AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Projekt dyplomowy

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, 2021

Spis treści

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 poprawną interpretację i zrozumienie otaczającego nas świata - tak jak robią to ludzie. Poprzez wykorzystanie obrazów cyfrowych i odpowiednich modeli tworzone oprogramowania potrafią 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. Narzędzie to ma też wiele innych zastosowań, jest pewnego rodzaju bazą dla bardziej rozbudowanych programów. W niniejszej pracy dyplomowej wykrywanie twarzy, będące podstawą wspomnianego wyżej algorytmu rozpoznawania twarzy, odegra znaczącą rolę. Będzie 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 zdecydowałam 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 społecznośiowe udostępniają specjalne filtry, nakładki na zdjęcia działające na podobnej zasadzie. Jednak większość dostępnego oprogramowania nie oferuje wglądu w kod źródłowy, są to rozwiązania komerycjne, 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 jest opracowanie algorytmu animacji awatara. Metoda będzie opierać się na identyfikacji punktów charakterystycznych (ang. landmarks) oznaczających położenie łuków brwiowych, powiek, oczu, nosa i warg. Na podstawie wyznaczonych punktów i ich przemieszczeń będą obliczane przemieszczenia analogicznych punktów awatara, według których jego obraz zostanie poddany lokalnym transformacjom.

Algorytm zostanie przedstawiony z wykorzystaniem nieskomplikowanej aplikacji webowej, która posłuży jako narzędzie walidacji.

1.3. Założenia projektu

Projekt stworzony na potrzeby tejże pracy powinien spełniać określone założenia, zgodne z poniższymi szczegółami:

- 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 zostaną 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 pokrótce zaznajomić odbiorcę z jego strukturą. Kolejne części zawierają szczegóły każdego z etapów animacji awatara. Natomiat 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 rozdział przedstawiający 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 z 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 wykrywania 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.

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

Rys. 2.1. Techniki detekcji twarzy według Younga

– metody bazujące na wiedzy - wykorzustujące ludzkie obeznanie na temat 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 wzorcom 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.

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 w trakcie 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 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

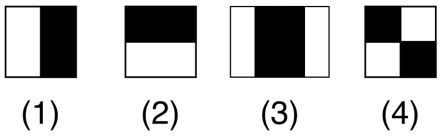
– utworzenie zintegrowanych obrazów w celu szybkiego znalezienia danej cechy

– 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 $24x24$ 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.



Rys. 2.2. Przykładowe funkcje Haara [7]

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 krawędzii na obrazie (1, 2) oraz prostych (3) i skośnych linii (4).

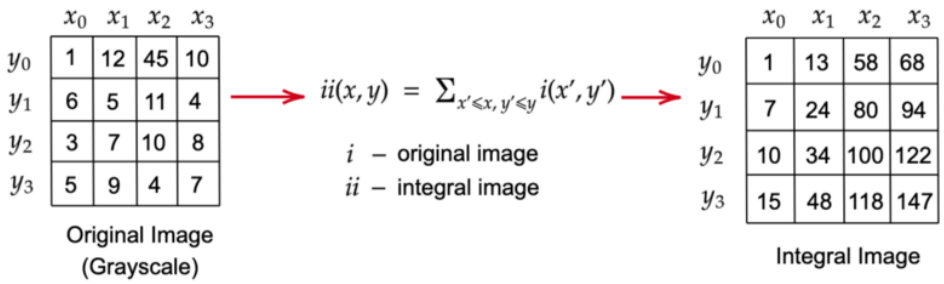
Każda funkcja Haara przemierza piksel po pikselu dany obraz. Dla każdego regionu (ciemnego jak i jasnego), który obejmuje owa cecha, sumowana jest wartość pikseli znajdujących się w tych obszarach. Następnie obliczana zostaje różnica między tymi sumami. Na podstawie tej różnicy, wyszukuje się lokalizację 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 znajdujacy 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. Rozwiązaniem jest stworzenie zintegrowanego obrazu, dzięki czemu złożoność obliczeń 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.

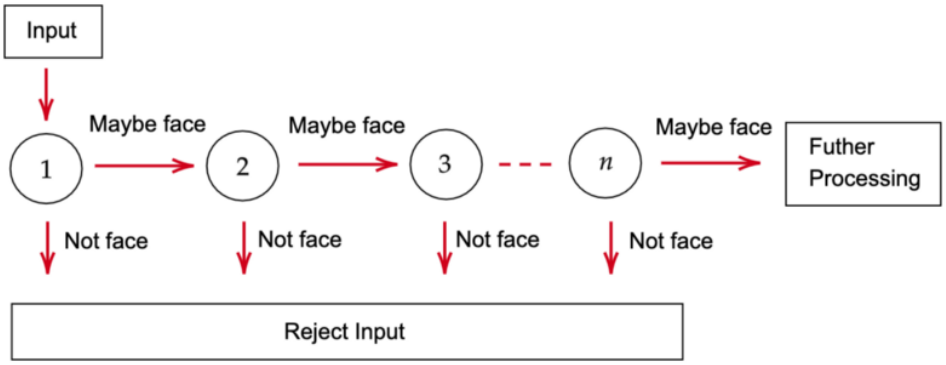


Rys. 2.4. Konwersja na obraz zintegrowany [7]

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 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 wyodrębniane na obrazie. W przypadku negatywnego rezultatu, tzn. niedopasowania jednej z cech sprawdzanych w 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 [7]

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 elementówch 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 skorzystać z deskryptora HOG (ang. Histograms of Oriented Gradients) oraz klasyfikatora SVM (ang. Support Vector Machine) [8].

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 [9]. Pierwszym z nich jest przygotowanie obrazu w odcieniach szarości, dla którego chcemy obliczyć histogram gradientów. Według autorów algorytmu rozmiar zdjęcia powinno się zmniejszyć do $128x64$ 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 poniższe jądra.

Rys. 2.6. Jądra służące do filtracji obrazu w celu obliczenia gradientów [9]

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

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

Kolejnym krokiem jest podział obrazu na jednakowe siatki o wymiarach $8x8$. Każdą komórkę da się przedstawić za pomocą 128 liczb. Pojedynczy fragment składa się z 64 pikseli, z każdym związane są ważne wartości dotyczące jego gradientu - wielkość i kierunek ($8x8x2 = 128$).

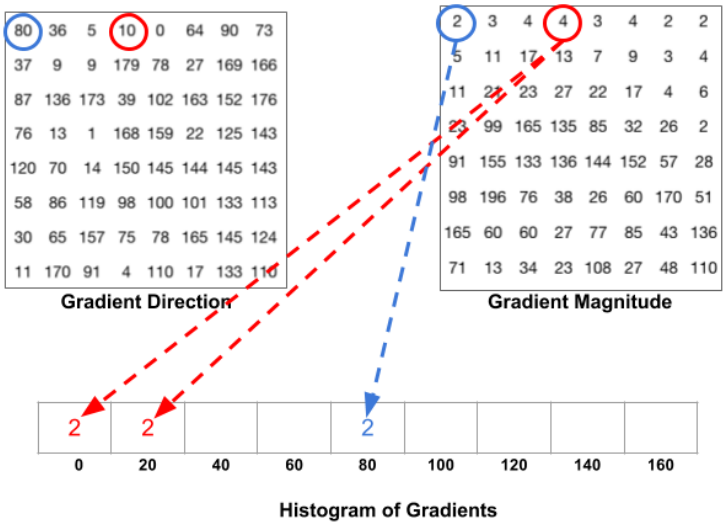
Aby skompresować dane dla każdej siatki, należy stworzyć histogram z podziałem na dziewięć oddzielnych pojemników. Każdy z nich odpowiada kątom z zakresu 0-160 z 20-stopniowym przyrostem. Przyporządkowując każdy piksel do jednego z pojemników uwzględnia się kierunek gradientu (jego kąt) oraz wielkość, 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 miedzy oba pola. Omawiana sytuacja dotyczy piksela oznaczonego na czerwono na Rys. 2.7.

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 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 stosuję się normalizację histogramu.

Bloki przekształca się w wektory elementów, a następnie dla każdego stosuje się normalizację. Blok o wymiarach $16x16$ posiada 4 histogramy, gdzie każdy da się przedstawić jako wektor $9x1$ co w sumie



Rys. 2.7. Schemat tworzenia histogramu dla komórki o wymiarach $8x8$ [9]

daje wektor $36x1$. Normalizacja następuje dla pierwszego okna, po czym jest ono przesuwane o 8 pikseli gdzie znormalizowany wektor jest obliczany ponownie. Wszystko powtarza się do momentu przebycia wszystkich pozycji.

Każdy z omawianych bloków reprezentowany jest przez wektor $36x1$ 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 on się 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.

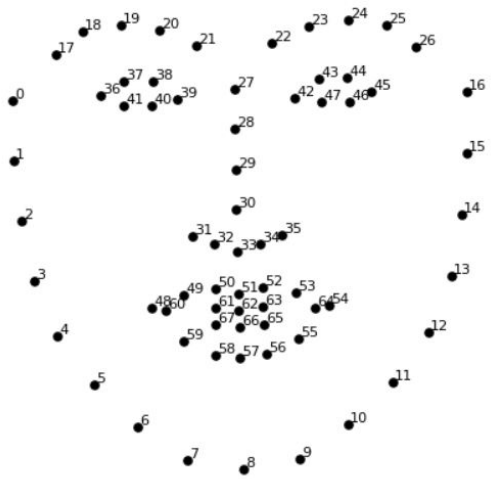
Analizowana w tej sekcji metoda stała się konkurecją dla algorytmu opisywanego we wcześniejszym podrozdziale. Dokładność jego działania jest dużo lepsza niż algorytmu opartego na klasyfikatorze kaskadowym. Minusem jest wykrywanie twarzy tylko w przypadku widoku z przodu, program nie działa poprawnie w momencie jej rotacji i zmian kąta. Podobnie jak algorytm przedstawiony w poprzednim rozdziale, to podejście wciąż cieszy się wielką popularnością. 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 których pojęcie rozumie się elementy, pomagające zidentyfikować jej najistotniejsze cechy. Zazwyczaj są to punkty utożsamiane z linią szczęki, ust, nosa, oczu i brwi [10].

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 rozpocząć 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 [11].



Rys. 2.8. Wizualizacja 68 punktów charakterystycznych twarzy

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 dowolonych 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.

2.3. Triangulacja Delaunaya

Poprzez pojęcie triangulacji, w kontekście matematycznym, można rozumieć podział figury geometrycznej na trójkąty, bądź też czworościany (określane jako sympleksy) w taki sposób, aby część wspólna dwóch sąsiadujących trójkątów (czworościanów) była ich wspólną ścianą, wierzchołkiem, bokiem, trójkątem lub zbiorem pustym [12].

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

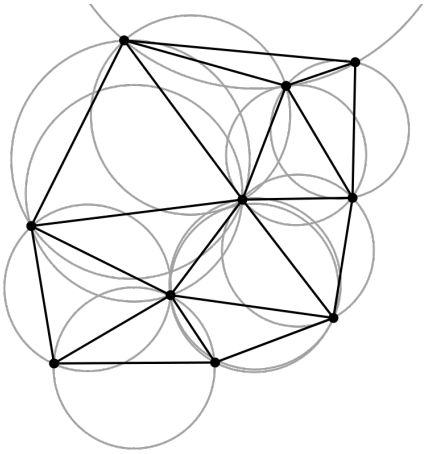
Triangulacja Delone [13] rozumiana jest jako triangulacja T przestrzeni R^{n+1}, którą definiuję się w poniższy sposób. Jako T określa się podział przestrzeni R^{n+1} na $(n+1)$ sympleksów, z punktami jako wierzchołki, spełniających okreśone warunki:

1. Każde dwa sympleksy należące do zbioru T posiadają wspólną ściane 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 dowolonym trójkącie ze zbioru T nie natkniemy się na sytuację, gdy wnętrze kuli będzie zawierało wierzchołki z pozostałych sympleksów zawartych w zbiorze T.

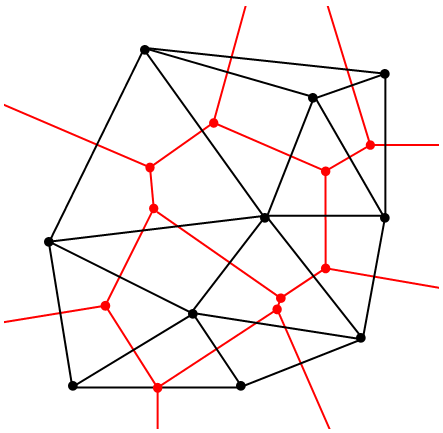
Wspominając o triangulacji Delone warto wspomnieć o diagramach Voronoi, które są ściśle powiązane z owym pojęciem. Diagram Voloroi dla zestawu punktów dzieli przestrzeń w taki sposób, że linie podziału znajdują się w równej odległości od punktów sąsiadujących.



Rys. 2.9. Przykładowa triangulacja Delone dla zbioru punktów [13]

Ważną własnością trangulacji Delone jest fakt, iż wraz z diagramem Voronoi tworzy ona graf dualny. Te definicje są powiązane, więc znając trangulację Delone dla zbioru punktów możemy w łatwy sposób obliczyć diagram Voronoi. Dwa trójkąty mające wspólną krawędź w triangulacji umożliwiają odnalezienie krawędzie w diagramie Voronoi.

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



Rys. 2.10. Graf przedstawiający komórki Vornoi oraz krawędzie triangulacji [13]

Istotną własnością 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 trangulacji 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 społecznośiowe 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 zbliżone 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 zamiana wypowiedzi jednej osoby, na wypowiedź innej, to samo dotyczy ruchów ciała. Istnieje także możliwość spreparowania dźwięku, przez co mamy wrażenie, że słyszymy słowa wypowiadane przez znajomą nam osobę, tymczasem jest to fikcja uzyskana z 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 kierownicy 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 [14].

Stworzenie filmu określanego 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 to 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 zostają 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 [15] 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 [16] to aplikacja mająca na celu, podobnie jak poprzedniczka, zapewnienie rozrywki użytkownikowi. Jest ona dostępna w wersji mobilnej tylko na Androida.

W odróżnieniu do wspomnianego wyżej programu to narzędzie posiada dużo więcej funkcji, przez co może sprawiać wrażenie bardziej atrakcyjne 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 [17] to bardzo podstawowa aplikacja mobilna dostępna zarówno na Androida 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, 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 ilość aplikacji podobnych do programów opisanych powyżej. Niektóre z nich są bardzo podstawowe 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.

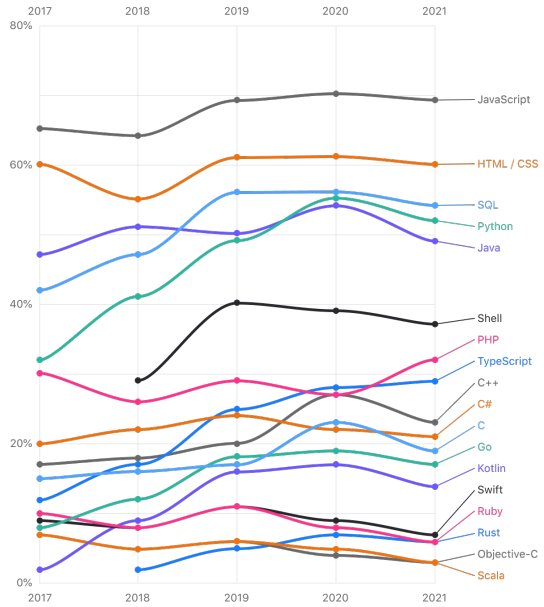
4. Wykorzystane narzędzia i technologie

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

4.1. Python

Język programowania Python [18] powstał we wczesnych latach dziewięćdziesiątych. Guido van Rossum jest uważany za głównego twórcę tego języka, jednak wkład w jego rozwój miały także inne osoby.

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 na wiele systemów operacyjnych, obsługuje większość używanych w dzisiejszych czasach platform, takich jak Windows, Linux, AIX, iOS i inne.



Rys. 4.1. Popularność wybranych języków programowania na przestrzeni lat

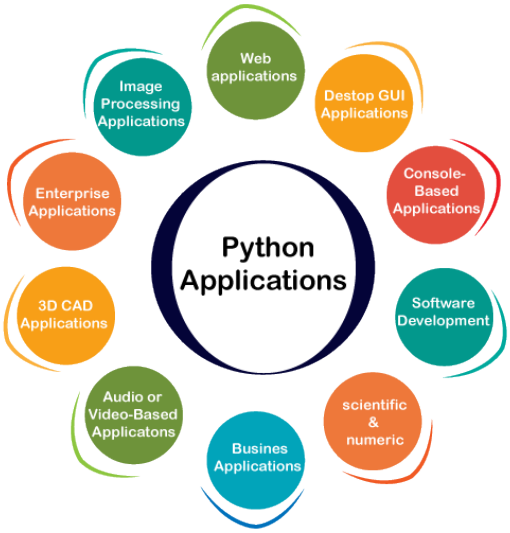
Projekt, w ramach którego zostaje rozwijany owy język zarządzany jest przez Python Software Foundation, będącą organizacją non-profit. Narzędzie zostało udostępnione jako otwarte oprogramowanie, przez co użytkownicy posiadają możliwość ingerowania w wydawany kod źrodłowy.

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.

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.



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

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 [19]. 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 [20] jest narzędziem wydanym jako otwarte oprogramowanie, wspomagające przetwarzanie obrazów oraz uczenie maszynowe. Została napisana w języku C, natomiasto posiada interfejsy dla innych języków, takich jak Java, Python bądź Matlab.

Zawiera około 2500 zoptymalizowanych algorytmów implementujących między innymi wykrywanie i rozpoznawanie 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 wyróżniać wydajnością, co jest bardzo istotne w przypadku działania na dużych zbiorach danych.

4.2.2. imutils

Pakiet imutils [21] 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 zdjeć 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 tenże 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++ [22]. 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 [23], podobnie jak opisane powyżej pakiety, zawiera szereg implementacji algorytmów powiązanych z przetwarzaniem obrazów. Została udostępniona jako otwarte oprogramowanie tworzone przez kilkadziesiąt osób.

Obejmuje ona 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 [24] 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 wymagałaby zapewnienia szczególnych bibliotek. Natomiast obsługuje rozszerzenia, ktore umożliwiają dodanie owych funkcji do aplikacji.

Flask dostarcza programiście 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. Zabezpieczna przed atakami poprzez wprowadzanie niezauwanych 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 ściśle 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 [25] 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).

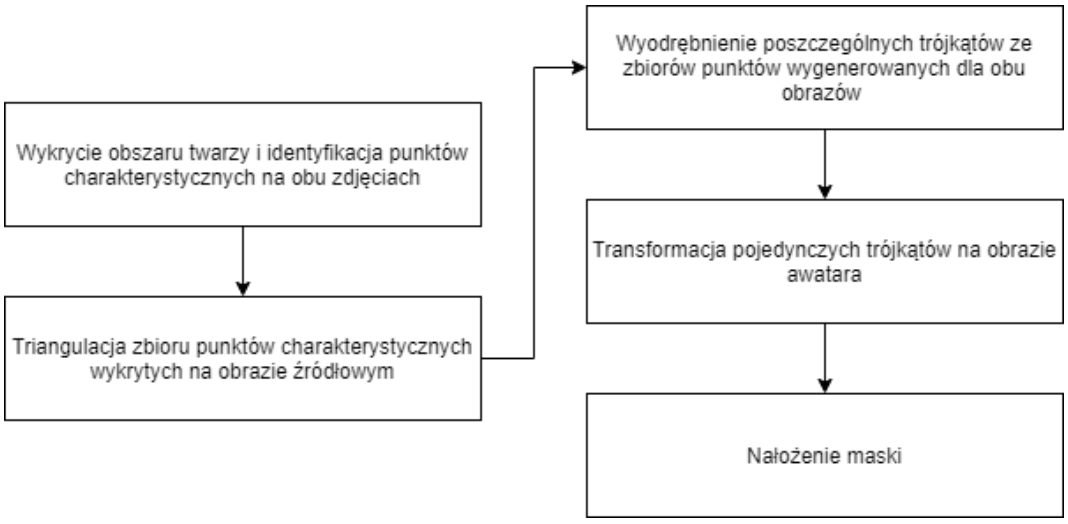
Ułatwia pracę z projektem dostarczając prosty interfejs konfiguracyjny. W przypadku tworzenia aplikacji webowej opartej na platformie Flask w nieskomplikowany sposób można wprowadzić wszystkie istotne ustawienia.

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 posłużć 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

```
def animate_avatar(avatar_img, avatar_points, src_img, src_points):
    new_avatar_face = np.zeros(src_img.shape, np.uint8)

    for src_triangle_points, avatar_triangle_points in
    ↪ delaunay_triangulation(src_points, avatar_points):
        avatar_img_cropped, avatar_triangle, _ =
        ↪ crop_single_triangle(avatar_triangle_points, avatar_img)
        src_img_cropped, src_triangle, b_rect =
        ↪ crop_single_triangle(src_triangle_points, src_img)

        warped_triangle = transform_triangle(avatar_triangle, avatar_img_cropped,
        ↪ src_triangle, src_img_cropped)
        add_triangle_to_new_face_area(new_avatar_face, warped_triangle, b_rect)

    return generate_new_face(avatar_img, avatar_points, new_avatar_face,
    ↪ src_points)
```

5.2. Wykrycie twarzy oraz punktów charakterystynczych

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, który również został udostępniony przez bibliotekę, umożliwiający poprawną lokalizację punktów charakterystycznych. Został on wyszkolony na podstawie algorytmu przedstawionego w sekcji 2.2. Wynikiem zastosowania danej funkcji jest obiekt, który generuje owy zestaw punktów zlokalizowanych na obrazie przekazanym jako dane wejściowe.

Wykrycie twarzy oraz punktów charakterystycznych

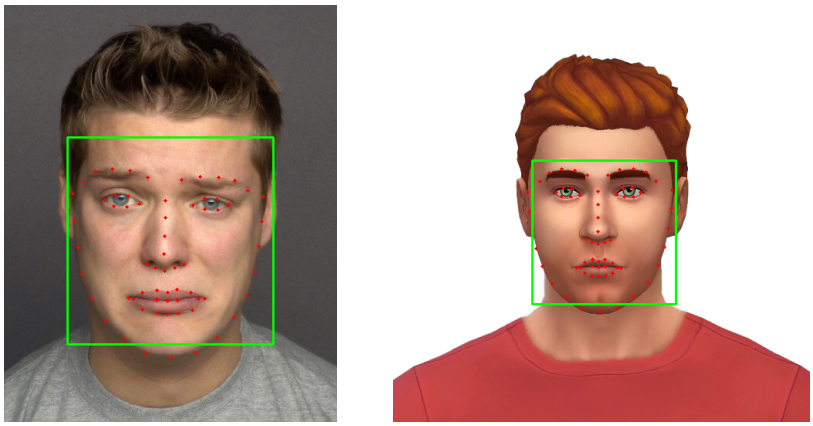
```
def detect_face_and_landmarks(img):
    detector = dlib.get_frontal_face_detector()
    predictor =
    ↪ dlib.shape_predictor('./data/shape_predictor_68_face_landmarks.dat')

    gray = cv2.cvtColor(imutils.resize(img), cv2.COLOR_BGR2GRAY)
    rects = detector(gray, 1)
    facial_landmarks = []

    if rects:
        for rect in rects:
            facial_landmarks = predictor(gray, rect)
```

```
12              facial_landmarks = face_utils.shape_to_np(facial_landmarks)
13
14      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.



(a) (b)

Rys. 5.2. Twarz i punkty wykryte na obrazie źródłowym (a) oraz na awatarze (b)

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

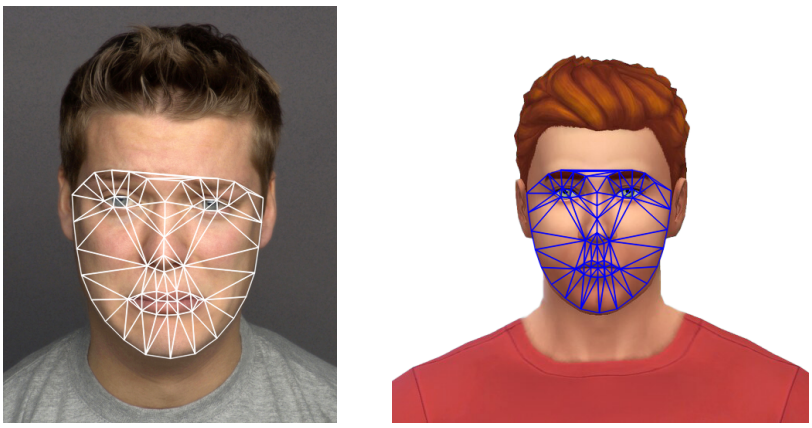
```
def delaunay_triangulation(src_points, avatar_points):
    points_list = list(src_points)
    delaunay_subdivision = cv2.Subdiv2D((*src_points.min(axis=0),
    ↪  *src_points.max(axis=0)))
    delaunay_subdivision.insert(points_list)

    for x1, y1, x2, y2, x3, y3 in delaunay_subdivision.getTriangleList():
        indexes_set = [(src_points == single_point).all(axis=1).nonzero()[0][0]
                       for single_point in [[x1, y1], [x2, y2], [x3, y3]]]

        src_triangles = generate_xy_for_indexes(src_points, indexes_set)
        avatar_triangles = generate_xy_for_indexes(avatar_points, indexes_set)
        yield [np.array(src_triangles), np.array(avatar_triangles)]


def generate_xy_for_indexes(points, indexes):
    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.



(a) (b)

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 wykrtyte trójkąty. Figury, które zostały otrzymane poprzez użycie funkcji z biblioteki OpenCV umożliwią 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. Fragemnt kodu przedstawionego poniżej umożliwia dokonanie tej operacji. Funkcja zwróci przycięty 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 indexes in
      ↪  single_triangle]
4     cropped_img = img[b_rect[1]:b_rect[1] + b_rect[3], b_rect[0]:b_rect[0] +
      ↪  b_rect[2]]
5
6     return cropped_img, triangle_img, b_rect
```

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 afinicznej
- *warpAffine()* - funkcja dokonuje przekształcenia obrazu za pomocą macierzy rotacji będącej wynikiem wcześniejszej operacji

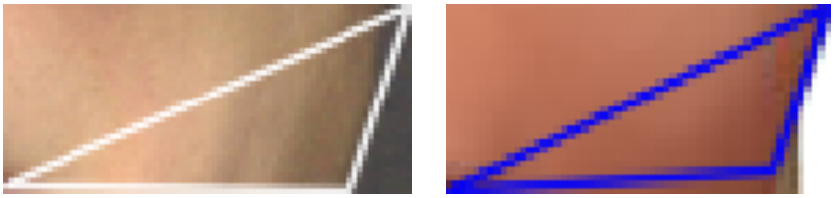
Funkcja pomocnicza transformująca trójkąty

```
1 def transform_triangle(avatar_triangle, avatar_img_cropped, src_triangle,
  ↪  src_img_cropped):
2     transform_matrix = cv2.getAffineTransform(np.float32(avatar_triangle),
      ↪  np.float32(src_triangle))
3
4     warped_triangle = cv2.warpAffine(avatar_img_cropped, transform_matrix,
5                                      (src_img_cropped.shape[1],
                                       ↪  src_img_cropped.shape[0]), None,
6                                      flags=cv2.INTER_LINEAR,
                                       ↪  borderMode=cv2.BORDER_REFLECT_101)
7
```

```
    mask = np.zeros(src_img_cropped.shape, dtype=np.uint8)
    mask = cv2.fillConvexPoly(mask, np.int32(src_triangle), (1.0, 1.0, 1.0), 16, 0)
    warped_triangle *= mask

    return warped_triangle
```

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.



(a) (b)

(c)

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

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 (5.6a).

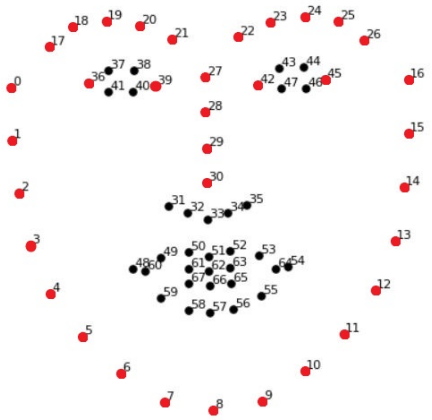
Funkcja dodająca transformowaną figurę do nowego obszaru twarzy

```
def add_triangle_to_new_face_area(new_face, warped_triangle, b_rect):
    (x, y, w, h) = b_rect
    new_face_gray = cv2.cvtColor(new_face[y: y + h, x: x + w], cv2.COLOR_BGR2GRAY)
    _, mask = cv2.threshold(new_face_gray, 0, 255, cv2.THRESH_BINARY_INV)
    warped_triangle = cv2.bitwise_and(warped_triangle, warped_triangle, mask=mask)

    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 których podstawie następuje ostateczna transformacja

– *PiecewiseAffineTransform()* - inicjalizacja pustego obiektu, który umożliwia transformację bazując na dwóch równoliczncych zestawach punktów. Jej działanie oparte jest na wykorzystaniu triangulacji Delone

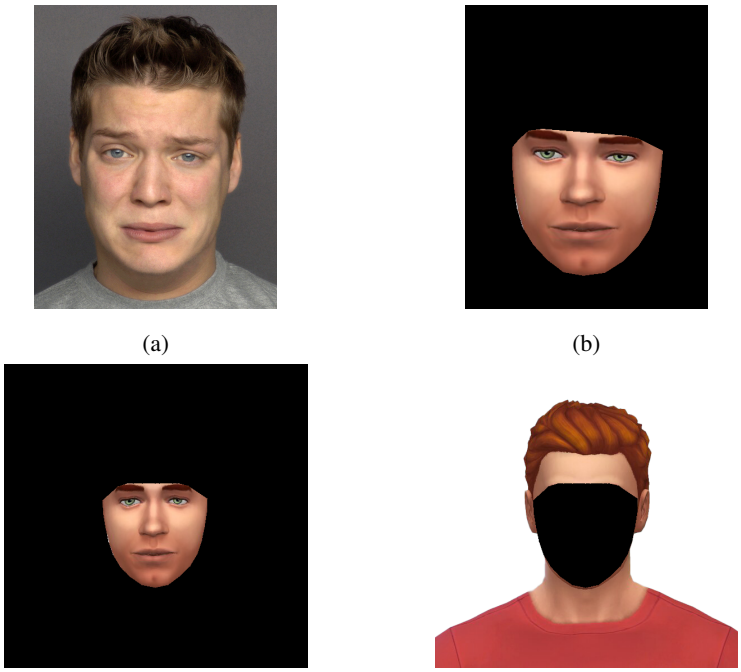
– *estimate()* - funkcja wywołana na otrzymanym wcześniej obiekcie przyjmuje dwa zestawy punktów, dla których ma nastapiąć przemapowanie. W przypadku działania owego programu wybrano elementy oznaczone na Rys. 5.5 kolorem czerwonym. Są to punkty charakterystyczne, które z reguły, w przypadku podstawowych 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, src_points):
2      transform = PiecewiseAffineTransform()
3      transform.estimate(choose_specific_points(avatar_points),
       ↪  choose_specific_points(src_points))
4
5      face = warp(new_avatar_face, transform, output_shape=avatar_img.shape, order=0,
       ↪  mode='wrap')
```

```
6       face = cv2.normalize(face, None, 0, 255, cv2.NORM_MINMAX, cv2.CV_8U)
7
8       face_gray = cv2.cvtColor(face, cv2.COLOR_BGR2GRAY)
9       _, mask = cv2.threshold(face_gray, 0, 255, cv2.THRESH_BINARY_INV)
10      new_avatar_img = cv2.bitwise_and(avatar_img, avatar_img, mask=mask)
11
12      new_avatar_img += face
13
14      return new_avatar_img
```

Po wykonaniu opisanych instrukcji należy dokonać normalizacji całego obrazu, aby przywrócić odpowiedni typ danych. Na Rys. 5.6c przedstawiono otrzymany 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.



(a) (b) (c) (d)

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

Aby podmienienić 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.

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

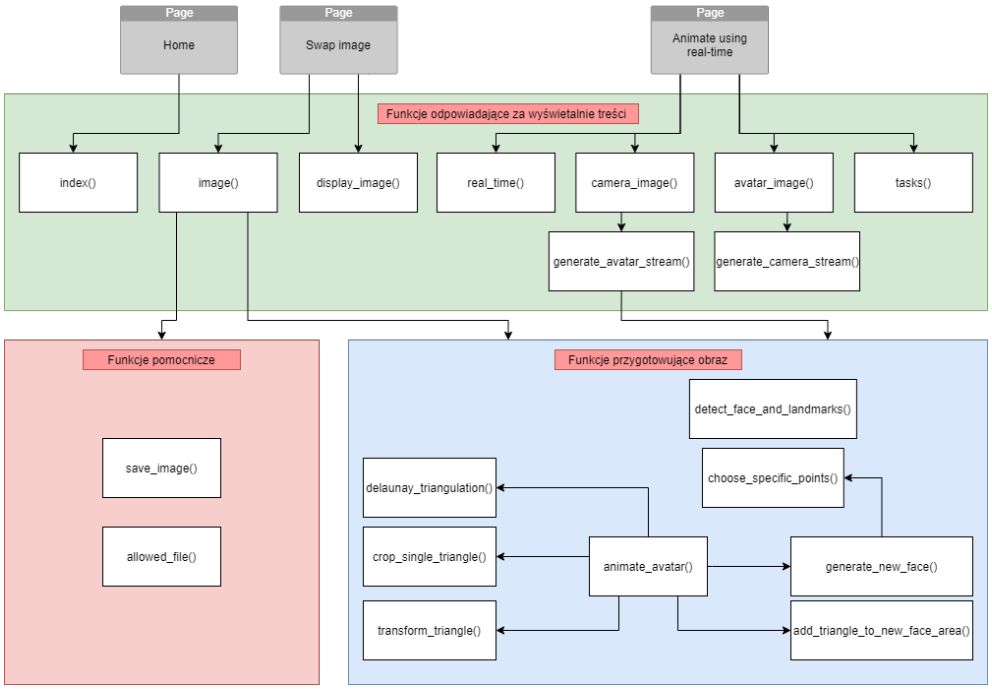
(a) (b)

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

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 przedstawia strukturę aplikacji webowej. Umieszczono na niej funkcje, które zostały opisane we wcześniejszych sekcjach oraz pozostałe moduły zaimplementowane na potrzeby stworzenia poprawnie działającej aplikacji.



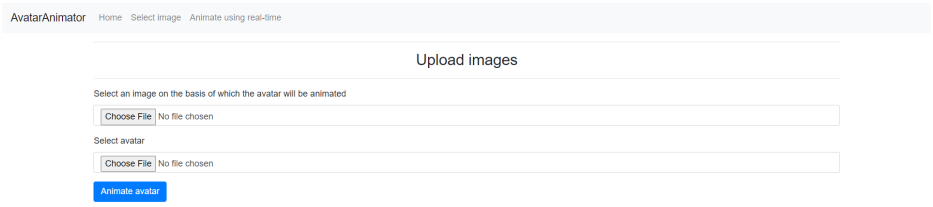
Rys. 5.8. Struktura aplikacji webowej.

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

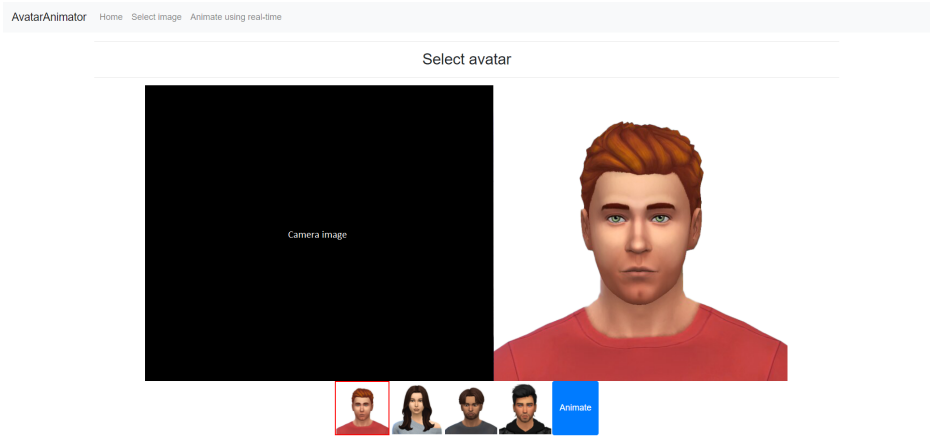
1. Krótka instrukcja zamieszczona na stronie głównej ma na celu zapoznanie użytkownika z możliwościami działania programu.

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 przyciśnięciu przycisku *Animate* następuje zastosowanie algorytmu na zdjęciu awatara dla aktualnego obrazu przechwyconego z kamery. W przypadku braku wykrycia twarzy, efekt nie zostanie osiągnięty.



(a) Wybór dwóch zdjęć wykorzystanych do animacji



(b) Wykorzystanie do animacji obrazu z kamery

Rys. 5.9. Interfejsy graficzne aplikacji webowej

6. Ewaluacja i rezultaty

6.1. Walidacja działania aplikacji

6.2. Ocena osiągniętych efektów

7. Podsumowanie

7.1. Wnioski

7.2. Weryfikacja początkowych założeń

7.3. Możliwy rozwój projektu

8. Opis używanych pojęć

Bibliografia

[1] Oskar Pacelt. *Sztuczna inteligencja a przyszłość ludzkości*. *https://botland.com.pl/blog/sztuczna-inteligencja-a-przyszlosc-ludzkosci/*. [Online, 07.01.2020].

[2] *Computer Vision - What it is and why it matters*. *https://www.sas.com/pl_pl/insights/analytics/computer-vision.html*. [Online, 09.03.2021].

[3] Corinne Bernstein. *Face Detection*. *https://searchenterpriseai.techtarget.com/definition/face-detection*. [Online, 02.2020].

[4] *Face Detection*. *https://en.wikipedia.org/wiki/Face_detection*. [Online, 05.08.2021].

[5] Ming-Hsuan Yang, David Kriegman i Narendra Ahuja. „Detecting Faces in Images: A Survey". W: *Institute of Electrical and Electronics Engineers* 24.1 (2002), s. 35, 36.

[6] Paul Viola i Michael J. Jones. „Rapid Object Detection using a Boosted Cascade of Simple Features". W: *Institute of Electrical and Electronics Engineers* 1 (2001), s. 1–6.

[7] Socret Lee. *Understanding Face Detection with the Viola-Jones Object Detection Framework*. *https://towardsdatascience.com/understanding-face-detection-with-the-viola-jones-object-detection-framework-c55cc2a9da14*. [Online, 22.05.2020].

[8] Navneet Dalal i Bill Triggs. „Histograms of Oriented Gradients for Human Detection". W: *Institute of Electrical and Electronics Engineers* (2005), s. 1–8.

[9] Satya Mallick. *Histogram of Oriented Gradients explained using OpenCV*. *https://learnopencv.com/histogram-of-oriented-gradients/*. [Online, 06.12.2016].

[10] *Facial landmark detector with dlib*. *https://www.pyimagesearch.com/2018/04/02/faster-facial-landmark-detector-with-dlib/*. [Online, 02.04.2018].

[11] Vahid Kazemi i Josephine Sullivan. „One millisecond face alignment with an ensemble of regression trees". W: *Institute of Electrical and Electronics Engineers* (2014), s. 1–8.

[12] *Triangulacja*. *https://pl.wikipedia.org/wiki/Triangulacja_(matematyka)*. [Online, 18.02.2020].

[13] *Delaunay triangulation*. *https://en.wikipedia.org/wiki/Delaunay_triangulation*. [Online, 15.11.2021].

[14] Sally Adee. *What Are Deepfakes and How Are They Created? Deepfake technologies: What they are, what they do, and how they're made. https://spectrum.ieee.org/what-is-deepfake*. [Online, 29.04.2020].

[15] *Aplikacja Wombo.ai. https://play.google.com/store/apps/details?id=com.womboai.wombo&hl=pl&gl=US*. [Online, 10.11.2021].

[16] *Aplikacja Anyface. https://play.google.com/store/apps/details?id=com.friendzy.Pereface*. [Online, 26.02.2021].

[17] *Aplikacja MotionPortrait. https://play.google.com/store/apps/details?id=com.motionportrait.MotionPortrait&hl=pl&gl=US*. [Online, 07.11.2019].

[18] *Python. https://pl.wikipedia.org/wiki/Python*. [Online, 03.11.2021].

[19] *Python Applications. https://www.javatpoint.com/python-applications*. [Online].

[20] *OpenCV Official Page. https://opencv.org/about/*. [Online].

[21] *Imutils Github Repository. https://github.com/PyImageSearch/imutils*. [Online, 15.01.2021].

[22] *dlib Official Page. http://dlib.net/*. [Online].

[23] *Scikit-image Official Page. https://scikit-image.org/*. [Online].

[24] *What is Flask Python. https://pythonbasics.org/what-is-flask-python/*. [Online].

[25] *PyCharm. https://pl.wikipedia.org/wiki/PyCharm*. [Online, 07.04.2021].