

# System for detecting product defects from the production line based on digital image processing using neural networks

Slawomir Serafin

2021-11-16



# Contents

<b>Wprowadzenie</b>	<b>3</b>
Cel projektu . . . . .	3
Założenia przedsięwzięcia oraz zdefiniowanie problemu . . . . .	3
Oczekiwany rezultat . . . . .	3
Wykorzystane narzędzia . . . . .	3
<b>Cyfrowe przetwarzanie obrazów z wykorzystaniem biblioteki OpenCV</b>	<b>4</b>
Protokół RTSP . . . . .	4
Biblioteka OpenCV . . . . .	6
Podstawowa składnia . . . . .	7
Środowisko pracy . . . . .	9
Conda . . . . .	9
Jupyter Notebook/Jupyter Lab/Google Colaboratory . . . . .	11
Podstawowa klasyfikacja obrazów . . . . .	13
Przetwarzanie obrazu . . . . .	16
Oczekiwany rezultat . . . . .	16
Schemat blokowy toru przetwarzania . . . . .	17
Grayscale . . . . .	18
Blurring . . . . .	18
Canny Edge Detection . . . . .	20
Dilation . . . . .	21
Threshold . . . . .	22
Bibliografia . . . . .	23

# Wprowadzenie

## Cel projektu

Celem projektu jest wykonanie programu, który będzie wykorzystywał cyfrowe przetwarzanie obrazów w celu automatyzacji procesów produkcyjnych w zakładzie przemysłowym specjalizującym się w branży meblarskiej. Dodatkowo w celu zwiększenia precyzji zostanie wykorzystana sieć neuronowa z wykorzystaniem biblioteki TensorFlow.

## Założenia przedsięwzięcia oraz zdefiniowanie problemu

Zadanie projektowe polega na zliczaniu otworów w środkowej sekcji komponentu (kolor czerwony na zamieszczonym rysunku). Obiekt będzie poruszał się dynamicznie po linii produkcyjnej z wcześniej zadaną prędkością. Program przed rozpoczęciem pracy powinien zostać dostosowany do aktualnych warunków oświetleniowych panujących na zakładzie. Jeżeli okaże się to niezbędne, konieczne może być zastosowanie dodatkowych źródeł światła aby uzyskać jednolite parametry pracy w dłuższej perspektywie czasowej. Dodatkowo środowisko powinno działać z wydajnością, która będzie umożliwiała wykonywanie operacji na obrazie w czasie rzeczywistym. Język programowania, który wykorzystałem jest Python wraz z dodatkowymi bibliotekami, które znacząco ułatwiają niektóre etapy przetwarzania. Dobór narzędzi nie może być przypadkowy. System będzie wykorzystywany w komercyjnym zakładzie przemysłowym, zatem wszystkie zastosowane narzędzia będą zdefiniowane przy pomocy licencji o swobodnym dostępie zarówno prywatnym jak i komercyjnym. Docelowo cała struktura na zakładzie oparta jest o język C# zatem obiekt, który zostanie zwrócony przez program musi być charakter uniwersalny, czyli taki który możemy wykorzystać również w innym języku niż Python.

## Oczekiwany rezultat

Docelowo program powinien zwracać dwie wartości. Pierwszą z nich będzie wartość typu boolowskiego, która określać będzie zgodność analizowanego artykułu z wartością zdefiniowaną przez użytkownika. Druga wartość będzie typu liczbowego, która reprezentować będzie liczbę otworów w zdefiniowanym przez użytkownika fragmencie. Dodatkowo w celu lepszej wizualizacji zostaną wwrócone współrzędne wykrywanego obiektu z wykorzystaniem sieci neuronowych.

## Wykorzystane narzędzia

W projekcie wykorzystane zostaną różne biblioteki, jednak wspólnym mianownikiem w każdym przypadku będzie język programowania Python. Analizując obraz pod kątem jego wstępnej obróbki wykorzystamy bibliotekę OpenCV, która posiada API dzięki czemu możemy swobodnie korzystać z niej bez względu na wykorzystywany język. W przypadku edytora



Figure 1: Zdefiniowanie problemu projektowego

będę korzystał ze środowiska **Jupyter Notebook** jak również z chmurowego odpowiednika oferowanego przez firmę Google **Google Colaboratory**. W tym drugim przypadku, aby zapewnić stałą synchronizację danych wykorzystamy **Google Drive**. Zdalny dostęp do danych pomiarowych zapewni nam protokół **RTSP**. Natomiast w przypadku sieci neuronowych posłużymy się biblioteką **TensorFlow** oraz nieco jej zmodyfikowaną wersją, która zamiast klasyfikacji umożliwia nam wykrywanie obiektów **TensorFlow Object Detection API**.

## Cyfrowe przetwarzanie obrazów z wykorzystaniem biblioteki OpenCV

### Protokół RTSP

Pierwszym krokiem zanim zaczniemy analizować jakiegokolwiek dane musimy uzyskać do nich dostęp. Możemy to zrobić na wiele różnych sposobów. Jednym z nich jest fizyczne zrobienie zdjęć nad obiektem, które potem będziemy analizować. Rozwiązanie wydaje się stosunkowo proste jednak niesie ze sobą wiele negatywnych aspektów. Rozdzielczość, którą będziemy aktualnie dysponować oraz inne parametry aparatu mogą się znacznie różnić od docelowego rozwiązania. Co więcej, analiza pojedynczej klatki nie pozwala nam w żaden sposób sprawdzić wydajności stosowanych algorytmów, które mogą okazać się zbyt wolne podczas pracy nad rzeczywistym obiektem. Zatem poszukiwane rozwiązanie powinno być już wstępnie znormalizowane do warunków, w jakich będzie działać w późniejszym etapie. Do-

datkowo aby jednocześnie kontrolować aspekt wydajnościowy przekazywany obraz powinien na bieżąco dostarczać kolejne klatki przechwyconego obiektu. W tym miejscu również możemy zastosować dwa podejścia zastosowania problemu. Jednym z nich jest zastosowanie kamery, która będzie się komunikować z komputerem przy użyciu złącza typu USB. Jest to dość proste rozwiązanie jednak wymaga fizycznego połączenia między kamerą a komputerem. Niestety warunki panujące na zakładzie uniemożliwiły takie podejście ze względu na ograniczoną ilość miejsca. Problem postanowiono zatem rozwiązać przy pomocy serwera chmurowego. Takie podejście znacznie ogranicza zastosowanie wszelkich fizycznych połączeń. Niestety musimy tutaj zaakceptować pewnie kompromisy. Jednym z nich jest ograniczenie rozdzielczości w taki sposób aby uniknąć braku płynności w przekazywanym obrazie. Ta cecha jak się okaże w kolejnych etapach okaże się bardzo problematyczna. Kolejnym aspektem jest konieczność zastosowania szybkiego łącza oraz świadomość, że w przypadku tymczasowego braku dostępu do sieci program po prostu nie będzie w stanie funkcjonować. Kolejnym krokiem po zaakceptowaniu wszystkich wad od zalet zastosowania bezprzewodowego jest wybór odpowiedniego narzędzia do przesyłu obrazu. Wybór padł tutaj na rozwiązanie w oparciu o protokół **RTSP** (*Real Time Streaming Protocol*). Jest to rozwiązanie, które znacznie zyskało na swojej popularności podczas okresu pandemicznego, gdzie większość z nas została zmuszona do przejścia w tryb pracy zdalnej. Całość systemu oparta jest na serwerze, który udostępnia nam dostęp do obrazu w czasie rzeczywistym. Dzięki takiemu podejściu wiele użytkowników przy pomocy odpowiedniej platformy może śledzić wydarzenia w tym samym momencie. Takie rozwiązanie jest często wykorzystywane do transmisji różnego rodzaju konferencji, posiedzeń zarządów, wydarzeń kulturowych, czy nawet wydarzeń sportowych. Poniższa grafika przedstawia uproszczony schemat działania. Platformą obsługującą w naszym przypadku będzie biblioteka *OpenCV*.

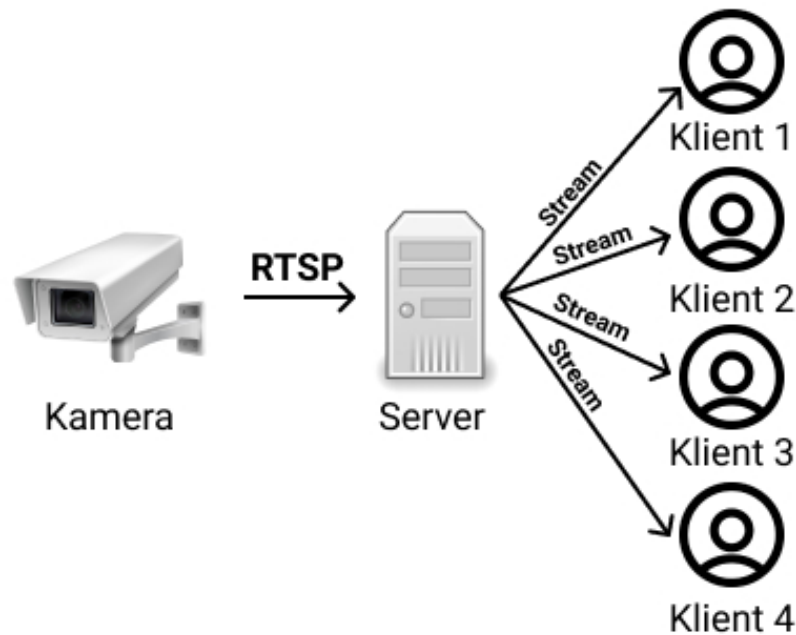


Figure 2: Wykorzystanie protokołu RTSP jako źródło informacji o produkcji

## Biblioteka OpenCV

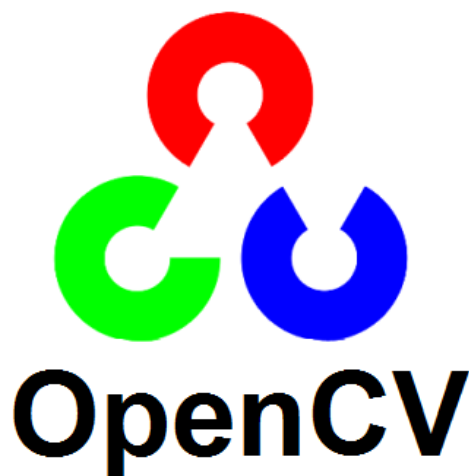


Figure 3: Logo biblioteki OpenCV

**OpenCV (Open Source Computer Vision Library)** jest biblioteką funkconującą na zasadach licencji otwartej. Jej głównym zastosowaniem jest wykonywanie operacji na obrazach w czasie rzeczywistym. W skład biblioteki wchodzi również algorytmy uczenia maszynowego, które znacząco zwiększają zastosowanie oraz wydajność operacji. Kluczowym założeniem przy konfiguracji był aspekt komptybilności z rozwiązaniami komercyjnymi. Biblioteka jest łatwa w modyfikacji oraz posiada przejrzystą strukturę kodu. W skład pakietu wchodzi zarówno klasyczne rozwiązania stosowane od wielu dekad do analizy obrazów jak i najnowocześniejsze struktury dostarczane przez społeczność, która znacząco przyczynia się do rozwoju produktu. Algorytmy mogą służyć między innymi do wykrywania twarzy na zdjęciach, identyfikacji obiektów, klasyfikacji nastroju w jakim znajduje się użytkownik, śledzenia obiektu w ruchu, redukcji czerownych oczu. W dobrej sytuacji penczemicznej program może wykrywać czy wszyscy uczestnicy spotkania mają założone maseczki. Aktualnie nasilony jest również problem migracyjny. Systemy bezpieczeństwa które nieustannie sledzą granice państw są wyposażone również w algorytmy inteligentnego przetwarzania obrazu. Dane na stronie producenta wskazują, że biblioteka została już pobrana przez ponad 18 milionów użytkowników. OpenCV jest ciągle rozwijana nie tylko przez grupy pastonatów, lecz również jest przedmiotem analiz grup badawczych a nawet organów rządowych. Wśród użytkowników znajdują się również wielkie korporacje tj. Google, Microsoft, Intel, IBM, Honda, Sony, VideoSurf, Zeitera. Znane wszystkim użytkownikom Google Maps używa właśnie OpenCV w swoich zastosowaniach. Cała składnia języka została zbudowana w oparciu o język *C++*. Wraz z rozwojem popularności producenci postanowili jednak przyciągnąć nowych użytkowników udostępniając **API**, dzięki czemu z biblioteką możemy się komunikować przy pomocy języka Python, Java oraz dostępne są pewnie funkcjonalne aspekty również w środowisku MATLAB. Interfejs wspiera najważniejsze systemy operacyjne jak: Windows, Mac OS, Linux oraz Android. Dodatkowo implementacja kodu może również

być wykonywana z wykorzystaniem mikrokontrolerów z rodziny Arduino oraz Raspberry Pi. Przy wykonywaniu bardziej zaawansowanych operacjach warto rozważyć akcelerację w oparciu o system **CUDA** oraz **OpenCL**. Jest to jednak możliwe tylko i wyłącznie, jeżeli dysponujemy kartą graficzną o wysokim standardzie. Poniższa grafika ilustruje porównanie czasu wykonywania obliczeń. Na potrzeby symulacji jako GPU została użyta karta *Tesla C2050*, której osiągi porównane zostały z CPU *Core i5-760 2.8Ghz*.

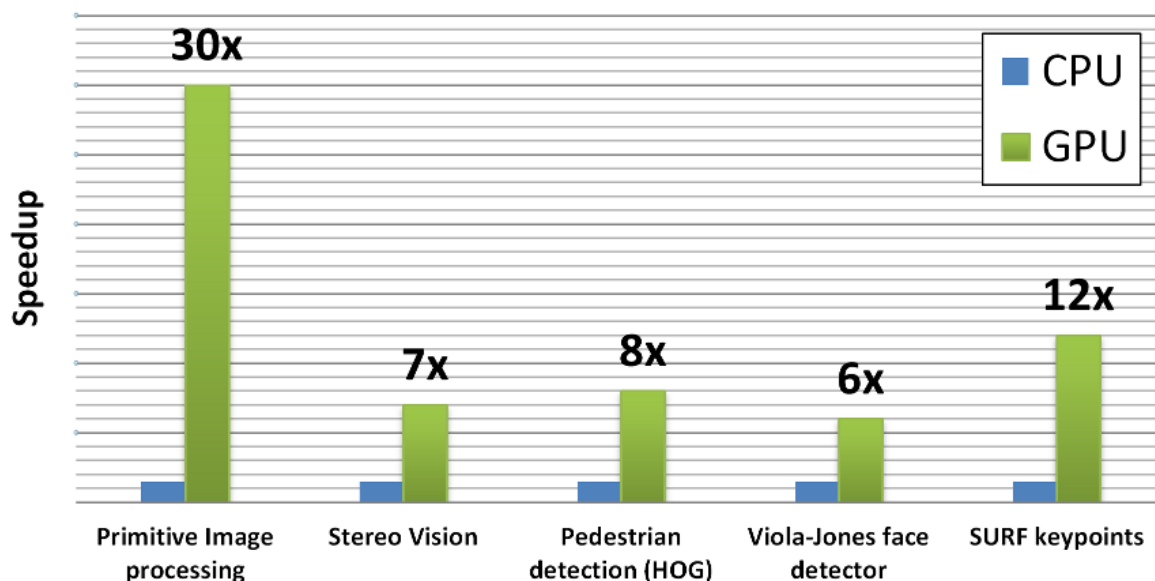


Figure 4: Porównanie czasu obliczeń dla CPU oraz GPU

## Podstawowa składnia

W przypadku gdy źródłem obrazu jest kamera dostarczająca określoną liczbę klatek na dany okres czasowy strukturę programu możemy podzielić na dwa bloki: wewnątrz pętli *while* oraz wartości lub funkcje zdefiniowane poza jej obrębem. Pierwszym etapem rozpoczęcia pracy jest import bibliotek, które będą wykorzystywane. W kolejnym kroku definiujemy źródło obrazu. W tym przypadku *cap = cv2.VideoCapture(0)* odnosi się do sprzętu zdefiniowanego przez komputer jako sygnał domyślny, w dalszej części zostanie on zastąpiony poprzez opisany wcześniej protokół **RTSP**. Kolejno zdefiniowane jest pętla *while*, która jest wykonywana dopóki użytkownik nie naciśnie przycisku powodującego przerwanie operacji (w tym przypadku jest to klawisz *q*). Funkcja *cap.read()* zwraca nam dwie wartości: *ret* oraz *frame*. Pierwsza z nich jest zmienna typu boolowskiego informująca nas czy obraz został przechwycony w sposób prawidłowy. Zmienna *frame* natomiast jest obrazem przedstawionym w postaci wektora.

```

import cv2
#Wybor domyślnego źródła
cap = cv2.VideoCapture(0)

while(cap.isOpened()):
    #Wczytaj obraz
    ret, frame = cap.read()
    ---Operacje na obrazie---
    ---Operacje na obrazie---
    #Zamknij okno gdy użytkownik wciśnie klawisz "q"
    if cv2.waitKey(20) & 0xFF == ord('q'):
        break

```



## Środowisko pracy

### Conda



Figure 5: Conda logo

Zarówno środowisko *Python* jak i wszystkie potrzebne pakiety zostały zainstalowane w oparciu o środowisko **Conda**. Jest to system zarządzania pakietami działający na wszystkich popularnych systemach operacyjnych. Najbardziej rozpowszechniony oraz wykorzystywany jest przez użytkowników zajmujących się przetwarzaniem oraz analizą danych. Mogą być to pliki tekstowe jak również multimedialne. Środowisko wspiera również inne popularne języki programowania tj. *R*, *Ruby*, *Java*, *C/C++*, *FORTRAN*, *JavaScript*. Conda z łatwością przechowuje, tworzy oraz obsługuje biblioteki, które są instalowane w oparciu o dany język programowania. Mając na uwadze fakt, że pisany program będzie w przyszłości dystrybuowany na wiele różnych urządzeń potrzebna jest funkcjonalność, która pozwoli zebrać wszystkie potrzebne pliki w jednym miejscu. W tym momencie z pomocą przychodzi nam *Wirtualne Środowisko*. Pozwoli nam to wyseparować wersje różnych bibliotek, gdzie zostały już wcześniej zainstalowane na komputerze co w prosty sposób pozwoli nam unikać możliwej w przyszłości niekompatybilności. Kolejnym ważnym aspektem jest separacja. Dokonując modyfikacji pakietów wyłącznie w obrębie wirtualnego środowiska nie oddziałujemy w żaden sposób na wersje zainstalowane globalne, co sprawia, że nie wpływamy na kompatybilność wcześniej wykonanych projektów. Do stworzenia oraz aktywacji wirtualnego środowiska służy poniższa składnia.

```

'''
Tworzenie zmiennej środowiskowej o nazwie engineering-thesis
w oparciu o wersje Pythona 3.10 (jeżeli wersja ta nie zostanie
podana system automatycznie wybierze postać domyślną)
'''

conda create --name engineering-thesis python=3.10

#Aktywowanie zmiennej środowiskowej

conda activate engineering-thesis

'''
Jeżeli komenty zostaną wykonane poprawie w wierszu poleceń
powinny być widoczny następujący tekst
'''

(engineering-thesis) C:\Current\Path

'''
Instalacja biblioteki z wykorzystaniem środowiska Conda
'''

conda install -c conda-forge opencv

'''
Alternatywnie możemy użyć menadżera pip
'''

pip install opencv-python

'''
W celu wylistowania wszystkich zainstalowanych pakietów
stosujemy poniższą składnię
'''

conda list

'''
Powinna się wyświetlić lista z wszystkimi pakietami,
przykładowo:
'''

# Name                                Version                                Build  Channel
anyio                                  3.1.0                                  pypi_0
argon2-cffi                           20.1.0                                 pypi_0
py-opencv                             3.4.2                                 py36hc319ecb_0

```

## Jupyter Notebook/Jupyter Lab/Google Colaboratory

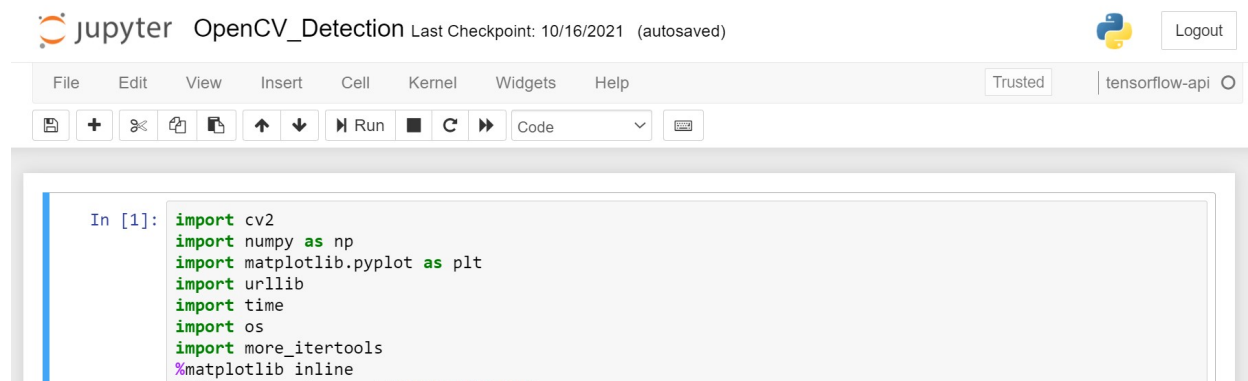


Figure 6: Okno programu Jupyter Notebook

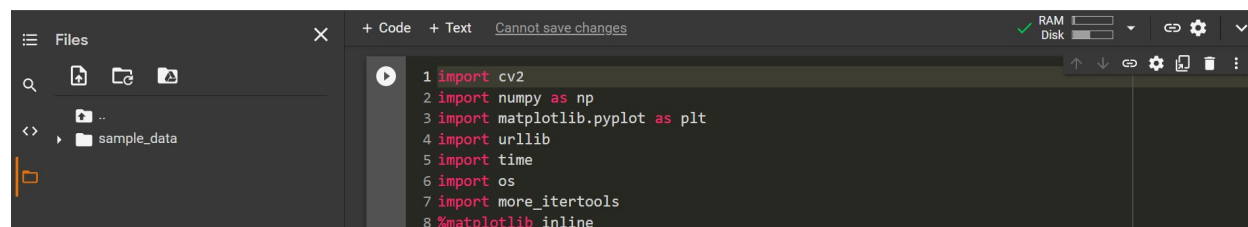


Figure 7: Okno programu Google Colaboratory

**Jupyter Notebook/Jupyter Lab** jest częścią projektu *Jupyter*, który wyewoluował z in-rearaktywnej wersji Pythona *IPython* w roku 2014. Jest to program o charakterze *non-profit* bazujący na bezpłatnej licencji. Środowisko znajduje szerokie zastosowanie w przetwarzaniu danych, obliczeniach numerycznych z wykorzystaniem różnych języków programowania. W odróżnieniu do klasycznego pisania kodu struktura podzielona jest na osobne sekcje zwane potocznie komórkami. Wykonywany program możemy uruchamiać poszczególnymi fragmentami. Po instalacji dodatkowych pakietów, możemy mieć podgląd do aktualnych wartości zmiennych, które są zdefiniowane w programie na zasadach podobnych do tego co oferuje nam pakiet MATLAB. Nowe elementy mogą przybierać również formę multimedialną w postaci zdjęć bądź równań matematycznych zapisywanych zgodnie ze składnią LaTeX.

**Google Colaboratory** jest chmurowym odpowiednikiem *Jupytera* dostarczonym przez firmę *Google*. Największą zaletą takiego rozwiązania jest brak konieczności instalowania dodatkowych bibliotek. Wszystkie potrzebne pakiety dostarczone są wraz ze środowiskiem. Program domyślnie używa najnowszych pakietów, dlatego jeżeli wymagamy konkretnej wersji biblioteki musimy to wcześniej zdefiniować. Należy mieć również na uwadze, że *Colab* zbudowany jest w oparciu o system Linux, dlatego aby sprawnie poruszać się po katalogach oraz zarządzać plikami należy posiadać podstawową wiedzę z zakresu pracy z *Terminalem*. Dodatkowo możemy cały projekt zsynchronizować z zewnętrznymi źródłami co daje możliwość obsługi plików z lokalnego poziomu komputera. W zależności czy zdecydujemy się na wykupienie wersji *Pro* czy zostaniemy na wariancie darmowym *Colab*

oferuje nam akcelerację GPU, co jest świetnym rozwiązaniem jeżeli mamy ograniczone możliwości sprzętowe. Niestety OpenCV w znacznym stopniu wykorzystuje bibliotekę *Qt* do tworzenia okienek w których znajdują się rezultaty programu, zatem z poziomu przeglądarki nie będziemy mieli do nich dostępu. Istnieją pewnie rozwiązania, które częściowo niwelują ten problem jest nie są na tyle wydajne, żeby w pełni zastąpić wersję stacjonarną. Biorąc pod uwagę wady i zalety takie rozwiązania najlepszym wyjściem okazuje się tutaj praca w trybie hybrydowym tj. część obliczeniową przeprowadzić w środowisku chmurowym, natomiast rezultaty przedstawić w oparciu o *Jupytera* w formie stacjonarnej.

## Podstawowa klasyfikacja obrazów

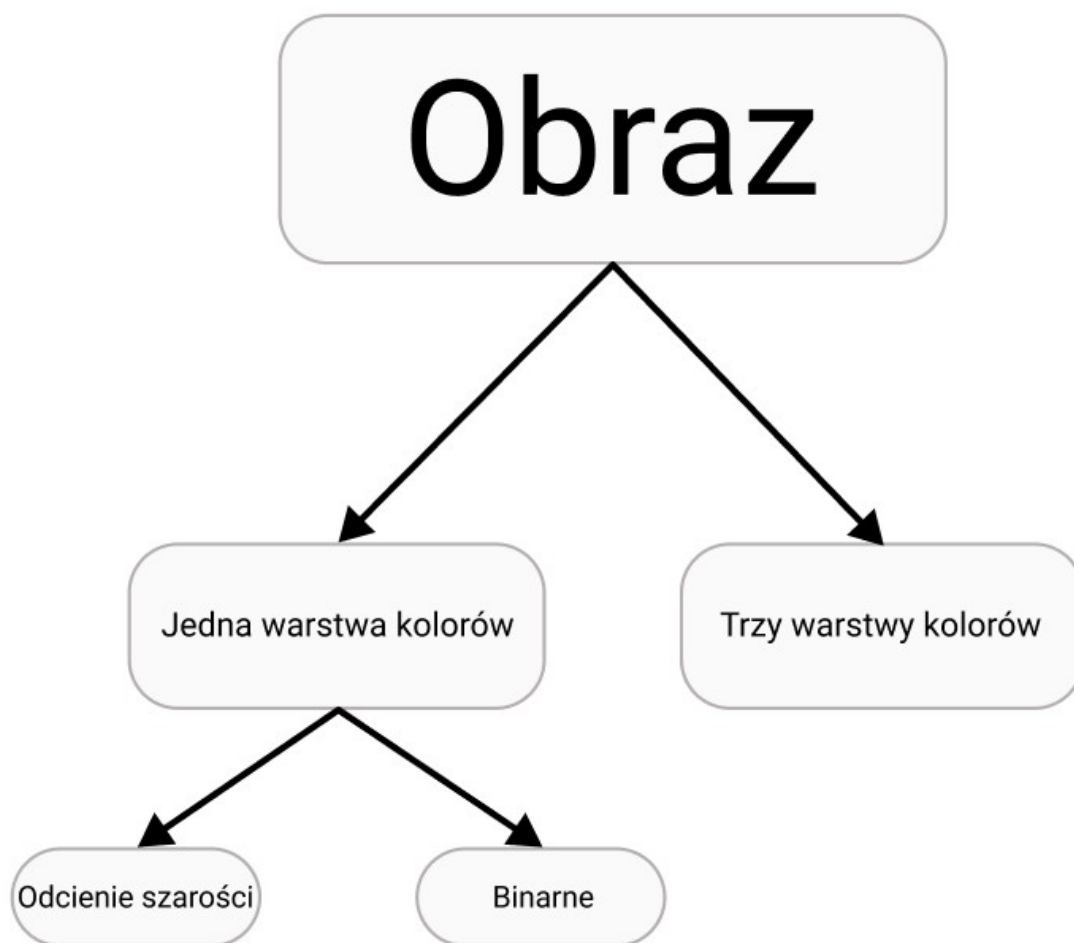


Figure 8: Klasyfikacja obrazów

Ze względu na matematyczną interpretację obrazu zostanie wprowadzona podstawowa klasyfikacja, która będzie odnośnikiem również do dalszej części pracy. W celu odwzorowania rzeczywistości wykorzystuje się wielowymiarowe struktury danych. Zależnie od źródła mogą się one nazywać *macierzami*, *tablicami*, *tensorami* bądź zwyczajnie *wektorami wielowymiarowymi*. Najprostrzym przypadkiem jest konstrukcja macierzy, która posiada pewną określoną liczbę wierszy oraz kolumn a jej strukturę możemy naszkicować na kartce papieru. Zdjęcie może przykazywać informacje tylko i wyłącznie, kiedy jego wnętrze wypełnione jest danymi. Wszystkie analizowane obiekty zostaną przedstawione w oparciu o rozszerzenie graficzne formatu **jpg**, które umożliwia wykorzystanie *8-bitowej* głębi kolorów. Oznacza to, że wnętrze macierzy może być wypełnione liczbami naturalnymi z przedziału od 0 do 255. Często w kolejnych etapach pracy wykonuje się operacje normalizowania tj. przeskalowania wszystkich liczb znajdujących się we wcześniej wspomnianym zakresie na przedział od 0 do 1. Takie podejście umożliwia przedstawienie zdjęcia tylko i wyłącznie w

odcieniu jednego koloru. W sytuacji gdy jest potrzeba zastosowania formatu wielokolorowego musimy rozszerzyć istniejącą warstwę o dwie dodatkowe i wykonać na każdej z nich kolejno mapowanie na kolor czerwony (R), zielony (G) oraz niebieski (B). Jeżeli pojedyncza wartość piksele może znajdować się w zakresie od 0 do 255 oraz uwzględnimy liczbę kanałów to otrzymujemy paletę barw zdolną do odwzorowania niemal *17 mln* kolorów. Ławtwa jednak zauważyć, że zdjęcie wielokolorowe zwiększa ilość dostarczanych informacji trzykrotnie co może nie być pożądane pod kątem wydajnościowym. Tutaj decyzja o formacie leży tylko i wyłącznie po stronie użytkownika, który musi określić czy identyfikacja kolorów pomoże mu w rozwiązaniu problemu czy jest wyłącznie dodatkowymi, zbędnymi informacjami. Analizując zdjęcie monochromatyczne, możemy przedstawić je w odcieniu jednego koloru (najczęściej głębia koloru szarego) lub zbinearyzować tj. uprościć obraz tylko i wyłącznie do jego warunków brzegowych (brak informacji lub jej obecność). Ta własność okaże się bardzo przydatna w kolejnych etapach przetwarzania.

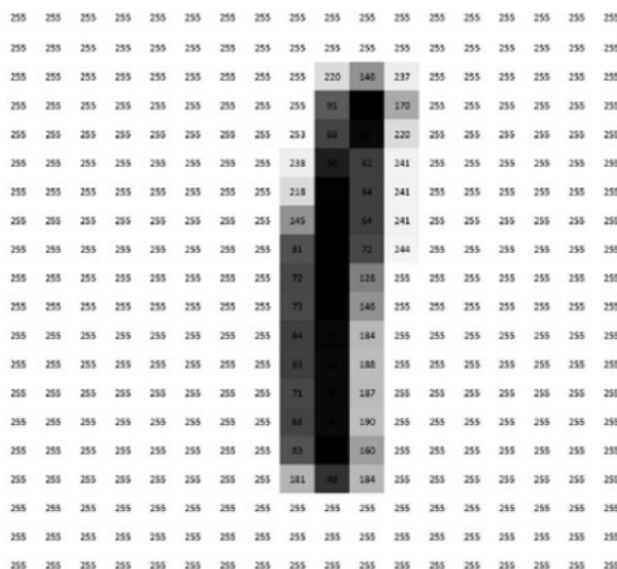


Figure 9: Obraz w różnych odcieniach szarości

255	255	255	255	255	255	255	255
255	255	255	255	0	0	255	255
255	0	0	0	255	0	0	255
255	0	255	255	255	255	0	255
255	0	255	255	255	255	0	255
255	0	0	0	0	0	0	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255

Figure 10: Zdjęcie w postaci binarnej

			14	31	44	96
		41	13	44	69	5
4	12	31	2	3	32	
123	76	3	45	65	3	
78	32	65	201	0		
1	43	21	0			

↖ Kanaly R,G,B

Figure 11: Sposób przedstawienia zdjęcia w formacie kolorowym

## Przetwarzanie obrazu

### Oczekiwany rezultat



Figure 12: Obiekt do przetworzenia

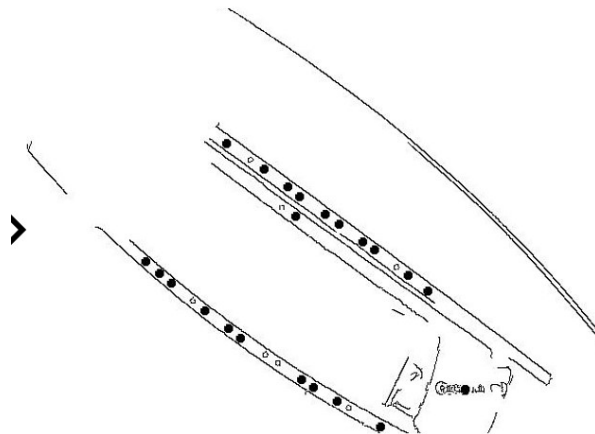


Figure 13: Efekt działania programu



Schemat blokowy toru przetwarzania

## Grayscale

Zdjęcie dostarczone przez kamerę w trybie domyślnym zawiera trzy kanały danych (R,G,B). W celu usprawnienia wykonywanych operacji oraz możemy wykonać transformacji na obraz w odcieniach szarości. Matematyczną implementację opisuje równanie (1). Korzystając z pakietu OpenCV operację tą możemy wykonać w oparciu o dwa dostarczone argumenty. Pierwszym z nich jest obraz, natomiast w drugim członie funkcji podanie zostany sposób konwersji, w tym przypadku z kanału kolorowego na odcienie szarości.

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

$$y = 0.299R + 0.587G + 0.114B \quad (1)$$

([1], n.d.)

## Blurring

Kolejnym krokiem jest usunięcie ze zdjęcia w jak największym stopniu, zakłóceń, które mogą w sposób inwazyjny zaburzać dalszą pracę. W tym celu wykonywana jest operacja wygładzania zdjęcia. Innym określeniem stosowanym do tego typu zabiegów jest również rozmazywanie. W ujęciu matematycznym możemy zdefiniować to jako proces filtracji. Wykonanie takiej operacji zostało przedstawione w postaci poniższego równania ([2], n.d.).

$$g(i, j) = \sum_{k,l} f(i + k, j + l)h(k, l) \quad (2)$$

Wyjściowy wektor  $g(i, j)$  otrzymywany jest poprzez sumę wartości pikseli w uwzględnieniem wag otrzymywanych przy zastosowaniu jądra. W równaniu został ono przedstawione w postaci  $h(k, l)$ . Filtr ten można zwizualizować przy pomocy okna zawierającego wartości współczynników. Możemy wyróżnić kilka rodzajów filtracji. Najprostszym podejściem jest utworzenie tablicy z wartości średnich w obrębie danego jądra (Richard Szeliski Springer 2010). Poniższa tożsamość ilustruje podstawową implementację.

$$K = \frac{1}{K_{\text{width}} \cdot K_{\text{height}}} \cdot \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \\ \cdot & \cdot & \cdot & \dots & 1 \\ \cdot & \cdot & \cdot & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix} \quad (3)$$

W przetwarzaniu obrazów najczęściej wykorzystujemy jednak giltrację w oparciu o okno Gaussa. Warto jednak w tym momencie wrócić uwagę, że rozwiązanie nie jest najkorzystniejsze pod względem wydajności, zatem jeżeli w kolejnych krokach zaobserwujemy spadek wydajności zmiana rodzaju okna może okazać się kluczową decyzją. Z racji, że obiekt na którym będziemy wykonywać operacje będzie posiadał dwa wymiary, poniższe równanie prezentuje opisywaną funkcję, natomiast poniżej widoczna jest jej graficzna interpretacja.

$$G_0(x, y) = Ae^{\frac{-(x-\mu_x)^2}{2\sigma_x^2} + \frac{-(y-\mu_y)^2}{2\sigma_y^2}} \quad (4)$$

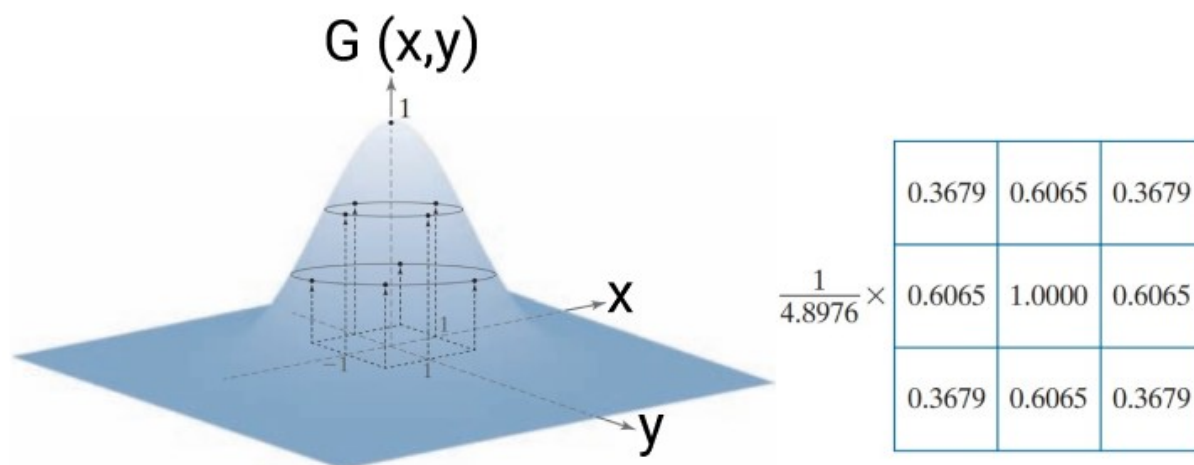


Figure 14: Kształt funkcji gaussa dla dwóch zmiennych parametrów wraz ze współczynnikami dla wymiarów okna 3x3 (Rafael C. Gonzalez, n.d.)

Implementacja przy użyciu dostarczeniu do funkcji trzech argumentów. Pierwszym z nich jest obraz powstający w wyniku poprzedniej operacji. Kolejno definiujemy rozmiar jądra w postaci listy statycznej. Opcjonalnym krokiem jest zdefiniowanie stanu, tak aby operacja była wykonywana na wartościach znormalizowanych. Domyślnie funkcja przybiera wartość *false*

```
img_blur = cv2.GaussianBlur(gray, (size, size), 1)
```

## Canny Edge Detection

Canny Edge Detection to jest z najbardziej rozpowszechnionych algorytmów służących do wyseparowania krawędzi z obrazu. Rozwiązanie zaproponował australijski informatyk John F. Canny w roku 1986. Jest to algorytm wieloetapowy, który wymaga kilku niezależnie związanych ze sobą operacji. Pierwszą z nich jest redukcja szumów, która została wykonana w poprzednim punkcie poprzez operację rozmazywania. Kolejnym krokiem jest przeprowadzenie obliczeń gradientowych wzdłuż osi x oraz y. W tym celu wykorzystywany jest algorytm Sobela. Matematycznie wykonaną operację przedstawia poniższy wzór.

$$EdgeGradient(G) = \sqrt{G_x^2 + G_y^2} \quad (5)$$

$$Angle(\theta) = \tan^{-1} \left( \frac{G_y}{G_x} \right) \quad (6)$$

Powyższą operację można przedstawić w sposób graficzny co prezentuje poniższa ilustracja. Składa się ona z trzech osobnych elementów. Na pierwszej grafice widnieje potencjalny element do wykrycia. Przeprowadzamy przez niego linię odniesienia, która będzie referencją w kolejnych krokach. Następnie liczony jest poziom nasycenia kolorów (grafika 2). Końcowym etapem są obliczenia gradientowe. W zależności od wyboru poziomu wyzwania otrzymujemy maksymalną oraz minimalną wartość pochodnej bo wskazuje, że wzdłuż badanego obszaru znajduje się obiekt, który chcieliśmy wykryć.

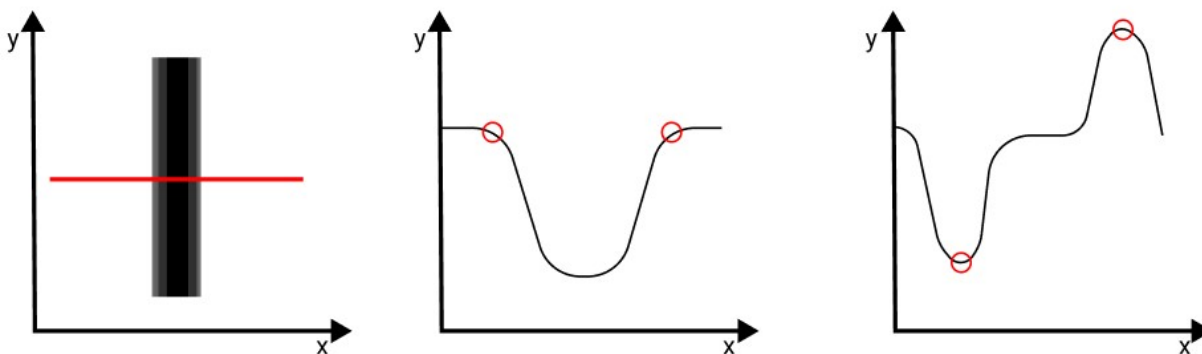


Figure 15: Obliczenia gradientowe

## Dilation

Dilation jest operacją, która ma na celu zwiększenie objętości wyseparowanych krawędzi. W tym przypadku jest to niezbędny proces gdyż zdjęcie jest o stosunkowo niewielkiej rozdzielczości co przekłada się na niewielką powierzchnię krawędzi. Dodatkowo taki proces jest potrzebny, ponieważ kontury uzyskane w wyniku poprzedniej operacji często nie tworzą obszarów o strukturze zamkniętej, co w dalszym procesie uniemożliwia zastosowanie filtrowania w oparciu o wymieniony parametr. Dodatkowo potrzeba określić dwa parametry zastosowanej funkcji, pierwszym z nich jest wejściowy wektor macierzy w postaci binarnej natomiast drugim rozmiar jądra. W tym momencie należy mieć na uwadze, że czym większy jego wymiar tym krawędzie bardziej zyskają na objętości. Należy jednak dobierać uważnie jego wartość, ponieważ można doprowadzić do sytuacji gdzie kontury zaczną tworzyć jeden większy co nie jest zjawiskiem porządnym. Poniższe równanie opisuje zastosowaną zależność.

$$\text{dst}(x, y) = \max_{(x', y') : \text{element}(x', y') \neq 0} \text{src}(x + x', y + y') \quad (7)$$

Z praktycznego punktu widzenia oznacza to, że jeżeli przynajmniej jeden element oznaczający wartość 1 z macierzy wejściowej pokryje się z elementem 1 zastosowanego jądra na wyjściu otrzymujemy 1. Następnie operację powtarzamy przesuwając wektor jądra w taki sposób aby przejść przez całą macierz wyjściową. Modyfikować możemy również liczbę iteracji.

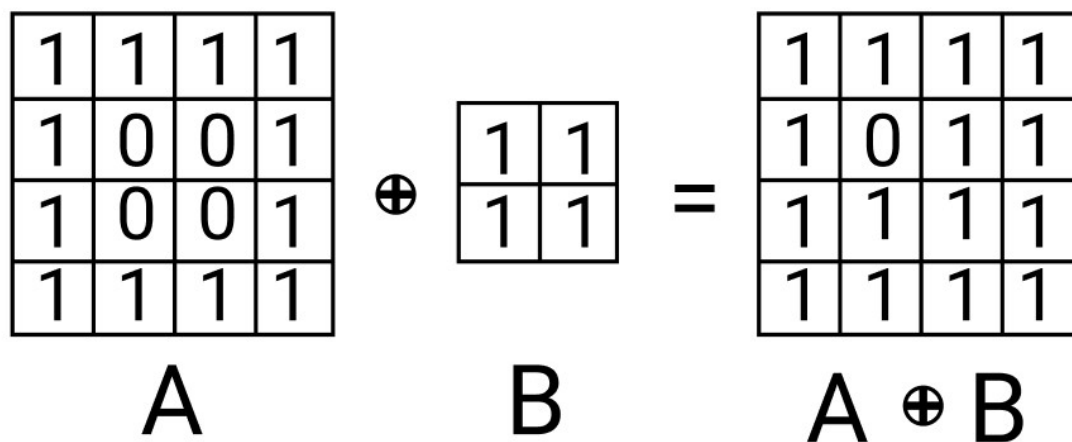


Figure 16: Przykładowy przebieg procesu

```
#Dilation process  
dsize = cv2.getTrackbarPos("Dilation_kernel", "Parameters")  
imgDil = cv2.dilate(edges, np.ones((dsize, dsize)), iterations=1)
```

## Threshold

Kolejnym etapem jest **Thresholding**, który polega na ustaleniu wartości granicznych dla których zdjęcie powinno zostać wyprogowane. Proces ten można nazwać również binaryzacją. Przetwarzanie polega na aplikacji dla każdego piksela z obrazu funkcji, która w zależności od wartości koloru piksela ustala jego wartość na 0 (kolor czarny) lub 255 (kolor biały). Należy jednak pamiętać że musimy dysponować obrazem przedstawionym w formacie różnych odcieni szarości, aby proces został przeprowadzony poprawnie. Poniżej interpretacja matematyczna oraz przykład użycia funkcji na której wejście należy podać cztery argumenty. Pierwszym z nich jest macierz obrazu. Kolejne dwa ostalają parametry progowania. Jeżeli wartość numeryczna piksela będzie większa niż drugi argument, na wyjściu otrzymamy wartość 255 w przeciwnym wypadku będzie to 0. W zależności od zapotrzebowania możemy zastosować różny algorytm, poniższa grafika prezentuje 5 podstawowych, jednak najważniejsze z nich w kontekście projektu to *BINARY* oraz *BINARY\_INV*.

$$\text{dst}(x, y) = \begin{cases} 0 & \text{if } \text{src}(x, y) > \text{thresh} \\ \text{maxval} & \text{otherwise} \end{cases} \quad (8)$$

```
#Color inverse
```

```
ret,thresh1 = cv2.threshold(imgDil,100,255,cv2.THRESH_BINARY_INV)
```

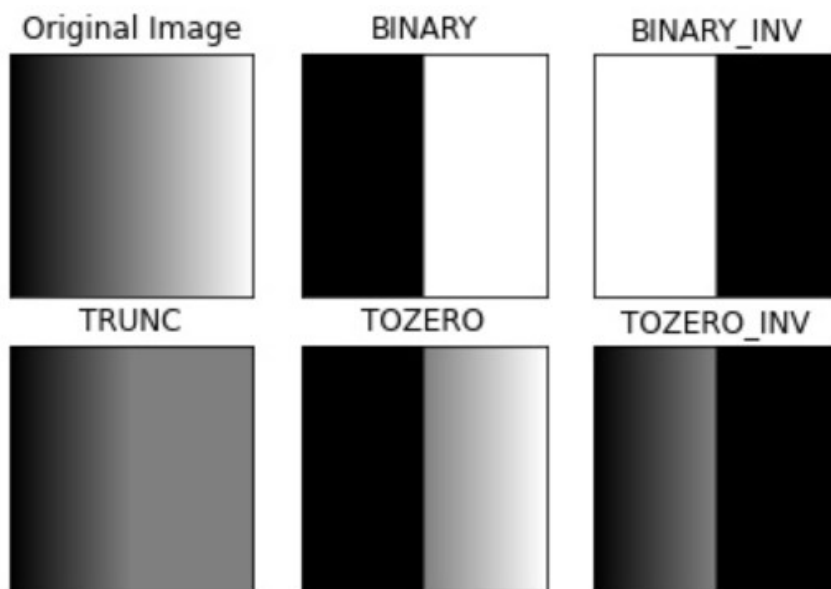


Figure 17: Podstawowe algorytmy progowania

## Bibliografia

- [1]. n.d. “[1] Image Color Conversions.” [https://docs.opencv.org/3.4.15/de/d25/imgproc\\_color\\_conversions.html](https://docs.opencv.org/3.4.15/de/d25/imgproc_color_conversions.html).
- [2]. n.d. “[2] Tutorial\_gaussian\_median\_blur\_bilateral\_filter.” <https://docs.opencv.org/4.5.3/dc/dd3/.html>.
- Rafael C. Gonzalez, Richard E. Woods [3]. n.d. “[3] Digital Image Processing 4th.”
- Richard Szeliski, [4]. Springer 2010. “[4] Computer Vision: Algorithms and Applications.”