**Analiza i Przetwarzanie Obrazów Biometrycznych**

**Sprawozdanie 2**

Michalik Piotr

Spis treści

[Wstęp 3](#_Toc500201990)

[Treshold 3](#_Toc500201991)

[Dilation 4](#_Toc500201992)

[Erosion 6](#_Toc500201993)

[Opening 7](#_Toc500201994)

[Closing 8](#_Toc500201995)

[Get Iris 9](#_Toc500201996)

[Pozostałe 12](#_Toc500201997)

[Wnioski 13](#_Toc500201998)

[Źródła 13](#_Toc500201999)

# Wstęp

W poniższym dokumencie przedstawię środki użyte do realizacji drugiego etapu projektu. Po kolei opisze i przedstawię metody zaimplementowane w tym projekcie. Następnie skupie się na części właściwej, to znaczy na wydzielaniu tęczówki z oka. Na samym końcu zamieszczę krótkie podsumowanie etapu. Dla każdej opisanej metody zostaną zamieszczone screenshoty prezentujące działanie poszczególnej funkcjonalności.

# Treshold

## Teoria

Każdy kolorowy obraz zawierający trzy kolory składowe (czerwony, zielony, niebieski) można przestawić w odcieniach szarości. To znaczy że każdy piksel będzie posiadał taką samą wartość koloru w miejscu pojedynczych składowych (czerwony = zielony = niebieski).

Wartość nowego koloru (szarości) uzyskujemy według wzoru

Obraz przedstawiony w skali szarości można następnie łatwo zamienić na obraz zbinaryzowany to znaczy taki który posiada pixele w dwóch wartościach : białe lub czarne. Operacje zamiany zdjęcia z czarno-białego na zbinaryzowane możemy nazwać tresholdingiem. Operacja taka polega na wyznaczeniu pewnego progu T, który przyjmuje wartości takie same jak każdy kolor [0 ;256]. Następnie każdy pixele poniżej tego progu malujemy na biało, a pozostałe na czarno.

## Implementacja

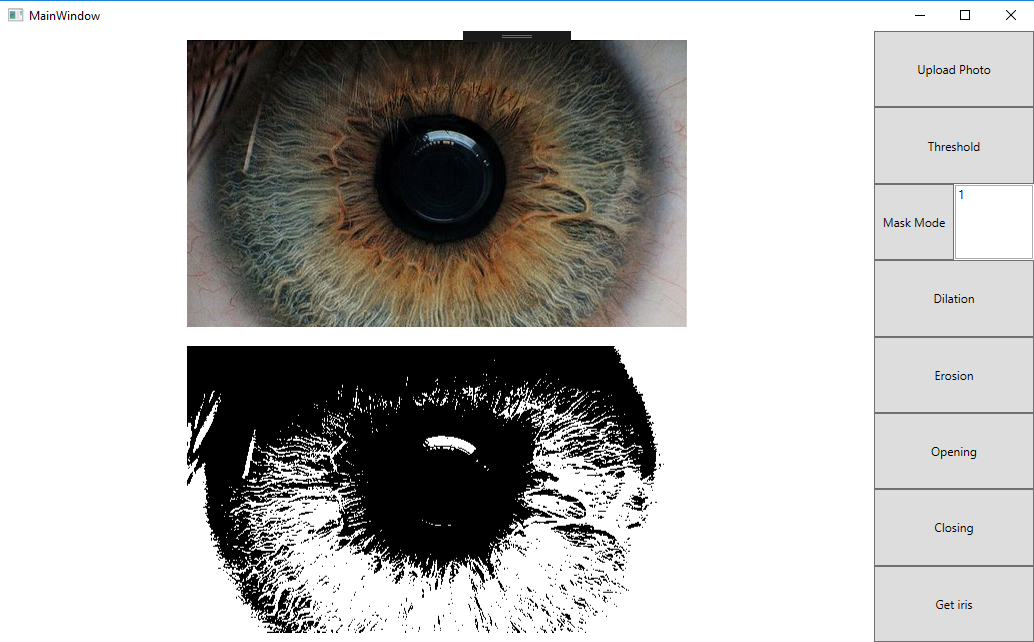
Program po wczytaniu zdjęcia, przechodzi po wszystkich jego pixelach. Następnie zgodnie z powyższym wzorem zamienia piksele kolorowe na czarno-białe. Jednocześnie sumuje wszystkie dotychczasowe wartości kolorów. Na końcu wartość ta zostaje podzielona przez liczbę wszystkich pixeli, dając nam poszukiwany treshold.

Gdzie:

* A(a,b) – pixel na pozycji (a,b) na czarnobiałym obrazie A
* h – wysokość obrazu A
* w – szerokość obrazu A

Uzyskawszy treshold program ponownie przechodzi po wszystkich pixelach obrazu i zamienia je na białe lub czarne z godnie z opisem teoretycznym.

## ScreenShot



Rysunek - 'Treshold’

# Dilation

## Teoria

W dylacji jak i kolejnych metodach będziemy mówili o tzw. operacjach morfologicznych. Operacje takie są łatwe do przeprowadzania na obrazach binarnych gdzie obraz można przedstawić jako macierz zer i jedynek. Zanim wyjaśnię na czym polegają poszczególne operacje, trzeba zapoznać się z pojęciem maski.

Maską nazywamy macierz kwadratową (w tym projekcie o wymiarach 3x3) która reprezentuje pewien sposób ustawienia pixeli. Każda komórka maski może przyjmować wartość 0 lub 1. Przykładowe maski mogą wyglądać następująco:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | X |  |
|  |  |  |

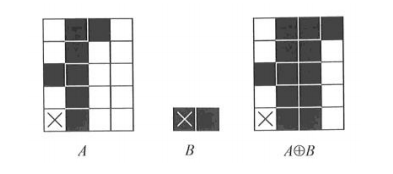
|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | X |  |
|  |  |  |

Jako ‘X’ oznaczamy pixel, dla którego podejmowane będą operacje morfologiczne w danej masce. Pierwsza maska oznacza taki wzór, w którym nad, pod, na lewo oraz na prawo od zaznaczonego pixela występują pixele czarne. Natomiast pixele po przekątnych są białe.

Dylacje samą w sobie można opisać następująco:

Oznacza to, że dla obrazu A oraz maski B, jeżeli środek *b* z maski B można odnaleźć w środku fragmentu *a* obrazu A, to wynikowy obraz jest sumą dwóch powyższych.

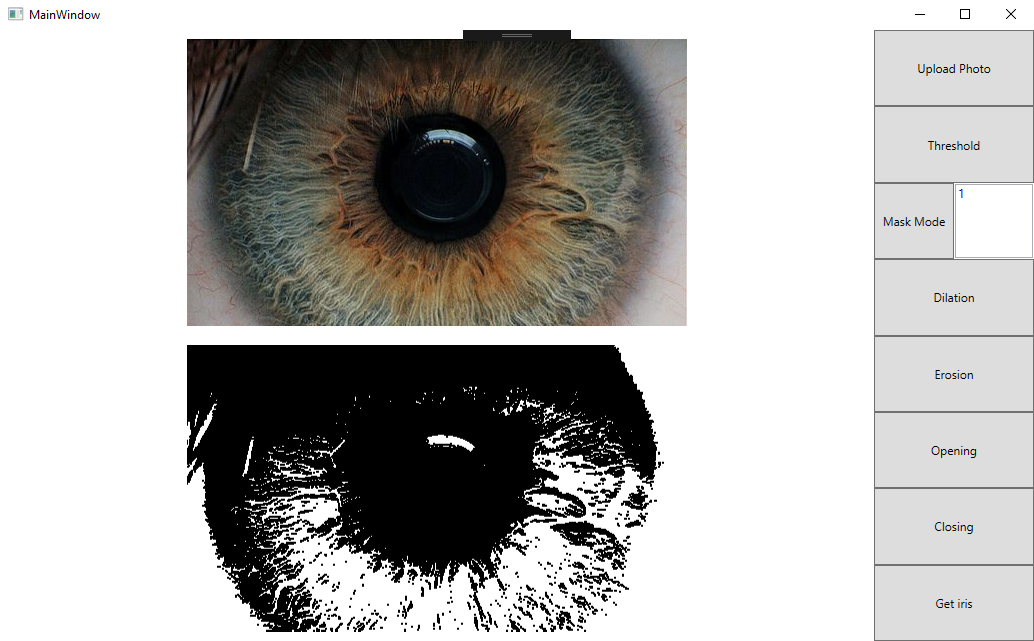
Przykład:



## Implementacja

Dla ułatwienia, w programie użyto założenia że środek maski jest zawsze elementem oznaczonym jako ‘X’. Program przechodzi po całym obrazie. Dla każdego pixela znajduje jego sąsiedztwo o promieniu = 1 ( tak że powstaje nam macierz 3x3 odpowiadająca masce). Jeżeli środek sąsiedztwa zgadza się ze środkiem maski, to kolorujemy na czarno piksele fragmentu oznaczone w masce jako czarne. Operacja kolorowania odbywa się na kopii obrazu, podczas gdy porównanie fragmentów na obrazie oryginalnym.

## ScreenShot



Rysunek - 'Dilation'

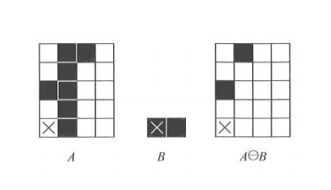
# Erosion

## Teoria

Erozję, jako kolejną operację morfologiczną możemy opisać następująco:

Oznacza to, że dla obrazu A oraz maski B element *a* pozostaje czarny jeżeli całość maski B jest zawarta w fragmencie A, w przeciwnym wypadku *a* zmienia wartość na białą.

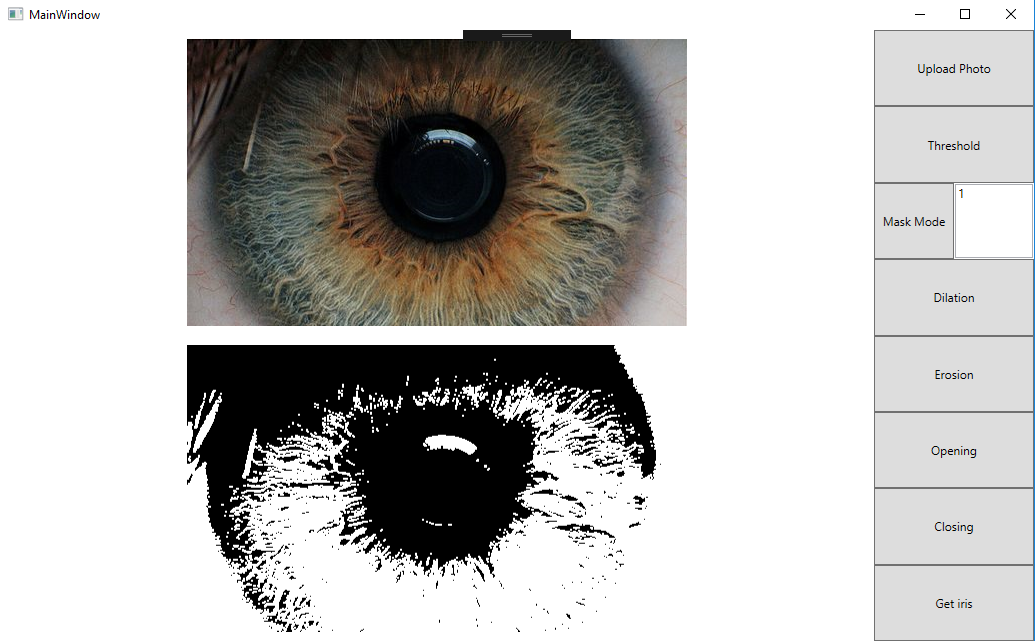
Przykład:



## Implementacja

Mechanizm działa analogicznie jak dla Dilation, z róznicą w zasadach zmiany koloru. Program przechodzi po całym obrazie. Dla każdego pixela znajduje jego sąsiedztwo o promieniu = 1 ( tak że powstaje nam macierz 3x3 odpowiadająca masce). Jeżeli środek sąsiedztwa zgadza się ze środkiem maski, to sprawdzamy czy czarne pixele maski są również czarne w wybranym fragmencie. Jeżeli tak to pixel zachowuje swoją wartość. W przeciwnym wypadku kolorujemy go na biało.

## ScreenShot



Rysunek - 'Erosion'

# Opening

## Teoria

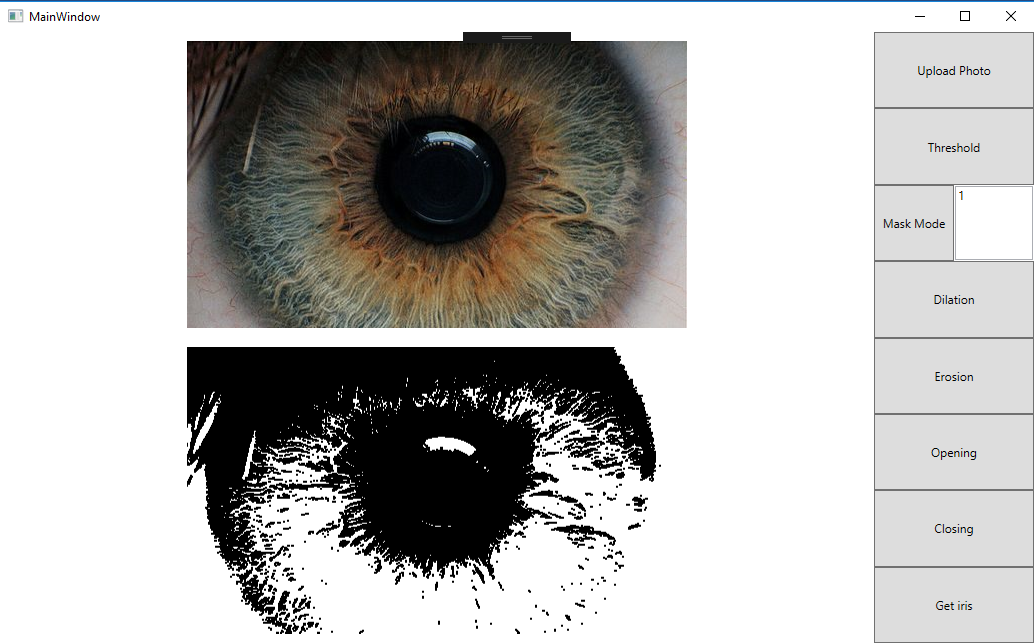
Operacje morfologiczne na zdjęciach można łączyć, stosować naprzemiennie czy wielokrotnie. Mogło by się wydawać że operacja Erozji, a następnie Dylacji pozwoli wrócić do oryginalnego obrazu. Jednak nie dość że jest kompletnie na odwrót, to jeszcze rezultat będzie różny w zależności od tego której z opisanych powyżej operacji użyjemy jako pierwszej.

Opening – czyli otwarcie – jest operacja powstałą w skutek połączenia ze sobą najpierw erozji a następnie dylacji. Zatem opening możemy opisać następująco :

## Implementacja

Operacja Opening jest zaimplementowana jako kolejne wywołanie metod Erosion i Dilation na tym samym binarnym obrazie.

## ScreenShot



Rysunek - 'Opening'

# Closing

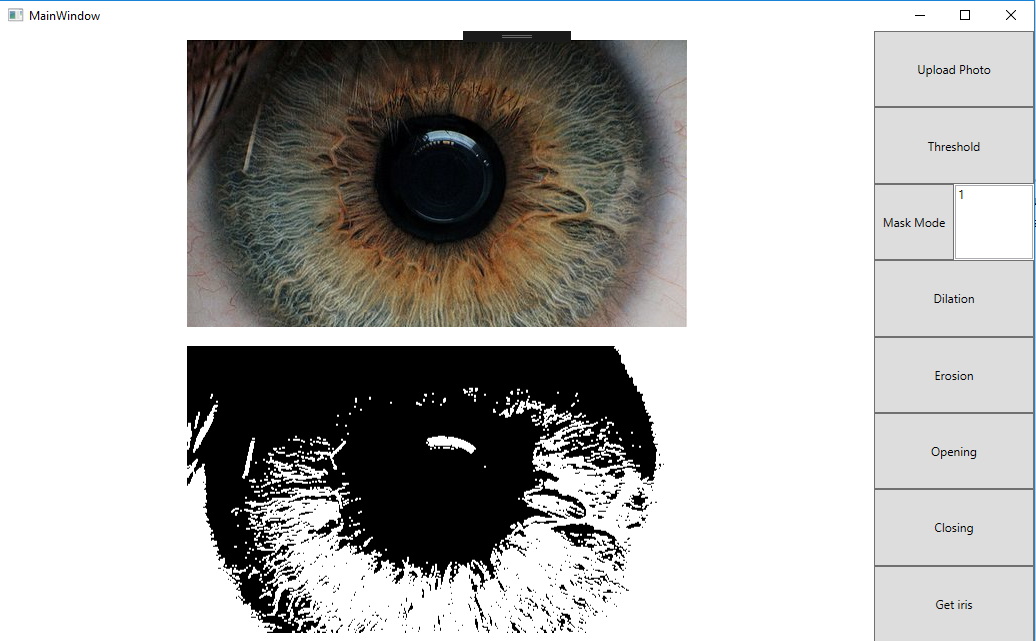
## Teoria

Nawiązując do opisu poprzedniej metody, zdefiniujmy kolejne połączenie dwóch operacji morfologicznych. Closing – czyli domknięcie – jest operacja powstałą w skutek połączenia ze sobą najpierw dylacji a następnie erozji. Zatem closing możemy opisać następująco :

## Implementacja

Operacja Opening jest zaimplementowana jako kolejne wywołanie metod Dilation i Erosion na tym samym binarnym obrazie.

## ScreenShot



Rysunek - 'Closing'

# Get Iris

## Teoria

Problem znalezienia tęczówki w biometrii jest szeroko znany i często używany. Również w powyżej opisywanym projekcie, zaimplementowałem szereg metod mających na celu odseparowanie tęczówki ze zdjęcia oka. Posłużyłem się tutaj między innymi wieloma operacjami morfologicznymi opisanymi wcześniej.

Sam proces znalezienia tęczówki podzieliłem na dwa oddzielne podprocesy które są do siebie bardzo podobne z drobnymi szczegółami: proces znalezienia źrenicy oraz proces znalezienia tęczówki. Każdy z tych procesów miał na cel ustalenie środka oraz promienia kolejno: źrenicy (FindPupil) i tęczówki (FindIris).

## FindPupil

Zacznijmy od znalezienia źrenicy. Opiszę teraz po kolei operacje które użyłem w tym procesie (Operacje opisane w poprzednim sprawozdaniu oznaczę w skrócie znakiem – ‘OWPS’):

1. **Przekształcenie zdjęcia do skali szarości** – OWPS
2. **Normalizacja histogramu** – OWPS
3. **Erozja** –   
   Ponieważ do Erozji potrzeba obrazu binarnego, binaryzujemy szary obraz. Treshold jaki jest do tego użyty ma wartość:  
   gdzie - jest użytym tresholdem, a *T* tresholdem opisanym powyżej. Następnie na uzyskanym obrazie stosujemy opisane wcześniej Erosion.
4. **Usunięcie ramek** –   
   Z racji tego że chcielibyśmy mieć jak najczystszy obraz, rozpoczynamy teraz usuwanie zbędnych elementów. Doświadczalnie zauważyłem, że usunięcie ramki o szerokości 2 pixeli dookoła obrazu, może polepszyć wyniki znajdowania źrenicy.
5. **Usuwanie szumów** –   
   Odszumianie jest kolejnym krokiem oczyszczania obrazu. W tym wypadku program usuwa każdy pixel który ma mniej niż 3 sąsiadów. W ten sposób pozbywamy się pojedynczych ugrupowań pixeli które mogą przeszkadzać w kolejnych krokach. Operacja jest powtarzana tak długo, aż w kolejnej iteracji nie zostanie usunięty ani jeden pixel.
6. **Dilation** –   
   W celu wzmocnienia naszej źrenicy stosujemy Dilation.
7. **Usunięcie ramek** –   
   Po raz kolejny usuwamy potencjalnie powstałe w wyniku dylacji ramki.
8. **Wypełnienie źrenicy** –   
   Często podczas robienia zdjęcia oka, w źrenicy pojawia się refleks światła. Aby późniejsze wykrywanie krawędzi nie potraktowało refleksu jako źrenicy należy go usunąć. W tym celu używam algorytmu Flood Fill. Ponieważ nie mamy krawędzi, zaczynamy w górnym lewym rogu i oznaczamy każdy napotkany biały pixel. Ponieważ biały refleks światła jest oddzielony źrenicą od krawędzi zdjęcia, nie zostanie ani on ani źrenica oznaczony jako biały. Następnie każdy oznaczony pixel zostawiamy białym, a pozostałe zamieniamy na czarny kolor. W ten sposób źrenica cała zostaje pokolorowana na czarno.
9. **Wykrywanie krawędzi** –   
   Ponieważ chcemy określić środek źrenicy i jej promień, musimy znaleźć jej kształt. To znalezienia kształtu będzie nam potrzebna krawędź którą znajdujemy opisanym w poprzednim sprawozdaniu Edge Detection z filtrem Sobela.
10. **Znalezienie Okręgu** –   
    Kiedy wreszcie mamy na naszym zdjęciu wykrytą krawędź źrenicy musimy określić jej środek i promień. Aby uniknąć problematycznego analizowania histogramu zdecydowałem się użyć algorytmu zwanego Circle Hough Transform. Polega on na symulowaniu okręgów z każdego możliwego miejsca zdjęcia i o każdym możliwym promieniu. Miejsce gdzie jest krawędź symulowanego okręgu zyskuje punkt w specjalnie przygotowanej trzy-wymiarowej tablicy. Nasz algorytm używamy tylko na pixelach które zostały oznaczone w poprzednim punkcie jako krawędź. Dzięki temu miejsca z największą ilością punktów są środkami okręgów. Następnie z współrzędnych tablicy z punktami możemy wyczytać współrzędne i promień każdego wykrytego okręgu. Na samym końcu wystarczy znaleźć ten okrąg który ma największy promień.

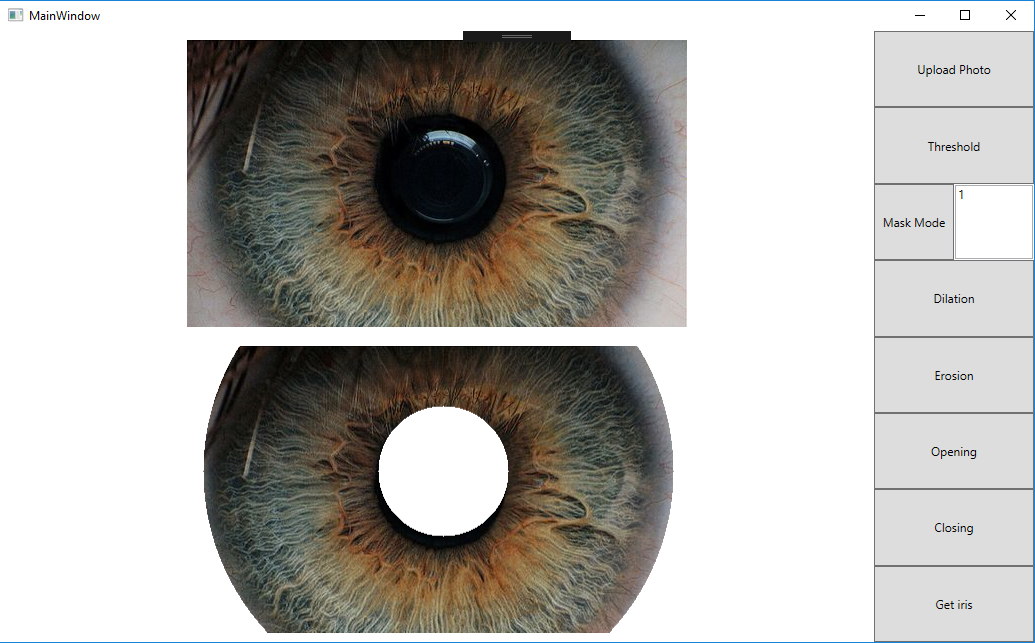
Jako rezultat tej sekwencji, otrzymujemy współrzędne środka źrenicy oraz jej promień.

## FindIris

W kolejnym kroku szukamy zewnętrznej krawędzi tęczówki:

1. **Przekształcenie zdjęcia do skali szarości** – OWPS
2. **Normalizacja histogramu** – OWPS
3. **Erozja** –   
   Treshold jaki jest użyty tym razem ma wartość:  
   gdzie - jest użytym tresholdem, a *T* tresholdem opisanym wcześniej. Następnie na uzyskanym obrazie stosujemy opisane wcześniej Erosion.
4. **Usunięcie ramek**
5. **Usuwanie szumów**
6. **Dilation** –   
   W celu wzmocnienia naszej źrenicy stosujemy Dilation dwa razy aby tęczówka była jeszcze wyraźniejsza.
7. **Usunięcie ramek**
8. **Usunięcie źrenicy** –   
   Aby oddzielić źrenice która jest zawarta w naszej tęczówce, posługujemy się wynikami poprzedniej sekwencji. Pixele które znajdują się w odległości mniejszej niż promień od środka źrenicy, zostają pokolorowane na biało.
9. **Wykrywanie krawędzi**
10. **Usunięcie ramek**
11. **Usuwanie szumów**
12. **Znalezienie Okręgu** –   
    Ponownie wyszukujemy okręgu na przygotowanym obrazie. Ponieważ wiemy, że środek tęczówki powinien się ‘mniej więcej’ pokrywać ze środkiem źrenicy, poszukiwania środka okręgu zawężamy do znanego środka źrenicy +/- 0.2 promienia źrenicy.
13. **Finalize** –   
    Na zakończenie, wszystkie pixele które są od środka tęczówki w odległości większej niż promień tęczówki, zostają pokolorowane na biało.

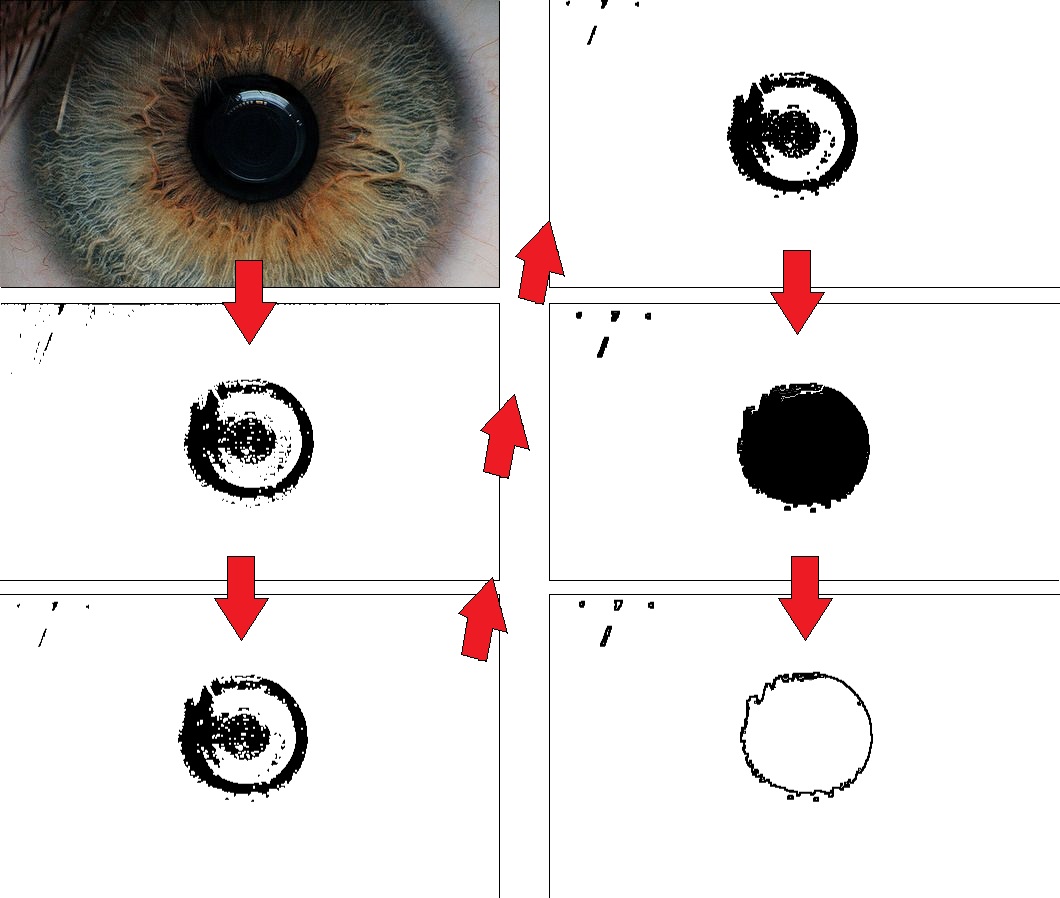
## ScreenShot



Rysunek - 'Get Iris'

# Pozostałe

## Przekształcenia obrazów w GetPupil oraz GetIris



# Wnioski

Aplikacja spełnia swoją funkcjonalność. Jednak ma ona wiele wad i niedociągnięć. Przede wszystkim aplikacja bardzo zależy od tego jakie tresholdy zostaną użyte do oddzielania tęczówki i źrenicy. Przecież dla takiego samego zdjęcia. Jeżeli tylko podmienimy błękitny kolor tęczówki na brązowy, uzyskany treshold będzie diametralnie inny. I wtedy cała reszta algorytmu jest kontynuowana na źle zbinaryzowanym zdjęciu.

Moim zdaniem, gdyby opracować jakąś metoda która pozwoliłaby subiektywnie obliczyć poziom tresoldu (np. w oparciu o wiele czynników takich jak całkowita jasność obrazu), można by przeprojektować cały algorytm tak aby był z jednej strony prostszy a z drugiej skuteczniejszy.

# Źródła

<https://en.wikipedia.org/wiki/Thresholding_(image_processing)>

<https://pl.wikipedia.org/wiki/Cyfrowe_przetwarzanie_obraz%C3%B3w_binarnych>

<https://en.wikipedia.org/wiki/Circle_Hough_Transform>

<http://www.mini.pw.edu.pl/~rafalkoj/www/?download=Project%202%20-%20requirements.pdf>

<http://www.mini.pw.edu.pl/~rafalkoj/www/?download=Morphological%20operations.pdf>