**Analiza i Przetwarzanie Obrazów Biometrycznych**

**Sprawozdanie 3**

Michalik Piotr

Spis treści

[Wstęp 3](#_Toc503826854)

[KMM 4](#_Toc503826855)

[K3M 9](#_Toc503826856)

[Porównanie metod 13](#_Toc503826858)

[Listy 14](#_Toc503826859)

[Źródła 14](#_Toc503826860)

# Wstęp

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

W tym sprawozdaniu opisane, zaprezentowane i porównane zostaną dwa algorytmy: KMM oraz K3M. Oba te algorytmy służą do przeprowadzenia operacji ścieniania obrazu.

Przed użyciem każdego z algorytmów, na początku przygotowujemy już binarne zdjęcie w celu uzyskania lepszych wyników. Tak zwany preprocessing obejmuje tutaj:

* dwukrotne zwiększenie kontrastu
* usunięcie pixeli stykających się z krawędzią obrazu
* operacja otwarcia
* usunięcie pojedynczych pixeli (tzw. szumów)

# KMM

## Teoria

Pierwszym algorytmem zaproponowanym do problemu ścieniania jest KMM. Algorytm ten ma za zadanie dla zadanego czarnobiałego obrazu sprawić, aby wszystkie linie sprowadzić do grubości jednego pixela. Jednocześnie kolejnym zadaniem KMM’a jest zachowanie odpowiednich kształtów obrazu.

Jako obraz wejściowy do algorytmu przyjmujemy mapę bitową. W tym celu oryginalny obraz binaryzujemy metodami opisanymi w poprzednich sprawozdaniach. Cechą obrazu binarnego jest to że wartości pixeli [0;1] odpowiadają mapie bitowej.

Zdefiniujmy sobie sąsiedztwo pixela. Jako sąsiedztwo rozumiemy 8 pikseli które stykają się bezpośrednio z analizowanym pixelem.

Algorytm prezentuje się następująco:

1. Każdą jedynkę która sąsiaduje z zerem zamieniamy na dwójkę.
2. Każde nie zero, które nad sobą, pod sobą, z lewej oraz z prawej mają pixel oznaczony zerem nazywamy pixelem narożnym oraz oznaczamy go trójką.
3. Każde nie zero które ma 2,3 lub 4 przylegających do siebie sąsiadów zamieniamy na czwórkę.
4. Wszystkie pixele oznaczone jako 4 zamieniamy na 0.
5. Jeżeli dwójki i trójki są potrzebne do zachowania ciągłości linii, zaznaczamy je jedynką, a pozostałe zamieniamy na zera.

Następnie punkty od 1 do 5 powtarzamy tak długo, aż w jednym cyklu nie zostanie usunięty ani jeden pixel.

Kolejnym wartym omówienia etapem jest definiowanie czy pixel jest niezbędny aby zachować ciągłość linii. Patrząc na najbliższe sąsiedztwo (3x3) możemy zauważyć że mamy 24 możliwości w których środkowy pixel jest zbędny do zachowania ciągłości.

Przydzielmy każdemu sąsiadującemu pixelowi wagę według wzoru:

|  |  |  |
| --- | --- | --- |
| 128 | 1 | 2 |
| 64 | X | 4 |
| 32 | 16 | 8 |

Zauważmy że pełne sąsiedztwo ( wszystkie pixele obecne ) daje nam sume 255, a brak jakiegokolwiek pixela da nam sumę zero. Dodatkowo z właściwości wielokrotności dwójki mamy zagwarantowane że nie da się jednej sumy pixeli uzyskać na więcej niż jeden sposób, dzięki temu każda suma mówi nam które pixele są obecne.

Algorytm przewiduje listę takich sum, które po przeanalizowaniu ukazują takie kombinacje pixeli które gwarantują zachowanie ciągłości linii. Oto następująca lista:

3, 5, 7, 12, 13, 14, 15, 20, 21, 22, 23, 28, 29, 30, 31, 48, 52, 53, 54, 55, 56, 60, 61, 62, 63, 65, 67, 69, 71, 77, 79, 80, 81, 83, 84, 85, 86, 87, 88, 89, 91, 92, 93, 94, 95, 97, 99, 101, 103, 109, 111, 112, 113, 115, 116, 117, 118, 119, 120, 121, 123, 124, 125, 126, 127, 131, 133, 135, 141, 143, 149, 151, 157, 159, 181, 183, 189, 191, 192, 193, 195, 197, 199, 205, 207, 208, 209, 211, 212, 213, 214, 215, 216, 217, 219, 220, 221, 222, 223, 224, 225, 227, 229, 231, 237, 239, 240, 241, 243, 244, 245, 246, 247, 248, 249, 251, 252, 253, 254, 255

## Implementacja

Algorytm został zaimplementowany dosłownie tak jak w opisie teoretycznym. Dlatego poniższy opis odnosi się do poszczególnych punktów.

1. W podwójnej pętli przechodzimy po obrazie. Dla każdego pixela który nie jest zerem sprawdzamy czy chociaż jeden jego sąsiad jest zerem. Jeżeli tak, zamieniamy jedynkę na dwójkę.
2. W podwójnej pętli przechodzimy po obrazie. Dla każdego pixela który nie jest zerem sprawdzamy czy: nad, pod, z lewej oraz z prawej znajdują się wartości różne od zera. Jeżeli tak, to sprawdzamy czy któryś z przekątnych pixeli jest równy zeru. Jeżeli tak to wartość pixela zmieniamy na trzy.
3. Stworzona została lista wartości, które otrzymane z sumowania wcześniej opisanych wag gwarantują istnienie 2,3 lub 4 sąsiadów stykających się ze sobą. W podwójnej pętli przechodzimy po obrazie. Dla każdego pixela który nie jest zerem sprawdzamy czy suma wag pixeli sąsiednich znajduje się w liście. Jeżeli tak to zamieniamy wartość pixela na cztery.
4. W podwójnej pętli przechodzimy po obrazie. Dla każdego pixela który ma wartość 4 zamieniamy na 0.
5. W podwójnej pętli przechodzimy po obrazie. Dla każdego pixela o wartości N = 2 liczymy wagę jego sąsiadów. Jeżeli suma wag znajduje się w liście określającej ciągłość linii, zapamiętujemy że dany pixel należy usunąć. Po oznaczeniu wszystkich pasujących pixeli, zamieniamy ich wartość na 0. Całość następnie powtarzamy dla N = 3.

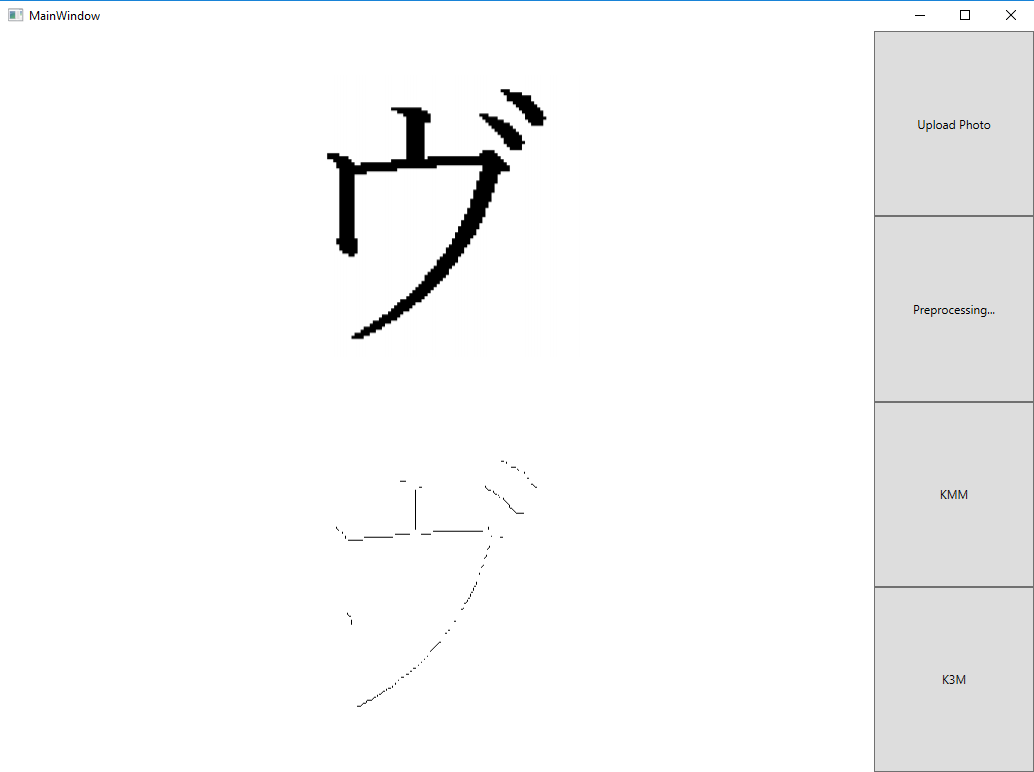
## ScreenShot



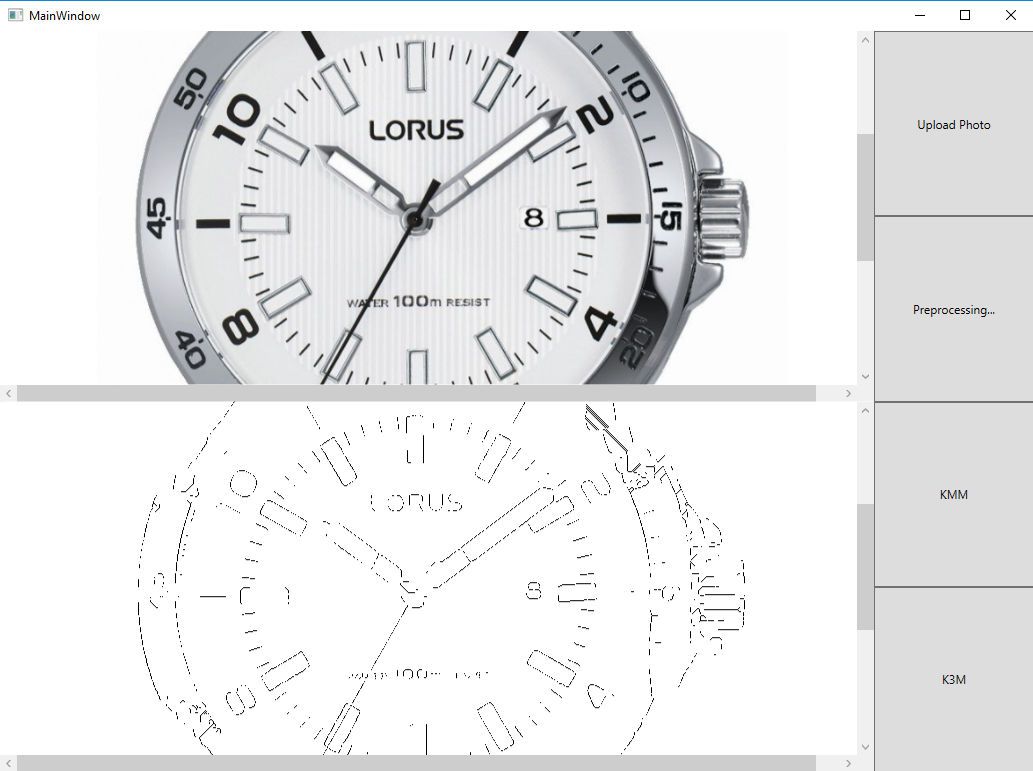
Rysunek 1- Przykład Preprocessingu



Rysunek 2- KMM pierwszy przykład



Rysunek 3- KMM drugi przykład



Rysunek 4- KMM trzeci przykład



Rysunek 5- KMM czwarty przykład

# K3M

## Teoria

Drugim algorytmem zaproponowanym do problemu ścieniania jest K3M. Algorytm został opracowany przez Panów: Khalida Saeed, Mariusza Rybnik, Marka Tabędzki oraz Marcina Adamski jako odpowiedź na niewystarczającą skuteczność algorytmu KMM. Jest to poniekąd jego rozwinięcie oraz udoskonalenie.

Idea K3M opiera się na eliminacji zbędnych pixeli w kolejnych fazach w oparciu o opisane wcześniej listy sum wag sąsiadów. Cały zestaw faz możemy regularnie powtarzać aż uzyskamy obraz o grubości jednego pixela.

Ze względu na podobieństwo algorytmów, opis przygotowania zdjęcia, definicja sąsiedztwa oraz definicja wag jest taki sam.

Algorytm prezentuje się następująco:

1. Każdy pixel który ma chodziaż jednego zerowego sąsiada oznaczamy jako 2.
2. Faza 1 – usuwamy wszystkie pixele oznaczone jako 2, które posiadają 3 przylegających do siebie sąsiadów.
3. Faza 2 – usuwamy wszystkie pixele oznaczone jako 2, które posiadają 3 lub 4 przylegających do siebie sąsiadów.
4. Faza 3 – usuwamy wszystkie pixele oznaczone jako 2, które posiadają 3, 4 lub 5 przylegających do siebie sąsiadów.
5. Faza 4 – usuwamy wszystkie pixele oznaczone jako 2, które posiadają 3, 4, 5 lub 6 przylegających do siebie sąsiadów.
6. Faza 5 – usuwamy wszystkie pixele oznaczone jako 2, które posiadają 3, 4, 5, 6 lub 7 przylegających do siebie sąsiadów.
7. Wszystkie pozostałe 2 oznaczamy jako 1.
8. Powtarzamy kroki 1 – 7 dopóki nie zostanie usunięty ani jeden pixel.
9. Dla każdego pixela oznaczonego jako 2 sprawdzamy czy jest on w linii o grubości jednego pixela. W przeciwnym wypadku usuwamy go

Punkty 1-9 powtarzamy dopóki nie zostanie zmieniony ani jeden pixel.

W celu określenia zarówno każdej liczby przylegających sąsiadów, jak również w celu sprawdzenia grubości linii posługujemy się siedmioma listami sum wag. Listy wyglądają analogicznie jak w poprzednim algorytmie. Różnią się jedynie wartościami, których nie będziemy tutaj przytaczać, a ich treść znajdzie się przed źródłami.

## Implementacja

Algorytm został zaimplementowany dosłownie tak jak w opisie teoretycznym. Dlatego poniższy opis odnosi się do poszczególnych punktów.

1. W podwójnej pętli przechodzimy po obrazie. Dla każdego pixela który nie jest zerem sprawdzamy czy chociaż jeden jego sąsiad jest zerem. W tym celu obliczamy sumę wag jego sąsiadów i sprawdzamy czy wartość jest w liście A0. Jeżeli tak, zamieniamy jedynkę na dwójkę.

W punktach: 2, 3, 4, 5, oraz 6 zaimplementowano następujące kroki:

W podwójnej pętli przechodzimy po obrazie. Dla każdego pixela który jest dwójką liczymy sumę wag odpowiadającą jego sąsiadom. Jeżeli waga znajduje się w odpowiedniej liście wag (kolejno A1, A2, A3, A4, A5), zamieniamy wartość pixela na 0.

1. W podwójnej pętli przechodzimy po obrazie. Dla każdego pixela który ma wartość 2 zmieniamy mu wartość na 1.
2. W podwójnej pętli przechodzimy po obrazie. Dla każdego pixela który nie jest zerem liczymy sumę wag odpowiadającą jego sąsiadom. Jeżeli waga znajduje się w liście wag OnePixel, zamieniamy wartość pixela na 0.

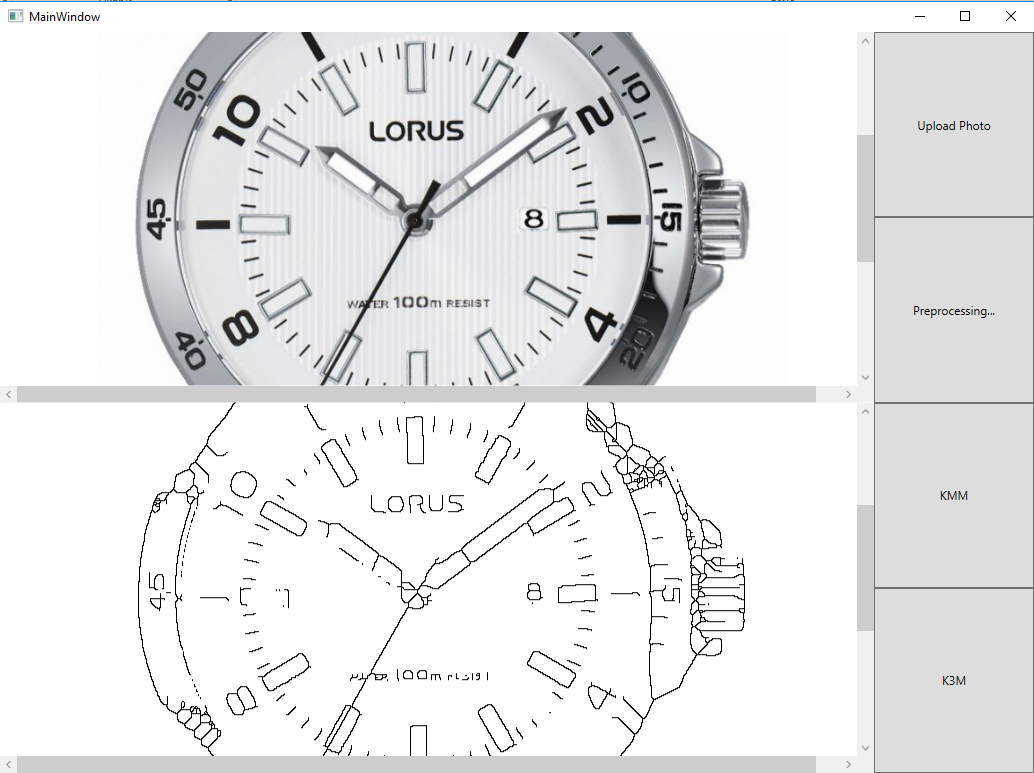
## ScreenShot

# 

Rysunek 6- K3M pierwszy przykład



Rysunek 7- K3M drugi przykład



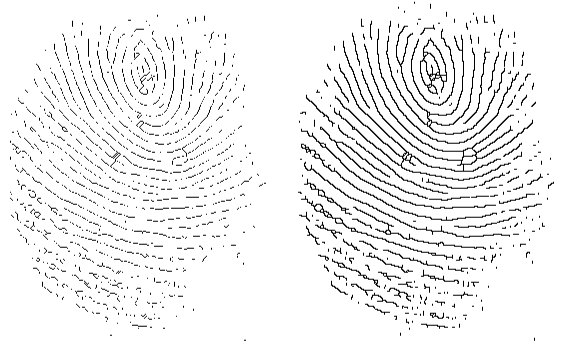
Rysunek 8- K3M trzeci przykład



Rysunek 9- K3M czwarty przykład

# Porównanie metod

W poniższym akapicie porównane zostaną oba algorytmy.



Rysunek 10- porównanie KMM - K3M

Jak widać na powyższym obrazie oraz wcześniejszych screenshotach, oba algorytmy spełniają swoje zadanie. I w jednym i w drugim przypadku możemy mówić o sukcesie w ścienianiu obrazu. Jednak już na pierwszy rzut oka widać różnice. Algorytm K3M daje wizualnie lepszy efekt niż KMM. Lepszą skuteczność można zauważyć na 3 płaszczyznach:

* Lepsze zachowanie ciągłości linii
* Większa liczba szczegółów
* Lepsze odwzorowanie kształtów

Z drugiej jednak strony z samej nazwy wynika, że głównym zadaniem obu algorytmów jest ścienianie. Pomimo niezachowania ciągłości linii, algorytm KMM wydaje się generować obraz o cieńszych krawędziach niż K3M kosztem jakości.

Podsumowując. W oparciu o obserwacje wyników można jednoznacznie stwierdzić że algorytm K3M dużo lepiej sprawdza się jako algorytm ścieniania.

# Listy

deletionTable = 3, 5, 7, 12, 13, 14, 15, 20, 21, 22, 23, 28, 29, 30, 31, 48, 52, 53, 54, 55, 56, 60, 61, 62, 63, 65, 67, 69, 71, 77, 79, 80, 81, 83, 84, 85, 86, 87, 88, 89, 91, 92, 93, 94, 95, 97, 99, 101, 103, 109, 111, 112, 113, 115, 116, 117, 118, 119, 120, 121, 123, 124, 125, 126, 127, 131, 133, 135, 141, 143, 149, 151, 157, 159, 181, 183, 189, 191, 192, 193, 195, 197, 199, 205, 207, 208, 209, 211, 212, 213, 214, 215, 216, 217, 219, 220, 221, 222, 223, 224, 225, 227, 229, 231, 237, 239, 240, 241, 243, 244, 245, 246, 247, 248, 249, 251, 252, 253, 254, 255

A0 = 3, 6, 7, 12, 14, 15, 24, 28, 30, 31, 48, 56, 60, 62, 63, 96, 112, 120, 124, 126, 127, 129, 131, 135, 143, 159, 191, 192, 193, 195, 199, 207, 223, 224, 225, 227, 231, 239, 240, 241, 243, 247, 248, 249, 251, 252, 253, 254

A1 = 7, 14, 28, 56, 112, 131, 193, 224

A2 = 7, 14, 15, 28, 30, 56, 60, 112, 120, 131, 135, 193, 195, 224, 225, 240

A3 = 7, 14, 15, 28, 30, 31, 56, 60, 62, 112, 120, 124, 131, 135, 143, 193, 195, 199, 224, 225, 227, 240, 241, 248

A4 = 7, 14, 15, 28, 30, 31, 56, 60, 62, 63, 112, 120, 124, 126, 131, 135, 143, 159, 193, 195, 199, 207, 224, 225, 227, 231, 240, 241, 243, 248, 249, 252

A5 = 7, 14, 15, 28, 30, 31, 56, 60, 62, 63, 112, 120, 124, 126, 131, 135, 143, 159, 191, 193, 195, 199, 207, 224, 225, 227, 231, 239, 240, 241, 243, 248, 249, 251, 252, 254

OnePixel = 3, 6, 7, 12, 14, 15, 24, 28, 30, 31, 48, 56, 60, 62, 63, 96, 112, 120, 124, 126, 127, 129, 131, 135, 143, 159, 191, 192, 193, 195, 199, 207, 223, 224, 225, 227, 231, 239, 240, 241, 243, 247, 248, 249, 251, 252, 253, 254

# Źródła

* „Algorytm do ścieniania obrazów: Implementacja i zastosowania”: Khalid Saeed, Mariusz Rybnik, Marek Tabędzki, Marcin Adamski.
* „K3M: A universal algorithm for image skeletonization and a review of thinning techniques” : Khalid Saeed, Mariusz Rybnik, Marek Tabędzki, Marcin Adamski.