, w ramach którego zostaje rozwijany owy język zarządzany jest przez Python Software Foundation, będącą organizacją non-profit. Narzędzie zostało udostępnione jako otwarte oprogramowanie, przez co użytkownicy posiadają możliwość ingerowania w wydawany kod źródłowy.

Python to język wielopoziomowy, którego przeznaczenie nie jest ściśle określone. Jego możliwości są niezwykle rozległe i uzależnione od stosowanych bibliotek, platform oraz gotowych skryptów, których wybór jest wyjątkowo obszerny.

Język nie wymusza od użytkownika jednego stylu, w którym tworzone są programy. Istnieje możliwość zastosowania różnych paradygmatów programowania (obiektowe, strukturalne oraz funkcyjne), co zdecydowanie wpływa na rozległość jego zastosowań. Ważną cechą jest także dynamiczna typizacja, którą oferuje. Podczas tworzenia zmiennych nie wymaga się definiowania ich typu. Co więcej Python posiada automatyczne zarządzanie pamięcią, które zwalnia programistę z tego obowiązku.



Rys. 4.2. Dziedziny, w których najczęściej wykorzystuje się język programowania Python [23]

Główną ideą towarzyszącą twórcy podczas opracowywania składni języka była wysoka czytelność kodu źródłowego, jego przejrzystość i zwięzłość. Okazało się to jednym z głównych czynników, które spowodowały, że stał się tak powszechnie wykorzystywany.

Dzięki swojej uniwersalności Python odnajduje zastosowanie w wielu różnych dziedzinach, które zostały przedstawione na Rys. 4.2.

Popularność w środowisku matematycznym zawdzięcza bibliotece SciPy stanowiącą darmową alternatywę do języka Matlab. Biblioteka Tensor Flow wspiera tworzenie sieci neuronowych, wykorzystywane w tworzeniu algorytmów w popularnych aplikacjach [23]. W przypadku dziedzin operujących na przetwarzaniu obrazów czy też analizowaniu danych Python dostarcza kilka pakietów ułatwiających te operacje.

Ze względu na powyższe fakty, to jest między innymi czytelność, uniwersalność oraz ogromne zasoby pakietów, algorytm będący tematem niniejszej pracy inżynierskiej został opracowany w języku Python. W następnej sekcji zostaną przedstawione biblioteki wykorzystane w jego implementacji.

4.2. Biblioteki

Biblioteki programistyczne dostarczają podprogramy, które użytkownik może wykorzystać w swoim kodzie źródłowym bez konieczności implementowania ich od podstaw. Jest to metoda na wielokrotne używanie identycznego kodu.

4.2.1. OpenCV

Biblioteka OpenCV [24] jest narzędziem wydanym jako otwarte oprogramowanie, wspomagające przetwarzanie obrazów oraz uczenie maszynowe. Została napisana w języku C, natomiast posiada interfejsy dla innych języków, takich jak Java, Python bądź Matlab.

Zawiera około 2500 zoptymalizowanych algorytmów implementujących między innymi wykrywanie i rozpoznanie twarzy, identyfikację i śledzenie ruchu obiektów. Znajdują się w niej także moduły dotyczące modeli 3D, ich tworzenia i przetwarzania.

Poza skomplikowanymi algorytmami OpenCV dostarcza wiele podstawowych operacji, które wykorzystuje się przy modyfikacji obrazów. Zaimplementowane funkcje zdają się wyraźnie wydajnością, co jest bardzo istotne w przypadku działania na dużych zbiorach danych.

4.2.2. imutils

Pakiet imutils [25] to kolejne przydatne narzędzie wykorzystywane w trakcie modyfikacji obrazów. Zawiera szereg funkcji ułatwiających ich przetwarzanie, takich jak obracanie, zmiana rozmiaru, przesunięcie, zastosowanie szkieletyzacji. Zaimplementowano także moduły ułatwiające poprawną prezentację zdjęć podczas ich wyświetlania.

Funkcje zawarte w owym pakiecie pochodzą z biblioteki OpenCV, zostały one odpowiednio połączone i zmodyfikowane, aby można było w prostszy sposób dokonywać podstawowych operacji na zdjęciach. Biblioteka OpenCV zawiera ogromną ilość komponentów, co może być przytłaczające dla osoby rozpoczynającej swoją przygodę z przetwarzaniem obrazów.

Ideą, dla której powstał ten pakiet była chęć zebrania podstawowych modułów i odpowiednie ich dostosowanie w celu ograniczenia zbędnych operacji, które należy zastosować korzystając z biblioteki OpenCV.

4.2.3. dlib

Dlib to biblioteka udostępniona na zasadach otwartej licencji, napisana w języku C++ [26]. Dostarcza ona algorytmy oparte na uczeniu maszynowym oraz narzędzia do tworzenia oprogramowania we wspomnianym wyżej języku.

Funkcje, z których można korzystać, implementują algorytmy umożliwiające wytrenowanie klasyfikatorów SVM oraz SVR. Dodatkowo istnieje możliwość wytrenowania własnego modelu wykrywającego twarz lub jej punkty charakterystyczne.

Zastosowania tej biblioteki obejmują takie dziedziny jak przemysł, robotyka oraz wszelkie obszary wymagające skorzystania z programów o dobrej wydajności obliczeniowej.

4.2.4. scikit-image

Biblioteka scikit-image [27], podobnie jak opisane powyżej pakiety, zawiera szereg implementacji algorytmów powiązanych z przetwarzaniem obrazów. Została ona udostępniona jako otwarte oprogramowanie tworzone przez kilkadziesiąt osób.

Obejmuje takie algorytmy jak transformacje, segmentacje obrazów, ich morfologię oraz manipulację przestrzenią kolorów. Pakiet korzysta z tablic biblioteki NumPy, wykorzystanych do przechowywania obrazów.

4.3. Flask

Flask [28] to minimalistyczny szkielet wspierający tworzenie aplikacji webowych (ang. microframework). Zawiera wiele przydatnych bibliotek oraz modułów umożliwiających ich proste implementowanie.

W odróżnieniu od pełnowymiarowej platformy programistycznej (ang. framework) nie posiada warstwy abstrakcji bazy danych, części walidacyjnej formularzy czy też innych elementów, które wymagałyby zapewnienia szczególnych bibliotek. Natomiast obsługuje rozszerzenia, które umożliwiają dodanie owych funkcji do aplikacji.

Flask dostarcza programistom wiele przydatnych modułów, dzięki którym nie musi się on zajmować obsługą wątków i protokołów. Oparty jest na zestawie kilku narzędzi, na które składają się:

- *Werkzeug* - biblioteka udostępniająca zestaw narzędzi WSGI (ang. Web Server Gateway Interface) rozumianych jako interfejs bramy serwera WWW. Implementuje on przetwarzanie żądań między serwerem a aplikacją.
- *jinja2* - biblioteka programistyczna umożliwiająca osadzanie danych na stronie w odpowiednich szablonach prezentacyjnych, której rezultat jest widoczny w przeglądarce.
- *MarkupSafe* - biblioteka rozszerzająca typ tekstowy, zapewniająca bezpieczne używanie znaków w HTML oraz XML. Zabezpieczona przed atakami poprzez wprowadzanie niezaufanych danych.
- *ItsDangerous* - pakiet odpowiadający za poprawną serializację danych. Służy do przechowywania sesji za pomocą plików cookies.

Głównymi zaletami opisywanego szkieletu jest jego prostota i brak ograniczeń związanych ze ścisłe ustalonymi regułami, którymi należy się kierować budując program. Jest niezwykle elastyczny, nadaje się do tworzenia małych, wewnętrznych aplikacji, ale jednocześnie daje możliwość rozbudowania ich do pełnowymiarowej struktury.

W związku z wymienionymi zaletami, Flask zdaje się być idealnym narzędziem do stworzenia prostej aplikacji mającej na celu walidację algorytmu tworzonego w owym projekcie.

4.4. Pycharm

Pycharm [29] jest zintegrowanym środowiskiem wspierającym dla języka programowania Python. Oprogramowanie jest dostępne na wielu platformach systemowych takich jak Windows, Linux i macOS. Oferuje inteligentny edytor kodu źródłowego z możliwością formatowania i autouzupełniania, który posiada funkcję weryfikacji błędów oraz ich kontrolowanie.

Zapewnia refaktoryzację języka, przez co utrzymana jest wysoka jakość systemowa. Elementy są wpasowywane w dane wzorce, dopasowywane do obowiązujących standardów.

Ważną kwestią jest udostępniane wsparcie dla różnych bibliotek i narzędzi. Pycharm wspiera tworzenie stron internetowych z wykorzystaniem biblioteki Django, czy też Flaska i Pyramid. Wspomaga tworzenie plików w językach powiązanych z technologią webową (HTML, CSS, JavaScript).

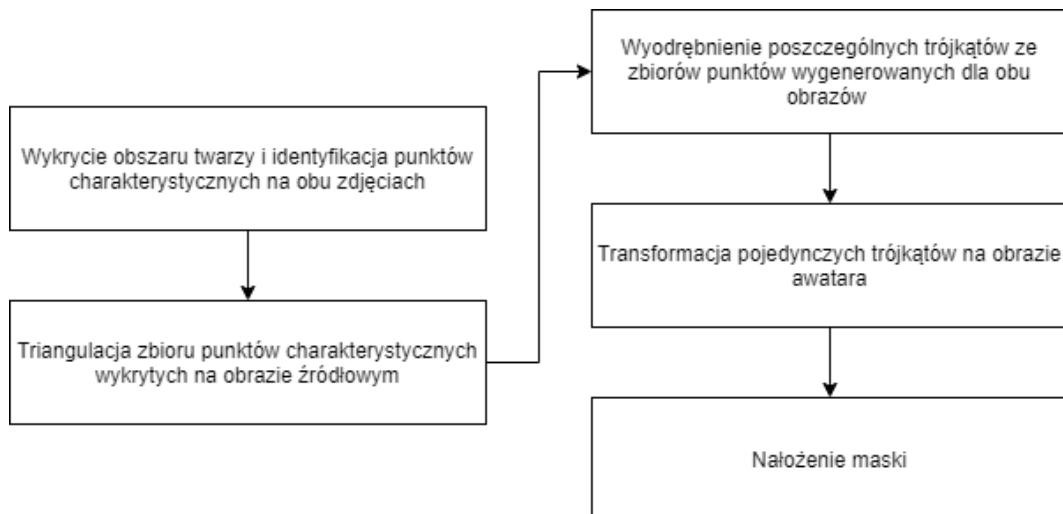
Ułatwia pracę z projektem dostarczając prosty interfejs konfiguracyjny. W przypadku tworzenia aplikacji webowej opartej na platformie Flask w nieskomplikowany sposób można wprowadzić wszystkie istotne ustawienia.

5. Algorytm animacji awatara

Jak wspomniano we wcześniejszych rozdziałach, działanie algorytmu zaimplementowanego na potrzeby tej pracy inżynierskiej można podzielić na kilka kluczowych etapów. Poniższy rozdział zawiera szczegółowy opis każdego z nich. Zostaną także zaprezentowane efekty otrzymane w konkretnych krokach.

5.1. Struktura programu

Do stworzenia algorytmu wykorzystano język programowania Python umożliwiający tworzenie aplikacji, których działanie oparte jest na wykorzystaniu różnego rodzaju napisanych wcześniej funkcji. W przypadku owego projektu zostały one zaimplementowane dla każdego istotnego etapu programu, które przedstawiono na Rys. 5.1.



Rys. 5.1. Etapy algorytmu animacji awatara

Kluczową kwestią jest wcześniejsze wczytanie danych, czyli obrazu źródłowego, przez który rozumie się zdjęcie mające posłużyć jako baza do przekształcenia awatara oraz jego obraz, na którym nastąpi animacja. Funkcja łącząca wszystkie działania została przedstawiona w formie poniższego kodu. Jej wynikiem jest obraz awatara ze zmienionym wyrazem twarzy

osiągniętym za pomocą odpowiednich przekształceń. W dalszych sekcjach zostaną opisane poszczególne moduły, z których korzystano.

Główna funkcja algorytmu

```

1 def animate_avatar(avatar_img, avatar_points, src_img, src_points):
2     new_avatar_face = np.zeros(src_img.shape, np.uint8)
3
4     for src_triangle_points, avatar_triangle_points in
5         delaunay_triangulation(src_points, avatar_points):
6         avatar_img_cropped, avatar_triangle, _ =
7             crop_single_triangle(avatar_triangle_points, avatar_img)
8         src_img_cropped, src_triangle, b_rect =
9             crop_single_triangle(src_triangle_points, src_img)
10
11     warped_triangle = transform_triangle(avatar_triangle,
12         avatar_img_cropped, src_triangle, src_img_cropped)
13     add_triangle_to_new_face_area(new_avatar_face, warped_triangle,
14         b_rect)
15
16     return generate_new_face(avatar_img, avatar_points, new_avatar_face,
17         src_points)
```

5.2. Wykrycie twarzy oraz punktów charakterystycznych

Pierwszym krokiem jest wykrycie obszaru twarzy oraz identyfikacja jej punktów charakterystycznych. Etap ten został zrealizowany z użyciem gotowych funkcji udostępnianych przez bibliotekę dlib:

- *get_frontal_face_detector()* - funkcja nie przyjmuje żadnych parametrów, zwraca wytrenowany obiekt umożliwiający detekcję twarzy. Model został wyszkolony przy pomocy algorytmu działającego w oparciu o klasyfikator SVM i deskryptor HOG, opisany w podrozdziale 2.1.3.
- *shape_predictor()* - parametrem wejściowym jest wytrenowany model, również udostępniony przez bibliotekę, umożliwiający poprawną lokalizację punktów charakterystycznych [30]. Został on wyszkolony na podstawie algorytmu przedstawionego w sekcji 2.2. Wynikiem zastosowania danej funkcji jest obiekt, który generujeowy zestaw punktów zlokalizowanych na obrazie przekazanym jako dane wejściowe.

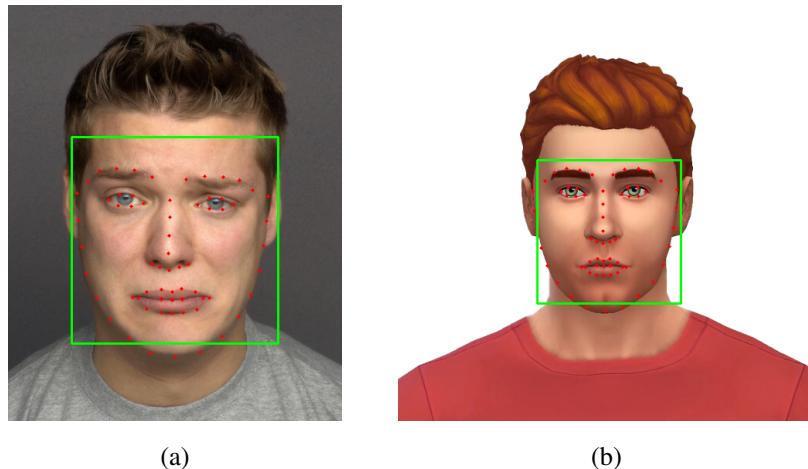
Wykrycie twarzy oraz punktów charakterystycznych

```

1 def detect_face_and_landmarks(img):
2     detector = dlib.get_frontal_face_detector()
3     predictor =
4         ↳ dlib.shape_predictor('./data/shape_predictor_68_face_landmarks.dat')
5
6     gray = cv2.cvtColor(imutils.resize(img), cv2.COLOR_BGR2GRAY)
7     rects = detector(gray, 1)
8     facial_landmarks = []
9
10    if rects:
11        for rect in rects:
12            facial_landmarks = predictor(gray, rect)
13            facial_landmarks = face_utils.shape_to_np(facial_landmarks)
14
15    return img, facial_landmarks

```

Wybrane funkcje zdają się być wystarczające do osiągnięcia zamierzonego rezultatu. Można założyć, iż przetwarzane obrazy zawierają dobrze widoczny obszar twarzy ustawionej w pozycji frontowej. Z tego powodu dokładność działania narzędzia wykrywającego twarz oraz punkty charakterystyczne jest wysoka.



Rys. 5.2. Elementy wykryte na obrazie źródłowym (a) [31] oraz na awatarze (b) [32]

Efektem wizualnym użycia powyższej funkcji są obrazy przedstawione na Rys.5.2. Na każdym ze zdjęć została zlokalizowana twarz, którą zaznaczono zielonym prostokątem oraz jej kluczowe elementy, w postaci 68 punktów oznaczonych kolorem czerwonym. W następnych etapach posłużą one jako baza, na której będą wykonywane wszelkie operacje.

5.3. Triangulacja zbioru punktów

Kolejnym etapem algorytmu animacji awatara jest triangulacja zbioru punktów charakterystycznych, zlokalizowanych na obszarze twarzy obrazu źródłowego. Celem jej zastosowania jest podział przestrzeni na nieskomplikowane trójkąty posiadające ostre kąty, co zapewni przejrzystość i równomierność figur.

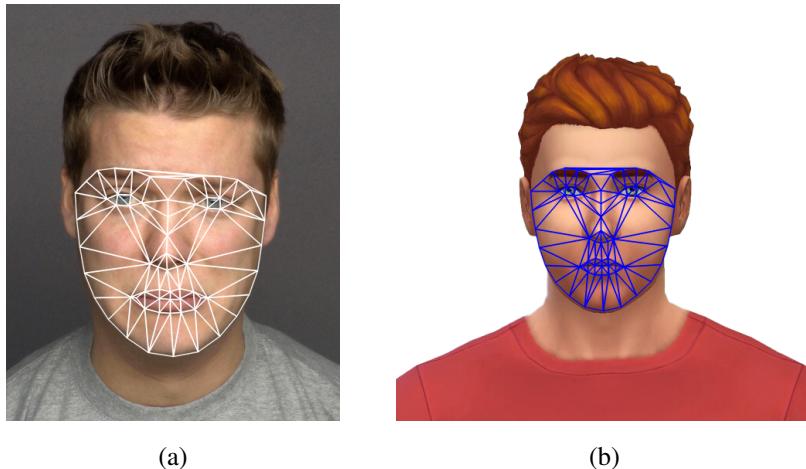
W bibliotece OpenCV dostępne są funkcje, generujące triangulację Delone dla zestawu punktów. Ich użycie jest bardzo proste i intuicyjne. Z wykorzystaniem modułu *Subdiv2D(Rect rect)*, dla którego parametrem wejściowym jest prostokątny obszar obejmujący wszystkie istotne punkty, zostaje wygenerowany pusty obiekt. Następnie za pomocą funkcji *insert()* można dodać do niego elementy charakterystyczne, dla których chcemy dokonać triangulacji. Ostatnim krokiem jest wywołanie funkcji *getTriangleList()*, która zwraca listę trójkątów wygenerowanych przez algorytm.

Triangulacja Delone dla zbioru punktów

```

1 def delaunay_triangulation(src_points, avatar_points):
2     points_list = list(src_points)
3     delaunay_subdivision = cv2.Subdiv2D((*src_points.min(axis=0),
4                                         *src_points.max(axis=0)))
5     delaunay_subdivision.insert(points_list)
6
7     for x1, y1, x2, y2, x3, y3 in delaunay_subdivision.getTriangleList():
8         indexes_set = [(src_points ==
9                         single_point).all(axis=1).nonzero()[0][0]
10                     for single_point in [[x1, y1], [x2, y2], [x3, y3]]]
11
12         src_triangles = generate_xy_for_indexes(src_points, indexes_set)
13         avatar_triangles = generate_xy_for_indexes(avatar_points,
14                                         indexes_set)
15         yield [np.array(src_triangles), np.array(avatar_triangles)]
16
17
18 def generate_xy_for_indexes(points, indexes):
19     return [points[single_index] for single_index in indexes]
```

Funkcja zaimplementowana w celu rozwiązania tej części problemu generuje opisaną powyżej triangulację. Kolejną istotną kwestią jest wyodrębnienie analogicznych trójkątów z obydwu zbiorów punktów. Moduł *getTriangleList()* zwraca współrzędne wszystkich trzech wierzchołków każdego z wygenerowanych trójkątów. W prosty sposób można odnaleźć ich indeksy w zbiorze punktów charakterystycznych obrazu źródłowego, a następnie zidentyfikować analogiczne elementy w zbiorze punktów wygenerowanych z obrazu awatara.



Rys. 5.3. Triangulacja punktów char. obrazu źródłowego (a) oraz awatara (b)

Na Rys. 5.3 można zaobserwować efekt działania tego etapu programu. Na pierwszym obrazie zaznaczono trójkąty otrzymane w wyniku zastosowania triangulacji Delone, natomiast na drugim zaznaczono analogicznie wykryte trójkąty. Figury, które zostały otrzymane poprzez użycie funkcji z biblioteki OpenCV umożliwiają klarowne wykonanie dalszych transformacji.

5.4. Transformacja trójkątów

Posiadając zbiory trójkątów dla obydwu obrazów, można przejść do następnego kroku algorytmu. Jego celem jest dopasowanie każdej z pojedynczych figur obrazu awatara do odpowiadającego mu trójkąta (punkty o tych samych indeksach) z obrazu źródłowego. Mówiąc prościej, należy dokonać transformacji płaszczyzny do odpowiadającego mu obszaru trójkąta.

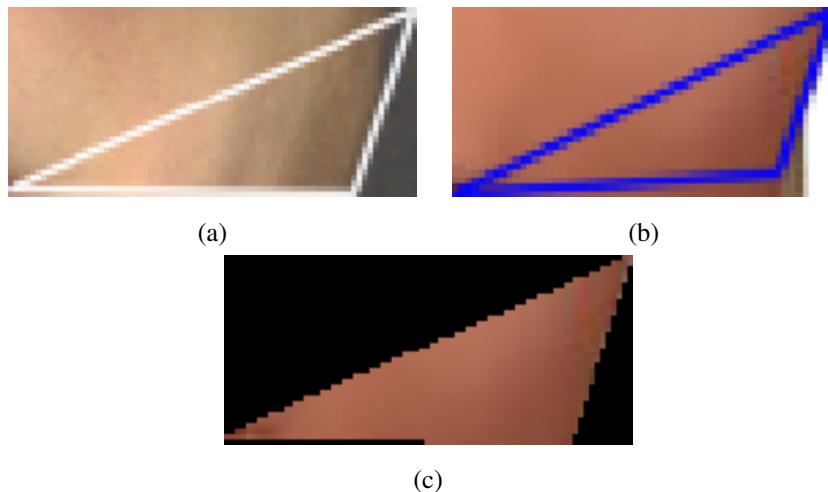
W celu uzyskania owych rezultatów potrzebna jest funkcja pomocnicza, która przygotuje przetwarzane obszary, tak aby można było dokonać transformacji. Na początku należy wyodrębnić obydwa fragmenty obrazów, w których znajdą się analizowane aktualnie figury. Fragment kodu przedstawionego poniżej umożliwia dokonanie tej operacji. Funkcja zwróci przyjęty region oraz nowe współrzędne trójkątów odnoszące się do mniejszego obszaru, a nie całego obrazu. Na Rys. 5.4a, 5.4b przedstawiono efekt zastosowania danego modułu.

_____ Funkcja pomocnicza przygotowująca trójkąty _____

```

1 def crop_single_triangle(single_triangle, img):
2     b_rect = cv2.boundingRect(single_triangle)
3     triangle_img = [(indexes[0] - b_rect[0], indexes[1] - b_rect[1]) for
4         ↳ indexes in single_triangle]
5     cropped_img = img[b_rect[1]:b_rect[1] + b_rect[3], b_rect[0]:b_rect[0]
6         ↳ + b_rect[2]]
7
8     return cropped_img, triangle_img, b_rect

```



Rys. 5.4. Odpowiadające sobie figury wyodrębnione z obszaru obrazu źródłowego (a) i awatara (b) oraz rezultat transformacji (c) płaszczyzny (b) w (a)

Po pobraniu odpowiednich danych dla dwóch przetwarzanych figur, możliwe jest wywołanie kolejnej funkcji, której celem jest transformacja fragmentu obrazu awatara. Skorzystano tutaj z modułów udostępnianych przez bibliotekę OpenCV:

- *getAffineTransform()* - jako dane wejściowe należy podać dwa zbiory punktów, dla których obliczana jest macierz transformacji afonicznej.
- *warpAffine()* - funkcja dokonuje przekształcenia obrazu za pomocą macierzy rotacji będącej wynikiem wcześniejszej operacji.

Funkcja pomocnicza transformująca trójkąty –

```

1 def transform_triangle(avatar_triangle, avatar_img_cropped, src_triangle,
2   ↵ src_img_cropped):
3   transform_matrix = cv2.getAffineTransform(np.float32(avatar_triangle),
4   ↵ np.float32(src_triangle))
5   warped_triangle = cv2.warpAffine(avatar_img_cropped, transform_matrix,
6   ↵ (src_img_cropped.shape[1],
7   ↵ src_img_cropped.shape[0]), None,
8   ↵ flags=cv2.INTER_LINEAR,
9   ↵ borderMode=cv2.BORDER_REFLECT_101)
10
11 mask = np.zeros(src_img_cropped.shape, dtype=np.uint8)
12 mask = cv2.fillConvexPoly(mask, np.int32(src_triangle), (1.0, 1.0,
13   ↵ 1.0), 16, 0)
14
15 return warped_triangle *= mask

```

Jako wynik otrzymano zmodyfikowany obraz. Na Rys. 5.4c zwizualizowano ten krok dla wybranej iteracji. Po otrzymaniu nowej figury, należy dodać ją do obszaru twarzy, który docelowo zastąpi aktualny rejon oblicza awatara. Funkcja `add_triangle_to_new_rect_area()` odpowiada za to zadanie. W celu usunięcia ewentualnych szumów etap ten poprzedzono zastosowaniem binaryzacji progowej.

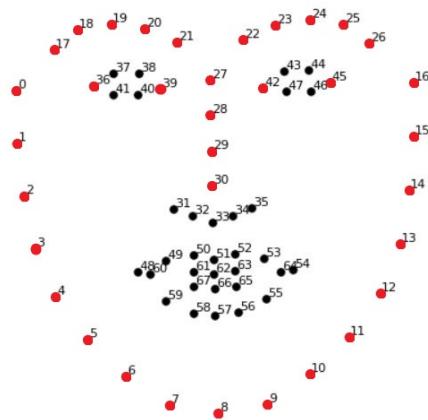
Schemat opisany w tej sekcji należy powtórzyć dla każdego trójkąta zwróconego przez funkcję `delaunay_triangulation()`. W skończonej pętli następuje modyfikacja wszystkich figur i jednoczesna rekonstrukcja danych fragmentów obszaru twarzy awatara. Finalny efekt można obserwować na Rys. 5.6b przedstawionym w następnej sekcji. Trójkąty zostały dopasowane do odpowiadających im figur z obrazu źródłowego, efektem jest zmodyfikowany obszar twarzy o takich samych wymiarach jak twarz źródłowa (Rys. 5.6a).

— Funkcja dodająca transformowaną figurę do nowego obszaru twarzy —

```
1 def add_triangle_to_new_face_area(new_face, warped_triangle, b_rect):
2     (x, y, w, h) = b_rect
3     new_face_gray = cv2.cvtColor(new_face[y: y + h, x: x + w],
4                                   cv2.COLOR_BGR2GRAY)
5     _, mask = cv2.threshold(new_face_gray, 0, 255, cv2.THRESH_BINARY_INV)
6     warped_triangle = cv2.bitwise_and(warped_triangle, warped_triangle,
7                                       mask=mask)
8
9     new_face[y: y + h, x: x + w] += warped_triangle
```

5.5. Nałożenie maski

Finalnym etapem opisywanego algorytmu jest ponowna transformacja, tym razem całego obszaru nowej twarzy awatara, tak aby odpowiadała ona jej wcześniejszym proporcjom. Ze względu na to, iż funkcja wykorzystana w poprzednim rozdziale (`getAffineTransform()`), przyjmuje pary tylko trzech lub czterech punktów, w tym przypadku użyto innych modułów. Zostały one udostępnione przez bibliotekę scikit-image:



Rys. 5.5. Punkty, na podstawie których następuje ostateczna transformacja

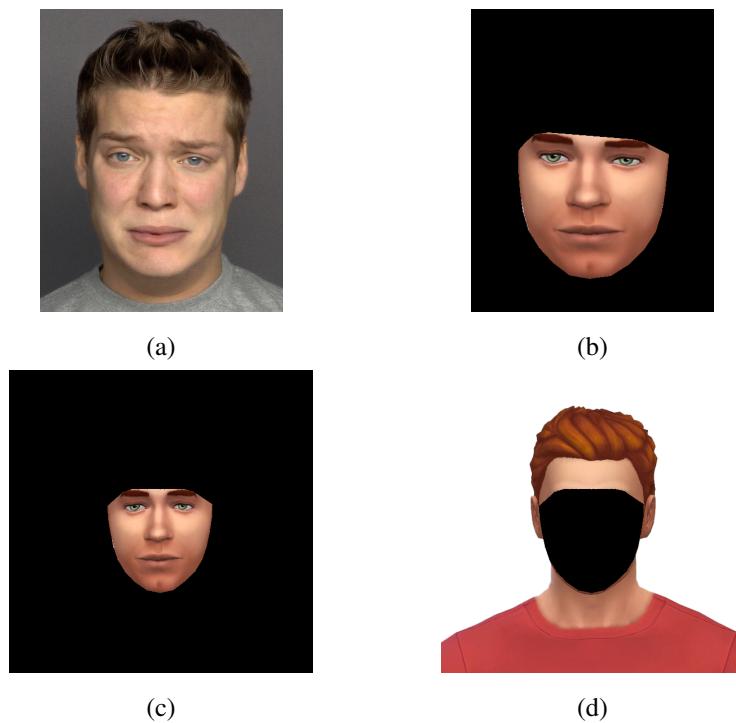
- `PiecewiseAffineTransform()` - inicjalizacja pustego obiektu, który umożliwia transformację bazując na dwóch równolicznych zestawach punktów. Jej działanie oparte jest na wykorzystaniu triangulacji Delone.
- `estimate()` - funkcja wywołana na otrzymanym wcześniej obiekcie przyjmuje dwa zestawy punktów, dla których ma nastąpić przemapowanie. W przypadku działania owego programu wybrano elementy oznaczone na Rys. 5.5 kolorem czerwonym. Są to punkty charakterystyczne, które z reguły, w przypadku podstawowych zmian mimiki twarzy pozostają niezmienne - linia szczęki, główny zarys nosa oraz kąciki oczu. Wybór akurat tych punktów zapewni zachowanie odpowiednich proporcji twarzy w trakcie transformacji obszaru.
- `warp()` - funkcja dokonująca transformacji płaszczyzny przekazanej jako dane wejściowe.

```

1 def generate_new_face(avatar_img, avatar_points, new_avatar_face,
2   ↪ src_points):
3     transform = PiecewiseAffineTransform()
4     transform.estimate(choose_specific_points(avatar_points),
5   ↪ choose_specific_points(src_points))

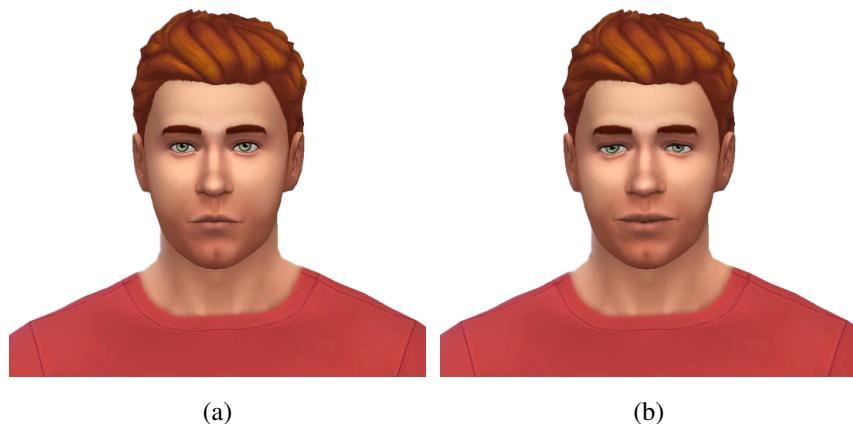
```

Po wykonaniu opisanych instrukcji należy dokonać normalizacji całego obrazu, aby przywrócić odpowiedni typ danych. Na Rys. 5.6c przedstawiono rezultat twarzy zmodyfikowanej do odpowiednich wymiarów. Jak widać otrzymany obraz składa się tylko ze zmodyfikowanego obszaru, przez co można swobodnie przejść do próby podmiany danego regionu na obrazie początkowym.



Rys. 5.6. Kolejne kroki działania etapu końcowego

Aby podmienić oblicze awatara na zmodyfikowany obszar wystarczy skorzystać z podstawowej operacji przetwarzania obrazów, czyli funkcji *bitwise_and()* udostępnionej przez bibliotekę OpenCV. Umożliwia ona zastosowanie na zdjęciu źródłowym maski o wymiarach nowego obszaru twarzy, czego rezultatem jest obraz przedstawiony na Rys. 5.6d.



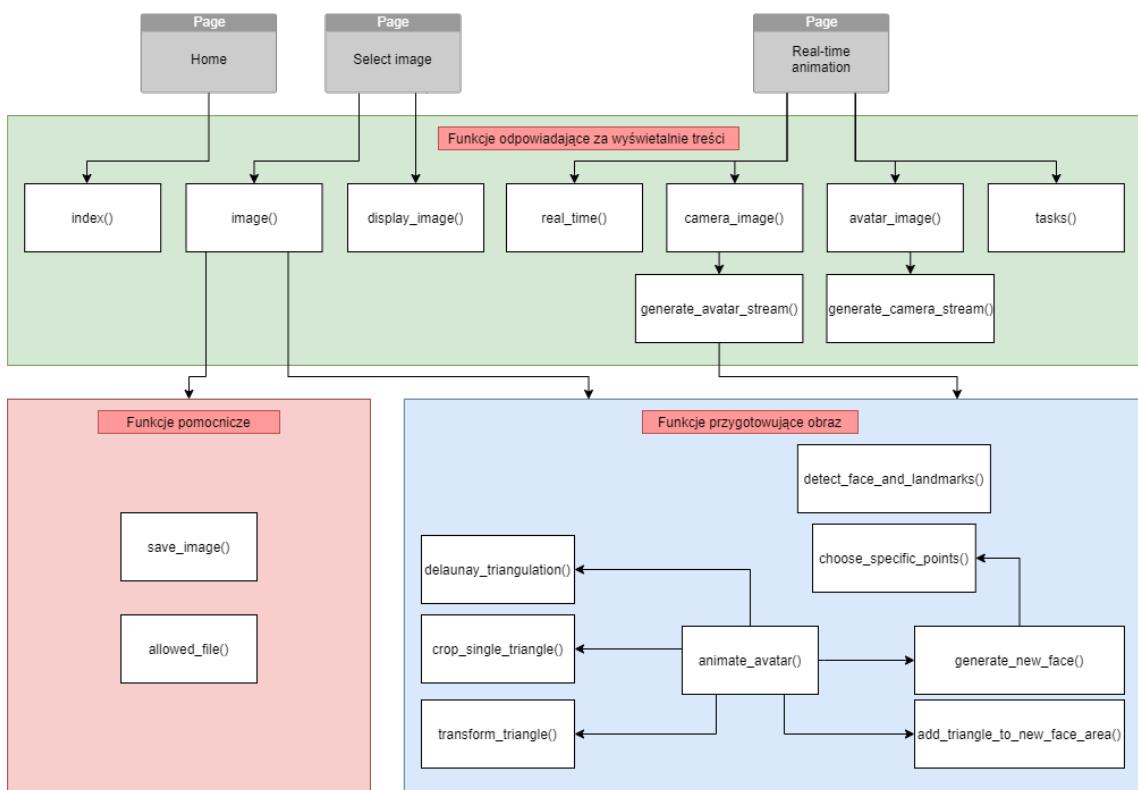
Rys. 5.7. Rezultat działania algorytmu animacji awatara (b) na zdjęciu źródłowym 5.6 (a) oraz na przykładowym obrazie awatara (a)

Po dokonaniu owej operacji wystarczy dodać do siebie obrazy 5.6c oraz 5.6d, aby otrzymać finalny rezultat przedstawiony na Rys. 5.7b, gdzie można obserwować zmodyfikowany wyraz twarzy awatara.

5.6. Aplikacja webowa

Na potrzeby przeprowadzenia ewaluacji algorytmu zaimplementowanego w tej pracy inżynierskiej, stworzono prostą aplikację webową zbudowaną na minimalistycznym szkielecie Flask. Skorzystano również z biblioteki Bootstrap, aby w prosty sposób dostosować wygląd strony do wymagań użytkownika.

Grafika zamieszczona poniżej (Rys. 5.8) przedstawia strukturę aplikacji webowej. Umieszczone na niej funkcje, które zostały opisane we wcześniejszych sekcjach oraz pozostałe moduły zaimplementowane na potrzeby stworzenia poprawnie działającej aplikacji.



Rys. 5.8. Struktura aplikacji webowej

Aplikacja webowa składa się z trzech podstron. Dla dwóch z nich na Rys. 5.9 przedstawiono wizualizację interfejsów graficznych. Funkcje, które pełnią opisano poniżej.

1. Krótka instrukcja zamieszczona na stronie głównej ma na celu zapoznanie użytkownika z możliwościami działania programu (zakładka **Home**).
2. Zakładka **Select image** umożliwia osobie zainteresowanej wgranie dwóch dowolnych zdjęć, gdzie jedno odpowiada obrazowi źródłowemu, na podstawie którego nastąpi animacja drugiego obrazu. Obsłużono wszelkiego rodzaju błędy, takie jak próba wczytania złego rozszerzenia pliku bądź zdjęcia, na którym nie da się wykryć twarzy i punktów charakterystycznych.

3. Podstrona ***Real-time animation*** umożliwia użytkownikowi przetestowanie działania algorytmu w czasie rzeczywistym na wybranym, z czterech możliwych, awatarze. Po przyściąnięciu przycisku *Animate* następuje zastosowanie algorytmu na zdjęciu awatara dla aktualnego obrazu przechwyconego z kamery. W przypadku braku wykrycia twarzy, efekt nie zostanie osiągnięty.

AvatarAnimator Home Select image Real-time animation

Select an image on the basis of which the avatar will be animated

Select avatar

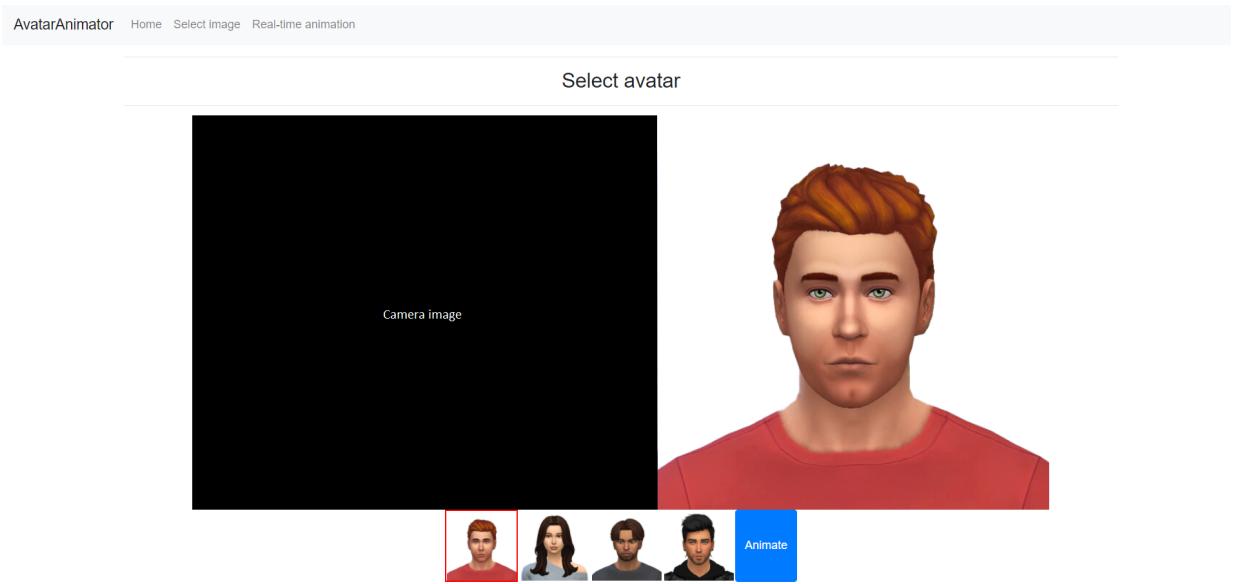
Upload images

Choose File No file chosen

Choose File No file chosen

Animate avatar

(a) Zakładka *Select image* - wybór dwóch zdjęć wykorzystanych do animacji



(b) Zakładka *Real-time animation* - wykorzystanie do animacji obrazu z kamery

Rys. 5.9. Interfejsy graficzne aplikacji webowej

6. Ewaluacja i rezultaty

Poniższy rozdział zawiera opis testów przeprowadzonych w celu zbadania poprawności działania algorytmu animacji awatara. Przedstawiono w nim również otrzymane rezultaty i wnioski płynące z opracowanej walidacji.

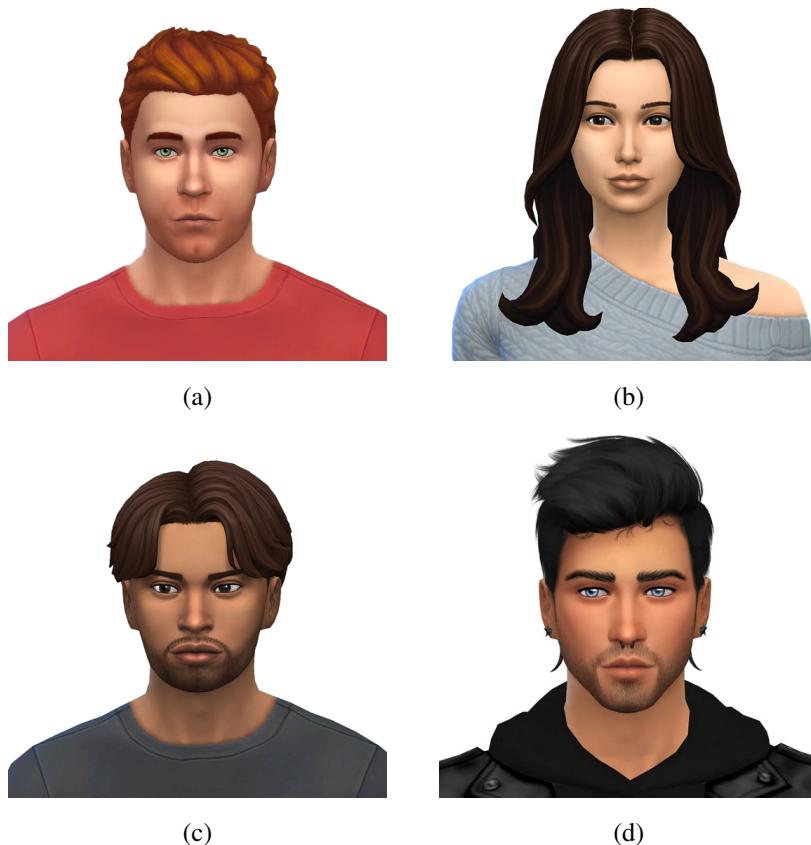
6.1. Sposób testowania

Uprednie przetestowanie stworzonego programu to kluczowa kwestia, dzięki której łatwiej wykryć błędy mogące pojawić się podczas użytkowania danego oprogramowania. Istotna jest również opinia osób testujących program. Niejednokrotnie zwracają oni uwagę na braki, których uzupełnienie potrafi zdecydowanie usprawnić korzystanie z aplikacji.

W czasie tworzenia algorytmu animacji awatara zostały przeprowadzone testy manualne poszczególnych funkcji, co pomogło usprawnić niektóre moduły i dopracować logikę ich działania. Ze względu na prostotę programu, stworzonego na potrzeby tej pracy inżynierkiej, wykonywanie bardziej zaawansowanych testów samego oprogramowania nie było konieczne.

Ważnym elementem stała się weryfikacja efektywności działania algorytmu na zarejestrowanych obrazach lub gotowych zbiorach danych, co wcześniej zostało określone w założeniach projektowych (sekcja 1.3). Z tego powodu zrealizowano testy użyteczności, które zostały podzielone na dwie części. W każdej z nich modyfikowano zdjęcia czterech różnych awatarów (Rys. 6.1), co pozwoliło na dokładniejsze przetestowanie algorytmu.

Dobór zdjęć użytych do badań nie był przypadkowy. W ramach niniejszej pracy jako awatar rozumie się nierealną cyfrową postać mającą formę osobową. Ważną cechą jest prostota takiego charakteru - wyraźne rysy twarzy, brak cieni. Ze względu na owe wymagania do testów wybrano podobizny z popularnej gry symulacyjnej The Sims 4, które zostały udostępnione na forum [32].



Rys. 6.1. Awatary użyte do przetestowania algorytmu

6.2. Walidacja z użyciem aplikacji

Pierwsza część wykonanych testów objęła wykorzystanie aplikacji webowej w celu zbadań efektywności działania algorytmu w przypadku animacji na podstawie obrazu przechwyconego z kamery.

Na potrzeby przeprowadzonego badania stworzono ankietę składającą się z czterech sekcji, gdzie każda dotyczy modyfikacji innego awatara. Głównym zadaniem miała być ocena rezultatów odwzorowania czterech podstawowych emocji [33], na które składają się:

- szczęście,
- smutek,
- strach/zaskoczenie,
- gniew/obrzydzenie.

Każda sekcja złożona jest z pięciu pytań:

- cztery z nich dotyczą testowania zakładki *Real-time animation*, gdzie awatar jest animowany na podstawie obrazu przechwyconego z kamery,
- jedno ma na celu porównanie efektów otrzymanych przez animację opartą na analizowaniu obrazu z kamery z modyfikacją na podstawie wczytanego zdjęcia (zakładka *Select image*).

Treść czterech pierwszych pytań, pod którymi kryją się kolejne zadania, różni się między sobą tylko aktualnie testowaną emocję. Dla przykładu pierwsze pytanie ma postać:

Odwzorowanie pierwszej podstawowej emocji (szczęście).

W tej części zastosowano ocenianie za pomocą skali od 1 do 5, gdzie konkretnym wartościami odpowiadały następujące etykiety:

1. Bardzo słaby
2. Słaby
3. Przeciętny
4. Dobry
5. Bardzo dobry

Ostatnie, piąte pytanie dotyczy oceny rezultatu osiągniętego w zakładce *Select image* dla danego awatara. Osoba ankietowana powinna wybrać jedno zdjęcie z udostępnionej bazy, odwzorowujące emocje, dla której animacja wypadła najgorzej. Następnie należało ocenić osiągnięty efekt, w stosunku do tego otrzymanego w zakładce *Real-time animation*. Tym razem użyto skali od 1 do 7 gdzie 1 oznacza *zdecydowanie gorszy* efekt, natomiast 7 *zdecydowanie lepszy* wynik.

Przed przystąpieniem do testowania użytkownicy zostali zapoznani z krótką instrukcją, w której zwrócono uwagę na czynniki mogące mieć wpływ na działanie algorytmu:

1. Ustaw twarz w pozycji frontowej, tak aby była ona dobrze widoczna w kamerze.
2. Odsłoń włosy oraz zdejmij okulary.
3. Staraj się nie wykonywać gwałtownych ruchów w momencie naciskania przycisku *Animate*.
4. W trakcie oceny rezultatów kieruj się głównie tym, czy emocja została poprawnie odwzorowana.

5. Możesz wielokrotnie dokonywać animacji dla danej emocji i ocenić sumaryczne rezultaty.

W ankiecie wzięło udział 11 osób w przedziale wiekowym od 19 do 60 lat. Większość z nich korzysta na co dzień z komputera, jednak tylko w podstawowych celach. W trakcie przeprowadzania ankiety nie wyniknęły żadne problemy. Aplikacja działała płynnie, przez co każdej ankietowanej osobie udało się ukończyć zadania bez przeszkód.

awatar	emocja	ocena				
		1	2	3	4	5
6.1a	szczęście	0%	9%	0%	45%	45%
	smutek	0%	0%	0%	64%	36%
	zaskoczenie/strach	0%	0%	18%	45%	36%
	gniew/obrzydzenie	0%	9%	18%	36%	36%
6.1b	szczęście	0%	9%	9%	45%	36%
	smutek	0%	27%	9%	45%	18%
	zaskoczenie/strach	9%	0%	27%	55%	9%
	gniew/obrzydzenie	9%	18%	45%	27%	0%
6.1c	szczęście	0%	0%	9%	45%	45%
	smutek	0%	0%	27%	55%	18%
	zaskoczenie/strach	0%	9%	18%	36%	36%
	gniew/obrzydzenie	0%	18%	18%	36%	27%
6.1d	szczęście	0%	0%	0%	18%	82%
	smutek	0%	9%	0%	55%	36%
	zaskoczenie/strach	0%	9%	9%	55%	27%
	gniew/obrzydzenie	0%	9%	9%	64%	18%

Tabela 6.1. Wyniki walidacji przeprowadzonej z użyciem aplikacji webowej

Procentowe wyniki ankiety przedstawiono w powyższej tabeli (Tab. 6.1). Zawarto w niej podsumowane odpowiedzi na pytania 1-4, czyli te dotyczące odwzorowania danej emocji. Dla każdego awatara i poszczególnych emocji dominujące wartości oznaczono kolorem zielonym.

Analizując wyniki można dojść do następujących wniosków:

- najlepsze oceny zanotowano dla awatara 6.1d. Odwzorowanie każdej z emocji zostało ocenione jako *dobre* lub *bardzo dobre* średnio w 89% przypadków,
- ocena *bardzo słaby* pojawiła się dwukrotnie, wyłącznie dla awatara 6.1b,

- największa liczba ocen *słaby* oraz *bardzo słaby* wystąpiła dla awatara 6.1b (średnio 18% odpowiedzi).

Podsumowując odpowiedzi na ostatnie pytanie, dotyczące porównania efektów osiągniętych w zakładce *Select image* oraz *Real-time animation*, można zauważyć, że:

- dla awatara 6.1a, w 91% przypadków, odpowiedzi skupiły się w przedziale od 4 do 7, co oznacza tendencję w kierunku *zdecydowanie lepszego* efektu,
- oceny awatara 6.1b kumulowały się między 1 a 4, przez co rozumie się osiągnięcie głównie gorszych rezultatów,
- dla awatara 6.1c koncentracja odpowiedzi przypadła na przedział 4-7, aż w 45% zdecydowano że efekt modyfikacji poprzez zakładkę *Select image* jest *zdecydowanie lepszy*,
- efekty dla ostatniego awatara (6.1d) zostały ocenione bardzo różnorodnie, żadna z odpowiedzi nie wydaje się znacząco dominować.

6.3. Walidacja na zbiorze danych

Drugi etap testów dotyczył oceny rezultatów algorytmu osiągniętych z wykorzystaniem gotowego zbioru danych. Celem tej części była ocena wcześniej zmodyfikowanych awatarów. Powinna ona zapewnić przejrzyste, nieobarcone błędem wnioski. Wyniki walidacji przedstawionej we wcześniejszej sekcji są narażone na przekłamanie, ponieważ efekty zależały od mimiki poszczególnych osób biorących udział w badaniu.

Przygotowanie danych przebiegało w następujący sposób:

- wykorzystano bazę zdjęć zawierającą podobizny mężczyzn oraz kobiet z różnymi wyrazami twarzy, odpowiadającymi określonym emocjom,
- dla każdej emocji, których podstawowy podział został podany w powyższym rozdziale, wybrano po jednym zdjęciu,
- na podstawie określonych obrazów dokonano modyfikacji każdego awatara. Finalnie otrzymano 16 przekształconych podobizn.

Ankieta utworzona na potrzeby badania składa się z 16 pytań, każde z nich posiada cztery możliwe odpowiedzi:

1. szczęście,
2. smutek,

3. strach/zaskoczenie,
4. gniew/obrzydzenie.

Zadaniem osoby ankietowanej była próba dopasowania emocji, którą odzwierciedla zmodyfikowany wyraz twarzy awatara. W celu łatwiejszego uporządkowania wyników zastosowano punktację 0-1.

W ankiecie wzięło udział 27 osób w różnym przedziale wiekowym. Poniżej przedstawiono wartości najważniejszych parametrów otrzymanych wyników:

- maksymalna liczba zdobytych punktów - 16
- najniższy wynik - 9
- średni wynik - 12,44/16
- mediana - 12

emocja	awatar			
	6.1a	6.1b	6.1c	6.1d
szczęście	74%	70%	85%	100%
smutek	78%	93%	85%	85%
zaskoczenie/strach	93%	74%	63%	63%
gniew/obrzydzenie	78%	41%	96%	67%
sumaryczna poprawność	81%	69%	82%	79%

Tabela 6.2. Ocena rezultatów osiągniętych z użyciem gotowego zbioru danych

Wyniki procentowe otrzymane po przeprowadzeniu badania zawarto w tabeli 6.2. Po szczególowej analizie, można zauważyć, że:

- najwyższej oceniono awatar 6.1c (82% poprawnych odpowiedzi), a najgorzej 6.1b (poprawność na poziomie 69%),
- w odwzorowaniu szczęścia najwyższy wynik, bo aż 100% zanotowano dla awatara 6.1d,
- smutny wyraz twarzy najtrajniej przyporządkowywano dla awatara 6.1b (93%),
- w przypadku emocji zaskoczenia/strachu najwyższą poprawność dopasowania otrzymano dla awatara 6.1a (93%),
- najlepszy rezultat dla gniewu/obrzydzenia osiągnięto na awatarze 6.1c (96%).

Obrazy, których odzwierciedlane emocje przyporządkowano z najlepszą skutecznością (kolejno 100% oraz 96%) przedstawiono na Rys. 6.2. Według ankietowanych obraz 6.2a bardzo dobrze oddaje szczęście, mimo pewnego defektu w postaci braku uzębienia. Natomiast Rys. 6.2b poprawnie ukazuje gniew/obrzydzenie poprzez odpowiednie ułożenie ust oraz oczu.



Rys. 6.2. Najtrajniej odgadnięte odwzorowania

6.4. Wnioski

Przeprowadzone testy pomogły sprawdzić poprawność działania algorytmu zaimplementowanego w niniejszej pracy dyplomowej. Według użytkowników algorytm zapewnia dobre efekty, a stworzona aplikacja jest intuicyjna w użyciu. Wiele osób stwierdziło, iż udział w badaniu był przyjemny i przysporzył sporo rozrywki.

Analiza przeprowadzona w powyższym rozdziale pozwoliła na wykrycie wad oraz zdefiniowanie potencjalnych kierunków rozwoju programu. Kilku użytkowników zwróciło uwagę na brak uzębienia animowanych awatarów, który rozpraszał w trakcie testowania algorytmu. Pojedyncze osoby wspominały także o braku uwzględniania ułożenia brwi. Niewykluczone, że poprawa tego elementu zdecydowanie udoskonaliłaby odwzorowanie emocji, które można osiągnąć z użyciem owego oprogramowania.

Głównym celem ewaluacji było zbadanie jakości działania programu ze względu na dobór awatara. Okazało się, że efektywność narzędzia w dużym stopniu zależy od obrazu, na którym następuje animacja. W pierwszej części testów najwyżej oceniono obraz 6.1d, natomiast w drugiej 6.1c. Najniższe oceny, w obu przypadkach, pojawiły się dla awatara 6.1b. Podczas jego modyfikacji można było zauważać sporo artefaktów psujących efekt animacji. Jak widać, mimo wykorzystania identycznych obrazów, dla różnych awatarów otrzymano odmienne wyniki. W przypadku kobiety znajdującej się na obrazie 6.1b włosy nachodzące na twarz mogły negatywnie wpływać na rezultaty.

Istotną kwestią była także część, w której porównywano działanie oprogramowania na gotowych obrazach z animacją na podstawie obrazu przechwyconego z kamery. Dla awatarów ocenionych najwyżej osiągnięte efekty były zbliżone. Dla awatara ocenionego najgorzej efektywność animacji na podstawie gotowych obrazów dawała lepsze wyniki. Można z tego wnioskować, iż poprawnie wykonana animacja z użyciem kamery nie wpływa negatywnie na wizualację, a w przypadku animacji niekorzystnego awatara może poprawić jej efekt. Nie należy jednak zapominać, że efekty są mocno zależne od mimiki danej osoby - dla każdej z nich inny awatar może sprawdzić się lepiej.

Uważam, iż przeprowadzone testy przebiegły pomyślnie. Stworzona aplikacja okazała się dobrym narzędziem walidacji. Ewaluacja na gotowym zbiorze danych pomogła zweryfikować otrzymane wnioski oraz zwrócić uwagę na inne ścieżki rozwoju.

7. Podsumowanie

Główym celem niniejszego projektu dyplomowego było opracowanie algorytmu animacji awatara. Zaimplementowana metoda miała opierać swoje działanie na identyfikacji punktów charakterystycznych twarzy, będących podstawą dalszych transformacji. W ramach walidacji oprogramowania należało napisać nieskomplikowaną aplikację webową.

Stworzenie owego algorytmu wymagało wiele zaangażowania oraz zgłębienia tematów, których znajomość pomogła rozwiązać problemy stawiane w poszczególnych etapach. Zadanie zakończyło się sukcesem, co potwierdzają opinie ankietowanych osób. Utworzone narzędzie zdaje się działać poprawnie i pozwala osiągnąć zadowalające efekty. Aktualna wersja programu spełnia określone na początku wymagania i tworzy dobrą podstawę do dalszego rozbudowywania. Udało się także zaimplementować aplikację webową, która ułatwia przedstawienie algorytmu oraz jego przetestowanie.

7.1. Weryfikacja początkowych założeń

W pierwszym rozdziale owej pracy (sekcja 1.3) wymieniono założenia, które w etapie końcowym powinien spełniać stworzony projekt. Poniżej zostanie przedstawiona ich analiza oraz weryfikacja osiągniętych efektów:

- Działanie algorytmu miało zostać powiązane z analizą punktów charakterystycznych twarzy, co w stu procentach osiągnięto. Algorytm rozpoczyna działanie od ich wykrycia. Na tej podstawie odbywają się kolejne etapy, które dotyczą modyfikacji obszarów wyznaczonych przez dane punkty.
- Samą identyfikację punktów udało się zrealizować z wykorzystaniem biblioteki dlib, dzięki której odbywa się wykrycie twarzy oraz jej istotnych elementów.
- Po stworzeniu projektu algorytm przetestowano według planu, na gotowym zbiorze danych, co umożliwiło sprawdzenie poprawności jego działania.
- Zgodnie z planem, zaimplementowano prostą aplikację webową, wykorzystaną do walidacji algorytmu.

Podsumowując, wszystkie założenia postawione na początku tworzenia programu zostały zrealizowane. Dzięki cennym informacjom zdobytym podczas przeprowadzonych badań pozostała kwestią jego optymalizacji i dalszego rozwoju.

7.2. Możliwy rozwój projektu

Działanie algorytmu zaimplementowanego w trakcie tworzenia niniejszej pracy dyplomowej można usprawnić na wiele sposobów. Udoskonalenie niektórych etapów z pewnością poprawiłoby osiągane rezultaty:

- Pierwszym ważnym elementem jest próba dodania rozwiązań, które zapewni lepszy efekt wizualny w przypadku, gdy usta awatara się rozszerzają - warto zastanowić się nad kwestią pokazania uzębienia.
- Ponadto dotychczasowy czas działania niektórych funkcji wykorzystanych w programie zdaje się być zbyt długi. Niewykluczone, że lepszym rozwiązaniem byłoby zastąpienie ich wydajniejszymi modułami - być może należy takowe opracować. Prawdopodobnie poprawa czasu działania algorytmu możliwałaby udoskonalenie aplikacji - stworzenie filtra animującego awatar w czasie rzeczywistym.
- Uważam również, że warto przemyśleć etap nałożenia maski i być może spróbować rozwiązać go w inny sposób. W tym momencie maska dopasowywana jest poprzez wykorzystanie funkcji z biblioteki scikit-image, która transformuje obraz na podstawie wybranych punktów charakterystycznych. Rezultat wydawał się zadowalający, jednakże podczas testów dwie osoby zwróciły uwagę na brak animacji brwi, co w tym momencie nie jest możliwe, ze względu na to że owe punkty biorą udział w transformacji - ich modyfikacja jest niemożliwa.
- Niewykluczone, że dobranie modelu z inną ilością punktów charakterystycznych również wpłynęłoby na poprawę efektów jakie zapewnia działanie algorytmu.

Wydaje mi się, że wszystkie wymienione wyżej propozycje poprawiłyby efekty, które w tym momencie można osiągnąć używając algorytmu animacji awatara. Warto również rozwinąć aplikację stworzoną na potrzeby ewaluacji, dokładniej przetestować jej działanie i popracować nad interfejsem graficznym.

Bibliografia

- [1] Oskar Pacelt. *Sztuczna inteligencja a przyszłość ludzkości*. URL: <https://botland.com.pl/blog/sztuczna-inteligencja-a-przyszlosc-ludzkosci/> (term. wiz. 2021-10-25).
- [2] *Computer Vision - What it is and why it matters*. URL: https://www.sas.com/pl_pl/insights/analytics/computer-vision.html (term. wiz. 2021-10-27).
- [3] Corinne Bernstein. *Face Detection*. URL: <https://searchenterpriseai.techtarget.com/definition/face-detection> (term. wiz. 2021-10-29).
- [4] *Face Detection*. URL: https://en.wikipedia.org/wiki/Face_detection (term. wiz. 2021-10-30).
- [5] Ming-Hsuan Yang, D.J. Kriegman i N. Ahuja. „Detecting faces in images: a survey”. W: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.1 (2002), s. 35–36. DOI: [10.1109/34.982883](https://doi.org/10.1109/34.982883).
- [6] 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. 511–518. DOI: [10.1109/CVPR.2001.990517](https://doi.org/10.1109/CVPR.2001.990517).
- [7] *Haar features image*. URL: https://commons.wikimedia.org/wiki/File:VJ_featureTypes.svg (term. wiz. 2021-11-07).
- [8] Socret Lee. *Understanding Face Detection with the Viola-Jones Object Detection Framework*. URL: <https://towardsdatascience.com/understanding-face-detection-with-the-viola-jones-object-detection-framework-c55cc2a9da14> (term. wiz. 2021-11-10).
- [9] 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](https://doi.org/10.1109/CVPR.2005.177).
- [10] Satya Mallick. *Histogram of Oriented Gradients explained using OpenCV*. URL: <https://learnopencv.com/histogram-of-oriented-gradients/> (term. wiz. 2021-11-05).

- [11] *Facial landmark detector with dlib*. URL: <https://www.pyimagesearch.com/2018/04/02/faster-facial-landmark-detector-with-dlib/> (term. wiz. 2021-11-12).
- [12] Vahid Kazemi i Josephine 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](https://doi.org/10.1109/CVPR.2014.241).
- [13] Pavel Korshunov i Sébastien Marcel. „Speaker Inconsistency Detection in Tampered Video”. W: 2018-09, s. 2377. DOI: [10.23919/EUSIPCO.2018.8553270](https://doi.org/10.23919/EUSIPCO.2018.8553270).
- [14] *Triangulacja*. URL: [https://pl.wikipedia.org/wiki/Triangulacja_\(matematyka\)](https://pl.wikipedia.org/wiki/Triangulacja_(matematyka)) (term. wiz. 2021-11-06).
- [15] *Triangulacja Delone*. URL: https://pl.wikipedia.org/wiki/Triangulacja_Delone (term. wiz. 2021-11-06).
- [16] *Delaunay triangulation*. URL: https://en.wikipedia.org/wiki/Delaunay_triangulation (term. wiz. 2021-11-06).
- [17] Sally Adee. *What Are Deepfakes and How Are They Created? Deepfake technologies: What they are, what they do, and how they're made*. URL: <https://spectrum.ieee.org/what-is-deepfake> (term. wiz. 2021-11-12).
- [18] *Aplikacja Wombo.ai*. URL: <https://www.wombo.ai/> (term. wiz. 2021-11-13).
- [19] *Aplikacja Anyface*. URL: <https://play.google.com/store/apps/details?id=com.friendzy.Perefase> (term. wiz. 2021-11-13).
- [20] *Aplikacja MotionPortrait*. URL: <https://play.google.com/store/apps/details?id=com.motionportrait.MotionPortrait&hl=pl&gl=US> (term. wiz. 2021-11-13).
- [21] *Python*. URL: <https://pl.wikipedia.org/wiki/Python> (term. wiz. 2021-11-17).
- [22] *The State of Developer Ecosystem 2021*. URL: <https://www.jetbrains.com/lp/devecosystem-2021/> (term. wiz. 2021-12-19).
- [23] *Python Applications*. URL: <https://www.javatpoint.com/python-applications> (term. wiz. 2021-11-18).
- [24] *OpenCV Official Page*. URL: <https://opencv.org/about/> (term. wiz. 2021-11-18).
- [25] *Imutils Github Repository*. URL: <https://github.com/PyImageSearch/imutils> (term. wiz. 2021-11-21).
- [26] *Dlib Official Page*. URL: <http://dlib.net/> (term. wiz. 2021-11-22).
- [27] *Scikit-image Official Page*. URL: <https://scikit-image.org/> (term. wiz. 2021-11-21).
- [28] *What is Flask Python*. URL: <https://pythonbasics.org/what-is-flask-python/> (term. wiz. 2021-11-21).

- [29] *PyCharm*. URL: <https://pl.wikipedia.org/wiki/PyCharm> (term. wiz. 2021-11-27).
- [30] *Dlib trained facial model*. URL: http://dlib.net/files/shape_predictor_68_face_landmarks.dat.bz2 (term. wiz. 2021-10-01).
- [31] Natalie Ebner, Michaela Riediger i Ulman Lindenberger. „FACES—A database of facial expressions in young, middle-aged, and older women and men: Development and validation”. W: *Behavior research methods* 42 (2010-02), s. 351–362. DOI: [10.3758/BRM.42.1.351](https://doi.org/10.3758/BRM.42.1.351).
- [32] *Sims community page*. URL: <https://simscommunity.info/> (term. wiz. 2021-12-01).
- [33] *Wszystko, co należy wiedzieć o podstawowych emocjach*. URL: <https://noizer.pl/emocje-podstawowe-4-podstawowe-emocje/> (term. wiz. 2021-12-28).