Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Informatyki

Praca dyplomowa inżynierska

na kierunku Informatyka
w specjalności Inżynieria systemów informatycznych

Opracowanie i implementacja algorytmu stabilizacji filmów wideo

Paweł Ostaszewski
Numer albumu 273888

promotor
dr inż. Grzegorz Nieradka

WARSZAWA 2020

Streszczenie

Opracowanie i implementacja algorytmu stabilizacji filmów wideo

Celem tej pracy było opracowanie, implementacja i przetestowanie działania nowego algorytmu stabilizacji filmów wideo, który dawałby dobre rezultaty i jednocześnie działałby w czasie rzeczywistym. Został zaproponowany nowy algorytm bazujący na znanych rozwiązaniach opisanych w literaturze. W czasie realizacji pracy zaproponowano kilka nowych elementów pozwalających na zmniejszenie czasu wykonywania algorytmu stabilizacji filmów wideo. Algorytm ten został zaimplementowany i przetestowany na przykładowych filmach z publicznie dostępnej bazy filmów. Wyniki działania zaproponowanego algorytmu zostały przedstawione w tej pracy i porównanie z wynikami działania innych algorytmów stabilizacji filmów wideo znanych z literatury.

Słowa kluczowe: Stabilizacja filmów wideo, wykrywanie punktów charakterystycznych, eliminacja niewłaściwych wartości, wygładzanie ruchu klatek filmu, wygładzanie wartości parametrów, przekształcenia afiniczne

Abstract

Design and implementation of video stabilization algorithm

The main goal of this thesis was to develop, implement and test a new video stabilization algorithm, which would give correct results and would be able to work in real-time applications. Some kind of novelty algorithm was proposed, which is based on solutions already existing in the literature. This algorithm also features some new solutions, which are capable of reducing time consumption of the video stabilization process. The algorithm has been implemented and tested on sample videos from publicly available benchmark base. Performance results of proposed algorithm were presented in this thesis and compared with performance results of other video stabilization algorithms existent in literature.

Keywords: video stabilization, keypoints detection, outliers elimination, path smoothing, parameter smoothing, affine transforms

załącznik nr 3 do zarządzenia
nr 28 /2016 Rektora PW

Politechnika Warszawska

Warszawa, 03.02.2020
miejscowość i data

Paweł Ostaszewski
imię i nazwisko studenta
273888
numer albumu
Informatyka
kierunek studiów

OŚWIADCZENIE

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

Jednocześnie oświadczam, że:

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

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

.....
czytelny podpis studenta

Spis treści

1. Wstęp

W dzisiejszych czasach powszechne jest wykorzystanie urządzeń mobilnych, takich jak telefony i tablety. Producenci urządzeń mobilnych starają się wytwarzać urządzenia, które mogą spełniać wiele różnych funkcji. W związku z tym większość urządzeń mobilnych jest wyposażonych w układy cyfrowe o sporej mocy obliczeniowej, w kamery cyfrowe, karty sieciowe, akcelerometry oraz inne podzespoły. W ostatnich latach widzimy jak tradycyjne kamery i aparaty cyfrowe są zastępowane przez urządzenia mobilne, które również potrafią wykonywać zdjęcia oraz nagrywać filmy wideo.

W ostatnich latach coraz bardziej popularne jest wykorzystanie dronów UAV (*ang. Unmanned Aerial Vehicles*), zdalnie sterowanych urządzeń wykonanych z lekkiego materiału i wyposażonych w śmigła dzięki czemu mogą latać. Drony są wyposażone w kamery i wysyłają obraz zaobserwowany z kamery do osoby sterującej dronem, dzięki czemu jest możliwe oglądanie filmu wideo nagranego przez drona z lotu ptaka. W obu powyżej wymienionych przypadkach mamy do czynienia z licznymi wibracjami i drganiami kamery utrudniającymi oglądanie filmu. W minionych dekadach zostało opracowanych wiele algorytmów stabilizacji filmów wideo. Celem tych algorytmów jest poprawa jakości filmów w taki sposób, aby nie było w nich żadnych poruszeń ani zniekształceń, co pozwala na bardziej komfortowe oglądanie. Wynik działania przykładowego algorytmu stabilizacji jest przedstawiony na rysunku 1.1.

Algorytmy stabilizacji filmów można podzielić na dwie grupy. Z których pierwsze wykonują stabilizację filmu wideo w czasie rzeczywistym, w trakcie nagrywania filmu a drugie wykonują algorytm stabilizacji filmu po całkowitym nagraniu filmu mając do dyspozycji więcej czasu oraz większą moc obliczeniową. Aby film był płynny to kamera nagrywająca dany film powinna generować klatki filmu z częstotliwością $f = 30\,Hz$. Ograniczenie czasowe na wykonywanie pełnego obiegu algorytmu jest odwrotnie proporcjonalne do ilości klatek na sekundę wykonywanych przez kamerę. Stąd wynika, że ograniczenie czasowe na ustabilizowanie jednej ramki filmu wynosi $T = \frac{1}{f} = \frac{1}{30} = 33.(333)$ ms. Algorytmy działające na bieżąco w czasie rzeczywistym muszą więc mieć niską złożoność obliczeniową aby zmieścić się w restrykcyjnych ramach czasowych. Algorytmy z tej grupy powinny mieć też niską złożoność pamięciową ze względu na to, że są one często wykorzystywane w układach wbudowanych w kamerach dronów lub wewnątrz urządzeń mobilnych. Jednakże rosnąca moc obliczeniowa urządzeń mobilnych pozwala na zastosowanie coraz bardziej wyrafinowanych algorytmów cyfrowej stabilizacji filmów wideo działających w czasie rzeczywistym w trakcie nagrywania filmu. Nowoczesne drony również zaczynają być wyposażone w wyspecjalizowane układy cyfrowe implementujące algorytmy stabilizacji filmu wideo w czasie rzeczywistym.

W ramach niniejszej pracy podjęto problem opracowania wydajnego algorytmu stabilizacji filmów wideo, który działałby sprawnie w systemach czasu rzeczywistego.

Rysunek 1.1. Przykład działania algorytmu stabilizacji. Na obu obrazach są pokazane średnie obrazy z 50 klatek filmu. Po lewej stronie średni obraz z filmu nie poddany żadnym modyfikacjom. Po prawej stronie średni obraz z filmu poddanego procesowi stabilizacji. Źródło zdjęć: [21]

2. Zagadnienie stabilizacji filmów w świetle literatury

W tym rozdziale zostaną omówione dotychczas opisane rozwiązania wykorzystywane w procesie stabilizacji filmów wideo. Zostaną przedstawione wady i zalety poszczególnych metod oraz zostaną one porównane ze sobą i sklasyfikowane w ogólne kategorie metod stabilizacji filmów wideo.

2.1. Przegląd istniejących rozwiązań

W pracach [5], [6], [7], [8], [9], [10] opisano i zastosowano standardowy algorytm stabilizacji filmów, który jest najbardziej powszechnie wykorzystywany w dzisiejszych czasach. W ogólnym zarysie algorytm ten działa w taki sposób, że przetwarza po kolei każdą kolejną parę ramek filmu. Najpierw poszukuje punktów charakterystycznych w sąsiednich ramkach, znajduje ich dopasowania pomiędzy sąsiednimi ramkami i eliminuje błędne dopasowania. Następnie znajduje optymalną macierz przekształcenia w celu korekcji kolejnej ramki filmu tak aby upodobnić ją do poprzedniej. Po wykonaniu przekształcenia korekcyjnego na kolejnych ramkach filmu dopasowane punkty charakterystyczne z sąsiednich ramek znajdują się o wiele bliżej siebie w ustabilizowanym ciągu ramek filmu niż w oryginalnym ciągu ramek.

W pracach [1] i [14] napisanych przez tych samych autorów zaproponowano algorytm podobny do tego opisanego w pracach [5], [6], [7] [8], [9], [10], ale z dodatkową modyfikacją polegającą na wprowadzeniu siatki przepływu (*ang. mesh flow*). W tym algorytmie każda ramka filmu jest dzielona na prostokątną siatkę i każdy fragment siatki jest przekształcany osobno standardowym algorytmem. Dzięki temu siatka przepływu potrafi płynnie reagując na zniekształcenia pojawiające się w filmie poprzez skalowanie i zmianę rozmiaru dowolnych obszarów ramek filmu. Dla każdego fragmentu siatki jest wyznaczana osobna ścieżka i są wyznaczane osobne przekształcenia korekcyjne, co powoduje że algorytm ten ma wysoką jakość działania ale też jest dość powolny i zasobochłonny przez co ciężko jest go zastosować w systemach wbudowanych czasu rzeczywistego.

W pracy [8] zmodyfikowano standardowy algorytm poprzez zastosowanie znajdywania optymalnego przepływu między ramkami filmu, tzn. zamiast znajdywania punktów charakterystycznych wyznacza się optymalny wektor przesunięcia pomiędzy sąsiednimi ramkami w celu zminimalizowania różnic między sąsiednimi ramkami. W tym algorytmie zastosowano też piramidy laplasowskie (*ang. laplacian pyramid*) polegające na tym, że ramka filmu jest powielana na kilka pięter piramidy, a na każdym piętrze dokonywane jest podpróbkowywanie (*ang. down-sampling*) obrazu z piętra powyżej. W ten sposób tworzy się piramida powielonych obrazów posortowana w kolejności od najbardziej szczegółowego do najbardziej zgrubnego obrazu. Porównywanie pikseli z różnych pięter piramidy i pomiędzy sąsiednimi ramkami filmu ułatwia znajdywanie optymalnego przepływu pomiędzy ramkami filmu.

W pracy [2] opisano algorytm stabilizacji filmów działający w całkowicie odmienny sposób niż standardowe algorytmy opisane powyżej. Do stabilizacji filmów zastosowano głęboką sieć neuronową StabNet, która uczy się procesu stabilizacji filmów na różnorodnym zestawie trenującym składającym się z oryginalnych, chaotycznych filmów oraz ich odpowiedników ustabilizowanych standardowymi algorytmami. Gdy taka sieć neuronowa nauczy się już procesu stabilizacji filmów to jest ona w stanie stabilizować dowolne filmy wideo z dużą częstotliwością sięgającą 93,3 klatek filmu na sekundę.

2.2. Porównanie i klasyfikacja znanych algorytmów stabilizacji filmów

Metody stabilizacji filmów wideo można podzielić na dwie zasadnicze grupy: standardowe metody stabilizacji bazujące na znajdywaniu punktów charakterystycznych, oraz metody stabilizacji wykorzystujące nowatorskie rozwiązania takie jak sztuczne sieci neuronowe, estymacja 3D, siatki przepływu (*ang. mesh flow*).

Przykłady standardowych metod z pierwszej grupy są przedstawione w pracach [5], [6], [7], [8], [9], [10]. Metody z drugiej grupy opisane zostały w pracach [2], [3], [4], [12], [13]. W pracach [1], [14] zostały opisane algorytmy, które znajdują się pomiędzy algorytmami obu grup. Algorytm zaproponowany przeze mnie zalicza się do pierwszej grupy, aczkolwiek są zastosowane w nim nowe rozwiązania, których nie ma w pracach innych autorów, m. in. własna metoda usuwania niewłaściwych dopasowań (rozdział 4.5) oraz własna metoda wygładzania parametrów przekształceń afinicznych (rozdział 4.7).

Zasadnicza różnica między standardowymi algorytmami pierwszej grupy opisanych przez różnych autorów polega na wprowadzaniu modyfikacji w poszczególnych etapach ogólnej metody. Modyfikacje te polegają m. in. na stosowaniu różnych algorytmów znajdywania punktów charakterystycznych takich jak Susan, Harris [9], SIFT [5], SURF, FAST [6], BRIEF, ORB, FREAK [7], albo na stosowaniu różnych metod selekcji najlepiej dopasowanych par punktów charakterystycznych, m. in. RANSAC [5], głosowanie komparatywne [8].

Różnica między algorytmami poszczególnych autorów polega też na metodach znajdywania dopasowań między punktami charakterystycznymi sąsiednich ramek filmu. Na przykład w pracy [6] wykorzystana jest metoda siłowa (*ang. brute force*) z wykorzystaniem miary SSD (*ang. Sum of Squared Differences*) do znajdywania dopasowań. W pracach [5], [7] opisane są algorytmy polegające na obliczaniu unikalnych punktów charakterystycznych ukierunkowanych wektorami określonych przez algorytmy SIFT, FAST dzięki czemu można łatwo rozróżnić właściwe punkty charakterystyczne z sąsiednich ramek. Algorytmy opisane w pracach [1] i [8] dzielą obszar ramki filmu na wiele prostokątnych obszarów i szukają dopasowań tych obszarów w sąsiednich ramkach zamiast dopasowań poszczególnych punktów charakterystycznych.

Inną różnicą między algorytmami pierwszej grupy jest metoda usuwania niewłaściwych dopasowań, np. w pracy [5] jest wykorzystana probabilistyczna metoda RANSAC. Z kolei w pracy [8] mamy do czynienia z selektywnym usuwaniem obszarów jednolitych (*ang. region uniformity*) oraz obszarów z powtarzającym się wzorcem (*ang. block aperture*). W [8] mamy też do czynienia z usuwaniem niedopasowań poprzez głosowanie komparatywne oraz z usuwaniem wektorów przesunięcia pomiędzy sąsiednimi ramkami, które nadmiernie odbiegają od normy, wektora-mediany. Ja w niniejszej pracy proponuję własną metodę usuwania niewłaściwych dopasowań omówioną w rozdziale 4.5.

Algorytmy drugiej grupy stosują innowacyjne rozwiązania takie jak sztuczne sieci neuronowe [2], stabilizacja 3D z wykorzystaniem rozbudowanych przekształceń homograficznych [3], [12], [13], stabilizacja z użyciem siatki przepływu, która potrafi odpowiednio skalować dowolne obszary ramek filmu [1], [14], stabilizacja z wykorzystaniem usuwania zjawiska powtarzającej się migawki (*ang. rolling shutter removal*) [11].

Ważnym kryterium podziału algorytmów stabilizacji filmów wideo może być czas ich wykonywania. Pierwszą grupą według tego kryterium są algorytmy stabilizacji działające w czasie rzeczywistym w trakcie nagrywania filmu. Drugą grupą według tego kryterium są algorytmy stabilizacji o długim czasie wykonywania, które mogą być używane tylko na gotowym całym filmie tuż po jego nakręceniu. Niektóre algorytmy mają też dodatkowe ograniczenia, m. in. muszą mieć do

dyspozycji całą historię dotychczas nakręconych klatek filmu albo muszą mieć do dyspozycji cały oryginalny film. Na przykład sieć neuronowa z [2] musi mieć na początku w całości nakręcone filmy z zestawów trenujących.

Do algorytmów stabilizacji filmów w czasie rzeczywistym można zaliczyć sieć neuronową [2], która przetwarza 93,3 klatek filmu na sekundę, algorytm opisany w pracy [6] o częstotliwości przetwarzania sięgającej 30 klatek filmu na sekundę, prosty algorytm opisany w pracy [8] o częstotliwości przetwarzania 25 klatek filmu na sekundę, algorytmy stosujące siatkę przepływu (*ang. mesh flow*) – około 22 klatki filmu na sekundę [14]. Do algorytmów stabilizacji filmów używanych tylko po nakręceniu całego filmu można zaliczyć wyrafinowane i rozbudowane algorytmy z prac [12], [13], algorytm z pracy [1] działający z częstotliwością 3,5 klatek filmu na sekundę oraz algorytm utworzony przez firmę Adobe – Adobe Premiere CS6, wspomniany w pracy [1], o częstotliwości przetwarzania 4.2 klatek filmu na sekundę.

Dodatkowym kryterium podziału metod algorytmów stabilizacji filmów może też być obszar zastosowania. Np. prace [5], [6] skupiają się na zastosowaniu swojego algorytmu w kamerach latających dronów szpiegowskich. W pracy [8] utworzono algorytm dla robotów – łazików, takich jak łaziki eksplorujące planetę Mars. W wyżej wymienionych zastosowaniach algorytmy muszą spełnić restrykcyjne wymogi systemów wbudowanych czasu rzeczywistego. W pracach [1][11][12][13] skupiono się na stabilizacji filmów na zwykłym komputerze PC tuż po jego nakręceniu i nie są istotne mała złożoność czasowa i pamięciowa dla tych algorytmów.

3. Przegląd istniejących technologii

Zanim przystąpiłem do opracowywania algorytmu stabilizacji filmów wideo to dokonałem przeglądu różnych technologii pod kątem ich przydatności w zagadnieniu stabilizacji filmów wideo.

Celem było opracowanie algorytmu mogącego działać w urządzeniach czasu rzeczywistego a więc musiałem wybrać technologię, która działa dość szybko.

Dobrą biblioteką do przetwarzania obrazów jest OpenCV (*ang. Open Source Computer Vision*). Jest to wolno dostępna i wieloplatformowa biblioteka mająca zastosowania w grafice komputerowej i w uczeniu maszynowym. Zawiera ponad 2500 zaimplementowanych algorytmów, w tym przetwarzanie scen 2D, 3D w czasie rzeczywistym, śledzenie obiektów i punktów charakterystycznych w obrazach i filmach, fragmentacja obrazów, tworzenie modeli 3D na podstawie obrazów. Ma wsparcie dla języków programowania C/C++, Python, Java, MATLAB/OCTAVE.

Dla mnie najważniejszymi modułami biblioteki OpenCV były śledzenie punktów charakterystycznych w obrazach, moduły do operacji na macierzach i algorytmy realizujące transformacje Fouriera.

Natomiast jeśli chodzi o języki programowania to język C# może być dobrym kandydatem bo jest językiem kompilowanym a nie interpretowanym, w związku z czym działa dość szybko. Zawiera też liczne biblioteki do przetwarzania obrazów, ale niestety nie są one wolno dostępne i są komercyjne tak samo jak środowisko MS Visual Studio. Zrezygnowałem więc z tej opcji bo chciałem wykorzystać jakąś darmową biblioteką wolno dostępną.

Język Java jest dość zasobochłonny i powolny gdyż na urządzeniu wbudowanym trzeba zainstalować wirtualną maszynę javy, która na bieżąco interpretuje każdą instrukcję. Ponadto w języku Java wykonywane są różne procesy w tle, niepotrzebne w naszym przypadku, takie jak oczyszczanie pamięci (*ang. garbage collector*). A więc język Java raczej nie jest dobrym kandydatem w tym przypadku. Język Python jest trochę lepszym kandydatem, jest wolno dostępny i są dla niego zbudowane wszystkie moduły biblioteki OpenCV, ale Python też jest językiem interpretowanym zamiast kompilowanym i używa oczyszczania pamięci co może spowodować spowolnienie działania.

Dobrym kandydatem jest język C/C++ gdyż jest językiem kompilowanym, nie interpretowanym i domyślnie nie używa oczyszczania pamięci. Język C/C++ jest z tego powodu dość szybki i jest powszechnie wykorzystywany w systemach wbudowanych czasu rzeczywistego. Dla tego języka stworzona jest też dobra wolnodostępna biblioteka Qt do wygodnego tworzenia interfejsów graficznych GUI (*ang. Graphical User Interface*).

3.1. Zastosowane technologie

Z wyżej wymienionych względów zdecydowałem się na wykonanie i implementację mojego algorytmu stabilizacji filmów wideo w języku C++11. Zastosowałem też biblioteki tego języka: biblioteka OpenCV, dostępna na licencji open source BSD oraz biblioteka Qt, dostępna na licencji open source LGPLv3. Aczkolwiek biblioteka Qt jest też dostępna w wersji komercyjnej. Do stworzenia aplikacji z interfejsem graficznym GUI (*ang. Graphical User Interface*) został wykorzystany zintegrowany interfejs programistyczny Qt IDE (*ang. Integrated Development Environment*), klasy biblioteki Qt oraz wygodne narzędzie do tworzenia interfejsów graficznych GUI – Qt Designer. Biblioteka OpenCV została wykorzystana do przetwarzania klatek filmów, ponieważ biblioteka ta zawiera mnóstwo gotowych funkcji do przetwarzania scen 2D oraz 3D i do operacji na obrazach.

3.2. Wykorzystana funkcjonalność biblioteki OpenCV

W celu przedstawienia możliwości biblioteki OpenCV przedstawione zostaną niektóre wykorzystane z niej funkcje z ich sygnaturami i krótkim opisem efektu ich działania.

```
void cvtColor(InputArray src, OutputArray dst, int code, int
dstCn=0); // code = CV_BGR2GRAY
```

Funkcja ta przyjmuje jako argument obraz wejściowy `src` i zwraca poprzez argument przetworzony obraz `dst` o innej przestrzeni barw określonej parametrem `code`.

```
void goodFeaturesToTrack(InputArray image, OutputArray corners,
int maxCorners, double qualityLevel, double minDistance,
InputArray mask=noArray(), int blockSize=3,
bool useHarrisDetector=false, doublek=0.04);
```

Funkcja ta przyjmuje jako argument obraz wejściowy `image` i znajduje w nim `maxCorners` najbardziej wyrazistych punktów charakterystycznych. Po wykonaniu tej funkcji znalezione punkty charakterystyczne są przechowywane w strukturze `corners`.

```
cv::Mat estimateRigidTransform(InputArray src, InputArray dst,
bool fullAffine);
```

Funkcja ta przyjmuje dwa zbiory `src` i `dst` dopasowanych punktów charakterystycznych z sąsiednich ramek i zwraca macierz `cv::Mat` przekształcającą obraz zawierający punkty charakterystyczne `src` w obraz zawierający punkty charakterystyczne `dst`.

```
void warpAffine(InputArray src, OutputArray dst, InputArray M,
Size dsize, int flags = INTER_LINEAR, int borderMode =
BORDER_CONSTANT, const Scalar& borderValue=Scalar());
```

W funkcji tej struktura `src` oznacza wejściowy obraz, `dst` oznacza wyjściową strukturę w której ma być wygenerowany przetworzony obraz. Struktura **`M`** to macierz przekształcenia, która zostanie zastosowana na każdym pikselu struktury `src` w celu utworzenia struktury `dst`.

```
MatExpr cv::Mat::inv(int method = DECOMP_LU) const;
```

`Inv` to funkcja składowa klasy `cv::Mat`. Funkcja ta wykonuje operację odwracania macierzy `cv::Mat`.

```
void dft(InputArray src, OutputArray dst, int flags=0,
int nonzeroRows=0)
```

Funkcja ta wykonuje dyskretną transformację Fouriera (DFT). Funkcja przyjmuje tablicę wejściową `src` zawierającą reprezentację sygnału w dziedzinie czasu oraz zwraca tablicę wyjściową `dst`, która zawiera reprezentację sygnału w dziedzinie częstotliwości.

4. Opis działania algorytmu stabilizacji filmu wideo

Dla każdej sąsiedniej pary ramek filmu F_i, F_{i+1}, $i = 0..n-2$ wykonujemy 8 lub opcjonalnie 9 odrębnych etapów algorytmu stabilizacji oraz pomiary jakości działania algorytmu. Metody oceny jakości działania algorytmu są dokładnie opisane w rozdziale 5.

W ogólnym zarysie wykonywane etapy są następujące. Etap pierwszy to przetworzenie kolejnej ramki filmu F_{i+1} z kolorowego obrazu na obraz w skali szarości dla uproszczenia następnych operacji. W drugim etapie znajdywane są punkty charakterystyczne w obu sąsiednich ramkach za pomocą algorytmu Harris-Shi-Tomasi. W trzecim etapie znajdywane są odpowiadające sobie pary punktów charakterystycznych z obu sąsiednich ramek za pomocą metody siłowej (*ang. brute force*) i miary SSD (*ang. Sum of Squared Differences*). Czwarty etap to przekształcenie odpowiadających sobie par punktów charakterystycznych z obu sąsiednich ramek na wektory przesunięcia między dwoma sąsiednimi ramkami, LMV (*ang. Local Movement Vectors*). W piątym etapie stosujemy algorytm eliminacji niewłaściwych wektorów LMV, gdyż w trzecim etapie mogły powstać niewłaściwe powiązania między parami punktów charakterystycznych. Szósty etap to wyznaczenie macierzy przekształcenia afinicznego $\mathbf{A}_i$ przekształcającego ramkę F_i w ramkę F_{i+1} na podstawie tych wektorów przesunięcia LMV co zostaną po eliminacji. Macierz tą następnie odwracamy w celu uzyskania macierzy odwrotnej przekształcenia afinicznego $\mathbf{A}_i^{-1}$, którą później można wykorzystać do poprawienia nowej ramki F_{i+1} upodobniając ją do poprzedniej ramki F_i. W siódmym etapie obliczamy parametry przekształcenia afinicznego $\mathbf{A}_i^{-1}$ i poddajemy je procesowi wygładzania parametrów. Dzięki temu ruch klatek filmu odbywa się bardziej płynnie, tzn. usuwane są przypadkowe przesunięcia o wysokich częstotliwościach przy jednoczesnym zachowaniu ruchu kamery o niskiej częstotliwości. Ósmy etap polega na zastąpieniu poprzednich parametrów macierzy przekształcenia $\mathbf{A}_i^{-1}$ nowymi, wygładzonymi parametrami uzyskanymi w poprzednim etapie. Następnie macierz wygładzonego odwrotnego przekształcenia afinicznego $\mathbf{A}_i^{-1'}$ jest stosowana na ramce F_{i+1} w celu upodobnienia jej do poprzedniej ramki F_i. Ostatni, opcjonalny etap 9 polega na usunięciu ze wszystkich ramek filmu czarnych obszarów na krawędziach każdej ramki pozostałych po wykonaniu poprzednich 8 etapów algorytmu. Po wykonaniu dziewiątego, opcjonalnego etapu znikają czarne obszary na krawędziach z całego filmu takie jak te widoczne na rys. 1.1 oraz rys. 5.1.1.

Dzięki temu po wykonaniu 8 lub opcjonalnie 9 wymienionych etapów algorytmu stabilizacji, klatki filmu są bardziej upodobnione do siebie i obraz widoczny w całym ciągu klatek filmu jest o wiele bardziej płynny i spójny jak na rys. 1.1.

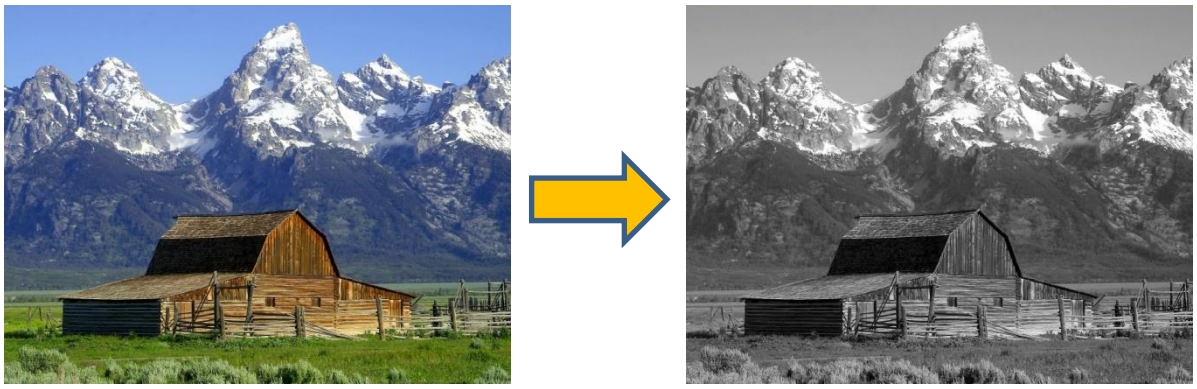
4.1. Etap pierwszy. Redukcja skali kolorów

W pierwszym etapie stabilizacji algorytmu dokonywana jest konwersja wczytanej ramki filmu z kolorowego formatu RGB do formatu w skali szarości, rys. 4.1.1. Dzięki temu upraszczają się następne operacje gdyż nie trzeba analizować obrazu w trzech kanałach czerwonym, zielonym i niebieskim tylko w jednym kanale odcieni szarości. Każdy piksel ma po konwersji tylko jedną wartość określającą jasność piksela (*ang. Illumination*), oznaczaną jako $I(x, y)$. Dzięki konwersji zaoszczędzamy też miejsce w pamięci gdyż w kolorowych obrazach w formacie RGB każdy piksel jest zazwyczaj przechowywany w trzech bajtach. Jeden bajt na każdy kolor. Natomiast w obrazach w skali szarości każdy piksel jest przechowywany w jednym bajcie określającym jasność piksela. Tzn. każdy piksel może przyjmować wartości z zakresu od 0 do 255. Stąd uzyskujemy trzykrotną oszczędność pamięci, jeden bajt na piksel w obrazach w skali szarości, w stosunku do trzech bajtów w

kolorowych obrazach. W aplikacji wykorzystywana jest następująca funkcja z biblioteki OpenCV służąca do konwersji między różnymi przestrzeniami barw wszystkich pikseli zawartych w obrazach.

```
void cvtColor(InputArray src, OutputArray dst, int code,
int dstCn=0) // code = CV_BGR2GRAY
```

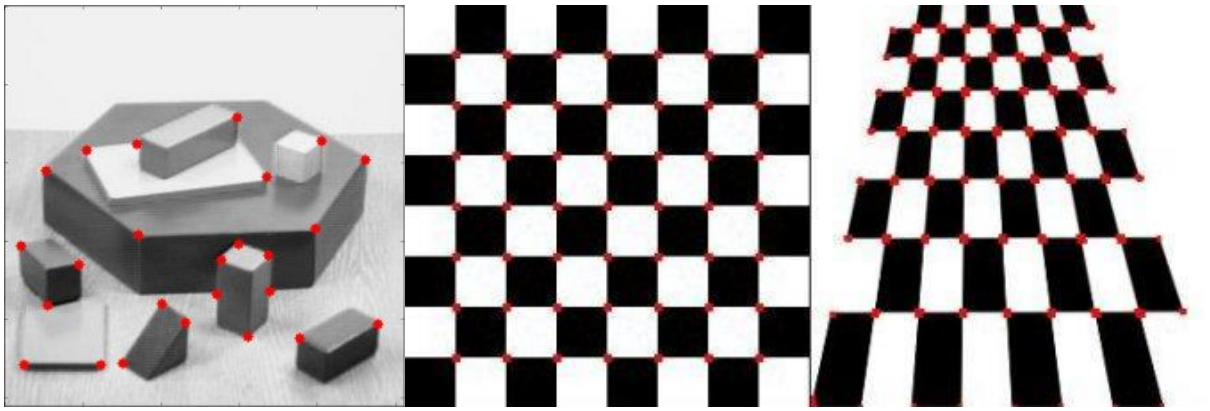
Funkcja ta przyjmuje jako argument obraz wejściowy `src` i zwraca poprzez argument przetworzony obraz `dst` o innej przestrzeni barw określonej parametrem `code`.



Rysunek 4.1.1. Przykład konwersji kolorowego obrazu do obrazu w skali szarości w pierwszym etapie algorytmu. Źródło zdjęcia: https://pl.wikipedia.org

4.2. Etap drugi. Wyszukiwanie punktów charakterystycznych

W tym etapie wykonywane jest znajdywanie punktów charakterystycznych w obu sąsiednich ramkach filmu F_i oraz F_{i+1}. Na rysunku 4.2.1 przedstawiono przykłady znalezienia unikatowych punktów wyróżniających się od pozostałych punktów w obrazach. Ważną cechą punktów charakterystycznych wykorzystywanych w algorytmach stabilizacji filmów jest to, że punkt charakterystyczny da się łatwo wykryć na obrazie nawet jeśli ten obraz zostanie poddany licznym przekształceniom afinicznym.



a b c

Rysunek 4.2.1. Przykłady zastosowania algorytmu Harris-Shi-Tomasi do znajdywania punktów charakterystycznych. (a) 25 najbardziej charakterystycznych punktów obrazu z figurami geometrycznymi (b) Wszystkie punkty charakterystyczne szachownicy. (c) Te same punkty charakterystyczne szachownicy poddanej przekształceniu afinicznemu. Źródło rysunku: https://docs.opencv.org

W wielu algorytmach stabilizacji filmów do znajdywania punktów charakterystycznych wykorzystywany jest algorytm opracowany przez Chris'a Harris'a oraz Mike'a Stephens'a przedstawiony w artykule [9] z 1988 roku. Algorytm ten został ulepszony przez Jianbo Shi oraz Carlo Tomasi i opisany w artykule [10] z 1994 roku. W literaturze pokrewnej korzysta się również z innych algorytmów bazujących na algorytmie Harris'a-Stephens'a, m.in. SIFT, SURF, FAST, BRIEF, ORB. Główną ideą algorytmu opracowanego przez Harris'a i Stephens'a jest znajdywanie takich pikseli które różnią się znacznie jasnością od sąsiednich pikseli. Ściślej według [9] można to zrobić poprzez obliczenie różnic w jasności pikseli przemieszczając się w każdym kierunku obrazu za pomocą wektora przesunięcia $\mathbf{d} = (u, v)$ startując od pozycji badanego piksela $p(x_0, y_0)$. Matematycznie można to zapisać w taki sposób

$$E = \sum_{(u,v)\in m\times n} \sum_{x=0}^{m} \sum_{y=0}^{n} w(x,y)[I(x_0+u, y_0+v) - I(x_0, y_0)]^2$$

W tej sumie m i n oznaczają szerokość i wysokość obrazu a w oznacza funkcję wagi, czyli okno, w którym obliczane są przesunięcia. Funkcja wagi w może być np. obszarem prostokątnym, które przyjmuje wartość 1 wewnątrz obszaru oraz wartość 0 na zewnątrz obszaru. Funkcja wagi może być obliczona na podstawie funkcji gaussowskiej, tzn. taką w którym jest płynne przejście od 0 dla pikseli odległych od $p(x_0, y_0)$ do 1 dla pikseli bliskich $p(x_0, y_0)$, podobnie jak w krzywej Gaussa.

Celem algorytmu jest znajdywanie punktów dla których wartość sumy E jest największa. Takie punkty są szukanymi punktami charakterystycznymi.

Po przekształceniach matematycznych zastosowanych w [9] można sprowadzić sumę E do następującej postaci:

$$E \approx [u, v]\mathbf{M}\begin{bmatrix} u \\ v \end{bmatrix}, \qquad \text{gdzie } \mathbf{M} = \sum_{x=0}^{m}\sum_{y=0}^{n} w(x,y)\begin{bmatrix} I_x I_x & I_x I_y \\ I_y I_x & I_y I_y \end{bmatrix}$$

$I_x = \frac{\partial I}{\partial x}$ oraz $I_y = \frac{\partial I}{\partial y}$ to pochodne cząstkowe obrazu, obliczone na podstawie jasności pikseli wzdłuż osi X oraz Y w obrazie. Następnie obliczamy wielomian charakterystyczny W macierzy **M**, który zgodnie z [18] wygląda następująco

$$W = \det[\mathbf{M} - \lambda \cdot \mathbf{I}]$$

gdzie **I** jest macierzą jednostkową o postaci

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

Po przekształceniach matematycznych dochodzimy do następującej postaci wielomianu W

$$W = \det[\mathbf{M}] - k(\mathrm{trace}[\mathbf{M}])^2$$

Ponadto z własności wartości własnych i wyznaczników wiemy też, że

$$\det[\mathbf{M}] = \lambda_1\lambda_2, \qquad \mathrm{trace}[\mathbf{M}] = \lambda_1 + \lambda_2$$

gdzie λ_1 oraz λ_2 to wartości własne macierzy **M**. Stąd można uzyskać równoważną postać wielomianu W

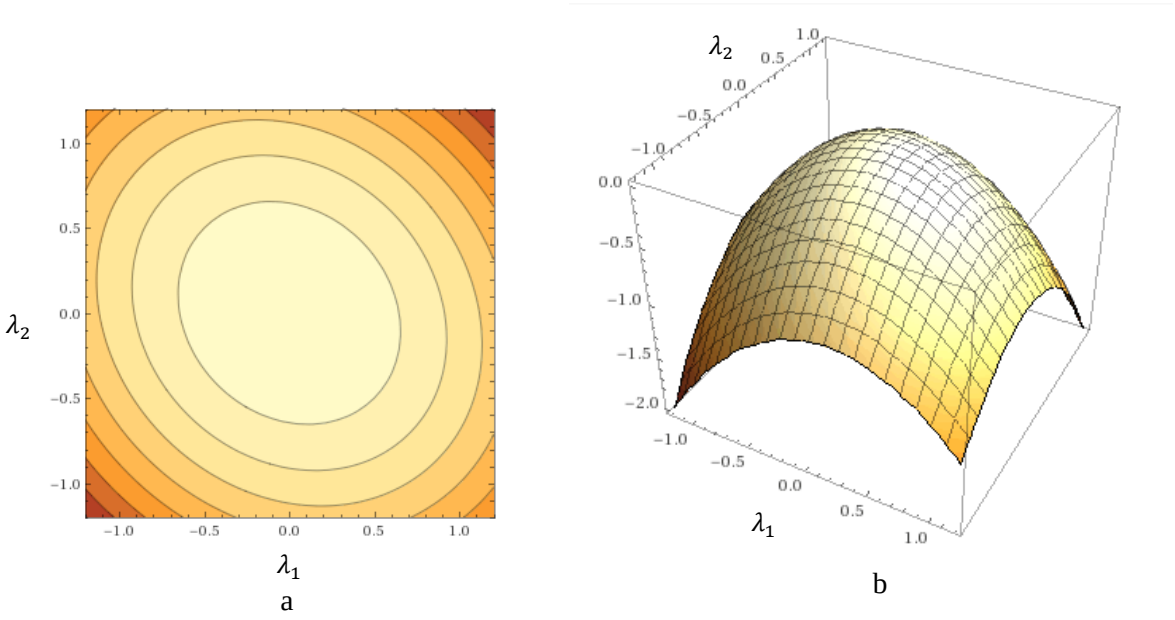
$$W = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$$

Po przekształceniach równanie to można zapisać w postaci

$$W = -k({\lambda_1}^2 + {\lambda_2}^2) + (1 - 2\,k)\lambda_1\lambda_2$$

Dla $k > \frac{1}{4}$ w powyższym równaniu dominuje czynnik ${\lambda_1}^2 + {\lambda_2}^2$. Stąd dla dużych wartości k mamy

$W \approx {\lambda_1}^2 + {\lambda_2}^2$. Równanie to przedstawia eliptyczną paraboloidę, w której pionowe przekroje to parabole, a poprzeczne przekroje to elipsy, rys. 4.2.2.

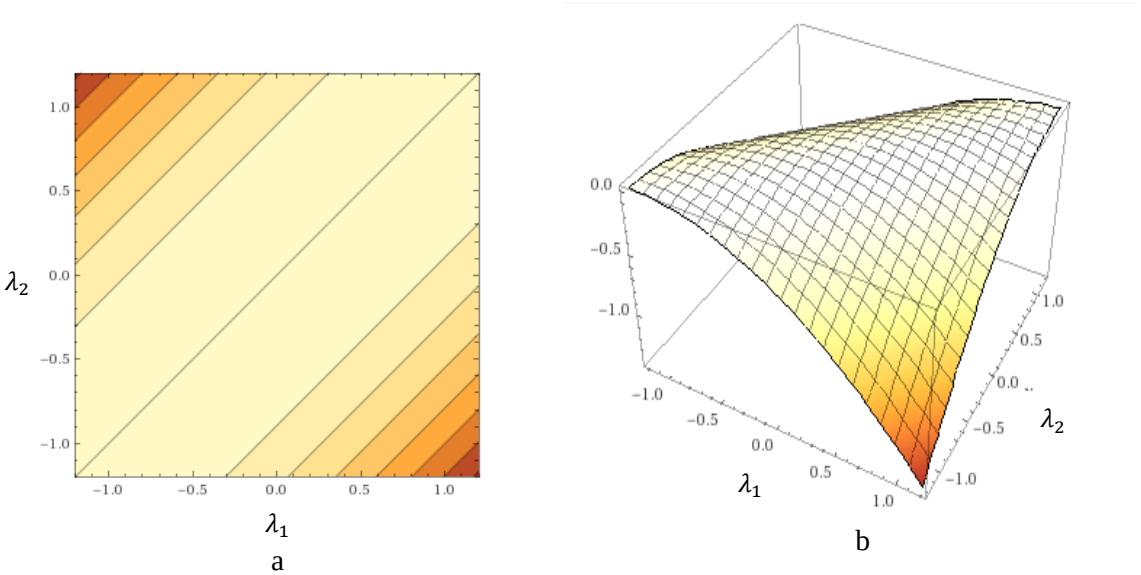


Rysunek 4.2.2. Wykresy eliptycznej paraboloidy generowane przez równanie $W = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$, dla $k > \frac{1}{4}$. (a) Wykres dwuwymiarowy dla $k = \frac{6}{10} > \frac{1}{4}$. (b) Wykres trójwymiarowy dla $k = \frac{6}{10} > \frac{1}{4}$.

Dla $k = \frac{1}{4}$ równanie to przyjmuje postać

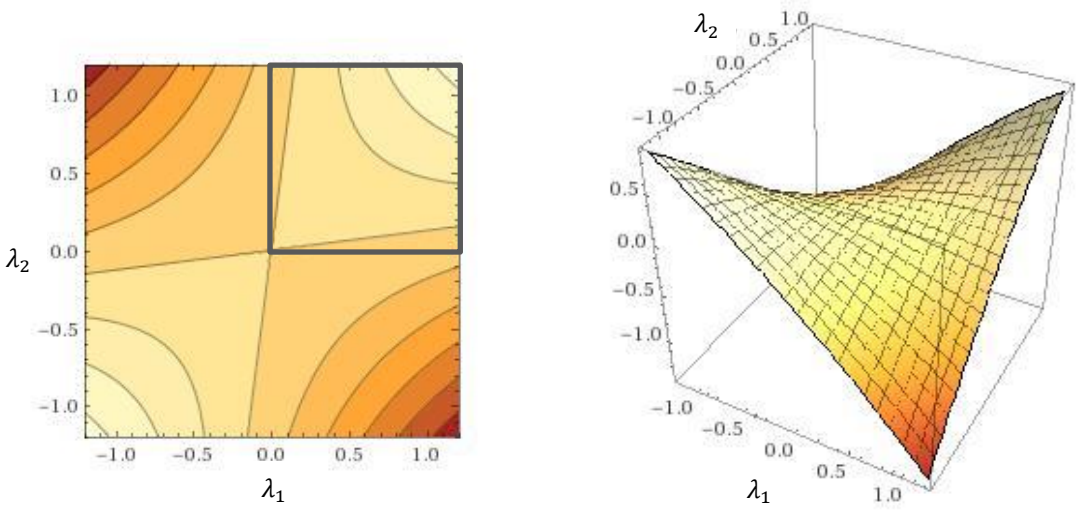
$$W = -\frac{1}{4}{\lambda_1}^2 - \frac{1}{4}{\lambda_2}^2 + \frac{1}{2}\lambda_1\lambda_2 = -\frac{1}{4}\,(\lambda_1 - \lambda_2)^2$$

Wtedy równanie to przedstawia paraboliczny cylinder, w którym pionowe przekroje to parabole, a poprzeczne przekroje to dwie równoległe linie, rys. 4.2.3.



Rysunek 4.2.3. Wykresy parabolicznego cylindra generowane przez równanie $W = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$, dla $k = \frac{1}{4}$. Po uproszczeniu $W = -\frac{1}{4}\,(\lambda_1 - \lambda_2)^2$. (a) Wykres dwuwymiarowy. (b) Wykres trójwymiarowy.

Dla $k < \frac{1}{4}$ w powyższym równaniu wielomianu charakterystycznego W dominuje czynnik $\lambda_1\lambda_2$. Stąd dla niskich wartości k mamy $W \approx \lambda_1\lambda_2$. Równanie to przedstawia hiperboliczną paraboloidę, w której pionowe przekroje to parabole, a poprzeczne przekroje to hiperbole, rys. 4.2.4.

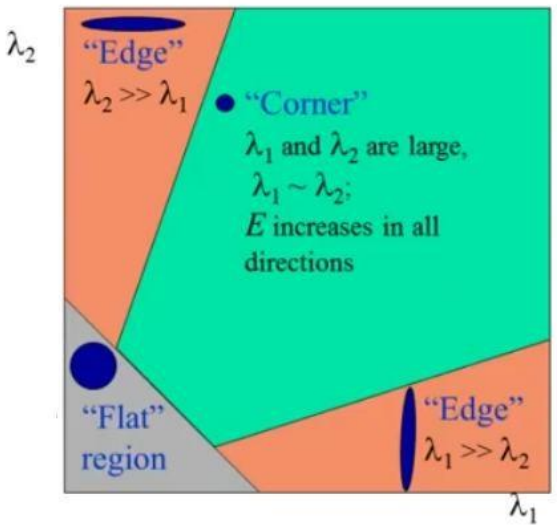


a b

Rysunek 4.2.4. Wykresy hiperbolicznej paraboloidy generowane przez równanie $W = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$, dla $k < \frac{1}{4}$. (a) Wykres dwuwymiarowy dla $k = \frac{1}{10} < \frac{1}{4}$. Na wykresie jest zaznaczona pierwsza ćwiartka współrzędnych, gdyż w procesie identyfikacji punktów charakterystycznych brane są pod uwagę tylko dodatnie wartości własne λ_1, λ_2. (b) Wykres trójwymiarowy dla $k = \frac{1}{10} < \frac{1}{4}$.

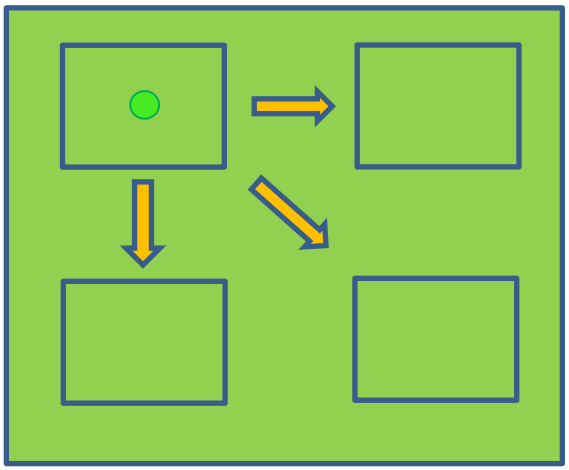
Wartość współczynnika k jest parametrem algorytmu znajdywania punktów charakterystycznych. W pracy [9] pokazano, że punkty charakterystyczne najłatwiej jest znaleźć dla wartości współczynnika $k < \frac{1}{4}$. W dalszych rozważaniach będziemy zakładać, że wartość k podana jako parametr dla algorytmu jest mniejsza od $\frac{1}{4}$ oraz przez równanie $W = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$ będziemy mieli na myśli wykres hiperbolicznej paraboloidy, rys. 4.2.4.

Na rysunku rys. 4.2.5 przedstawiono wykres pierwszej ćwiartki wielomianu charakterystycznego $W = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$ wraz z oznaczeniem charakterystycznych obszarów, zależnych od wartości własnych λ_1 i λ_2.



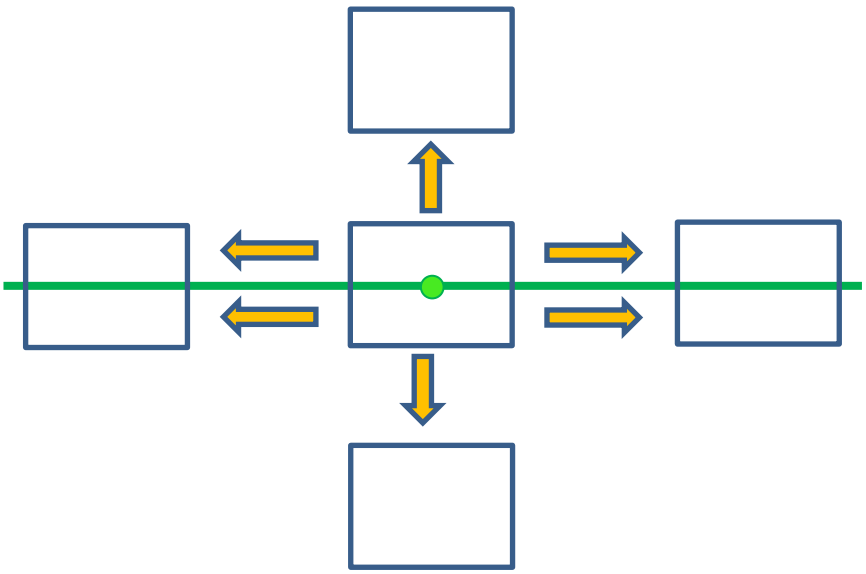
Rysunek 4.2.5. Wykres I ćwiartki wielomianu $W = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$ przedstawiający zależności między wartościami własnymi a charakterystykami badanych obszarów obrazu. Rysunek zaczerpnięty z https://docs.opencv.org

Gdy $|W|$ jest małe, co następuje dla małych λ_1 i λ_2, to badany punkt obrazu znajduje się w jednolitym, jednorodnym obszarze, np. jak na rys. 4.5.1. Cechą charakterystyczną takiego obszaru jest to, że przesuwanie okna poszukiwań w dowolnym kierunku od badanego punktu, piksela nie powoduje prawie żadnych zmian w sumie różnic jasności pikseli znajdujących się w wewnątrz okna poszukiwań, rys. 4.2.6.



Rysunek 4.2.6. Wizualizacja jednolitego obszaru. Gdziekolwiek byśmy nie przesunęli okna poszukiwań to suma różnic jasności pikseli wewnątrz niego przyjmuje niemalże identyczne wartości.

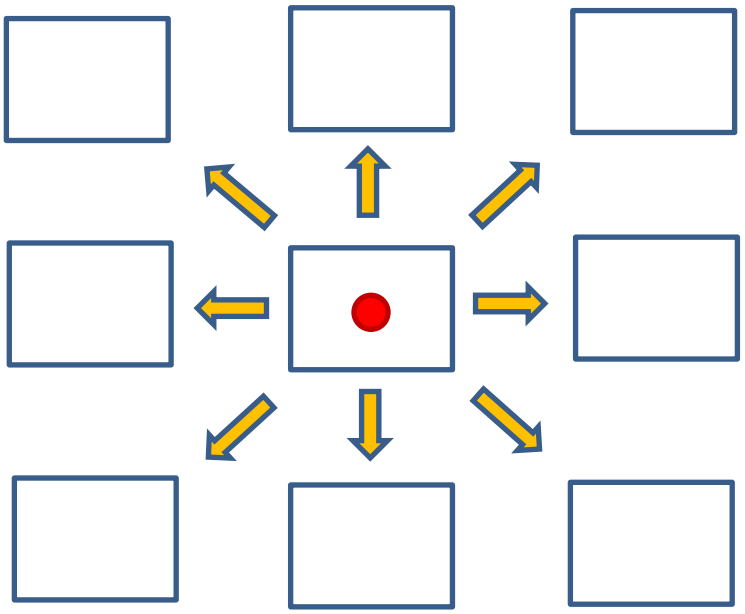
Gdy $W < 0$, co następuje dla $\lambda_2 \gg \lambda_1$ lub $\lambda_1 \gg \lambda_2$, to badany punkt obrazu znajduje się w obszarze zwanym krawędzią, np. jak na rys. 4.5.2. Obszar taki charakteryzuje się tym, że przesuwanie okna poszukiwań wzdłuż krawędzi nie powoduje zbyt dużych zmian w sumie różnic jasności pikseli wewnątrz okna poszukiwań, podobnie jak w przypadku jednolitego obszaru. Natomiast przesuwanie okna poszukiwań prostopadle lub pod kątem w stosunku do krawędzi powoduje pewne zmiany w sumie różnic jasności pikseli wewnątrz okna poszukiwań niemalże jak w przypadku punktu charakterystycznego, rys. 4.2.7.



Rysunek 4.2.7. Obszar będący krawędzią. Przesuwanie okna poszukiwań wzdłuż krawędzi nie powoduje zbytnich zmian w sumie różnic jasności pikseli wewnątrz okien poszukiwań. Przesuwanie okna poszukiwań prostopadle lub pod kątem do krawędzi powoduje pewne zmiany w sumie różnic jasności pikseli wewnątrz okien poszukiwań.

Gdy $|W|$ jest duże, co następuje dla dużych λ_1 i λ_2, to badany punkt obrazu jest punktem charakterystycznym, czyli takim jakiego szukamy. W przypadku takiego punktu, piksela przesunięcie okna poszukiwań w dowolnym kierunku powoduje dużą zmianę sumy różnic jasności pikseli znajdujących się wewnątrz okna poszukiwań, rys. 4.2.8. Na przykład bardzo jasny punkt na ciemnym tle albo bardzo ciemny punkt na jasnym tle albo punkt będący częścią unikalnego wzorca nie występującego nigdzie indziej na obrazie.

Naszym celem jest znalezienie najbardziej wyraźnych punktów charakterystycznych, które znacznie różnią się jasnością od wszystkich sąsiednich pikseli, tak jak na rys. 4.2.8, a więc algorytm powinien znajdywać takie piksele obrazu dla którego $k < \frac{1}{4}$ oraz obliczone λ_1 i λ_2 jest duże, powyżej ustalonego progu τ.



Rysunek 4.2.8. Wizualizacja punktu charakterystycznego. Dowolne przesunięcie okna poszukiwań powoduje dużą różnicę w sumie różnic jasności pikseli znajdujących się wewnątrz okna poszukiwań.

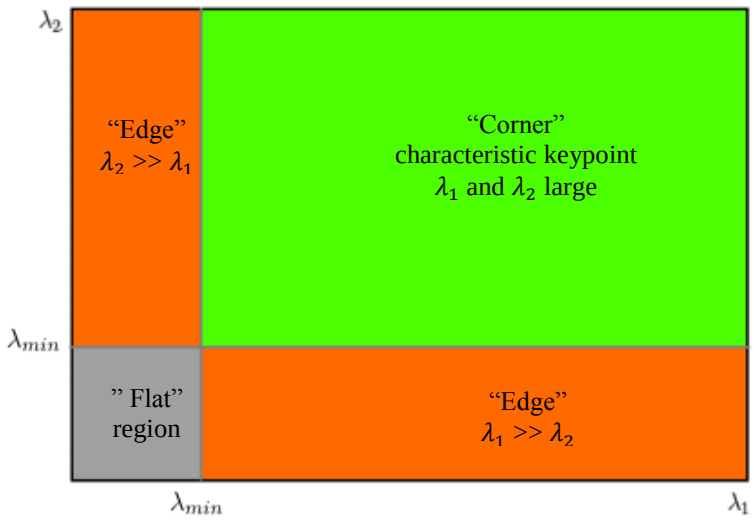
W 1994 roku Jianbo Shi oraz Carlo Tomasi wprowadzili w artykule [10] zmianę do pierwotnego algorytmu zaproponowanego przez Chris'a Harris'a i Mike'a Stephens'a. A mianowicie zamiast równania na wielomian charakterystyczny W

$$W = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$$

zostało zaproponowane uproszczenie

$$W = min(\lambda_1, \lambda_2)$$

Wykres tego równania został przedstawiony na rys. 4.2.9. Zaproponowane równanie daje podobny wykres co pierwotne równanie, rys. 4.2.5. Dzięki temu równaniu łatwiej jest policzyć czy badany punkt jest punktem charakterystycznym. Wystarczy, że każda z wartości własnych λ_1, λ_2 będzie większa niż λ_{min}.



Rysunek 4.2.9. Wykres funkcji $W = min(\lambda_1, \lambda_2)$ identyfikującej punkt charakterystyczny zgodnie z algorytmem Shi-Tomasi. Źródło rysunku: https://docs.opencv.org

W bibliotece OpenCV istnieje gotowa funkcja, która wykonuje algorytm Harris-Shi-Tomasi znajdywania punktów charakterystycznych.

```
void goodFeaturesToTrack(InputArray image, OutputArray corners,
int maxCorners, double qualityLevel, double minDistance,
InputArray mask=noArray(), int blockSize=3,
bool useHarrisDetector=false, double k=0.04)
```

W funkcji tej struktura `image` oznacza obraz, w którym poszukiwane są punkty charakterystyczne, `corners` to struktura która będzie przechowywać znalezione punkty charakterystyczne po wykonaniu tej funkcji. Zmienna całkowita `maxCorners` oznacza maksymalną liczbę punktów charakterystycznych jaka ma być zwrócona w wyniku działania tej funkcji. Zmienna całkowita `blockSize` określa rozmiar sąsiedztwa w którym znajdywane są punkty charakterystyczne. Zmienna rzeczywista `k` oznacza parametr który występuje w równaniu na wielomian charakterystyczny $W = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$.

Na rysunku 4.2.10 przedstawione zostały efekty działania zaimplementowanego w aplikacji algorytmu do znajdywania punktów charakterystycznych na przykładzie ramki obrazu testowanego filmu.

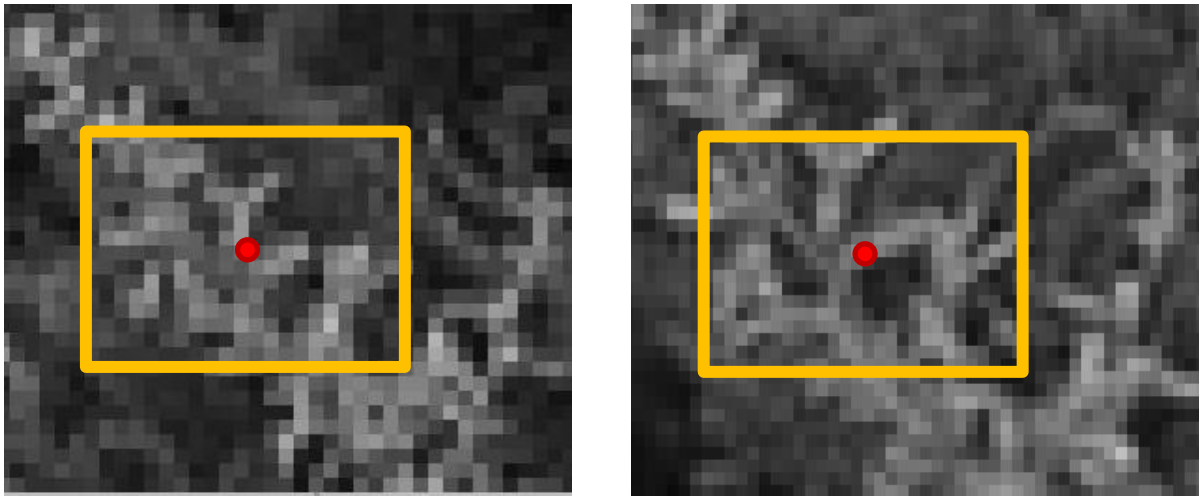
Rysunek 4.2.10. Ilustracja działania zaimplementowanego w aplikacji algorytmu do znajdywania punktów charakterystycznych. Na zdjęciu przedstawiono 5 ramkę przykładowego filmu wraz z zaznaczonymi najbardziej wyraźnymi punktami charakterystycznymi w tej ramce filmu. Źródło zdjęcia – benchmark [22]

4.3. Etap trzeci. Dopasowanie odpowiadających sobie punktów charakterystycznych

W tym etapie odnajdujemy odpowiadające sobie pary punktów charakterystycznych z dwóch sąsiednich ramek za pomocą metody siłowej (*ang. brute force*) z wykorzystaniem miary SSD (*ang. Sum of Squared Differences*), czyli sumy kwadratów różnic. Dokładniej mamy dwa zestawy punktów charakterystycznych znalezionych w poprzednim etapie algorytmu, jeden zestaw z poprzedniej ramki filmu a drugi zestaw z obecnej ramki filmu. W podwójnej pętli obiegającej dwa zestawy punktów charakterystycznych robimy dopasowanie punktów, każdy z każdym. Tzn. dla każdego punktu charakterystycznego z pierwszego zestawu wybierany jest punkt charakterystyczny z drugiego zestawu, który najbardziej pasuje do tego punktu. Sprawdzanie dopasowania obu punktów charakterystycznych wykonywane jest w ten sposób, że porównywane są wszystkie piksele sąsiadujące z dopasowanymi punktami charakterystycznymi znajdujące się w pewnym oknie wokół obu punktów, rys. 4.3.1. Okno sąsiedztwa punktu charakterystycznego jest najczęściej obszarem prostokątnym lub kwadratowym. W literaturze pokrewnej są stosowane również obszary koliste.

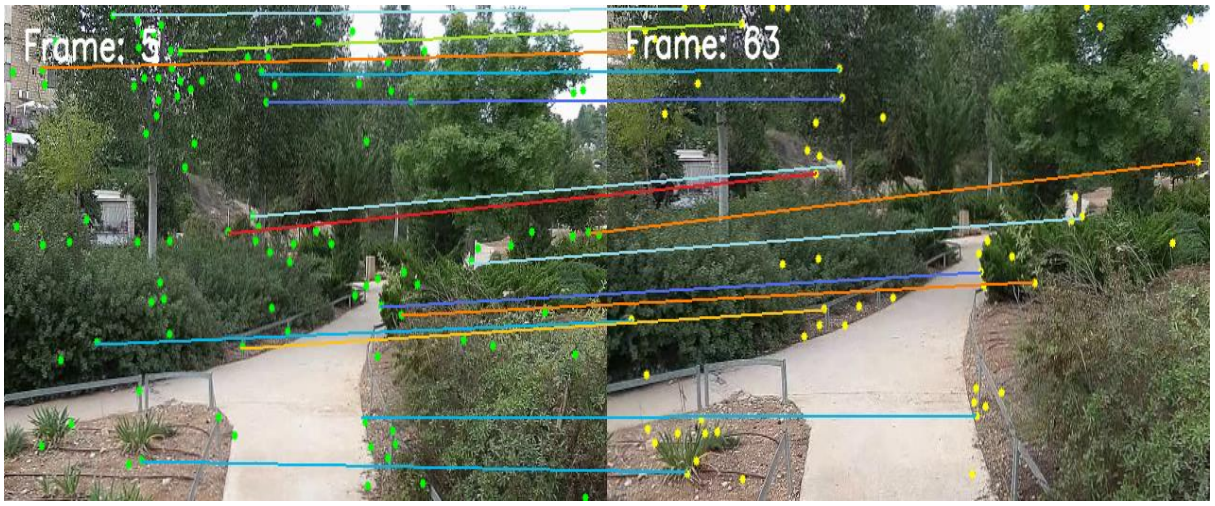
Stopień podobieństwa punktów charakterystycznych i ich sąsiednich pikseli mierzony jest za pomocą miary SSD (*ang. Sum of Squared Differences*), czyli sumy kwadratów różnic. Miarę SSD oblicza się poprzez zsumowanie kwadratów różnic między wszystkimi pikselami znajdującymi się wewnątrz okien sąsiedztwa o wymiarach $M \times N$, których środkiem są odpowiadające sobie punkty charakterystyczne $p(x_0, y_0)$. Ściślej miarę SSD oblicza się zgodnie ze wzorem:

$$SSD = \frac{1}{(2M+1)(2N+1)} \sum_{i=-M}^{M} \sum_{j=-N}^{N} [I_{n+1}(x_0 + i, y_0 + j) - I_n(x_0 + i, y_0 + j)]^2$$



Rysunek. 4.3.1. Sprawdzanie dopasowania dwóch punktów charakterystycznych z sąsiednich ramek poprzez porównanie wszystkich pikseli znajdujących się wewnątrz okna z użyciem miary SSD.

Rezultat wykonania dopasowania punktów charakterystycznych za pomocą metody siłowej (*ang. brute force*) i miary SSD (*ang. Sum of Squared Differences*) można zobaczyć na dwóch ramkach przykładowego filmu, rys. 4.3.2. Na rysunku można zauważyć, że nie wszystkie punkty charakterystyczne z pierwszej ramki mają swoje odpowiedniki wśród punktów charakterystycznych z drugiej ramki. Jest to spowodowane tym, że w drugim etapie algorytmu punkty charakterystyczne są znajdywane w sposób czysto arbitralny.



Rysunek 4.3.2. Rezultat wykonania etapu dopasowania punktów charakterystycznych z dwóch ramek przykładowego filmu. Liniami połączone są odpowiadające sobie punkty charakterystyczne. Źródło zdjęcia - benchmark [22]

Należy zwrócić tutaj uwagę na to, że aby punkty charakterystyczne z sąsiednich ramek mogły się dopasować to musimy korzystać z dobrej kamery cyfrowej, która generuje klatki filmu z dużą częstotliwością, np. około 30 klatek na sekundę. Gdybyśmy korzystali ze słabszych kamer, które generują klatki filmu z niską częstotliwością to punkty charakterystyczne z sąsiednich ramek mogłyby się nie dopasować i mogło by powstać sporo błędów w działaniu algorytmu stabilizacji.

Gdy stosujemy dobrą kamerę cyfrową, która generuje klatki filmu z dużą częstotliwością to zmiany w sąsiednich klatkach filmu są nieznaczne. A przede wszystkim zmiany rotacji i skalowania klatek filmu są na tyle nieznaczne, że odpowiadające sobie piksele z sąsiednich ramek nie różnią się zbytnio od siebie. Dla przykładu rotacja kamery wokół własnej osi zgodnie z ruchem wskazówek zegara powoduje rotację klatek filmu zgodnie ze ruchem wskazówek zegara. Nawet gdyby kamera obracała się z prędkością kątową 30 stopni na sekundę to przy częstotliwości generowania klatek filmu równej 30 klatek na sekundę otrzymalibyśmy efekt, że obraz między sąsiednimi klatkami filmu obraca się o 1 stopień. Przy takim obrocie odpowiadające sobie piksele z sąsiednich klatek mogą być przesunięte względem siebie o 0-3 pikseli w pionie i poziomie, co nie powinno powodować zbyt dużych błędów przy dopasowaniu punktów charakterystycznych za pomocą metody siłowej (*ang. brute force*).

4.4. Etap czwarty. Obliczenie wektorów przesunięcia między obrazami

W tym etapie dla każdej pary dopasowanych punktów charakterystycznych tworzymy obiekt – wektor przesunięcia, który upraszcza dalsze operacje algorytmu. W literaturze pokrewnej również stosuje się wektory przesunięcia pomiędzy sąsiednimi ramkami filmu, zwane LMV (*ang. Local Movement Vectors*), zamiast par dopasowanych punktów charakterystycznych. Na rysunku 4.4.1 przedstawione zostały dopasowane pary punktów charakterystycznych. Punkty charakterystyczne oznaczone kolorem niebieskim to punkty znalezione w obecnej ramce filmu. Punkty charakterystyczne oznaczone kolorem zielonym to punkty znalezione w poprzedniej ramce filmu. Czerwone strzałki to wektory przesunięcia. Pokazują one dokąd przemieściły się punkty charakterystyczne z poprzedniej ramki filmu w kolejnej ramce filmu. Dzięki wektorom przesunięcia jesteśmy w stanie określić w jaki sposób przesunęła się kolejna ramka filmu względem poprzedniej ramki filmu.

Rysunek 4.4.1. Punkty charakterystyczne z dwóch sąsiednich ramek oraz wektory przesunięcia łączące pary odpowiadających sobie punktów charakterystycznych. Źródło zdjęcia - benchmark [22]

W aplikacji realizującej omawiany algorytm stabilizacji filmu została utworzona odpowiednia klasa dla wektorów przesunięcia zwana *MotionVector*. Klasa ta ma następującą strukturę

```
class MotionVector
{
public:
    int x, y; // calculated from difference
    // between endPoint and startPoint
    cv::Point startPoint;
    cv::Point endPoint;
    double magnitude; // from 0 to infinity
    double angle; // from -pi to pi
    void calculateMagnitude();
    void calculateAngle();
    static bool magnitudesComparer(MotionVector firstMotionVector,
    MotionVector secondMotionVector);
    static bool anglesComparer(MotionVector firstMotionVector,
    MotionVector secondMotionVector);
}
```

Wszystkie informacje o parze punktów charakterystycznych mamy zawarte zwięźle w jednej strukturze. Mamy też w tej klasie gotowe funkcje do porównywania kątów i długości wektorów przesunięcia, które upraszczają operacje sortowania wektorów przesunięcia w następnych etapach.

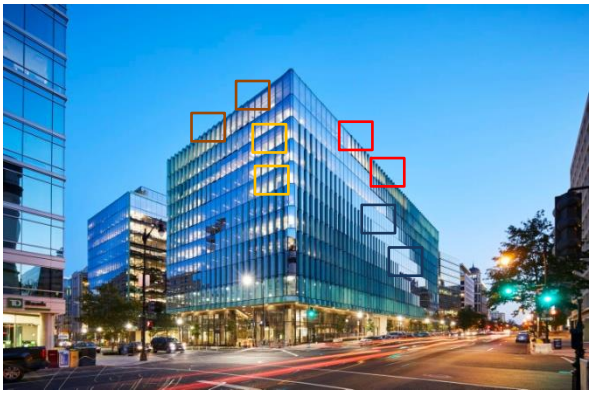
4.5. Etap piąty. Usuwanie błędnych wektorów przesunięcia

W tym etapie wykonujemy eliminację niewłaściwych wartości (*ang. outliers*) uzyskanych w etapach 3 i 4 algorytmu. Jest to konieczne gdyż metoda stosowane w etapie 3 jest dość prosta i mogą powstać liczne błędy dopasowania punktów charakterystycznych nawet gdy mamy do czynienia z filmem nagranym z dużą częstotliwością przetwarzania klatek przez dobrą kamerę cyfrową. Błędy dopasowania punktów charakterystycznych mogą powstać z kilku powodów. W pracy [8] są omówione dwa zasadnicze powody, które przyczyniają się do generowania wielu błędów dopasowań w większości algorytmach stabilizacji filmów opartych na dopasowaniu punktów charakterystycznych.

Pierwszy z tych powodów to jednolitość obszarów w klatkach filmów (*ang. region uniformity*). Charakteryzują się one tym, że są jednorodne i pozbawione punktów charakterystycznych, rys. 4.5.1. Takimi jednorodnymi obszarami są na przykład bezchmurne niebo, filmy kręcone w ciemności albo filmy z rozmazanym obrazem. W pierwszym przypadku wszystkie piksele mają kolor niebieski, w drugim czarny, a w trzecim każdy piksel jest zbliżony kolorystycznie do sąsiednich. W takich obszarach punkty charakterystyczne albo wcale nie występują albo są bardzo niewyraźne i mogą się niejednoznacznie dopasować z wieloma innymi punktami z jednolitego obszaru.

Drugi powód błędów dopasowań omawiany w pracy [8] to wielokrotne wystąpienie wzorca (*ang. block aperture limitation*). Charakteryzuje się to tym, że pewien wyraźny wzorzec pojawia się w obrazie wiele razy. Przykładem są krawędzie obiektów albo wiele obiektów z identyczną teksturą, rys. 4.5.2. W takich przypadkach punkty charakterystyczne na takich wzorcach mogą niejednoznacznie dopasować się z wieloma innymi punktami występującymi na podobnych wzorcach.

Rysunek 4.5.1. Przykład jednolitego obszaru obrazu. Źródło zdjęcia: https://pl.wikipedia.org



Rysunek 4.5.2. Przykład powtarzającego się wzorca. Prostokątami oznaczono identyczne dopasowania. Źródło zdjęcia: https://pl.wikipedia.org

W literaturze pokrewnej opisano różne metody eliminacji tych niewłaściwych dopasowań par punktów charakterystycznych, czyli wektorów przesunięcia pomiędzy ramkami filmu. W wielu metodach stosowano średnią do wyznaczenia najbardziej prawidłowych wartości oraz odchylenie standardowe do wyznaczania najbardziej skrajnych i niepasujących wektorów przesunięcia między ramkami.

W pracy [8] zastosowano metodę wykrywania dwóch głównych rodzajów błędów dopasowań. Według [8] błędy typu jednolitość obszarów w klatkach filmów (*ang. region uniformity*) można wykrywać poprzez zastosowanie pomiaru entropii informacji zawartej w poszczególnych obszarach ramki filmu. Zgodnie z [8] błędy typu wielokrotne wystąpienie wzorca (*ang. block aperture*

limitation) można wykrywać poprzez badanie podobnych dopasowań punktów charakterystycznych znajdujących się w bliskim sąsiedztwie pierwotnego punktu charakterystycznego. Pozostałe niewłaściwie wyznaczone wektory przesunięcia można, zgodnie z [8], wyeliminować poprzez głosowanie porównujące. Głosowanie to polega na dobieraniu w pary wszystkich wektorów przesunięcia. Jeśli różnica w długościach lub kątach tych wektorów jest powyżej pewnego progu ustalonego z góry to oba wektory w parze są kandydatami do głosowania. Wektory z największą liczbą głosów są usuwane. Eliminacja ta kończy się gdy już nie będzie żadnego wektora kandydującego do usunięcia.

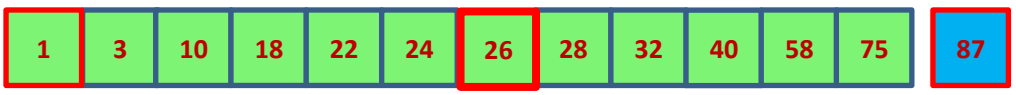
Z kolei w pracy [5] nie są zastosowane metody wykrywania nieprawidłowych wektorów przesunięcia tylko jest zastosowana probabilistyczna metoda RANSAC do eliminacji najbardziej wyróżniających się wektorów (*ang. outliers*) ze zbioru wszystkich wektorów. Ta metoda ona dość czasochłonna i może powodować opóźnienia w systemach czasu rzeczywistego.

Ja w swoim algorytmie stabilizacji filmu proponuję nową metodę eliminacji niewłaściwych wektorów przesunięcia, która daje dość dobre rezultaty przy niewielkim nakładzie pracy procesora. Rezultaty działania proponowanego w tej pracy algorytmu stabilizacji filmów wideo można zobaczyć w rozdziale 6. Proponowany przeze mnie algorytm eliminacji niewłaściwych wartości – wektorów przesunięcia jest następujący:

1) posortuj wszystkie wektory przesunięcia po długości lub po kącie,
2) dopóki wszystkich wektorów przesunięcia jest więcej niż ustalone $\varkappa$ wykonuj,
3) wyznacz medianę – wektor przesunięcia znajdujący się na środku posortowanego ciągu,
4) usuń najmniejszy lub największy wektor przesunięcia w zależności od tego który bardziej się różni od wektora – mediany,

Algorytm 4.5.1. Algorytm eliminacji niewłaściwych dopasowań

Przykładowy wynik uruchomienia tego algorytmu jest pokazany na rysunku 4.5.3. Do wykonania tego algorytmu zastosowałem w aplikacji specjalną strukturę ze standardowej biblioteki C++, kolejkę o dwóch końcach o nazwie deque (*ang. double-ended queue*). Dzięki tej strukturze wszystkie operacje wykonywane w tym algorytmie oprócz sortowania wykonywane są w czasie stałym – $O(1)$. Tzn. operacje sprawdzania pierwszego, środkowego i ostatniego elementu oraz operacje usuwania pierwszego i ostatniego elementu. Operacja sortowania elementów wykonywana jest w tym algorytmie tylko raz, na początku i ma złożoność czasową równą $O(nlogn)$. A więc algorytm ten wykonuje jedno sortowanie oraz wiele operacji o koszcie stałym w pętli zależnej od liczby elementów n. Stąd wynika, że algorytm ten nie pochłania zbytnio zasobów procesora, jeśli mamy do czynienia z około 300 wyznaczonymi parami punktów charakterystycznych. Tego typu algorytm można stosować w systemach czasu rzeczywistego w przeciwieństwie do niektórych czasochłonnych metod eliminacji takich jak RANSAC.



Rysunek 4.5.3. Rysunek ilustrujący działanie algorytmu eliminacji niewłaściwych wartości. Na początku ciąg danych jest sortowany. Potem wyznaczana jest mediana. Następnie usuwane są najbardziej skrajne wartości, najmniejsza lub największa w zależności od tego, która bardziej różni się od mediany do momentu gdy liczba wartości zmniejszy do ustalonej liczby $\varkappa$. Na rysunku zaznaczona jest na niebiesko najbardziej skrajna wartość do usunięcia.

4.6. Etap szósty. Wyznaczenie macierzy przekształcenia afinicznego

W tym etapie obliczamy macierz przekształcenia afinicznego $\mathbf{A}_i$, $i = 0 \,..\, n-1$, gdzie n to liczba ramek stabilizowanego filmu. Wygenerowana macierz przekształcenia afinicznego $\mathbf{A}_i$ mówi nam w jaki sposób trzeba przekształcić ramkę F_i aby upodobnić ją do następnej ramki filmu F_{i+1}. Macierz przekształcenia afinicznego $\mathbf{A}_i$ ma następującą postać

$$\mathbf{A}_i = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}, \qquad \mathbf{M} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

gdzie t_x oraz t_y oznaczają przesunięcia wzdłuż osi X i Y oraz a_{11}, a_{12}, a_{21}, a_{22} mogą przyjmować arbitralne wartości pod warunkiem zachowania nieosobliwości macierzy $\mathbf{M}$.

Uzyskana macierz przekształcenia afinicznego $\mathbf{A}_i$ jest następnie odwracana w celu uzyskania macierzy odwrotnej $\mathbf{A}_i^{-1}$, którą można wykorzystać do poprawienia ramki filmu F_{i+1}, tak aby upodobnić ją do poprzedniej ramki F_i. W kolejnych etapach algorytmu parametry macierzy odwrotnej $\mathbf{A}_i^{-1}$ są obliczane i poddawane procesowi wygładzania. Po czym tworzona jest z tych parametrów macierz wygładzonego, odwrotnego przekształcenia afinicznego $\mathbf{A}_i^{-1'}$, która służy do przetworzenia kolejnej ramki filmu F_{i+1} tak aby upodobnić ją do poprzedniej ramki filmu F_i. W ten sposób uzyskujemy ustabilizowany ciąg ramek filmu, w którym każda kolejna ramka jest stabilizowana i dołączana do ustabilizowanego ciągu ramek filmu, rys. 1.1.

Ściślej etap 6 algorytmu stabilizacji polega na tym, że wyselekcjonowane najlepsze wektory przesunięcia pomiędzy sąsiednimi klatkami filmu F_i, F_{i+1} uzyskane w poprzednim etapie są wykorzystywane do wyznaczenia przekształcenia afinicznego $\mathbf{A_i}$. W tym celu korzystamy z następującej funkcji z biblioteki OpenCV

```
cv::Mat estimateRigidTransform(InputArray src, InputArray dst,
bool fullAffine)
```

Funkcja ta przyjmuje dwa argumenty zawierające zbiory `src` i `dst` dopasowanych punktów charakterystycznych z sąsiednich ramek i zwraca macierz `cv::Mat`, czyli $\mathbf{A}_i$ przekształcającą obraz zawierający punkty charakterystyczne `src` w obraz zawierający punkty charakterystyczne `dst`, czyli ramkę filmu F_i w ramkę F_{i+1}. Funkcja ta działa w ten sposób, że znajduje macierz $\mathbf{A}$ dla której wartość następującej sumy jest minimalna, dla wszystkich wektorów przesunięcia v_k o punkcie początkowym p_k i punkcie końcowym q_k , $k = 1 \,..\, m$.

$$S = \sum_{k}^{m} (q_k - \mathbf{A} \cdot p_k)^2$$

4.7. Etap siódmy. Korekta parametrów macierzy przekształcenia afinicznego

W tym etapie dokonujemy zasadniczego procesu stabilizacji filmu a mianowicie poddajemy procesowi wygładzania parametry odwrotnego przekształcenia afinicznego $\mathbf{A}_i^{-1}$ i uzyskujemy odwrotne wygładzone przekształcenie afiniczne $\mathbf{A}_i^{-1'}$. Uzyskane wygładzone przekształcenia afiniczne $\mathbf{A}_i^{-1'}$ wykorzystujemy później do ustabilizowania sąsiednich ramek filmów F_i, F_{i+1} poprzez zastosowanie macierzy $\mathbf{A}_i^{-1'}$ na ramce F_{i+1} w celu upodobnienia jej do poprzedniej ramki F_i. Ma to na celu usunięcie chaotycznych przesunięć o wysokich częstotliwościach i umożliwienie ruchu kamery o niskiej częstotliwości w całym filmie.

Ściślej, najpierw pobieramy parametry z macierzy przekształcenia $\mathbf{A}_i^{-1}$, przesunięcie wzdłuż osi X, przesunięcie wzdłuż osi Y, skalę oraz rotację. Następnie poddajemy je procesowi wygładzania, które wygląda następująco. Każdy z 4 wyżej wymienionych parametrów p_i jest przekształcany za pomocą równania

$$p_i' = \alpha \cdot p_i + (1 - \alpha) \cdot 1, \text{ dla parametru będącego skalą}$$
$$p_i' = \alpha \cdot p_i, \text{ dla pozostałych parametrów}$$

Równania te opisują proces wygładzania w taki sposób, że każdy wygładzony parametr jest średnią ważoną oryginalnego parametru oraz parametru macierzy przekształcenia identycznościowego

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Dla przekształcenia identycznościowego **I** parametry są następujące, przesunięcie t_x wzdłuż osi X wynosi 0, przesunięcie t_y wzdłuż osi Y wynosi 0, skala S wynosi 1 oraz rotacja θ wynosi 0. Parametry te można łatwo wyliczyć dokonując porównania

$$\begin{bmatrix} S\cos\theta & -S\sin\theta & t_x \\ S\sin\theta & S\cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Drugie równanie wygładzania parametrów pokazane powyżej można zapisać również w taki sposób $p_i' = \alpha \cdot p_i = \alpha \cdot p_i + (1 - \alpha) \cdot 0$. Wartości 0 lub 1 występujące w równaniach na wygładzanie są parametrami macierzy przekształcenia identycznościowego **I**. Po wykonaniu powyższych równań wygładzania, tworzona jest z powrotem wygładzona macierz przekształcenia afinicznego $\mathbf{A}_i^{-1'}$ na podstawie wygładzonych parametrów p_i'.

Istotą tych równań jest stopniowe amortyzowanie przekształcenia afinicznego $\mathbf{A}_i^{-1}$ dopasowującego kolejną ramkę filmu F_{i+1} do poprzedniej F_i identycznościowym przekształceniem **I**, które umożliwia ruch kamery.

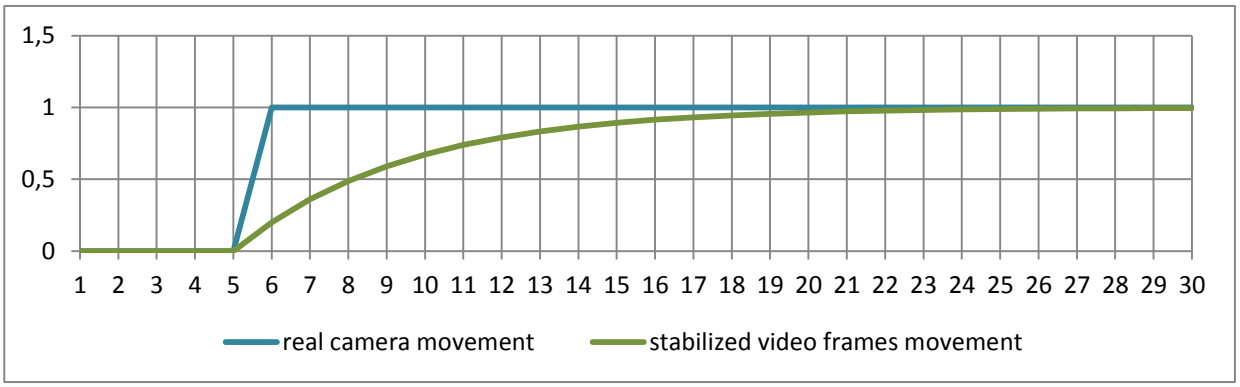
Dla przykładu jeśli $\alpha = 1$ to parametry przekształcenia p_i są takie same jak z przekształcenia afinicznego $\mathbf{A}_i^{-1}$, $p_i' = p_i$, $\mathbf{A}_i^{-1'} = \mathbf{A}_i^{-1}$ i rezultat jest taki, że każda kolejna ramka filmu jest tak dołączana do poprzedniej, żeby wszystkie punkty charakterystyczne widoczne w obrazie były ciągle w tym samym miejscu. W rezultacie obraz widoczny z kamery stoi sztywno w miejscu tam gdzie są punkty charakterystyczne pierwszej ramki filmu, jeśli kamera będzie się stopniowa przesuwać w jakimś kierunku to obraz będzie stopniowo zapełniał się coraz większą liczbą czarnych pikseli aż stanie się całkowicie czarny gdy kamera będzie skierowana w inne miejsce niż na początku. Ustawienie $\alpha = 1$ jest dobre tylko w przypadku gdy kamera jest cały czas skierowna w ten sam punkt przed kamerą przez cały czas trwania filmu.

Natomiast jeśli $\alpha = 0$ to $\mathbf{A}_i^{-1'} = \mathbf{I}$ i rezultat jest taki, że każda kolejna ramka filmu jest przekształcana macierzą identycznościową $\mathbf{I}$, czyli nie jest w ogóle przekształcana. A więc w takim przypadku ustabilizowany film jest taki sam jak oryginalny film, tzn. nie została wprowadzona żadna modyfikacja do oryginalnego filmu.

Z kolei jeśli $0 < \alpha < 1$ to mamy wartość pośrednią, coś pomiędzy swobodnym ruchem kamery a kurczowym dopasowaniem kolejnych ramek filmu tak, żeby punkty charakterystyczne zbytnio się nie przesuwały.

W mojej aplikacji ustawiłem wartość $\alpha = 0.8$, a więc pomiędzy każdą parą ramek jest umożliwiany swobodny ruch kamery w 20%, a punkty charakterystyczne mają się "przyciągać" w 80%, tzn. po każdym przekształceniu mają się przybliżyć do siebie o 20% różnicy dystansu pomiędzy punktami charakterystycznymi stabilizowanych ramek filmu a punktami oryginalnych ramek filmu.

Na poniższym rysunku 4.7.1 oraz tabeli 4.7.1 jest pokazany przykład działania algorytmu stabilizacji, który wygładza parametry macierzy $\mathbf{A}_i^{-1}$ zgodnie z równaniami omówionymi powyżej. Ruch kamery jest na tym wykresie oznaczony niebieską krzywą. Kamera na początku nie rusza się, potem wykonuje gwałtowny skok o jedną jednostkę długości w 5 ramce filmu i pozostaje w tej samej pozycji do końca filmu. Zieloną krzywą oznaczony został ruch punktów charakterystycznych w ustabilizowanych ramkach filmu. Ustabilizowane ramki filmu starają się łagodnie reagować na gwałtowne ruchy kamery. Wartości dla zielonej krzywej zostały obliczone zgodnie ze wzorami na stabilizację parametrów omówionych powyżej: $p_i' = \alpha \cdot p_i$. Dla przykładu po zastosowaniu macierzy $\mathbf{A}_i^{-1'}$ na każdym punkcie ramki filmu F_{i+1} mamy równanie na współrzędną x punktów charakterystycznych $x_{i+1}' = (1-\alpha) \cdot (x_{i+1} - x_i') + x_i'$.

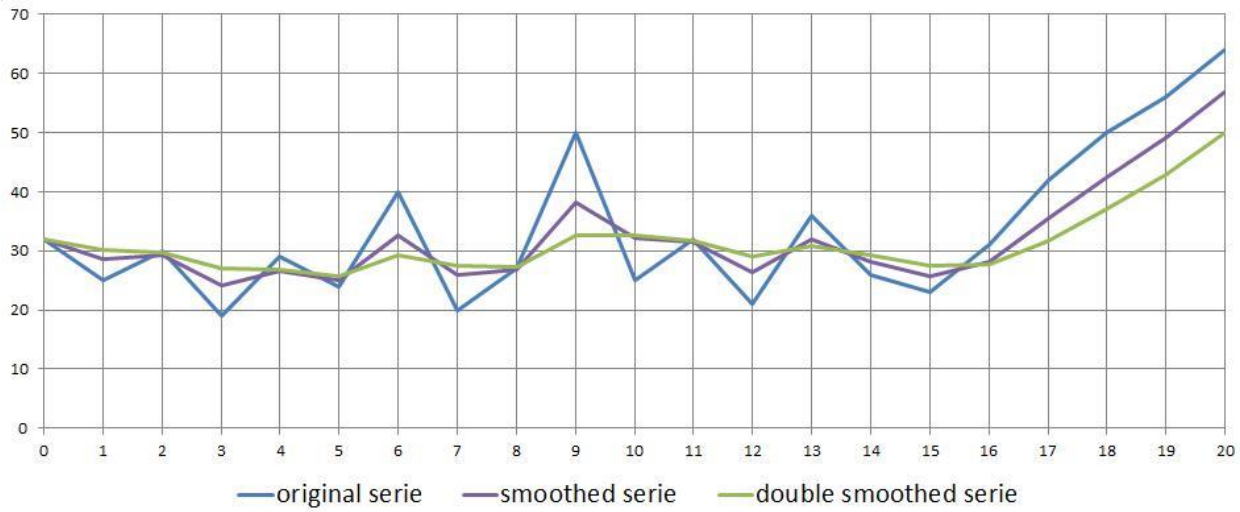


Rysunek 4.7.1. Rysunek ilustrujący sposób reagowania algorytmu stabilizacji ramek filmu na zmiany położenia kamery dla $\alpha = 0.8$. Na wykresie widoczna jest niebieska krzywa przedstawiająca oryginalny ruch kamery oraz zielona krzywa przedstawiająca ruch punktów charakterystycznych ustabilizowanych ramek. Kamera wykonuje gwałtowny skok o jedną jednostkę długości w 5 ramce filmu i pozostaje w tej samej pozycji. Punkty charakterystyczne łagodnie reagują na gwałtowne ruchy kamery.

Tabela 4.7.1. Tabela przedstawiająca wartości punktów dla krzywych z rysunku 4.7.1. Wartości są wyliczone zgodnie z równaniami na stabilizację omówionych w tym rozdziale, $x_{i+1}' = (1-\alpha) \cdot (x_{i+1} - x_i') + x_i'$. Parametr α wynosi 0.8 (80%).

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
real camera movement	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
stabilized video frames movement	0	0	0	0	0	0,2	0,36	0,49	0,59	0,67	0,74	0,79	0,83	0,87	0,89	0,91	0,93	0,95

Na rysunku 4.7.2 przedstawiono efekt wykonywanego procesu wygładzania parametrów przekształceń $\mathbf{A}_i^{-1}$. Wykresy parametrów stają się o wiele stabilniejsze dzięki amortyzacji macierzy przekształceń $\mathbf{A}_i^{-1}$ macierzami jednostkowymi $\mathbf{I}$. Zredukowane są chaotyczne ruchy o wysokich częstotliwościach przy jednoczesnym zachowaniu przesunięć kamery o niskich częstotliwościach.



Rysunek 4.7.2. Rysunek ilustrujący efekt działania procesu wygładzania parametrów. Niebieską linią oznaczono oryginalny, nieprzetworzony ciąg parametrów przekształceń $\mathbf{A}_i^{-1}$. Fioletową linią oznaczono ciąg parametrów poddany jednokrotnemu wygładzaniu zgodnie z procedurą opisaną w tym rozdziale. Zieloną linią oznaczono ciąg parametrów, który został dwukrotnie poddany procesowi wygładzania, tzn. został uzyskany poprzez wygładzenie ciągu parametrów oznaczonego fioletową linią.

Przedstawiona metoda wygładzania parametrów p_i' wszystkich przekształceń $\mathbf{A}_i^{-1}$, $i = 1..n$ jest dość optymalna ponieważ wykorzystujemy tylko 4 zmiennych do przechowywania wszystkich obecnych parametrów w pamięci. Ponadto dla każdego parametru wykonujemy tylko 1 operację mnożenia oraz 1 operację dodawania w przypadku gdy parametr jest skalą, a więc wykonujemy w sumie 5 operacji arytmetycznych dla wszystkich 4 parametrów w trakcie jednego obiegu algorytmu. Przedstawiona metoda jest dość oszczędna i nie pochłania zbytnio pamięci oraz czasu procesora, a więc może być wykorzystywana w systemach czasu rzeczywistego, w których występują ograniczenia na czas wykonywania oraz ilość dostępnej pamięci.

Dla porównania w literaturze pokrewnej są wykorzystywane dość czasochłonne i wymagające dużej ilości pamięci metody wygładzania wykresów. W pracy [15] do wygładzania wykresów ruchu kamery zastosowano optymalizację numeryczną. W pracy [16] zastosowano filtrowanie Kalmana. W wielu metodach stosuje się też funkcje gaussowskie oraz filtry dolnoprzepustowe do przepuszczania ruchu kamery o niskiej częstotliwości oraz blokowania chaotycznych wibracji o wysokich częstotliwościach. W pracy [6] zastosowano dość pracochłonną metodę wygładzania chaotycznego wykresu ruchu kamery poprzez aproksymację tego wykresu wygładzoną funkcją $g(t)$ typu splajn (*ang. cubic smoothing spline*) za pomocą wzoru

$$\mu \sum_{j=0}^{n-1} w(j)\left|p_j - g(j)\right|^2 + (1-\mu)\int \lambda(t)\left|\frac{d^2 g(t)}{dt^2}\right|^2 dt$$

Jak można zauważyć do zaproksymowania takiej funkcji potrzeba znacznie więcej operacji wykonanych na procesorze niż w przypadku rozwiązania zaproponowanego w niniejszej pracy.

4.8. Etap ósmy. Ustabilizowanie kolejnej ramki filmu

Ósmy etap prezentowanego algorytmu stabilizacji filmu wykonywany dla każdej pary sąsiednich ramek filmu F_i, F_{i+1} to zastosowanie uzyskanego, odwrotnego, wygładzonego przekształcenia afinicznego $\mathbf{A}_i^{-1'}$ na kolejnej ramce filmu F_{i+1} w celu upodobnienia jej do poprzedniej ramki filmu F_i. W ten sposób uzyskujemy ustabilizowany ciąg ramek filmu w którym każda kolejna jest upodobniona do poprzedniej. W tym stabilnym ciągu ramek filmu nie występują już chaotyczne wibracje o wysokich częstotliwościach a jednocześnie zachowany jest ruch kamery o niskiej częstotliwości. W celu przetworzenia kolejnej ramki filmu F_{i+1} na każdym pikselu tej ramki wykonywana jest operacja przekształcenia afinicznego $\mathbf{A}_i^{-1'}$

$$\underset{p=(x,y,1)}{\forall} \quad \mathbf{A}_i^{-1'} \cdot p = \begin{bmatrix} S\cos\theta & -S\sin\theta & t_x \\ S\sin\theta & S\cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = p'$$

W bibliotece OpenCV istnieje gotowa funkcja, która wykonuje powyższą operację zastosowania macierzy przekształcenia na wszystkich pikselach obrazu

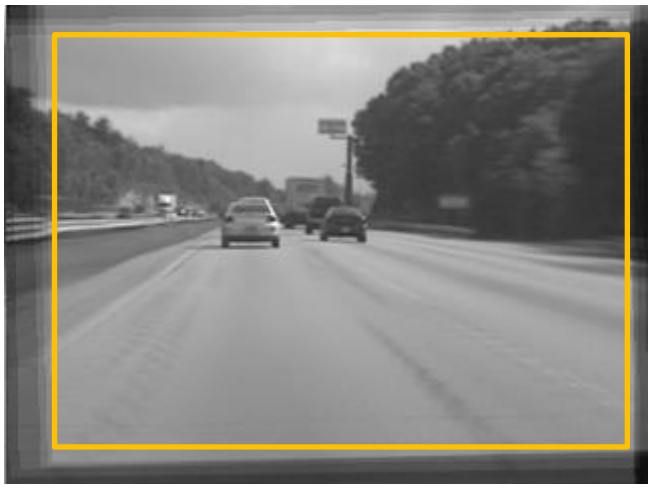
```
void warpAffine(InputArray src, OutputArray dst, InputArray M,
Size dsize, int flags=INTER_LINEAR, int borderMode=BORDER_CONSTANT,
const Scalar& borderValue=Scalar())
```

W funkcji tej struktura `src` oznacza wejściowy obraz, ramkę filmu do przetworzenia, `dst` oznacza wyjściową strukturę w której ma być wygenerowany przetworzony obraz, ramka filmu. Struktura `M` to macierz przekształcenia, która zostanie zastosowana na każdym pikselu struktury `src` w celu utworzenia struktury `dst`.

4.9. Etap dziewiąty. Korekcja kolejnej ramki filmu

W tym ostatnim, opcjonalnym etapie wykonujemy zabieg usunięcia czarnych obszarów występujących na krawędziach prawie każdej ramki, widocznych na rys. 1.1, oraz rys. 5.1.1. Takie czarne obszary pojawiają się po wykonaniu poprzednich ośmiu etapów algorytmu stabilizacji, a ściślej po wykonaniu funkcji `warpAffine` z ósmego etapu algorytmu. Funkcja `warpAffine` transformuje całą ramkę filmu F_{i+1} w inne miejsce tak, żeby upodobnić ją do poprzedniej ramki filmu F_i pozostawiając przy okazji czarne obszary na krawędziach w miejscach poza przekształconą ramką F_{i+1}.

Są dwa sposoby usuwania tego typu czarnych obszarów na krawędziach ramek filmu. Pierwszy sposób, prostszy i stabilniejszy, ale powodujący utratę większej ilości informacji polega na znalezieniu jak największego prostokąta pozbawionego czarnych pikseli spośród wszystkich ramek filmu. Następnie wszystkie ramki filmu są obcinane jednakowo tym największym prostokątem tak, że każda ramka po obcięciu (*ang. crop*) ma właśnie rozmiar tego największego prostokąta i jest pozbawiona czarnych obszarów na krawędziach. Sposób działania tej metody jest ukazany schematycznie na rysunku 4.9.1.



Rysunek 4.9.1. Rysunek przedstawiający I sposób usuwania czarnych obszarów poza przekształconymi ramkami na krawędziach ramek. Ze wszystkich ramek jest wycinany największy, poprawny prostokąt, który jest dobry dla wszystkich ramek jednocześnie. Na rysunku jest widoczna średnia z 50 sąsiednich ramek oraz największy poprawny prostokąt oznaczony pomarańczową ramką. Źródło zdjęcia: https://www.mathworks.com

Drugi sposób powodujący utratę mniejszej ilości informacji, ale trochę bardziej skomplikowany i wprowadzający dodatkowy, niepożądany ruch wewnątrz filmu polega na wycinaniu największego, poprawnego z każdej ramki osobno a nie jednakowo ze wszystkich jak w pierwszej metodzie. W tej metodzie ramki filmu są mniej obcinane, ale za to pojawia się niepożądany efekt przybliżania i oddalania pomiędzy wszystkimi ramkami filmu. Ten efekt jest spowodowany tym, że niektóre przekształcone ramki filmu mają mało czarnych obszarów na krawędziach i są prawie w ogóle nie obcinane, a niektóre przekształcone ramki filmu mają dużo czarnych obszarów na krawędziach i są bardzo mocno obcinane. Wtedy przejście w filmie z tych mało obciętych ramek do mocno obciętych ramek powoduje efekt przybliżania. Natomiast przejście w filmie w drugą stronę z mocno obciętych ramek do mało obciętych ramek powoduje efekt oddalania. Sposób działania tej metody jest przedstawiony schematycznie na rysunku 4.9.2.

Rysunek 4.9.2. Rysunek przedstawiający II sposób usuwania czarnych obszarów poza przekształconymi ramkami na krawędziach ramek. Największy poprawny prostokąt jest wycinany z każdej ramki osobno, a nie jednakowo ze wszystkich ramek. Na rysunkach największy, poprawny prostokąt jest oznaczony pomarańczową ramką. Źródło zdjęć - benchmark [22]

4.10. Szczegóły implementacji algorytmu

Prezentowany algorytm stabilizacji filmów wideo był uruchamiany na komputerze HP Compaq 6200 Pro MT z 10 GB RAM i z procesorem Intel Core i7-2600 (3.40 GHz, 8MB cache, 4 rdzenie). Aplikacja wykonująca algorytm zajmowała około 10 MB pamięci RAM w zależności od rozmiarów klatek filmu. Pojedyncza klatka filmu była na tym komputerze przetwarzana od 0.1 do 10 sekund w zależności od przyjętych parametrów. Sposób pomiaru czasu przetwarzania ramek filmów był następujący. W celu pomiaru czasu ustawiałem odpowiednią flagę i uruchamiałem funkcję do mierzenia czasu. Funkcja ta zliczała ilość dotychczas przetworzonych ramek filmu i mierzyła upływ czasu od początku pomiaru. Czas przetwarzania pojedynczej ramki filmu był obliczany według wzoru:

```
frameProcessingTime = WholeVideoProcessingTime /
                      numberOfFramesInVideo
```

Parametry działania prezentowanego algorytmu stabilizacji można ustawić na 2 zasadnicze sposoby. Pierwszy zestaw parametrów daje wyższą jakość filmu kosztem dużej ilości czasu potrzebnej na obliczenia, ok. $T = 5$ sekund na ramkę, także przy takim zestawie parametrów aplikacji nie da się raczej używać w wymagających systemach czasu rzeczywistego. Natomiast dla drugiego zestawu parametrów aplikacja generuje ustabilizowane filmy gorszej jakości ale za to działa szybciej, ok. $T = 0.1$ sekund na ramkę, czyli w ciągu jednej sekundy przetwarzane jest 10 klatek, $f = \frac{1}{T} = \frac{1}{\frac{1}{10}S} = 10\ Hz$. A więc dla takiego zestawu parametrów prezentowany algorytm mógłby być używany w mało wymagających systemach czasu rzeczywistego, w których częstotliwość przetwarzania wynosi około $f = 15 \div 30\ Hz$.

5. Metody oceny jakości algorytmów stabilizacji filmów wideo

W celu obiektywnego określenia jakości działania algorytmów stabilizacji filmów wideo wprowadzono pewne miary jakości działania tych algorytmów. Dzięki temu jest możliwa obiektywna ocena jakości działania algorytmów stabilizacji zamiast subiektywnej oceny jakości "na oko". W literaturze pokrewnej, m. in. [1][2][3][4] dotyczącej algorytmów stabilizacji filmów przyjęło się stosować następujące 3 miary jakości działania tych algorytmów: miara ucięcia, miara zniekształcenia i miara stabilności. Sposób obliczania tych miar zostanie szczegółowo omówiony poniżej. W dalszej części pracy zostaną również przedstawione otrzymane wyniki jakości działania algorytmu stabilizacji filmów przedstawionego w tej pracy, wraz z porównaniem z wynikami działania algorytmów innych autorów.

Sposób wykonania oceny jakości działania algorytmu jest następujący. W trakcie działania algorytmu stabilizacji filmu analizujemy każdą sąsiednią parę ramek filmu F_i, F_{i+1}, $i = 0..n-2$. Dla każdej sąsiedniej pary ramek F_i, F_{i+1} wyznaczamy przekształcenie afiniczne $\mathbf{A}_i^{-1}$ poprawiające ramkę F_{i+1} i upodabniające je do ramki F_i. Z przekształceń $\mathbf{A}_i^{-1}$ pobieramy parametry, m.in. przesunięcia wzdłuż osi X i Y, skalę, rotację oraz wartości własne, które są wykorzystywane do obliczenia miar. Na koniec po przeanalizowaniu całego filmu wykonujemy dodatkowe operacje na miarach, np. znajdujemy wartość minimalną, wartość średnią lub wykonujemy operację DFT (*ang. Discrete Fourier Transform*) dla całego ciągu parametrów w przypadku miary stabilności.

Każda z miar jakości działania filmów jest rzeczywistą wartością ułamkową mieszczącą się w przedziale $[0..1]$. Im większa wartość miar tym, lepsza jakość działania algorytmu. Warto zwrócić uwagę na to, że można by zaproponować taki naiwny algorytm stabilizacji filmów, który nie przekształcałby żadnych ramek filmu tylko pozostawiałby je takie jakie były pierwotnie po nagraniu. Wszystkie trzy miary dla takiego algorytmu byłyby równe 1, co można sprawdzić poniżej w szczegółowym omówieniu każdej z trzech miar. Jednakże taki naiwny algorytm nie jest brany pod uwagę. W literaturze są analizowane tylko te algorytmy stabilizacji, których wartości miar są mniejsze niż 1, czyli takie które powodują jakiekolwiek przekształcenia korygujące ramki filmu.

5.1. Miara ucięcia

Pierwszą miarą jakości działania algorytmu stabilizacji jest miara ucięcia (*ang. cropping ratio*). Jest ona najprostsza ze wszystkich miar. Główną ideą tej miary jest zmierzenie stosunku ilości pikseli objętych przekształceniem poprawiającym ramkę do ilości wszystkich pikseli występujących w ramce filmu. Na rysunku 5.1.1 można zobaczyć przykładową ramkę filmu poddaną przekształceniu korygującemu. Piksele nieobjęte przekształceniem poprawiającym ramkę są zaznaczane na czarno. Miara ucięcia to, najprościej mówiąc, stosunek ilości kolorowych pikseli do ilości wszystkich pikseli ramki filmu również czarnych, nieobjętych przekształceniem.

Rysunek 5.1.1. Rysunek przedstawiający ramkę filmu poddanej przekształceniu korygującemu. Piksele nieobjęte przekształceniem znajdują się na brzegach ramki i są zaznaczone na czarno. Miara ucięcia to stosunek ilości pikseli objętych przekształceniem do ilości wszystkich pikseli ramki filmu. Źródło zdjęcia - benchmark [22]

Dokładniej miara ucięcia filmu jest ułamkiem o wartości rzeczywistej mieszczącej się w przedziale $[0..1]$. Podobnie jak w przypadku pozostałych miar stabilności im większa wartość miary tym lepiej. W celu jej obliczenia dla każdej sąsiedniej pary ramek filmu F_i, F_{i+1} , $i = 0..n-2$ wyznaczamy przekształcenie afiniczne $\mathbf{A}_i^{-1}$ poprawiające ramkę F_{i+1} i upodabniające je do ramki F_i. Wartość pojedynczego współczynnika ucięcia C_i obliczamy dla każdego przekształcenia $\mathbf{A}_i^{-1}$. Na koniec wyciągamy wartość średnią ze wszystkich współczynników ucięcia C_i i otrzymujemy w ten sposób poszukiwaną miarę ucięcia dla całego filmu $C = \frac{C_i}{n}$, $i = 0..n-1$.

Obliczenie pojedynczego współczynnika ucięcia C_i dla przekształcenia $\mathbf{A}_i$ wygląda w następujący sposób. Powielamy ramkę filmu F_{i+1} przetworzoną do formatu w skali szarości. Usuwamy z niej wszystkie całkowicie czarne piksele o wartości równej 0, zamieniając je np. na wartość 1. W skali szarości wszystkie piksele mogą przyjmować wartości od 0 do 255, czyli 2^8 różnych wartości. Zamiana wartości pikseli nie powoduje żadnych błędów w działaniu algorytmu stabilizacji gdyż działamy na powielonej ramce. Następnie wykonujemy na przetworzonej ramce F_{i+1} przekształcenie $\mathbf{A}_i^{-1}$ upodabniające tą ramkę do ramki F_i . Po takim przekształceniu na ramce filmu powstają całkowicie czarne obszary na krawędziach ramki filmu, jak widoczne na rys. 5.1.1 oraz rys. 1.1. Na przykład przekształcenie, które przesuwa całą ramkę do góry powoduje pojawienie się czarnego obszaru na dolnej krawędzi ramki filmu. Każdy piksel znajdujący się w takim obszarze ma wartość 0. W celu obliczenia pojedynczego współczynnika ucięcia C_i obliczamy stosunek wszystkich pikseli, które nie są czarne, do wszystkich pikseli ramki filmu, również tych czarnych o wartości 0. Liczba wszystkich pikseli w ramce filmu to po prostu szerokość ramki pomnożona przez jej wysokość, tzn. rozdzielczość ramki filmu mierzona w pikselach. Bezpośrednio w kodzie aplikacji stosunek C_i oblicza się w następujący sposób:

```
croppingRatio[i] = (numberOfAllFramePixels - numberOfBlackPixels)
                   / numberOfAllFramePixels
```

gdzie:

`croppingRatio[i]` to i-ty współczynnik miary ucięcia C_i ,

`numberOfAllFramePixels` to liczba wszystkich pikseli w analizowanym obrazie,

`numberOfBlackPixels` to liczba czarnych (o wartości 0) pikseli w analizowanym obrazie

5.2. Wartość zniekształcenia

Drugą miarą jakości działania algorytmu stabilizacji jest wartość zniekształcenia (*ang. distortion value*). Główną ideą tej miary jest zmierzenie jak bardzo są zniekształcone ramki filmu po wykonaniu przekształceń korygujących ramki filmu. Zniekształcenie (*ang. distortion*) mówi nam jak bardzo są spłaszczone lub rozciągnięte ramki filmu. Zniekształcenie obrazu może nastąpić w dowolnym kierunku. Wizualny przykład zniekształcenia wzdłuż osi horyzontalnej jest zilustrowany na rys. 5.2.1.

Matematycznie rzecz ujmując zniekształcenie jest powiązane ze skalowaniem anizotropowym wykonywanym w trakcie przekształcenia korygującego ramkę filmu. Skalowanie może być izotropowe lub anizotropowe. Skalowanie izotropowe charakteryzuje się tym, że skalowanie jest wykonywane równomiernie we wszystkich kierunkach, rys. 5.2.2a. Skalowanie anizotropowe charakteryzuje się tym, że skalowania wzdłuż pewnych kierunków mogą być inne niż skalowania dokonywane wzdłuż innych kierunków ortogonalnych względem siebie. Zniekształcenie występuje tylko w przypadku skalowania anizotropowego. Wartość zniekształcenia jest wtedy liczbą zawierającą się w przedziale otwartym $[0..1)$. W przypadku skalowania izotropowego zniekształcenie nie występuje. Wartość zniekształcenia przyjmuje wtedy wartość równą 1, czyli brak zniekształcenia.

a b

Rysunek 5.2.1. Rysunek przedstawiający zniekształcenie obrazu. (a) Oryginalny obraz bez zniekształcenia. (b) Zniekształcony obraz, który powstał na skutek wykonania na oryginalnym obrazie skalowania anizotropowego wzdłuż osi horyzontalnej o ujemnej wartości. Źródło zdjęcia: https://pl.wikipedia.org

Zgodnie z metodyką przedstawioną w pracy [1] wartość zniekształcenia oblicza się w następujący sposób. Dla każdej sąsiedniej pary ramek filmu F_i, F_{i+1} , $i = 0..n-2$ wyznaczamy odwrotne przekształcenie afiniczne $\mathbf{A}_i^{-1}$ poprawiające ramkę F_{i+1} i upodabniające je do ramki F_i. Potem wyłączamy podmacierz $\mathbf{M}_i$ o wymiarze 2×2 z macierzy przekształcenia afinicznego $\mathbf{A}_i^{-1}$ (3×3) o postaciach

$$\mathbf{M}_i = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \qquad \mathbf{A}_i^{-1} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

W kolejnym kroku obliczamy stosunek dwóch wartości własnych przekształcenia $\mathbf{M}_i$, $S_i = \frac{\lambda_1}{\lambda_2}$. Wartość zniekształcenia otrzymujemy poprzez wybranie najmniejszego stosunku S_i ze wszystkich

przekształceń afinicznych $\mathbf{A}_i^{-1}$ dla wszystkich ramek F_i , $i = 0..n-1$. Wybieramy najmniejszy stosunek S_i dlatego, że niska wartość zniekształcenia może popsuć jakość całego ciągu klatek, nawet gdy sąsiednie klatki mają bardzo wysokie wartości.

Dokładniej wartość zniekształcenia filmu jest współczynnikiem o wartości rzeczywistej mieszczącej się w przedziale [0..1]. Podobnie jak w przypadku pozostałych miar stabilności im większa wartość współczynnika tym lepiej. Dla każdej sąsiedniej pary ramek filmu $F_i, F_{i+1}, i = 0..n-2$ wyznaczamy przekształcenie afiniczne $\mathbf{A}_i^{-1}$ przekształcające ramkę F_{i+1} i upodabniające je do ramki F_i. Przekształcenie afiniczne $\mathbf{A}_i$ jest wyznaczane na podstawie dopasowanych punktów charakterystycznych z sąsiednich ramek F_i i F_{i+1} za pomocą funkcji z biblioteki OpenCV:

```
cv::Mat estimateRigidTransform(InputArray src, InputArray dst,
bool fullAffine)
```

Funkcja ta przyjmuje dwa zbiory `src` i `dst` dopasowanych punktów charakterystycznych z sąsiednich ramek i zwraca macierz `cv::Mat`, czyli $\mathbf{A}_i$ przekształcającą poprzednią ramkę w następną. Macierz odwrotną $\mathbf{A}_i^{-1}$ przekształcającą ramkę następną w poprzednią uzyskujemy poprzez odwrócenie macierzy `cv::Mat`, czyli $\mathbf{A}_i$ uzyskanej dzięki zastosowaniu powyższej funkcji. Operację odwracania macierzy kwadratowych, w tym $\mathbf{A}_i$, można wykonać za pomocą funkcji składowej klasy `cv::Mat` z biblioteki OpenCV

```
MatExpr cv::Mat::inv(int method = DECOMP_LU) const
```

Następną ramkę F_{i+1} da się przekształcić za pomocą przekształcenia $\mathbf{A}_i^{-1}$ tak, żeby była podobna do poprzedniej F_i wykonując na każdym pikselu $p_{i,j}$ drugiej ramki następującą operację

$$p'_{i,j} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \mathbf{A}^{-1} \cdot p_{i,j} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Trzeba zwrócić uwagę, że indeksy i oraz j oznaczają tutaj pozycję piksela $p_{i,j}$ w ramce, czyli coś innego niż indeks i w przekształceniach $\mathbf{A}_i$ i ramkach F_i, F_{i+1}, który oznacza numer klatki filmu. Powyższą operację przekształcania ramek filmu, piksel po pikselu da się wykonać za pomocą funkcji z biblioteki OpenCV

```
void warpAffine(InputArray src, OutputArray dst, InputArray M,
Size dsize, int flags=INTER_LINEAR, intborderMode=BORDER_CONSTANT,
const Scalar& borderValue=Scalar())
```

W funkcji tej struktura `src` oznacza wejściowy obraz, ramkę filmu do przetworzenia, `dst` oznacza wyjściową strukturę w której ma być wygenerowany przetworzony obraz, ramka filmu. Struktura M to macierz przekształcenia, która zostanie zastosowana na każdym pikselu struktury `src` w celu utworzenia struktury `dst`.

Należy zwrócić uwagę tutaj na to, że korzystamy ze współrzędnych jednorodnych (*ang. homogeneous coordinates*), tzn. współrzędnych w dwuwymiarowej przestrzeni zawierające dodatkowy trzeci wymiar. Wymiar ten jest równy 1 w wektorach punktów oraz może przybierać dowolne wartości w dodatkowym trzecim wierszu w macierzach przekształceń. Zgodnie z [17] korzystanie ze współrzędnych jednorodnych ma szereg zalet w grafice komputerowej. Jedną z nich jest możliwość przedstawienia złożenia trzech fundamentalnych przekształceń: translacji, skalowania i obrotu za pomocą jednej macierzy we współrzędnych jednorodnych zamiast iloczynu i sum kilku mniejszych macierzy w zwykłych współrzędnych.

$$\mathbf{T}\cdot\mathbf{S}\cdot\mathbf{R}=\begin{bmatrix}1 & 0 & t_x\\ 0 & 1 & t_y\\ 0 & 0 & 1\end{bmatrix}\cdot\begin{bmatrix}S & 0 & 0\\ 0 & S & 0\\ 0 & 0 & 1\end{bmatrix}\cdot\begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}=\begin{bmatrix}S\cos\theta & -S\sin\theta & t_x\\ S\sin\theta & S\cos\theta & t_y\\ 0 & 0 & 1\end{bmatrix}$$

Następnie, gdy już mamy do dyspozycji podmacierz $\mathbf{M}_i$ o wymiarze 2×2 przekształcenia afinicznego $\mathbf{A}_i^{-1}$ o wymiarze 3×3 obliczamy jej wielomian charakterystyczny W. Zgodnie z [18] wielomian charakterystyczny W ma stopień n równy wymiarowi macierzy kwadratowej **A**. Ponadto wielomian charakterystyczny W o stopniu n ma n pierwiastków, które są jego wartościami własnymi λ_i. Wartości własne λ_i mogą być rzeczywiste albo zespolone. Zgodnie z [18] wielomian charakterystyczny W oblicza się według wzoru

$$W=\det[\mathbf{A}-\lambda\cdot\mathbf{I}]$$

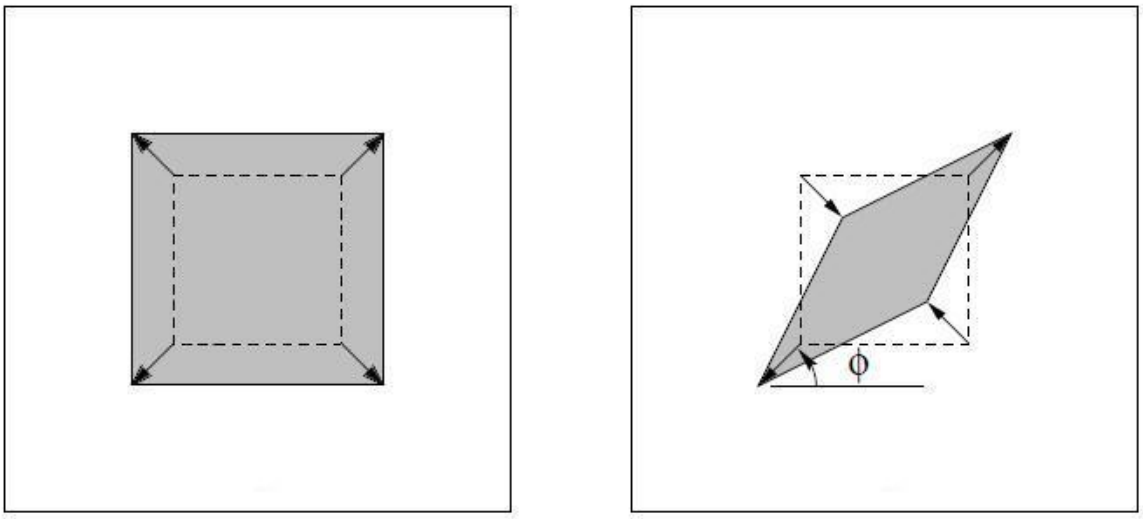
gdzie **I** jest macierzą jednostkową o postaci

$$\mathbf{I}=\begin{bmatrix}1 & 0 & \cdots & 0\\ 0 & 1 & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \cdots & 1\end{bmatrix}$$

Po rozwiązaniu równania $W=\det[\mathbf{A}-\lambda\cdot\mathbf{I}]=0$ otrzymamy 2 wartości własne λ_1 i λ_2 macierzy przekształcenia $\mathbf{M}_i$. Zgodnie z [17] macierz przekształcenia afinicznego **A** ma 3 wartości własne, jedna równa 1, a dwie pozostałe równe współczynnikom skalowania anizotropowego, stąd $\det[\mathbf{A}]=\lambda_1\lambda_2$. Przekształcenie afiniczne **A** różni się od prostszego przekształcenia euklidesowego (podobieństwo) $\mathbf{H}_S$ tym, że potrafi wykonać skalowanie wzdłuż jednej osi inne niż wzdłuż drugiej osi prostopadłej do pierwszej, rys. 5.2.2b. Zwykłe przekształcenie euklidesowe (*ang. similarity*) o postaci

$$\mathbf{H}_S=\begin{bmatrix}S\cos\theta & -S\sin\theta & t_x\\ S\sin\theta & S\cos\theta & t_y\\ 0 & 0 & 1\end{bmatrix}$$

potrafi wykonać tylko skalowanie równomierne, izotropowe o wartości S, jednakowe we wszystkich kierunkach, rys. 5.2.2a.



a b

Rysunek 5.2.2. Rysunek przedstawiający 2 rodzaje skalowania. (a) Skalowanie izotropowe – skalowanie jednakowe we wszystkich kierunkach. (b) Skalowanie anizotropowe – skalowanie wzdłuż jednego kierunku może być inne niż skalowanie wzdłuż ortogonalnych kierunków. Rysunki zaczerpnięte z [17]

Według [17] przekształcenie euklidesowe (podobieństwo) $\mathbf{H}_S$ ma 4 stopnie swobody: przesunięcie wzdłuż osi X, t_x , przesunięcie wzdłuż osi Y, t_y , skalowanie S oraz rotację θ. Natomiast przekształcenie afiniczne **A** stoi wyżej w hierarchii przekształceń i ma 6 stopni swobody: 4 takie same jak przekształcenie $\mathbf{H}_S$, a do tego jeszcze kąt ϕ określający kierunek skalowania oraz stosunek współczynników skalowania anizotropowego $\frac{\lambda_1}{\lambda_2}$. Ten właśnie stosunek dwóch parametrów jest miarą zniekształcenia algorytmu stabilizacji filmów. Określa on jak bardzo klatki filmu są spłaszczone albo rozciągnięte.

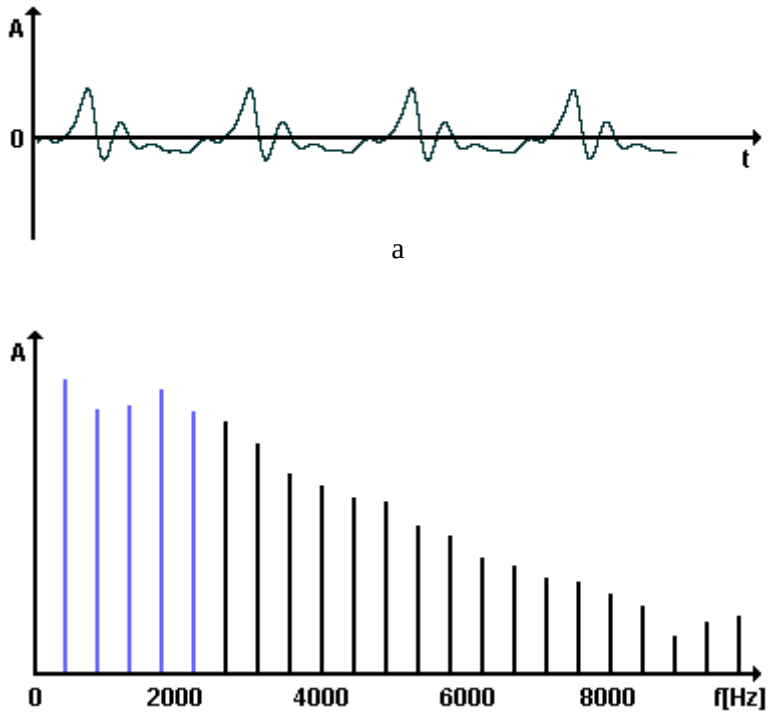
W tym stosunku określającym wartość zniekształcenia (*ang. distortion*) $S_i = \frac{\lambda_1}{\lambda_2}$ dzielimy mniejszą wartość własną przez większą. Jeśli obie są równe to otrzymujemy $S_i = 1$ czyli brak zniekształcenia, skalowanie równomierne (izotropowe) klatek filmu.

5.3. Stabilność

Trzecią miarą jakości działania algorytmu stabilizacji filmów wideo jest stabilność (*ang. stability score*). Główną ideą tej miary jest wyznaczenie jak bardzo stabilny jest obraz, tzn. jak bardzo płynnie lub jak bardzo chaotycznie jest pokazywany obraz w kolejnych ramkach filmu. Ciężko jest wyznaczyć uniwersalną metodę, która byłaby lepsza od pozostałych oraz dobra dla wszystkich rodzajów filmów, gdyż wrażenie stabilności obrazu jest pojęciem subiektywnym. W pracy [1] zaproponowano metodę, która daje empirycznie dobre rezultaty. Autorzy pracy [1] wysunęli założenie, że stabilny, przetworzony film to taki, w którym kolejne klatki filmu przesuwają się względem siebie ruchami o niskiej częstotliwości. Natomiast niestabilny, przetworzony film to taki, w którym kolejne klatki filmu przesuwają się względem siebie ruchami o wysokiej częstotliwości. W związku z tym widmo częstotliwościowe sygnału reprezentujące ruch klatek filmu powinno mieć jak najwięcej energii skupionej w części widma o niskiej częstotliwości i jak najmniej energii skupionej w części widma o wysokiej częstotliwości, rys. 5.3.1b.

Dokładniej stabilność filmu jest współczynnikiem o wartości rzeczywistej mieszczącej się w przedziale $[0..1]$. Podobnie jak w przypadku pozostałych miar stabilności im większa wartość współczynnika tym lepiej. Po przetworzeniu filmu algorytmem stabilizacji wyznaczamy jego stabilność w następujący sposób. Najpierw ze wszystkich przekształceń afinicznych $\mathbf{A}_i^{-1}$, $i = 0\,..\,n-1$ poprawiających każdą ramkę F_i, $i = 0\,..\,n-1$ pobieramy parametry tych przekształceń: przesunięcie wzdłuż osi X, przesunięcie wzdłuż osi Y oraz rotację. W ten sposób uzyskujemy 3 n–elementowe ciągi parametrów p_i, $i = 0\,..\,n-1$, czyli jednowymiarowe dyskretne sygnały o ograniczonym czasie trwania.

Według metody przedstawionej w pracy [1] stabilność jest obliczana jako stosunek energii 5 prążków w widmie częstotliwości, od 2 do 6, do całkowitej energii wszystkich prążków w widmie częstotliwości, rys 5.3.1. Pomijany jest pierwszy prążek dla $f = 0$, czyli sygnał stały (DC).



Rysunek 5.3.1. Wykres funkcji oraz jego widmo częstotliwościowe. Z okresowości sygnału czasowego wynika dyskretne widmo częstotliwości (a) Wykres funkcji ciągłego i okresowego sygnału w dziedzinie czasu. (b) Wykres widma częstotliwościowe go danego sygnału w dziedzinie częstotliwości. Kolorem niebieskim oznaczono prążki widma od 2 do 6.

Aczkolwiek moim zdaniem metoda stosowana przez autorów pracy [1] jest trochę nie w pełni adekwatna do tego zastosowania, gdyż takie postępowanie byłoby dobre gdyby ciąg parametrów p_i określonych na podstawie przekształceń $\mathbf{A}_i^{-1}$, $i = 0..n-1$, był sygnałem ciągłym i okresowym, zamiast dyskretnym i nieokresowym. Zgodnie z [19][20] taki sygnał faktycznie miałby wtedy dyskretne widmo częstotliwości o skończonej liczbie prążków i taki sygnał dałoby się zaproksymować szeregiem Fouriera o skończonej liczbie wyrazów.

Natomiast w dyskretnym, nieokresowym ciągu danych parametrów p_i przekształceń $\mathbf{A}_i^{-1}$, $i = 0..n-1$ mamy do czynienia wieloma próbkami ciągłego widma częstotliwości. Liczbę tych próbek widma częstotliwości w przekształceniu DFT można ustalić arbitralnie zgodnie z [20]. Zazwyczaj ustala się ją jako liczbę próbek zwykłego, dyskretnego sygnału w dziedzinie czasu, czyli w naszym przypadku tyle ile liczba ramek n. Zgodnie z [20] w DFT można zawsze ustalić liczbę próbek widma częstotliwości na tyle dużą, że energia pierwszych 5 prążków widma staje się dowolnie mała mniejsza od zadanego ε. Domyślam się, że autorzy pracy [1] aproksymują dyskretny ciąg parametrów p_i przekształceń $\mathbf{A}_i^{-1}$, $i = 0..n-1$ ciągłą i okresową funkcją a potem znajdują szereg Fouriera dla tej funkcji i obliczają stosunek energii 5 prążków w widmie częstotliwości do całkowitej energii wszystkich prążków w widmie częstotliwości dla tego szeregu. Taka metoda może wprowadzać pewien błąd oszacowania i dać w rezultacie trochę zniekształconą wartość stabilności.

Proponuję następującą metodę obliczania stopnia stabilności dla algorytmu stabilizacji filmów wideo, która powinna obliczać wartość stabilności pozbawioną błędów oszacowania. Najpierw wykonujemy dyskretną transformację Fouriera (DFT) na 3 dyskretnych, nieokresowych sygnałach reprezentujących ciągi parametrów p_i, $i = 0..n-1$ obliczonych z przekształceń afinicznych $\mathbf{A}_i^{-1}$, $i = 0..n-1$. Tzn. wykonujemy DFT dla ciągu przesunięć wzdłuż osi X, przesunięć wzdłuż osi Y oraz rotacji. W ten sposób uzyskujemy dla każdego ciągu parametrów dyskretne n–elementowe widmo częstotliwościowe, P_k, $k = 0..n-1$, reprezentujące częstotliwości z jakimi zmienia się każdy parametr w ciągu.

$$P_k = \mathcal{F}_{\mathcal{D}}(p[n]) = \sum_{i=0}^{n-1} p_i e^{-jki\frac{2\pi}{n}}$$

Ponieważ energię dyskretnego, skończonego sygnału o n próbkach w dziedzinie czasu oblicza się zgodnie ze wzorem:

$$E = \sum_{i=-\infty}^{\infty} |x_i|^2 = \sum_{i=0}^{n-1} |x_i|^2$$

Natomiast zgodnie z twierdzeniem Parsevala [19][20] energię sygnału na podstawie widma częstotliwościowego w przypadku dyskretnej transformaty Fouriera (DFT) można obliczyć w następujący sposób:

$$E = \sum_{i=0}^{n-1} |x_i|^2 = \frac{1}{n} \sum_{i=0}^{n-1} |X_i|^2$$

Jako że niskie częstotliwości mieszczą się w pierwszej połowie widma częstotliwościowego to proponuję obliczać wartość stabilności jako stosunek energii prążków pierwszej połowy widma częstotliwości, pomijając częstotliwość zerową $f = 0$, do energii wszystkich prążków całego widma częstotliwości zgodnie ze wzorem:

$$S = \frac{\sum_{k=1}^{\frac{n}{2}} |P_k|^2}{\sum_{k=0}^{n} |P_k|^2}$$

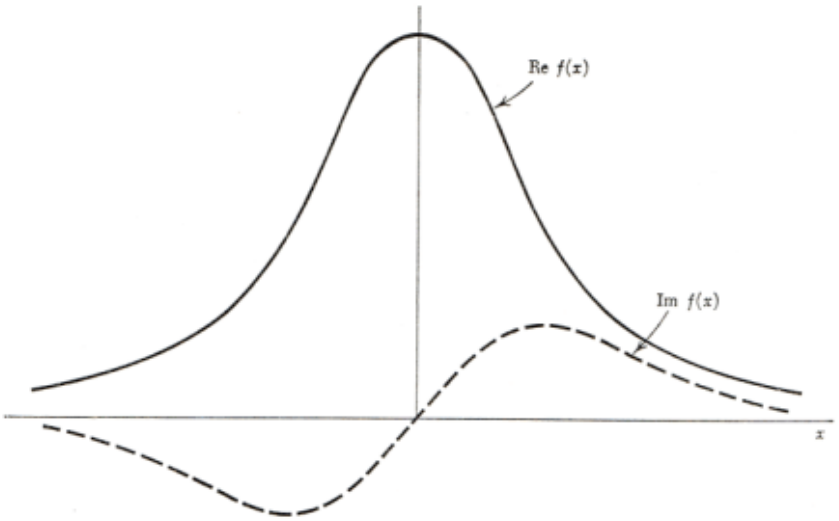
Opisując szczegóły implementacji należy zauważyć, że parametry przekształceń $\mathbf{A}_i^{-1}$, $i = 0 \,..\, n-1$ nie są liczbami zespolonymi tylko liczbami czysto rzeczywistymi. A więc w naszym przypadku mamy do czynienia z dyskretnymi ciągami rzeczywistych parametrów, czyli dyskretnymi, skończonymi sygnałami w dziedzinie czasu o wartościach rzeczywistych. Natomiast z pracy [19] wynika, że transformata Fouriera każdego sygnału o wartościach rzeczywistych jest funkcją hermitowską o wartościach zespolonych. Funkcja hermitowska to taka, która spełnia następującą zależność o nazwie CCS (*ang. Complex Conjugate Symmetry*):

$$f^*(x) = f(-x)$$

Mianowicie, sprzężenie zespolone wartości funkcji dla dowolnego argumentu x ma taką samą wartość jak wartość funkcji dla tego samego argumentu z odwróconym znakiem. Z definicji funkcji hermitowskiej wynika, że część rzeczywista funkcji f jest funkcją parzystą, a część zespolona funkcji f jest funkcją nieparzystą. Funkcje parzyste f_p i nieparzyste f_n spełniają następujące zależności:

$$f_p(x) = f_p(-x) \qquad f_n(x) = -f_n(-x)$$

Stąd wynika, że w naszych rozważaniach wystarczy analizować tylko połowę widma, tzn. wartości widma tylko o dodatnich częstotliwościach, gdyż widmo dla ujemnych częstotliwości jest w pewnym sensie odbiciem lustrzanym widma dla dodatnich częstotliwości. Przykład funkcji hermitowskiej jest pokazany na rysunku 5.3.2.



Rysunek 5.3.2. Wykres przykładowej funkcji hermitowskiej. Część rzeczywista takiej funkcji jest funkcją parzystą. Część zespolona takiej funkcji jest funkcją nieparzystą. Transformata Fouriera funkcji rzeczywistej jest funkcją hermitowską. Transformata Fouriera funkcji hermitowskiej jest funkcją rzeczywistą.

Biblioteka OpenCV [21] dostarcza funkcję, która implementuje dyskretną transformatę Fouriera (DFT)

```
void dft(InputArray src, OutputArray dst, int flags=0,
int nonzeroRows=0)
```

Funkcja ta korzysta z faktu, że widmo częstotliwościowe sygnału o wartościach rzeczywistych jest funkcją hermitowską, która spełnia zależność CCS (*ang. Complex Conjugate Symmetry*). Ponadto w przypadku DFT widmo częstotliwościowe również jest spróbkowane i w funkcji `dft` ma tyle samo próbek (prążków) co wejściowy sygnał rzeczywisty. Dzięki temu tablica `dst` zawierająca wynikowe widmo ma tyle samo elementów co wejściowa tablica `src` zawierająca sygnał rzeczywisty o n próbkach. Wynika to stąd, że tablica wejściowa `src` ma n elementów, tzn. n próbek sygnału wejściowego w dziedzinie czasu. A w przypadku tablicy `dst` zawierającej wynikowe widmo wystarczy analizować tylko połowę widma o dodatnich częstotliwościach. Każdy prążek w widmie jest liczbą zespoloną, a więc mamy $\frac{n}{2}$ prążków o 2 elementach, części rzeczywistej i części zespolonej. Stąd $\frac{n}{2} \cdot 2 = n$.

Funkcja `dft` potrafi przetworzyć tablicę `src` zawierającą sygnał „w miejscu”, tzn. jako argument tablicy docelowej `dst` zawierającej widmo można podać tą samą tablicę co źródłowa `src`. A więc funkcja `dft` potrafi wykonać operacje w pamięci i zapisać rezultat do tablicy źródłowej, co zmniejsza ilość zużywanej pamięci i ilość operacji w trakcie wykonywania dyskretnej transformaty Fouriera (DFT).

6. Wyniki analizy jakości działania opracowanego algorytmu stabilizacji filmów

W tym rozdziale przedstawione są wyniki pomiarów działania opracowanego przeze mnie algorytmu oraz porównanie go z wynikami działania algorytmów opracowanych przez innych autorów. Do obiektywnego porównania jakości działania algorytmów stabilizacji filmów wideo są wykorzystywane 3 obiektywne miary jakości działania omówione w rozdziale 5: miara ucięcia, wartość zniekształcenia oraz stabilność. Dzięki temu możemy obliczyć obiektywną ocenę jakości działania algorytmów stabilizacji filmów wideo zaproponowaną przez poszczególnych autorów, zamiast spekulacyjnej metody „na oko" lub pomiarów wadliwymi, nieobiektywnymi metodami.

6.1. Baza filmów testowych do oceny jakości działania algorytmów stabilizacji filmów

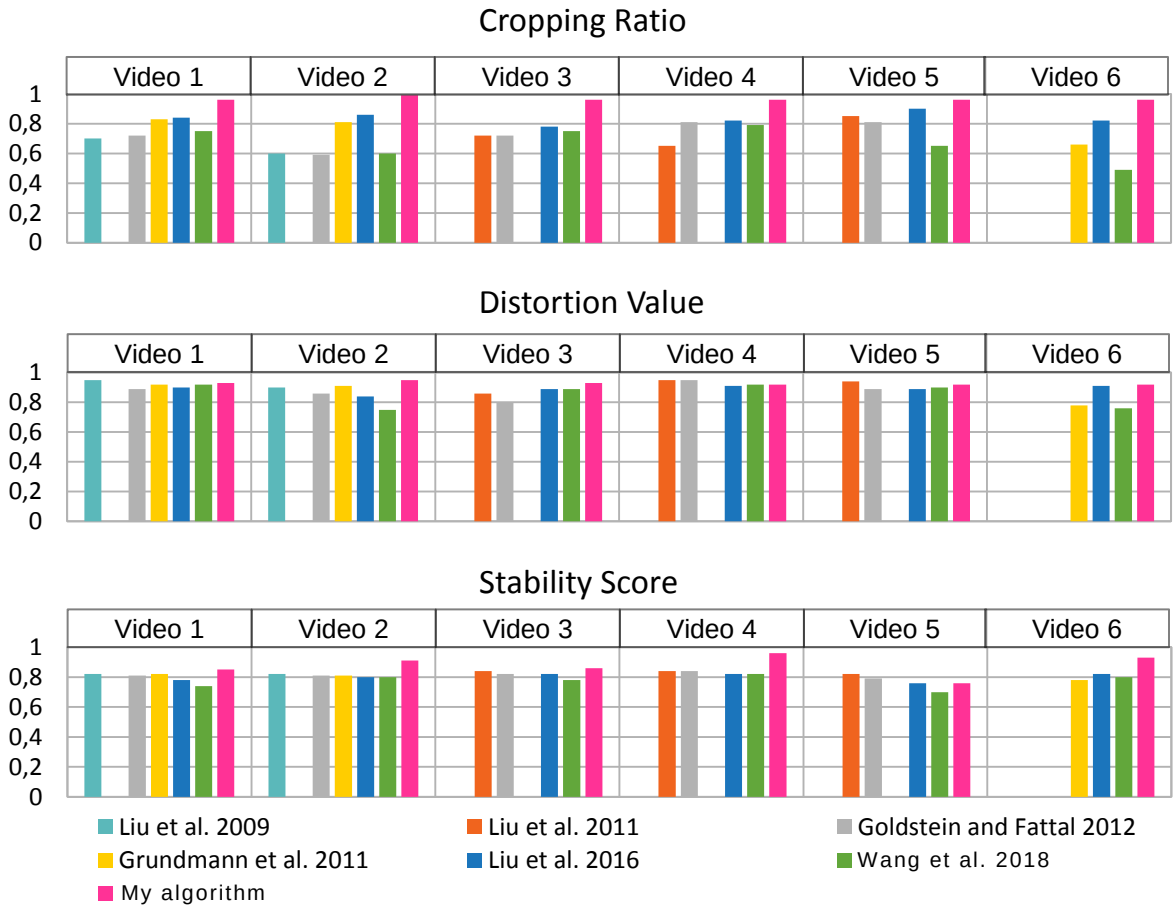
Autorzy pracy [1] udostępnili ogólnodostępną bazę danych 174 filmów zawierających różną ilość chaotycznych przesunięć ramek filmu. Baza ta służy do określania jakości działania algorytmów stabilizacji – benchmark [22]. Filmy zostały zebrane w jeden duży zbiór testowy przez autorów pracy [1] na podstawie poprzednich prac naukowych, internetu oraz własnoręcznie nakręconych filmów. Filmy te trwają od 10 do 60 sekund i są podzielone na 7 kategorii w zależności od różnych sytuacji oraz ilości losowych przesunięć w nich występujących. Siedem kategorii filmów w tym benchmarku to: (I) zwykłe, (II) szybka rotacja, (III) przybliżanie i oddalanie, (IV) duża paralaksa, (V) jazda pojazdem, (VI) tłum, (VII) bieganie. Spora część badaczy pracujących nad algorytmami stabilizacji filmów, m.in. autorzy [1], [2], [3], [4], po zaprezentowaniu swojego algorytmu stabilizacji w swojej pracy prezentuje również wynik jakości jego działania na filmach z udostępnionego benchmarku aby porównać jakość swojego algorytmu z innymi algorytmami opracowanymi dotychczas.

6.2. Porównanie jakości prezentowanego algorytmu z innymi algorytmami na przykładzie 6 wybranych filmów testowych

W pracach [1], [2], [3], [4] oraz innych zostało przedstawione zestawienie jakości działania algorytmu stabilizacji na przykładowych filmach z kategorii (I) zwykłe, wraz z porównaniem działania z algorytmami z poprzednich prac. Poniżej są pokazane wybrane kadry z filmów, w sumie 6, które zostały wykorzystane do oceny jakości działania algorytmów stabilizacji w pracach [1], [2], [3], [4].

Rysunek 6.2.1. Kadry z sześciu filmów wideo z bazy testowej [22] użyte do porównania jakości prezentowanego algorytmu oraz jakości innych autorów.

Na rys. 6.2.2 przedstawione zostały wykresy obliczenia wartości miar określających jakość działania mojego algorytmu stabilizacji powyższych 6 filmów (rys. 6.2.1) wraz z porównaniem wartości miar obliczonych dla 6 algorytmów stabilizacji innych autorów.



Rysunek 6.2.2. Wykresy trzech miar jakości działania algorytmów stabilizacji filmów wideo dla sześciu filmów z kategorii (I) zwykłe z benchmarku [22]. Na wykresach są przedstawione pomiary działania prezentowanego algorytmu oraz algorytmów innych autorów.

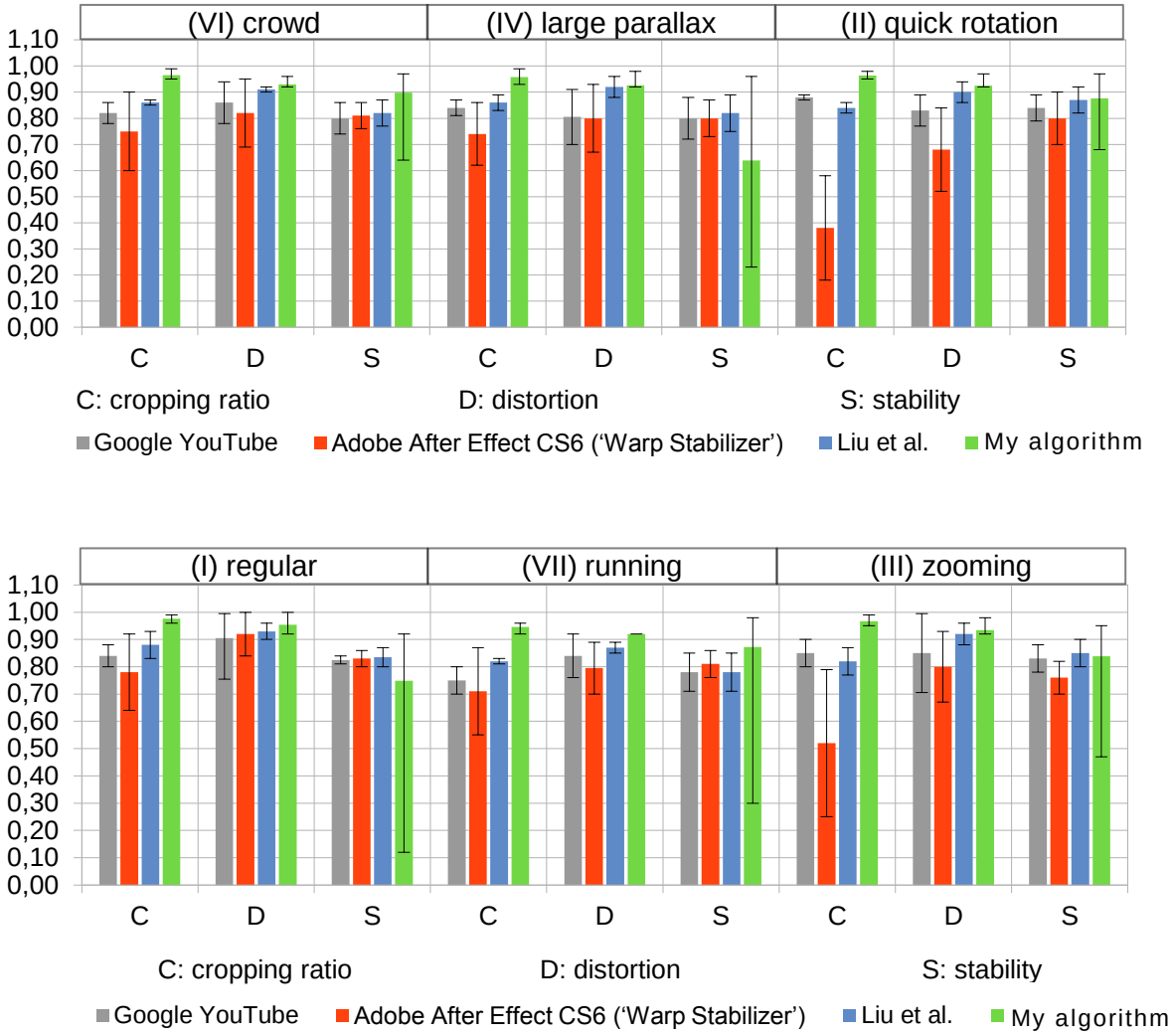
Algorytm oznaczony na wykresie jasnoniebieskim słupkiem oraz opisem „Liu et al. 2009” został przedstawiony w artykule [13]. Algorytm ten używa siatki przepływu (*ang. mesh flow*) podobnie jak algorytm opisany w artykule [1]. Algorytm oznaczony na wykresie jasnoczerwonym słupkiem oraz opisem „Liu et al. 2011” został omówiony w artykule [12] i stosuje metodę stabilizacji podprzestrzeni obrazów 3D.

Algorytm oznaczony na wykresie szarym słupkiem oraz opisem „Goldstein and Fattal 2012” został zaprezentowany w artykule [3]. Algorytm ten stosuje metodę stabilizacji z użyciem geometrii dwubiegunowej (*ang. epipolar geometry*), tzn. używane są dwa punkty widzenia oraz dwie kamery, lewa i prawa. Algorytm oznaczony na wykresie żółtym słupkiem oraz opisem „Grundmann et al. 2011” został przedstawiony w artykule [4]. Algorytm ten wykorzystuje optymalizację unormowanej ścieżki L1. Algorytm oznaczony na wykresie ciemnoniebieskim słupkiem oraz opisem „Liu et al. 2016” został opublikowany w artykule [14]. Wykorzystuje on siatkę przepływu (*ang. mesh flow*) z minimalnym opóźnieniem przetwarzania ramki w systemie czasu rzeczywistego. Osiąga on razem z moim algorytmem najlepsze rezultaty dla wybranych 6 filmów wideo. Algorytm oznaczony na wykresie zielonym słupkiem oraz opisem „Wang et al. 2018” został opisany w artykule [2]. Algorytm ten używa głębokiej sieci neuronowej, która jest trenowana pulą filmów z każdej z 7 kategorii filmów benchmarku [22]. Jak można zauważyć algorytm opisany w pracy [2] osiąga całkiem dobre rezultaty przy bardzo niskim czasie przetwarzania ramek, ok. 10 mikrosekund.

Mój algorytm jest oznaczony na wykresie jasnofioletowym słupkiem oraz opisem „My algorithm”. Jak można zauważyć mój algorytm osiąga całkiem dobre rezultaty w porównaniu z innymi algorytmami, aczkolwiek to dlatego, że badane 6 filmów należy do kategorii (I) zwykle. Mój algorytm osiąga trochę gorsze rezultaty dla filmów z pozostałych kategorii co można zobaczyć w następnym podrozdziale.

6.3. Porównanie jakości prezentowanego algorytmu z innymi algorytmami na wszystkich filmach testowych

Firmy Google, YouTube oraz Adobe również opracowały komercyjne metody służące do stabilizacji filmów wideo. Autorzy prac [1] oraz autorzy algorytmów firmy Google, YouTube oraz Adobe wykonali zbiorcze zestawienie jakości działania ich algorytmów na podstawie przetworzenia wszystkich filmów z benchmarku [22]. Uruchomili oni swój algorytm stabilizacji i przetworzyli nim po kolei wszystkie filmy ze wszystkich kategorii benchmarku [22], po czym stworzyli wykres jak widoczny na rysunku 6.2.2. Na wykresie widzimy średnie wartości miar działania algorytmu obliczone na podstawie wszystkich filmów danej kategorii. Miary jakości działania algorytmu zostały omówione wcześniej, w rozdziale 6, są to: miara ucięcia (*ang. cropping ratio*), wartość zniekształcenia (*ang. distortion value*), oraz stabilność (*ang. stability score*). Na wykresie przedstawione są czarnymi kreskami maksymalne odchylenia od średniej. Oprócz danych pomiarów algorytmów innych autorów dodałem też dla porównania wyniki pomiaru działania mojego algorytmu. Metoda stabilizacji opracowana przez Google i YouTube została oznaczona na wykresie szarym słupkiem i opisem „Google YouTube”. Metoda firmy Adobe została oznaczona na wykresie czerwonym słupkiem i opisem „Adobe After Effects CS6 (‘Warp Stabilizer’)”. Metoda stabilizacji filmów wideo opracowana przez autorów pracy [1] jest oznaczona na wykresie niebieskim słupkiem i opisem „Liu et al.”. Mój algorytm stabilizacji jest oznaczony na wykresie zielonym słupkiem i opisem „My algorithm”.



Rysunek 6.2.2. Wykresy 3 miar jakości działania algorytmów stabilizacji filmów wideo dla wszystkich filmów z benchmarku [22] oprócz kategorii jazda pojazdem. Należy zwrócić uwagę, że kategorie są posortowane alfabetycznie. Na wykresach są przedstawione uśrednione pomiary działania prezentowanego algorytmu oraz algorytmów innych autorów dla wszystkich filmów wideo. Czarnymi kreskami zostały oznaczone najbardziej skrajne pomiary.

Komercyjna metoda stabilizacji opracowana przez Google i YouTube opiera się na połączeniu dwóch metod. Pierwsza metoda to optymalizacja unormowanej ścieżki L1 opisana w artykule [4]. Druga metoda opiera się na wyznaczaniu przekształceń homograficznych bez wstępnej kalibracji parametrów opisana w artykule [11]. Na wykresie można zauważyć, że wypada całkiem nieźle, niewiele gorzej niż metoda opracowana przez autorów pracy [1]. Metoda stabilizacji filmów z Adobe After Effects (*ang. Warp Stabilizer*) w dużej mierze opiera się na stabilizacji podprzestrzeni obrazów 3D opisanej w artykule [12]. Metoda ta wypada na tym zestawieniu najsłabiej ze wszystkich w każdej kategorii filmów. Metoda stabilizacji filmów opracowana przez autorów pracy [1] stosuje siatkę przepływu, tzn. dzieli obszar ramki filmu na wiele prostokątnych obszarów i szuka dopasowań tych obszarów w sąsiednich ramkach zamiast dopasowań poszczególnych punktów charakterystycznych. Metoda ta wypada właściwie najlepiej w tym zestawieniu. Mój algorytm osiąga trochę niższe wartości miar dla kategorii (IV) duża paralaksa oraz dla kategorii (III) przybliżanie i oddalanie, trochę lepiej dla pozostałych. Moja metoda ma też trochę większe odchylenia od wartości uśrednionej niż pozostałe metody.

7. Wnioski

Po przeanalizowaniu metod stabilizacji filmów wideo dotychczas opracowanych w literaturze udało się opracować algorytm, różny od już znanych i opisanych w literaturze. Algorytm przedstawiony w niniejszej pracy zawiera autorskie rozwiązania wymyślone przeze mnie i nieobecne w literaturze pokrewnej. Miedzy innymi zaproponowałem nową metodę usuwania niewłaściwych dopasowań (rozdział 4.5), która jest bardzo oszczędna, ma niską złożoność obliczeniową $O(nlogn)$ i może być z powodzeniem wykorzystywana w algorytmach stabilizacji działających w czasie rzeczywistym.

Opracowałem metodę wygładzania parametrów przekształceń afinicznych korygujących kolejne ramki filmu (rozdział 4.7). Jest ona bardzo oszczędna pod względem obliczeniowym i pamięciowym w porównaniu z innymi stosowanymi metodami gdyż przechowuje w pamięci tylko 4 zmienne typu `float` oraz wykonuje tylko 5 operacji arytmetycznych pomiędzy każdą parą ramek filmu. Stąd wynika, że ta metoda bardzo dobrze nadaje się do wykorzystywania w rygorystycznych urządzeniach wbudowanych działających w czasie rzeczywistym.

Przedstawiony algorytm stabilizacji filmów wideo udało się z powodzeniem zaimplementować używając technologii Qt C++. Algorytm ten osiąga całkiem dobre rezultaty wizualne dla wielu filmów z benchmarku [22]. Prezentowany algorytm został porównany z algorytmami innych autorów i wyniki porównania również zostały przedstawione w pracy. Z wykresów porównujących jakość algorytmów można wywnioskować, że mój, prezentowany algorytm osiąga dobre wyniki w przypadku obliczania miary ucięcia (*ang. cropping ratio*) oraz stabilności (*ang. stability score*). Trochę niższe wyniki osiągane są w przypadku obliczania miary zniekształcenia (*ang. distortion value*). Jednocześnie na wykresach można zauważyć, że mój algorytm osiąga dobre średnie wyniki ale ma spore odchylenia standardowe i rozrzut pomiarów jakości, tzn. w niektórych rzadkich przypadkach mój algorytm może sobie nie poradzić, np. w chaotycznym biegu.

Prezentowany algorytm działa też niedostatecznie szybko, żeby utworzył się płynny obraz w urządzeniu wbudowanym czasu rzeczywistego. Pojedyncza klatka filmu jest przetwarzana od 0.1 do 10 sekund w zależności od przyjętych parametrów, co może być niewystarczające. Przyczyną powolnego działania jest trzeci etap algorytmu wykorzystujący czasochłonną metodę siłową (*ang. brute force*). Ten etap jest wąskim gardłem całego algorytmu. Prezentowany algorytm mógłby działać z powodzeniem w czasie rzeczywistym gdyby w trzecim etapie zastąpić metodę siłową inną metodą np. iteratywną metodą Lucasa Kanade'a z piramidami laplasowskimi (*ang. laplacian pyramid*).

Podsumowując udało się osiągnąć stawiane cele realizacji pracy. Opracowany algorytm działa prawidłowo osiągając nie gorsze wyniki od znanych i opisanych w literaturze. Może być on z powodzeniem stosowany do uzyskiwania stabilnych filmów nagranych różnymi urządzeniami, które zawierają przypadkowe ruchy klatek filmu.

Bibliografia

[1] S. Liu, L. Yuan, P. Tan, and J. Sun. *Bundled camera paths for video Stabilization*. ACM Trans. Graph, no. 4, pp. 78:1–78:10, 2013. http://doi.acm.org/10.1145/2461912.2461995

[2] M. Wang, G.Y. Wang, J.K. Lin, A. Shamir, S.P. Lu, S.M. Hu. *Deep Online Video Stabilization*. IEEE, arXiv:1802.08091v1 [cs.GR] 22 Feb 2018

[3] A. Goldstein, R. Fattal. *Video stabilization using epipolar geometry*. ACM Trans. Graph. (TOG) 31, 5, 126:1–126:10, 2012

[4] M. Grundmann, V. Kwatra, I. Essa. *Autodirected video stabilization with robust l1 optimal camera paths*. In Proc. CVPR, 2011

[5] A. Walha A. Wali A. M. Alimi. *Video Stabilization for Aerial Video Surveillance*. AASRI Conference on Intelligent Systems and Control. Available online 27 November 2013

[6] Y. Wang, Z. Hou, K. Leman and R. Chang. *Real-Time Video Stabilization for Unmanned Aerial Vehicles*. MVA2011 IAPR Conference on Machine Vision Applications, June 13-15, 2011, Nara, JAPAN.

[7] A. Alahi, R. Ortiz, P. Vandergheynst. *FREAK: Fast Retina Keypoint*. 2012 IEEE Conference on Computer Vision and Pattern Recognition. 16-21 June 2012.

[8] R. Chereau and T. P. Breckon. *Robust Motion Filtering as an Enabler to Video Stabilization for a Tele-operated Mobile Robot*. in Proc. SPIE Security and Defence: Electro-Optical Remote Sensing, SPIE, 2013.

[9] C. Harris, M. Stephens. *A Combined Corner and Edge Detector*. Plessey Research Roke Manor, United Kingdom, The Plessey Company plc. 1988.

[10] J. Shi, C.Tomasi. *Good Features to Track*. IEEE Conference on Computer Vision and Pattern Recognition (CVPR94) Seattle, June 1994.

[11] M. Grundmann, V. Kwatra, D. Castro and I. Essa. *Calibration-free rolling shutter removal*. In Proc. ICCP, 2012.

[12] F. Liu, M. Gleicher, J. Wang, H. Jin and A. Agarwala. *Subspace video stabilization*. ACM Trans. Graph. 30, 2011.

[13] F. Liu, M. Gleicher, H. Jin, and A. Agarwala. *Content-preserving warps for 3d video stabilization*. ACM Trans. Graph., vol. 28, no. 3, pp. 44:1– 9, 2009.

[14] S. Liu, P. Tan, L. Yuan, J. Sun, and B. Zeng. *MeshFlow: Minimum Latency Online Video Stabilization*. Cham: Springer International Publishing, pp. 800–815. [Online]. Available: https://doi.org/10.1007/978-3-319-46466-4 48, 2016.

[15] K. Lee, Y. Chuang, B. Chen, M. Ouhyoung. *Video Stabilization Using Robust Feature Trajectories*. National Taiwan University, 2009.

[16] A. Litvin, J. Konrad and W. Karl. Probabilistic *Video Stabilization Using Kalman Filtering and Mosaicking*. IS&T/SPIE Symposium on Electronic Imaging, Image and Video Communications and Proc., 2003.

[17] R. Hartley, A. Zisserman. *Multiple View Geometry in Computer Vision*, 2 ed, pp. 32–44. Cambridge University Press, New York, NY, USA, 2003.

[18] J. Klukowski, I. Nabiałek. *Algebra dla studentów*. Wydanie czwarte uzupełnione. Wydawnictwa Naukowo-Techniczne, Warszawa, 2004, strony 210–245.

[19] J. Szabatin. *Podstawy teorii sygnałów*. Wydanie trzecie. Wydawnictwa Komunikacji i Łączności, Warszawa, 2000, strony 24–140, 224–244.

[20] J. M. Wojciechowski. *Sygnały i systemy*. Wydanie pierwsze. Wydawnictwa Komunikacji i

Łączności, Warszawa 2008, strony 412–476.
[21] Dokumentacja OpenCV – 3.4.8, adres https://docs.opencv.org/3.4.8 [Dostęp 07.01.2020]
[22] Baza danych filmów testowych, benchmark, adres http://liushuaicheng.org/SIGGRAPH2013/database.html [Dostęp 07.01.2020]
[23] Strona domowa programu Wolfram Alpha, adres https://www.wolframalpha.com [Dostęp 07.01.2020]
[24] Strona domowa Mathworks, adres https://www.mathworks.com [Dostęp 07.01.2020]
[25] https://www.mathworks.com/help/vision/examples/video-stabilization-using-point-feature-matching.html [Dostęp 07.01.2020]
[26] Strona domowa Wikipedii, adres https://pl.wikipedia.org [Dostęp 07.01.2020]

Spis rysunków

Spis tabel