Warkod — Wykrywanie loga Warszawskich Kolei Dojazdowych na zdjęciach

Radosław Świątkiewicz

21 stycznia 2019

Spis treści

1 Opis

1.1 Stosowanie

Logo Warszawkich Kolei Dojazdowych ma kształt dwóch strzałek z literami pomiędzy.

1.2 Wymagania

Program jest napisany w C++ w standardzie `c++11`. Do pracy używa biblioteki OpenCV, testowane pod wersją `4.0.1`, ale używa jedynie podstawowych funkcji do wczytywania i zapisywania plików, które to się nie zmieniły od wcześniejszych wersji. Kompilacja jest możliwa za pomocą CMake.

Rysunek 1: Logo Warszawskich Kolei Dojazdowych

1.3 Uruchomienie

Warkod jest uruchamiany z konsoli i przyjmuje dwa lub trzy argumenty.

Nazwa pliku wejściowego to plik o dowolnym formacie, czytanym przez OpenCV.

Nazwa pliku wyjściowego jest miejscem, gdzie program wypisze obraz wynikowy, będzie to obraz wejściowy z zaznaczonym środkiem znalezionego loga WKD.

***Opcjonalny* katalog pomocniczy** do którego będą zapisywane pliki z parametrami znalezionych obiektów, obrazy z obiektami i wstępne dopasowania.

Przykładowe uruchomienie: `./warkod zdjęcie.jpg zaznaczone.jpg /tmp`

1.4 Spodziewany wynik

Program powinien wygenerować obraz wejściowy z zaznaczonym środkiem loga WKD. Pozycja, rozmiar i obrót, a także małe zmiany perspektywy nie powinny wpływać na odpowiedź.

Zakłada się minimalną wielkość obiektu w stosunku do rozmiarów zdjęcia. Oświetlenie i materiał, na którym wydrukowane jest logo, mogą wpływać na wykrywanie kolorów.

2 Działanie

Program wykonuje szereg osobnych operacji. Wszystkie parametry kolorów zakładają, że piksel ma zakres jasności od 0,0 do 1,0.

2.1 Filtracja

Obraz filtrowany jest filtrem medianowym o określonej wielkości, zależnej od wielkości obrazu. Filtr jest kwadratowy, o promieniu (ilości pikseli w linii od środkowego), który wynika z pomnożenia przez 0,002 długość mniejszego boku obrazu. Zwykle jest to 1 lub 2. Przy krawędziach program nie bierze pod uwagę obszaru poza obrazem, zmniejsza ilość porównywanych pikseli, wybiera medianę z mniejszej ilości.

Filtracja jest zaimplementowana wielowątkowo, po jednym wątku na wiersz.

2.2 Oddzielanie kolorów

Zastosowana została metoda porównywania geometrycznego punktów. Ponieważ piksel może być zaprezentowany jako punkt we wnętrzu sześcianu o boku 1, gdzie na osiach odłożone są składowe kolory czerwony, zielony i niebieski, można zastosować proste porównania.

Wykryć należy piksele odpowiednio czerwone i niebieskie, gdyż logo składa się z takich kolorów. Ponieważ materiał, na którym może być logo, bywa odbijający, warto zwiększyć zakres akceptowalności dla jaśniejszych pikseli.

Ciemne kolory mogą miewać różne odcienie, zależne mocno od oświetlenia, zanieczyszczeń atmosfery, itp. Mogą więc nieprawidłowo zostać zakwalifikowane. W tym przypadku należy odciąć zbyt ciemne piksele płaszczyzną prostopadłą do szukanego koloru.

Wynikowo otrzymujemy obcięty stożek, którego szersza podstawa leży na płaszczyźnie jasnych kolorów, a obcięta podstawa jest na płaszczyźnie obcinającej po ciemnej stronie. Stożek ma wysokość równoległą do osi szukanego koloru.

Każdy stożek jest identyfikowany przez 3 parametry.

Zmienna	Niebieski kolor	Czerwony kolor
Średnica ciemna	0,3	0,3
Średnica jasna	0,9	0,7
Płaszczyzna ciemna	0,2	0,2

Tak powstałe dwa obrazy binarne są gotowe do dalszej obróbki.

2.3 Otwieranie

Kilkukrotne wykonanie erozji, a następnie dylacji pozwala się pozbyć małych obiektów, głównie szumu, który został zakwalifikowany do koloru, a nie został usunięty przez filtr medianowy.

Głębokość otwierania zależy od wielkości obrazu, dzięki temu można w ten sam sposób przetwarzać obrazy o dowolnych rozmiarach. Długość mniejszego boku jest mnożona przez 0,003 i zaokrąglana do części całkowitej. Dla większości obrazów jest to około 3.

Otwieranie znacząco przyspiesza późniejsze wykrywanie obiektów, ale jeśli jest za duże, może uszkodzić literę na obrazie.

Algorytm stosuje się na obu obrazach binarnych. Jest zaimplementowany wielowątkowo, w identyczny sposób, jak filtr medianowy.

2.4 Znajdywanie obiektów

Najdroższa obliczeniowo część przetwarzania. Należy podzielić obraz binarny na wiele spójnych podobiektów i obliczyć charakterystyki dla każdego z nich.

Odbywa się to w pętli. Program tworzy nowy obraz, na którym zaznacza odwiedzone piksele. Za każdym razem program szuka pierwszego zapalonego piksela, idąc wierszami od lewego-górnego rogu. Po znalezieniu takiego, zaczyna rozlewać, zapamiętuje znaleziony piksel w kolejce i przechodzi do głównej pętli rozlewania:

1. Pobranie następnego piksela z kolejki.

2. Sprawdzenie, czy któryś z sąsiadów nie jest zapalony.

3. Dodanie każdego zapalonego sąsiada do kolejki.

4. Wyłączenie każdego sąsiada w oryginale.

5. Wyłączenie przetwarzanego piksela w oryginale.

6. Włączenie przetwarzanego piksela w nowym obrazie.

Nowopowstały obraz jest zwracany. Algorytm nie może być wewnętrznie zrównoleglony. Przetwarzane nim są dwa obrazy binarne.

2.5 Obliczanie cech

Dla każdego obiektu obliczany jest:

- Pełen zestaw niezmienników od $M1$ do $M10$.
- Środek geometryczny obiektu.
- Wypełnienie ekranu (wielkość), czyli ilość pikseli obiektu podzielona przez ilość pikseli obrazu.

Dzieje się to w jednym przelocie po wszystkich pikselach, więc zrównoleglenie tej operacji nie daje dużych przyspieszeń.

Następnie dla każdego obiektu obliczana jest *odległość ważona* w 10-wymiarowej przestrzeni od punktów idealnych obiektów. Wagi łączą w sobie przydatność niezmiennika przy identyfikowaniu obiektu, odchylenie standardowe cech uczących obiektów, oraz zerują nieprzydatne niezmienniki. Wielkość obiektu także jest brana pod uwagę, za małe obiekty są odrzucane.

Początkowe wartości niezmienników zostały wyznaczone przez program, następnie zmodyfikowane ręcznie w celu uzyskania najlepszych rezultatów. Wykrycie nieużytecznych niezmienników odbyło się poprzez stworzenie wykresów, porównujących interesujące obiekty i pozostałe obiekty na płaszczyźnie, dla każdej pary niezmienników. Wybrano te niezmienniki, dla których obiekty były najbardziej oddzielone od pozostałych. Zapisane są w poniższej tabeli.

Obiekt	M4	M7	M8	M9
Strzałka	0.00079014	0.000100286	0	-4.03119e-05
Wagi	1.1	1.2	0	0.4
Litera W	0	-1.54905e-05	0	-1.10516e-05
Wagi	0	1.5	0	1.5
Litera K	0.000242899	0	-3.99919e-05	-3.10566e-06
Wagi	0.8	0	0.8	1
Litera D	0	0	-5.13758e-05	-5.55874e-06
Wagi	0	0	1	0.6

2.6 Sortowanie

Odległości ważone są sortowane rosnąco dla zbioru odległości dla każdego obiektu. Na obrazku roboczym zaznaczane są obiekty o najlepszym dopasowaniu.

Jednak ze względu na różne obrazy, nie zawsze najbardziej dopasowane obiekty muszą być tymi prawidłowymi. W szczególności wśród niebieskich obiektów może się zdarzyć, że te same obiekty zostaną zakwalifikowane do jednego zbioru. Na przykład, drugim najlepszym dopasowaniem niebieskiej strzałki jest obiekt, który test także pierwszym najlepszym dopasowaniem litery W.

2.7 Szukanie konfiguracji

Przeszukując pierwsze 5 najlepszych dopasowań, program liczy odległości pomiędzy obiektami i szuka takiej konfiguracji, że spełnia ona pozycje obiektów w logo. Jest to tym kosztowniejsza operacja, im głębiej decydujemy się przeszukiwać listy obiektów. Następnie liczony jest środek loga i zapisywany w buforze razem z punktacją konfiguracji. Znalezione dopasowania są sortowane względem liczby punktów, jakie zyskują za spełnianie podobieństw odcinków.

Najlepsze dopasowanie jest ustawiane na wyjście.

Zauważono, że promowanie bliskości obiektów, daje najlepsze rezultaty. Algorytm przydziela odwrotnie proporcjonalną ilość punktów do odległości pomiędzy strzałkami i pomiędzy strzałkami, a literą K.

Zwiększanie głębokości przeszukiwania pozwala wykryć kilka logów, ale także zwiększa prawdopodobieństwo znalezienia nieprawidłowego loga.

Algorytm jest odporny na przypadek niewykrycia jednej ze składowych, każde porównanie porównuje odległości z daną składową identyczną ilość razy. To powinno zwiększać wykrywanie loga, jeśli będą problemy z dopasowywaniem cech obiektów we wcześniejszych krokach.

3 Wyniki

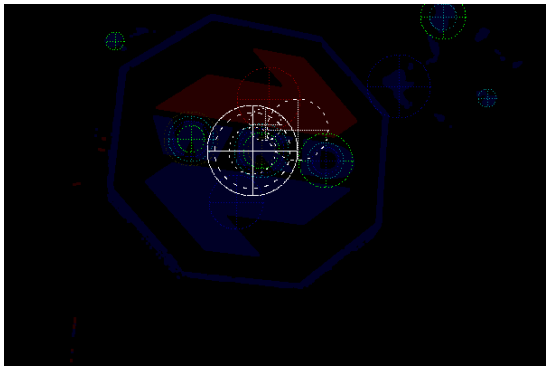
W ogólnym przypadku program zawsze zwróci wynik. Nawet, jeśli na obrazie nie ma loga WKD. Zwyczajnie znajdzie najlepsze dopasowanie do najlepszych obiektów. Może nie zwrócić wyniku, jeśli na przykład żaden obiekt nie będzie spełniał wymogów minimalnej wielkości, w szczególności strzałki.

Na lewym obrazku jest wejściowy obraz z zaznaczonym znalezionym logiem.

Na prawym obrazku są zaznaczone obiekty, wyznaczone w trakcie pracy. Małymi, przerywanymi okręgami zaznaczono wstępne dopasowania poszczególnych obiektów. Im większy okrąg, tym lepsze dopasowanie. Odcienie zielonego oznaczają kolejne litery.

Białym krzyżem zaznaczono środek, czyli punkt między czerwoną, a niebieską strzałką w najlepszym dopasowaniu. Przerywanymi białymi krzyżykami oznaczono najlepsze kolejne dopasowania logów. Przy dostatecznie dużej głębokości porównywania i dobrym algorytmie szukania konfiguracji, będą to kolejne znalezione na obrazie loga.

(a) Obraz wyjściowy



(b) Znalezione obiekty

4 Możliwości rozwoju

Ponieważ program jest napisany obiektowo, z dużą ilością abstrakcji, możliwa jest implementacja innego algorytmu w oparciu o istniejące klasy.

Można zrównoleglić poszukiwanie obiektów do dwóch wątków, co na wielordzeniowym komputerze przyspieszy działanie. Takich mniejszych akcji, jak sortowanie, nie trzeba przyspieszać, gdyż i tak dzieją się wystarczająco szybko, w porównaniu z innymi obliczeniami.

Wykrywanie kilku logów na raz jest możliwe do dodania. Do tego jest wymagane pogłębienie szukania wśród obiektów, co jednak może wprowadzić więcej nieprawidłowych dopasowań. Potrzebny jest sposób oceny granicy między prawidłowymi znaleziskami pasujących konfiguracji, a fałszywymi.

Przede wszystkim, program powinien posiadać lepszy algorytm ustalania konfiguracji obiektów. Można brać pod uwagę nie tylko pozycje znalezionych obiektów, ale także ich wzajemne wielkości. Należy na tej podstawie wykryć nieprawidłowe dopasowania liter jako niebieskiej strzałki. Trzeba jednak wciąż mieć na uwadze odporność algorytmu na brakujące składowe. Stworzenie takiego algorytmu jest czasochłonne, gdyż ciężko przewidzieć rezultaty.

Aby ulepszyć jakość wykrywania, warto by użyć jakiegoś zaawansowanego systemu do obliczania parametrów programu. Przy ręcznym wskazywaniu obiektów do obliczeń cech, każda bardziej znacząca zmiana algorytmu może powodować różne wyznaczenie kolejnych obiektów i potrzebę zaktualizowania numerów obiektów.