

Wykorzystanie algorytmu Hougha do detekcji obramowań tabel na dokumentach

Projekt z przedmiotu CPOO

Michał Krawczak

16 września 2017

Spis treści

1 Temat projektu	2
1.1 Zakres projektu	2
2 Algorytm Hougha	2
2.1 Transformacja Hougha	2
2.2 Pseudokod algorytmu	3
3 Implementacja	3
4 Testowanie	4
4.1 Obrazy testowe	4
4.2 Wyniki	4
4.3 Omówienie wyników	8
5 Wnioski	8

1 Temat projektu

W ramach projektu z przedmiotu CPOO zdecydowałem pochylić się nad problemem, na który natrafiłem podczas prac nad jednym z projektów, które realizowałem w pracy zawodowej, tj. problemem wykorzystania algorytmu Hougha do detekcji linii w dokumentach finansowych.

1.1 Zakres projektu

W ramach niniejszego projektu zaimplementowałem algorytm Hougha w języku Java, a następnie przetestowałem go na obrazach kilku dokumentów, zarówno oryginalnych, jak i celowo zaszumionych. Ponadto, na tych samych obrazach, uruchomiłem gotowy moduł wykonujący algorytm Hougha, wchodzący w skład popularnego pakietu `imagemagick` [1]. Następnie dokonałem porównania wyników działania obu tych podejść.

2 Algorytm Hougha

Algorytm Hougha, opisany pierwotnie w [2], stał się jednym z podstawowych algorytmów komputerowego przetwarzania obrazów. Najczęstszym zastosowaniem tego algorytmu jest detekcja prostych, jednak można go łatwo zaadaptować do wykrywania dowolnych kształtów, które dają się opisać za pomocą analitycznego wzoru[3]. Podstawowym pojęciem leżącym u podstaw tej techniki jest transformacja Hougha.

2.1 Transformacja Hougha

Transformacja Hougha przekształca przestrzeń kartezjańską w przestrzeń Hougha. Przestrzeń Hougha to dwuwymiarowa przestrzeń, w której każda prosta¹ z obrazu oryginalnego jest reprezentowana przez pojedynczy punkt.

Osie, na których rozpięta jest przestrzeń Hougha, reprezentują następujące skalary:

- r — reprezentuje długość najkrótszego odcinka łączącego prostą ze środkiem układu współrzędnych $(0,0)$ oryginalnego obrazu.
- θ — reprezentuje kąt między ww. odcinkiem, a osią OX oryginalnego obrazu.

Ponadto, każdy punkt w przestrzeni Hougha ma przypisaną wartość całkowitoliczbową n , która oznacza liczbę punktów w obrazie pierwotnym, które spełniają równanie prostej.

¹Od tego miejsca zakładam, że wykrywaną figurą jest prosta.

2.2 Pseudokod algorytmu

Algorytm można zrealizować za pomocą następujących kroków:

1. Parametry wejściowe:

- δ_θ — rozmiar kroku θ
- t — próg binaryzacji (tj. graniczna jasność poniżej której piksele mają być uznane za czarne).
- p — próg filtrowania wyników (tj. minimalna liczba punktów na prostej, aby była ona godna zwrócenia użytkownikowi).

2. Weź obraz wejściowy I o wymiarach $n \times m$.

3. Utwórz tablicę H liczb całkowitych o wymiarach $\theta_s \times d$, gdzie $\theta_s = \frac{2\pi}{\delta_\theta}$,
 $d = \sqrt{n^2 + m^2}$ i wypełnij ją zerami.

4. Dla każdego piksela $I_{n,m}$ w obrazie wejściowym I :

(a) Dla każdej wartości θ od 0 do π z krokiem co δ_θ :

i. Oblicz $r = \lfloor x \cdot \cos \theta + y \cdot \sin \theta \rfloor$

ii. Jeśli jasność piksela $I_{n,m}$ jest poniżej progu t , to zwiększą wartość $H_{\theta,r}$ o 1

5. Wybierz te elementy tablicy H , których wartość jest większa niż p . Zwróć je użytkownikowi.

Wynik może zostać zwrócony w postaci listy prostych, gdzie każda jest opisana przez parę parametrów (θ, r) . Wygodniej jest jednak, jeżeli wynik ma postać graficzną, np. wykryte linie zostają narysowane i nałożone na obraz wejściowy. W takiej postaci zostaną zaprezentowane wyniki w kolejnych rozdziałach.

3 Implementacja

Kod programu znajduje się w repozytorium pod adresem:

<https://github.com/ciekawylogin/rough>

Właściwy kod programu został umieszczony w pliku `HoughLineDetector.java`. Realizuje on niemal dokładnie algorytm opisany w 2.2. Jedynym drobnym usprawnieniem było wprowadzenie do kroku 5. dodatkowego warunku: linie są zwracane tylko wtedy, gdy są lepsze (tj. mają więcej punktów), niż ich 4 bezpośredni sąsiedzi w przestrzeni Hougha. Warunek ten sprawdzany jest w liniach 207–211. Nie wpływa on znacząco na poprawność, a jedynie ogranicza zbiór wynikowy. W oryginalnej postaci algorytm często zwraca grupy bardzo podobnych do siebie linii – np. obróconych o 1° albo przesuniętych o 1 piksel. Wprowadzenie tego dodatkowego sprawdzenia powoduje, że zwrócona zostaje tylko jedna (potencjalnie najtrajniejsza) linia z takiej grupy, przez co wynikowy obraz jest czytelniejszy.

4 Testowanie

W celu przetestowania powyższego programu i porównania go z narzędziem wchodząącym w skład programu **imagemagick**, oba programy zostały uruchomione na 5 obrazach testowych.

Własna implementacja była uruchamiana z następującymi parametrami: próg binaryzacji $r = 80$, próg algorytmu Hougha $p = 400$, rozmiar kroku kątowego $\delta_\theta = \frac{2\pi}{360}$ (tj. 360 kroków).

Imagemagick był wywoływany w następujący sposób:

```
convert <obraz_wejsciowy>.jpg -colorspace gray \
+threshold 2 -normalize -negate -hough-lines \
100x100+250 -transparent white temp.png
composite temp.png <obraz_wejsciowy> <obraz_wynikowy>.jpg
rm temp.png
```

Powyższa komenda wykonuje binaryzację (z użyciem progowania Otsu), następnie wykonuje algorytm Hougha i zapisuje uzyskane linie do pliku **temp.png**. Następnie linie są nakładane na oryginalny obraz i tworzony jest obraz wynikowy.

4.1 Obrazy testowe

Oba programy zostały przetestowane na pięciu obrazach testowych. Obrazy 1 i 2 to prawdziwe skany faktur, wykonane w zadowalającej jakości ², natomiast obrazy 3, 4 i 5 są to obrazy z arbitralnie nałożonym szumem:

- W obrazie 3 wartości 10% pikseli (wybranych losowo) zostały zamienione na losowe.
- W obrazie 4 podobnie jw., tyle że zmodyfikowane zostało 25% pikseli.
- W obrazie 5 zostało zastosowane rozmycie Gaussa.

Wszystkie te dokumenty zawierają tabelki (z produktami i z podsumowaniem stawek VAT), które należy wykryć. Idealnie, algorytm powinien wykryć tylko linie tworzące tabele, co pozwoliłoby wykryć położenie samej tabeli przez zlokalizowanie obszaru przecięcia linii.

Miniaturki obrazów są widoczne na ilustracji 1, zaś pełne wersje znajdują się w repozytorium w katalogu **docs/testing**.

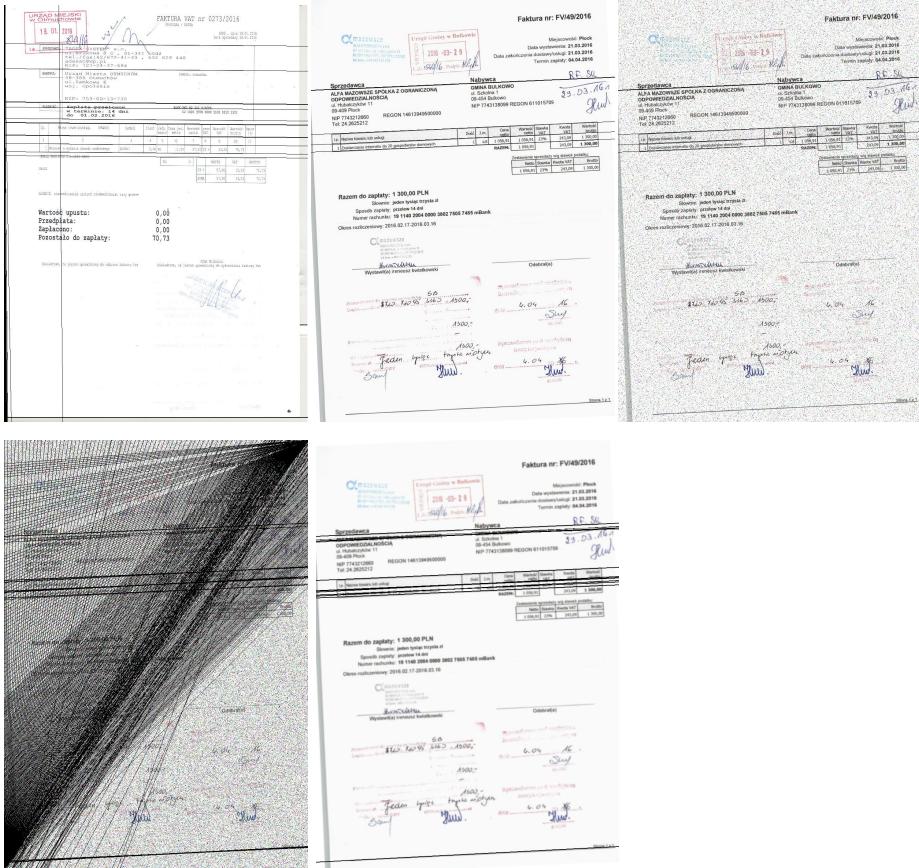
4.2 Wyniki

Wyniki działania zaimplementowanego programu zostały pokazane na ilustracji 2. Dla porównania, wyniki działania **Imagemagick** przedstawia ilustracja 3.

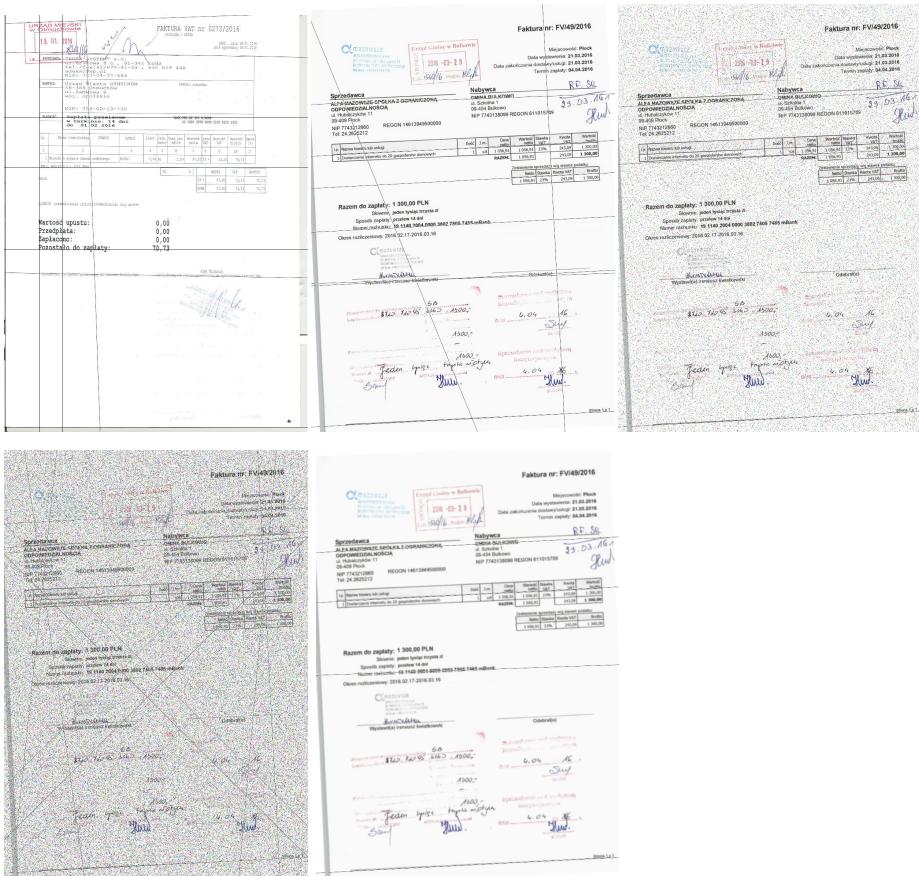
²Skany zostały udostępnione przez urzędy gmin Bulkowo i Otmuchów jako informacja publiczna. Na mocy ustawy z dnia 6 września 2001 r. o dostępie do informacji publicznej, obrazy takie mogą być swobodnie przetwarzane i rozpowszechniane.



Rysunek 1: Obrazy testowe



Rysunek 2: Obrazy testowe z nałożonymi liniami wykrytymi przez własną implementację



Rysunek 3: Obrazy testowe z nałożonymi liniami wykrytymi przez Imagemagick

Pełne wersje obrazów zostały zamieszczone w repozytorium, odpowiednio w katalogach `docs/own` i `docs/im`.

W obu przypadkach, linie zostały nałożone na oryginalne obrazy.

4.3 Omówienie wyników

1. Obraz 1 jest „najłatwiejszym” z obrazów tesowych, nie powinno zatem dziwić, że oba programy znalazły poziome linie tworzące tabele. **Imagemagick** znalazł również jedną z linii pionowych. Jednak zwrócone zostały nie tylko one - w obu przypadkach część zdetekowanych linii to po prostu linia bazowa tekstu.
2. Obraz 2 — podobnie jak w poprzednim przypadku, obie implementacje znalazły większość poziomych linii tworzących tabele, a **Imagemagick** także jedną z linii pionowych. Oba detektory znalazły też poziomą linię oddzielającą napisy „sprzedawca” i „nabywca” od sekcji danych sprzedawcy i nabywcy. **Imagemagick** „znalazł” też skośną linię, której nie widać gołym okiem – jest to zbiór krótkich, skośnych linii, będących w większości fragmentami liter (np. „y”, „W”), które okazują się być współliniowe.
3. Obraz 3 to obraz zaszumiony. Tu w przypadku mojej implementacji nie widać różnicy w stosunku do obrazu 2, zaś w przypadku programu **Imagemagick** zostało wykrytych znacznie mniej linii – wynika to z faktu, że algorytm Otsu zwrócił inną wartość progu binaryzacji na skutek istnienia szumu.
4. Obraz 4 jest zaszumiony tą samą metodą co poprzedni, jednak w dużo większym stopniu. Na tym obrazie własna implementacja zaczęła wyraźnie „gubić się” i znajdować wiele nieistniejących linii. Linie te w większości są po prostu zbiorami punktów szumu, które przypadkowo akurat okazały się współliniowe. **Imagemagick** poradził sobie nieporównywalnie lepiej – „znalazł” co prawda kilka nieistniejących linii, ale było ich znacznie mniej.
5. Obraz 5, rozmyty za pomocą rozmycia Gaussa, w przypadku własnej implementacji spowodował „rozszczepienie” znalezionych prostych. Ponadto, więcej linijek tekstu zostało uznanych za linie. W przypadku **Imagemagicka** wpływ rozmycia był wyraźnie mniejszy.

5 Wnioski

- Zaimplementowany przeze mnie algorytm w przypadku niezaszumionych i słabo zaszumionych obrazów ma skuteczność podobną do gotowego rozwiązania.
- W przypadku obrazów zaszumionych (4 i 5), **Imagemagick** poradził sobie zdecydowanie lepiej, choć również popełniał błędy.

- Na żadnym z obrazów testowych żadne z rozwiązań nie wykryło obramowania tabeli z pełną skutecznością. Wynika to z faktu, że linie tekstu są w przestrzeni Hougha na tyle „podobne” do prostych, że algorytm zwyczajnie nie jest w stanie ich rozróżnić. Można próbować dostosować wartość progowania p , jednak ustawienie jej na zbyt wysoką wartość powoduje, że linie tworzące obamowanie tabeli w większości nie zostaną wykryte, a zbyt niską – że większość tekstu zostanie uznana za linie. Sugeruje to, że wykrywanie obramowań tabeli na tego typu dokumentach wymaga zastosowania algorytmu bardziej zaawansowanego niż algorytm Hougha.

Literatura

- [1] *Use ImageMagick® to create, edit, compose, or convert bitmap images*,
<https://www.imagemagick.org/script/index.php>
- [2] Hough, Paul
Method and means for recognizing complex patterns,
<https://www.google.com/patents/US3069654>
- [3] Duda, R. O., Hart P. E.
Use of the Hough Transformation to Detect Lines and Curves in Pictures,
Communications of the ACM, nr 15, str. 11–15 (styczeń 1972)