Tomasz Kryjak, Piotr Pawlik 16 stycznia 2019

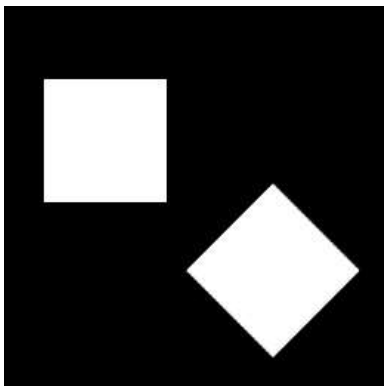
10 — Indeksacja. Klasyfikacja obiektów.

10.1 Cel

- zapoznanie z dwuprzebiegowym algorytmem indeksacji (ang. *connected component labelling*),
- zapoznanie z prostą klasyfikacją obiektów z wykorzystaniem współczynników kształtu.
- zadanie domowe: zmodyfikowana indeksacja dwuprzebiegowa i indeksacja jednoprzebiegowa.

10.2 Indeksacja

Większość dotychczas poznanych i wykorzystywanych algorytmów przetwarzania obrazu wykonywała operacje na całym obrazie (lub co najwyżej dokonywany był podział obiekt/obiekty — tło. Np. różne metody segmentacji).



Rysunek 10.1: Dwa kwadraty, obraz binarny.

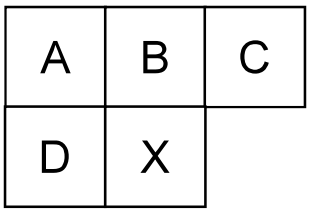
Patrząc na rysunek 10.1 człowiek widzi dwa kwadraty. Dla automatycznego systemu analizy i rozpoznawania obrazów przejście od etapu obiekty/tło do wyróżnienia dwóch kwadratów nie jest takie oczywiste i wymaga zastosowania jakieś formy indeksacji – czyli przypisania każdemu z pikseli uznanych za należące do obiektów (tu białych) jakieś etykiety (w naszym przypadku będą to dwie etykiet, ew. dodatkowa oznaczająca tło).

W tej części ćwiczenia zaprezentowany zostanie tzw. dwuprzebiegowy algorytm indeksacji (ang. *two-pass connected component labelling*). Jego nazwa związana jest z koniecznością wykonania dwukrotnego odwiedzenia każdego z pikseli obrazu (dwie pętle/iteracje po obrazie). Istnieją również tzw. jednoprzebiegowe algorytmy indeksacji – por. zadnie domowe.

Opis algorytmu:

1. Na wejściu mamy obraz po binaryzacji – zakładamy, że piksele białe (’1’, ’255’) należą do obiektów, tło jest czarne.

2. W pierwszej iteracji obraz analizujemy linia po linii, od lewej do prawej, aż do napotkania pierwszego piksela o wartości różnej od '0'. W tej sytuacji dokonujemy analizy otoczenia piksela:



gdzie: **X** – rozważany piksel, **A,B,C,D** – sąsiedzi z otoczenia. Uwaga. W przykładzie pokazano tzw. sąsiedztwo 8-elementowe. Niekiedy stosuje się również wersję 4-elementową. Wtedy analizie podlegałyby by tylko piksele **B i D**.

3. Możliwe są następujące przypadki dla otoczenia **A,B,C,D**:
   a wszystkie należą do tła ($A,B,C,D == 0$). Wtedy znaleziony piksel **X** należy do nowego obiektu – nadajemy mu zatem etykietę **L+1** (`X = L+1`) – przez **L** rozumiemy poprzednią etykietę. Uwaga. Przed uruchomieniem algorytmu **L** należy zainicjować wartością '1'.
   b jeden lub więcej pikseli ma przypisaną aktualną etykietę **L**. Wtedy rozważanemu pikselowi przypisujemy etykietę **L** (`X = L`).
   c w otoczeniu występują piksele o różnych etykietach np. **L1** i **L2**. Wtedy przyjmuje się zasadę, że rozważanemu pikselowi **X** przypisuje się mniejszą z wartości **L1** i **L2** (`X = min(L1,L2)`).

Implementacja:

1. Otwórz program Matlab. Ustal ścieżkę *Current Directory* na swój własny katalog roboczy. Utwórz nowy m-plik (*New Script*) lub (*New->Script*). Na początku wykonaj polecenia `close all; clearvars; clc;` Wczytaj obraz *ccl1.png*. Wyświetl go. Po krótkiej analizie można zauważyć, że występuje na nim 7 odrębnych obiektów. Ich "wykrycie" będzie naszym celem.
2. Na podstawie opisu algorytmu podanego wcześniej zaimplementuj tzw. pierwszą fazę indeksacji (ang. *first pass*).
   - uwaga ogólna – algorytm jest dość prosty i łatwy w implementacji,
   - obliczenia powinny odbywać się w pętlach `for` – iteracja po całym obrazku. Warto pominąć pierwszy wiersz i pierwszą oraz ostatnią kolumnę, aby wyeliminować problem z brakiem kontekstu (można np. założyć, że pikseli brzegowych nie ma, bo wcześniej wykonane zostało morfologiczne czyszczenie brzegu).
   - dla przypomnienia iteracja po macierzy w Matlabie – pętla zewnętrzna – wiersze, pętla wewnętrzna – kolumny — tj. pierwsza współrzędna to wiersz, a druga kolumna,
   - rozmiar obrazka można odczytać np. poleceniem: `[YY, XX] = size(obraz);` Przy takim przypisaniu `YY` oznacza liczbę wierszy (wysokość obrazka), a `XX` liczbę kolumn (szerokość obrazka),
   - działania podejmujemy tylko w przypadku, gdy aktualnie analizowany piksel ma wartość różną od zera,
   - najtrudniejszym elementem jest analiza otoczenia piksela i stwierdzenie, z którym z przypadków (a), (b) czy (c) mamy do czynienia. Problem można rozwiązać jakkolwiek (tylko dobrze), poniżej prezentowana jest jedna z możliwości,
   - na początku tworzymy wektor pikseli, które stanowią otoczenie piksela X (A, B, C, D)

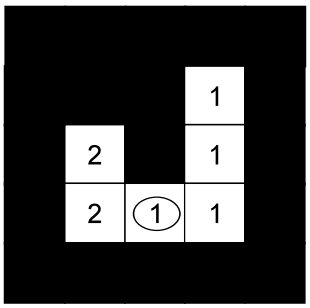
np: `sasiedzi= [obraz(x-1,y-1) obraz(x-1,y) obraz(x-1,y+1) obraz(x,y-1)];`

- następnie sprawdzamy czy nie występuje przypadek (a) – czyli czy suma sąsiadów nie wynosi 0 – `suma = sum(sasiedzi);`, jeżeli tak to `X = L; oraz L=L+1;`. Uwaga proszę nie zapomnieć zainicjalizować etykiety `L` wartością 1.
- jeżeli `suma > 0` mamy do czynienia z przypadkiem (b) lub (c). Eliminujemy zera z wektora `sasiedzi` (funkcja `nonzeros`), a następnie znajdujemy minimum i maksimum nowego wektora `sasiedzi` (funkcje `min` i `max`). Uwaga: proszę pamiętać aby nie nazywać wyników tych operacji min i max,
- z przypadkiem (b) będziemy mieli do czynienia, gdy **minimum = maksimum**. Wtedy jako etykietę przypisujemy `X = minimum = maksimum;`. Gdy **minimum != maksimum** mamy przypadek (c). Wtedy etykieta `X = minimum` – taką przyjmujemy konwencję,
- uwaga 1 – algorytm operuje (czyta punkty z otoczenia i zapisuje kolejne indeksy L) na tym samym obrazie wejściowym,
- uwaga 2 – przed implementacją metody obsługi konfliktów przypadki (b) i (c) działają tak samo, ale dla potrzeb dalszych kroków należy je rozróżnić,

3. Wykonaj indeksację obrazu za pomocą zaimplementowanego algorytmu. Jeżeli wszystko zostało poprawnie napisane to wynik powinien wyglądać mniej więcej tak jak w pliku *ccl1Result.png*. Takie porównanie stanowić będzie pierwszy test poprawności implementacji algorytmu.

Dyskusja:

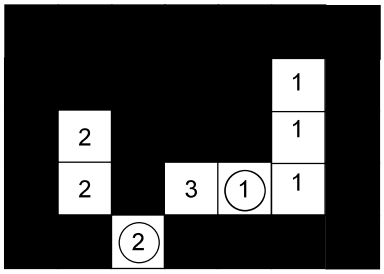
1. W wyniku indeksacji uzyskujemy obraz, na którym wyróżnionych jest wiele obiektów (dokładnie 48) – a faktycznie jest tylko 7. Zastanówmy się z czego to wynika.
2. Zacznijmy od prostego przypadku przedstawionego na rysunku 10.2.



Rysunek 10.2: Indeksacja – przykład pierwszy.

Obraz analizowany jest linia po linii. Zatem pierwszą etykietę dostanie słupek po prawej. Później ten po lewej. Postępując zgodnie z podanym algorytmem w pewnym momencie (piksel wyróżniony na obrazku) dojdzie do sytuacji, w której w otoczeniu danego piksela znajdą się dwie różne etykiety (tu: '1' i '2'). Wtedy zgodnie z przyjętą metodologią przypisujemy niższą, tj. '1'. Problem jaki powstaje to brak zapamiętania informacji o tym, że wystąpił taki konflikt tj. że powinno nastąpić połączenie etykiet '1' i '2'. Z tego powodu wynik działania algorytmu nie jest poprawny.

3. Przeanalizujmy jeszcze jeden, bardziej złożony przykład – rysunek 10.3.

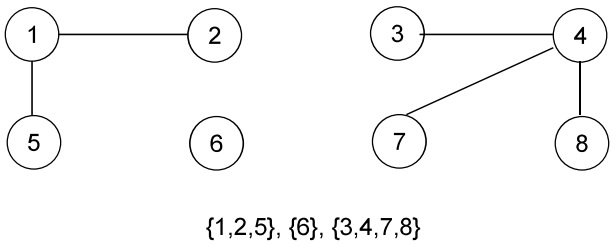


Rysunek 10.3: Indeksacja – przykład drugi.

Słupek po prawej dostaje indeks '1', a po lewej '2'. W czwartej linii występuje piksel, który ma w swoim sąsiedztwie (A,B,C,D) same piksele czarne. Dlatego dostaje etykietę '3'. Jednak w następnym kroku okazuje się, że następuje konflikt '3' z '1' (połączenie). W kolejnym wierszu występuje konflikt '2' i '3'. Warto zwrócić uwagę, że w tym przypadku trzeba uniknąć sytuacji "utraty" informacji o połączeniu '1' z '2' przez '3'.

4. W literaturze zaproponowano wiele sposobów reprezentacji i rozwiązywania przedstawionych konfliktów:
   - 2-krotki,
   - n-krotki,
   - tablica dwuwymiarowa,
   - grafy + przeszukiwanie grafu wgłąb.

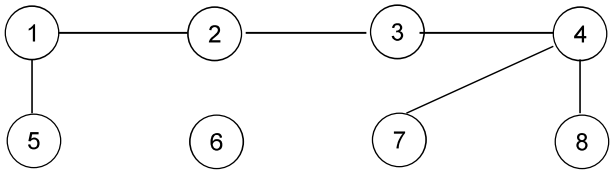
   W obecnym ćwiczeniu zastosujemy ostatnie podejście, które jest najprostsze do realizacji.
5. Opisane konflikty możemy przechowywać w strukturze zbiorów rozłącznych (ang. *union find*). Jest to zagadnienie znane z przedmiotu "Algorytmy i struktury danych". Poniżej zostanie zaprezentowane krótkie przypomnienie.
6. Mamy N obiektów. W naszym przypadku to jest N etykiet. Chcemy przechowywać informację o sytuacji, w której następuje łączenie etykiet tj. interesują nas zbiory obiektów połączonych. Przykład trzech zbiorów zamieszczono na rysunku 10.4.



Rysunek 10.4: Przykład trzech połączonych zbiorów.

7. W ramach rozważanej struktury implementuje się dwie operacje:
   - *find* (znajdź) – sprawdzenie czy dwa obiekty należą do tego samego zbioru połączonego,
   - *union* (połącz) – wprowadź połączenie pomiędzy dwoma obiektami.

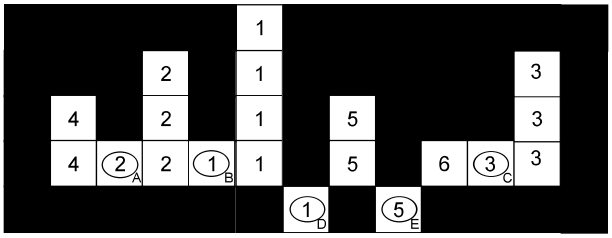
   Z punktu widzenia naszych potrzeb ważna jest funkcja *union*. Przykład dodania połączenia pomiędzy obiektami '2' a '3' pokazano na rysunku 10.5.



{1,2,3,4,5,7,8}, {6}

Rysunek 10.5: Przykład łączenia obiektów.

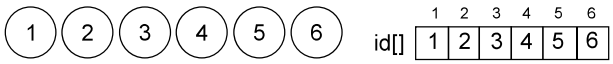
8. Strukturę oraz obie operacje można zaimplementować na kilka sposobów. Zainteresowanych odsyłam do literatury przedmiotu. My zastosujemy podejście *quick-union*. Nie jest ono specjalnie wydajnie, ale bardzo proste do implementacji.
9. Nasz graf, w którym wierzchołki oznaczają etykiety, a krawędzie połączenia między etykietami, zapiszemy w tablicy jednowymiarowej `id[]` o rozmiarze N (maksymalna liczba etykiet). Interpretacja pola w tablicy: `id[i]` jest rodzicem `i`. Korzeń elementu `i` jest dany jako: `id[id[id[...id[i]...]]]`.
10. Przeanalizujmy jak to działa na prostym przykładzie. Mamy dany poetykietowany obraz – rysunek 10.6.



Rysunek 10.6: Indeksacja – przykład 3

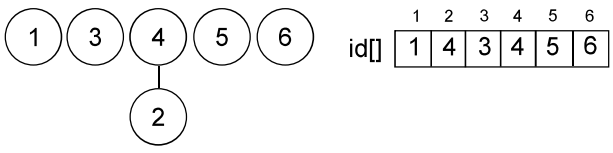
Występuje na nim pięć konfliktów oznaczonych A, B, C, D i E. (uwaga w trakcie działania algorytmu pojawią się właśnie w takiej kolejności).

11. Sytuację wyjściową opisuje poniższy rysunek:



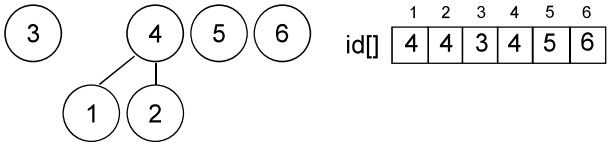
Tablica `id` wypełniona jest kolejnymi indeksami.

12. W algorytmie *quick-union*, aby połączyć elementy p i q należy ustawić *id* korzenia q pod adres korzenia elementu p. Uwaga. Zakładamy, że $p < q$. Można to zapisać jako: `id[root(p)] = root(q)`. Nasze pierwsze połączenie (A) to 4 z 2. Po tej operacji sytuacja będzie wyglądać następująco:



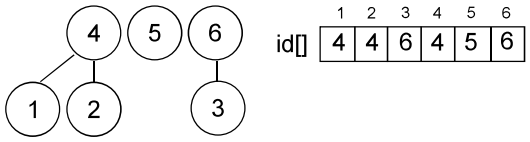
W tym przypadku korzenieniem obu elementów są same elementy, zatem operacja jest dość prosta.

13. Kolejny krok tj. połączenie 2 z 1 – sytuacja B:

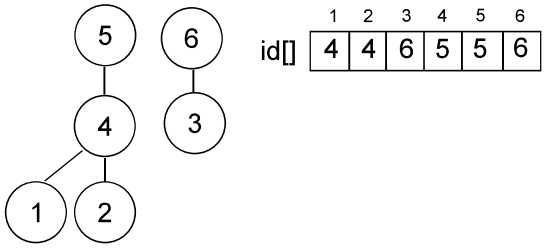


Element 1 jest korzeniem. Element 2 już nie, ponieważ `id[2]==4`. Sprawdzamy zatem `id[4]`. Okazuje się, że `id[4] == 4` tj. jest to korzeń. Zatem korzeniem dla elementu 2 jest 4. Dokonujemy stosownej modyfikacji w tablicy `id[1] = 4`.

14. Kolejne połączenie to 6 z 3 (C):

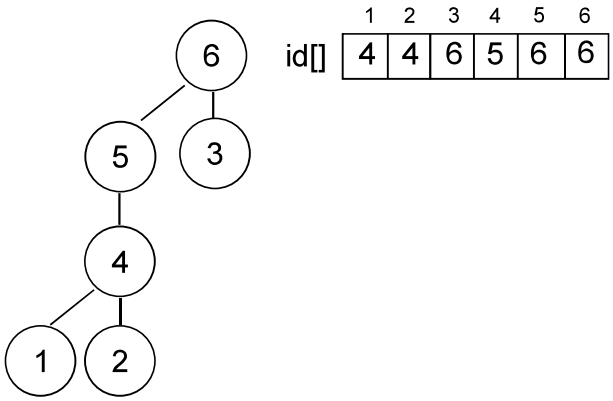


15. Kolejne połączenie to 5 z 1.



Szukamy korzenia elementu 1. Znajdujemy, że to jest 4. Zatem łączymy 5 i 4.

16. Ostatnie połączenie to 6 z 5.



17. Ostatni krok to wykorzystanie informacji zawartej w tablicy `id` do wyznaczenia przekodowania LUT (ang. *look-up table*), które pozwoli nam nadać poprawne etykiety wszystkim pikselom. Zasadniczo sprowadza się on do obliczenia dla każdej możliwej etykiety jej korzenia. W naszym przypadku zawsze otrzymamy 6, gdyż wszystkie piksele są ze sobą połączone i stanowią jeden obiekt.

Implementacja:

1. Implementacja jest dużo prostsza i krótsza od powyższego opisu. Istotne jest tylko dobre zrozumienie algorytmu *union-find*.
2. Potrzebne będą nam dwie funkcje pomocnicze:

- `root` – obliczanie korzenia zgodnie z podanym opisem (odpowiednia pętla `while`). Funkcja pobiera indeks elementu oraz tablicę, a zwraca indeks korzenia.
- `union` – realizacja operacji unii. Argumenty to indeksy `p` i `q` oraz tablica, a wyniki to zmodyfikowana tablica.

3. Na początku, przed pierwszym przebiegiem algorytmu indeksacji, tworzymy tablicę `id` i inicjujemy ją wartościami od 1 do N. Dla rozważanego przykładu wystarczy $N = 100$.
4. Następnie, w przypadku wystąpienia konfliktu (przypadek ()c)) tworzymy unię pomiędzy etykietą mniejszą i większą.
5. Po pierwszym przebiegu tworzymy, w opisany powyżej sposób, przekodowanie LUT (pętla po `id` i funkcja `root`). Powstaje nam tablica `lut`.
6. Implementujemy drugi przebieg po obrazie. Jest on bardzo prosty. Dla każdego piksela, który nie jest tłem (o etykiecie większej od 0) realizujemy przekodowanie LUT (czyli `I2(j,i) = lut(I1(j,i));`). Otrzymany w ten sposób obraz wyświetlamy. Powinniśmy uzyskać poprawne etykietowanie.

10.3 Indeksacja wbudowana w IPT Matlab'a

W Matlabie dostępna jest funkcja `bwlabel`, która pozwala wykonać indeksację. Jedynym jej parametrem jest wybór sąsiedztwa: 4-elementowego (za sąsiadów uznajemy piksele "stykające się" brzegami) i 8-elementowego (za sąsiada uznajemy piksele "stykające się" brzegami i wierzchołkami – w ten sposób działa zaimplementowany wcześniej algorytm).

Implementacja:

1. W nowym m-pliku wczytaj obraz *ccl2.png*. Wyświetl go.
2. Przeprowadź indeksację za pomocą funkcji `bwlabel` z sąsiedztwem 4 i 8 elementowym. Wyniki porównaj (np. na wspólnym rysunku).

Uwaga. Warto wiedzieć, że w bardzo popularnej bibliotece do przetwarzania i analizy obrazów OpenCV tego typu indeksacja nie występuje. Tam do podobnych (ale nie identycznych) celów stosuję się znajdywanie konturów.

10.4 Rozpoznawanie obiektów z wykorzystaniem współczynników kształtu

Współczynniki kształtu są pewnymi parametrami liczbowymi opisującymi kształt obiektu. Pozwala to na użycie ich do automatycznego rozpoznawania obiektów. Teoretycznie współczynniki kształtu (dobre) powinny być niezależne od zmiany położenia obiektu, jego orientacji i wielkości. W praktyce okazuje się jednak, że wartości współczynników zmieniają się w pewnym zakresie nawet dla obiektów należących do tych samych klas (te same kształty – przykład obracany kwadrat). Wynika z tego konieczność uwzględnienia pewnej tolerancji wartości współczynników kształtu wykorzystywanych w praktycznych zastosowaniach analizy i rozpoznawania obrazu.

W ćwiczeniu wykorzystane zostaną następujące współczynniki:

- *Compactness* – stosunek pola obiektu do pola najmniejszego prostokąta w jakim się obiekt mieści.
- *Rmin/Rmax* – pierwiastek stosunku minimalnej odległości konturu od środka ciężkości do maksymalnej odległości konturu od środka ciężkości.
- *Blair - Bliss* – $\frac{S}{2\pi\sum_i r_i^2}$, gdzie: S – pole powierzchni obiektu, r_i – odległość piksela obrazka od środka ciężkości, i – indeks piksela obiektu.
- *Haralick* – $\sqrt{\frac{(\sum_i d_i)^2}{n\sum_i d_i^2 - 1}}$, gdzie: d_i – odległość piksela konturu od środka ciężkości, n - liczba pikseli konturu.

- *M7* – niezmiennik momentowy – $M7 = N_{20}N_{02} - N_{11}^2$, N – moment znormalizowany: $N_{pq} = \frac{M_{pq}}{m_{00}^{\zeta}}$, gdzie: $\zeta = \frac{p+q}{2} + 1$, M – moment centralny, m – moment zwykły.

Implementacja:

1. W nowym m-pliku wczytaj obraz *shapes.png*. Wyświetl go. Pierwszym etapem zadania będzie wybranie współczynników kształtu i przedziałów ich wartości – takich, które pozwolą na wykrycie konkretnych kształtów. Do wyboru jest kwadrat lub krzyżyk (koło jest zbyt łatwe).
2. Wykorzystując dostarczoną funkcję – `obliczWspolczynniki.m` wyznacz współczynniki kształtu dla wszystkich obiektów z obrazu *shapes.png*. Funkcja jako argument przyjmuje obraz po indeksacji, a zwraca macierz dwuwymiarową – wiersze oznaczają konkretne kształty (kolejność taka jak numery indeksów przypisane podczas indeksacji), a kolumny kolejne współczynniki – *Compactness, Rmin/Rmax, Blair - Bliss, Haralick, M7*. Przeanalizuj otrzymają macierz – głównie pod kątem zadania detekcji kształtów. Uwaga. Funkcja wykorzystuje wbudowaną w IPT funkcję `regionprops`, która pozwala wyliczyć pewne współczynniki kształtu. Szczegóły w dokumentacji.
3. Analizę warto przeprowadzić w arkuszu kalkulacyjnym (Excel, Calc) — można wtedy opisać wiersze i kolumny. Na początku trzeba upewnić się, że dobrze oznaczamy konkretne kształty. Kolejność wierszy w macierzy zwracanej przez funkcję `obliczWspolczynniki` jest taka jak indeksów na obrazie (czyli wg. rosnącej jasności). Można np. "podpisać" kształty:

```
r = regionprops(labeled,'Centroid');
for i=1:length(r)
        text(r(i).Centroid(1),r(i).Centroid(2),['\color{magenta}',
    num2str(i)]);
end
```

   gdzie: `labeled` – obraz po indeksacji lub odczytać wartości z wykresu (narzędzie *Data Cursor*).
4. Następnie warto wyszukać taki współczynniki (współczynnik), które pozwolą odróżnić kwadraty od krzyżyków i ustalić pewien przedział wartości współczynnika. Dodatkowo trzeba zapewnić, żeby nie zostały także wykryte koła.
5. Zaimplementuj logikę, która pozwoli wyeliminować z obrazka niepożądane obiekty. Wykonaj iterację po całym poindeksowanym obrazku. Odczytaj aktualną wartość piksela, jeżeli jest różna od zera to wykorzystując macierz zwracaną przez funkcję `obliczWspolczynniki` wyzeruj dany piksel lub nie. Przykład:

```
piksel = labeled(i,j); % obliczanie srodka ciezkosci
if ( piksel ~= 0 && ~(wsp(piksel,2) > 0.33 && wsp(piksel,2) < 0.66)  )
        labeled(i,j) = 0;
end
```

   Zapis ten oznacza, że jeżeli współczynnik drugi (*Rmin/Rmax*) nie zawiera się w przedziale (0.33:0.66) to piksel należy wyzerować.
6. Wczytaj obraz *shapesReal.png*. Wyświetl go. Wykorzystując poznane metody przetwarzania obrazu doprowadź obraz do postaci binarnej (usuń ew. zakłócenia itp.). Następnie rozpoznaj na obrazie wybrany wcześniej kształt (postępowanie analogiczne jak w punktach 2-6).
7. Uwagi:
   - należy z obrazu wyeliminować małe grupki pikseli – tak aby nie zostały poindeksowane,

- obiektów powinno być 13,
- może się okazać, że zajdzie potrzeba modyfikacji warunków na współczynniki albo wręcz wprowadzenia dodatkowych współczynników.
- warto zaznaczyć, że opisane podejście tj. "ręczne" wyznaczanie przedziałów parametrów stosuje się tylko do bardzo prostych problemów. W rzeczywistych aplikacjach rozpoznawania obrazu stosuje się tzw. uczenie maszynowe i klasyfikatory typu sztuczne sieci neuronowe, algorytm k-NN, czy SVM.

8. Wyniki zaprezentuj prowadzącemu.