Wykrywanie migacza samochodu zjeżdżającego z kończącego się pasa ruchu

Piotr Majorczyk

13 czerwca 2016

Streszczenie

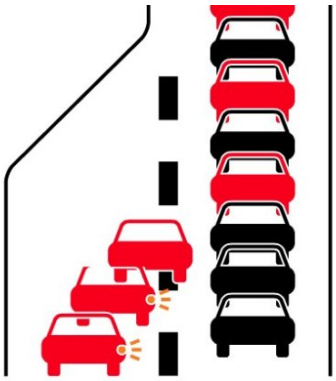
W opracowaniu przedstawiono kroki podjęte w celu poprawy działania algorytmu wykrywającego migacz pojazdu zjeżdżającego z kończącego się pasa ruchu oraz proces powstania bazy nagrań służącej do testowania stworzonych funkcjonalności.

Spis treści

1 Wstęp

1.1 Cel projektu

Celem projektu było usprawnienie istniejącego kodu wykrywającego migacz auta poprzedzającego nasz pojazd w celu usprawnienia tz. jazdy na suwak/zamek błyskawiczny. Jazdą na suwak nazywa się metodę dojazdu do zwężenia jezdni,polegającym na kontynuowaniu ruchu na obu pasach do samego momentu zwężenia, a następnie przepuszczania przez każdy pojazd jednego pojazdu z pasa, który się kończy. W niektórych krajach jazda na suwak wymuszana jest przez przepisy kodeksu drogowego.



Rysunek 1: Schemat ideowy jazdy na suwak

1.2 Sytuacja wyjściowa

Jak już wspomniano na wstępie, projekt polega na rozwinięciu kodu źródłowego wykrywającego migacze który dostępny jest na stronie http://dsp.org.pl. Plik źródłowy zawierający rzeczony kod jest również umieszczony na płycie dołączonej do opracowania.

2 Etapy prac

Prace w projekcie zostały wykonane według poniższej kolejności:

1. Zapoznanie się z gotowym kodem
2. Przygotowanie materiału wideo
3. Przeprowadzenie testów skuteczności gotowego kodu
4. Wprowadzenie zmian usprawniających działanie algorytmu
5. Testy końcowe

W dalszej części opracowania poszczególne kroki wykonywania projektu zostały szczegółowo opisane.

2.1 Przygotowanie materiału źródłowego

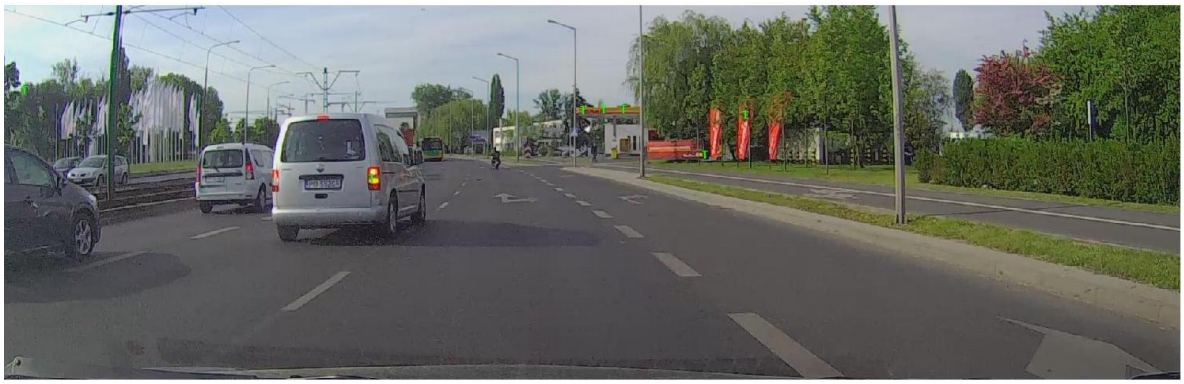
Materiał źródłowy do testów skuteczności działania programu pochodzi z wideorejestratora samochodowego MIO MiVue 538 montowanego centralnie w górnej części przedniej szyby samochodowej. Nagrania charakteryzują się następującymi parametrami technicznymi:

- Liczba klatek na sekundę nagrania: **29,97 fps**
- Rozdzielczość: **1920x1080**
- Format danych: **.MOV**

Nagrania zostały następnie zdekompresowane do formatu .avi, pozbawione ścieżki dźwiękowej oraz przycięte do długości 1-2s. Zredukowano również liczbę klatek na sekundę do 27fps. Wszystkie te operacje wykonano za pomocą programu Virtual Dub. Ostatecznie udało się zebrać 10 różnych materiałów wideo które posłużyły do testowania kodu. Co ważne fragmenty wideo były wykonywane przy różnych warunkach atmosferycznych i oświetleniowych co pozwoli na ocenę jakości kodu również pod względem czułości na te czynniki.

2.2 Testy skuteczności zastanego kodu

Kolejnym krokiem było dokonanie testów skuteczności działania zastanego kodu dla zebranego materiału wideo. Dla arbitralnie wybranych pojedynczych klatek z każdego z 11 filmów migacz udało się wykryć dla 4 na 10. Na wynik wpływ może mieć fakt, że kod programu został napisany dla kamery zamontowanej na zewnątrz samochodu a testy przeprowadzone zostały dla kamery wewnątrz pojazdu. Dwie udane oraz dwie nieudane próby zostały zaprezentowane na odpowiednio rysunkach 2, 3, 4 oraz 5.



Rysunek 2: Ilustracja działania przed zmianami

Przeprowadzone testy pomogły w zdefiniowaniu podstawowych problemów stojących na drodze do poprawnego wykrywania migaczy. Były to między innymi:

- Zmienne warunki oświetleniowe,
- zabrudzona szyba przednia,
- algorytm stworzony do kamery montowanej na zewnątrz,
- występowanie w kadrze elementów czerwono-żółtych które imitują (dla programu) światła pozycyjne oraz migacze,
- brak ustandaryzowanych kształtów oraz położeń świateł tylnych samochodów,
- zmatowienie świateł tylnych niektórych pojazdów.

Rysunek 3: Ilustracja działania przed zmianami

Rysunek 4: Ilustracja działania przed zmianami

2.3 Dodane funkcjonalności

2.3.1 Przycięcie obszaru nagrania

Pierwszą funkcjonalnością jaka została wprowadzona było zautomatyzowanie przycinania klatek wideo w celu zminimalizowania fałszywych wykryć migacza jak i zmniejszenia rozmiaru danych poddanych analizie. Funkcja przyjmuje jako argument plik graficzny a zwraca macierz reprezentującą przycięty obraz.

```
function [M]=crop(obraz)
x_min = 0;
y_min = 430;
szer = 1900;
wys = 450;
przytnij_v = [x_min y_min szer wys];
I = imread(obraz);
M = imcrop(I, przytnij_v);
```

2.3.2 Wczytanie wideo i zapis pojedynczych klatek

Drugą funkcjonalnością jaka została wprowadzona było wczytanie pliku .avi i zapisanie jego pojedynczych klatek to plików .bmp. Dodatkowo stworzono wektor nazw klatek w celu usprawnienia odwoływania się do poszczególnych plików. Funkcja przyjmuje jako argument film w formacie .avi z jakiego mają zostać wyciągnięte pojedyncze klatki. Zwraca liczbę zapisanych klatek oraz wektor indeksów nazw plików z klatkami.

Rysunek 5: Ilustracja działania przed zmianami

```
function [B,ilosc] = wczytaj(film)
B=[]; %wektor nazw klatek
mov = aviread(film);
rozm=size(mov);
ilosc=rozm(2);
for a=1:1:ilosc
    obrazek=mov(1,a).cdata;
    if a<10
        nazwa=['klatka000', num2str(a), '.bmp'];
    else
    if (a>=10 && a<100)
        nazwa=['klatka00', num2str(a), '.bmp'];
    else
    if (a>=100 && a<1000)
        nazwa=['klatka0', num2str(a), '.bmp'];
    else
        nazwa=['klatka', num2str(a), '.bmp'];
    end
    end
    end
imwrite(obrazek,nazwa,'bmp');
%obrazek2=crop(obrazek1);
B = [B; nazwa];
end
end
```

2.3.3 Tworzenie mapy zmian pomiędzy klatkami

Kolejną funkcjonalnością dodaną w celu usprawnienia działania algorytmu wykrywania migaczy był kod pozwalający na tworzenie map zmian pomiędzy klatkami x oraz x-2. Dzięki tej funkcjonalności udało się w znacznym stopniu zredukować fałszywe wykrycia migaczy. Jest to związane z tym, że w pierwotnym kodzie wyszukiwano obiekty odpowiadające migaczom pod kątem barwy a nie brano pod uwagę „zmiennej" natury migacza i z zasady „niezmiennej" natury elementów otoczenia (z wyłączeniem elektronicznych bilbordów itp.). Najlepsze działanie tego algorytmu można zaobserwować gdy pojazd z którego wykonywane jest nagranie nie porusza się dzięki czemu otoczenie nie zmienia się. Jest to również sytuacja najbliższa jeździe na suwak. Funkcja jako argumenty przyjmuje plik .avi a zwraca maskę zmian.

```
function [Z]=porownanie(obraz1,obraz2)
obraz1=crop(obraz1);
obraz2=crop(obraz2);
```

```
obraz1 = uint8(obraz1);
obraz2 = uint8(obraz2);
Z = imsubtract(obraz1,obraz2);
end

function [Bin8] = binaryzacja(film)
B = wczytaj(film);
%Odjęcie od siebie kolejnych klatek
mapa=porownanie(B(13,:),B(11,:)); %przykładowe dwie klatki dla wyboru statycznego
%Konwersja do skali szarości
I = rgb2gray(mapa);
%Binaryzacja
BW = im2bw(I, 0.1);
%Stworzenie elementu strukturalnego i operacja otwarcia w celu usunięcia
%zanieczyszczeń
se = strel('disk',1);
Bin = imopen(BW,se);
%Operacja zamknięcia w celu zgrupowania i uszczelnienia
se = strel('disk',5);
Bin = imclose(Bin,se);
%Zmiana typu zmiennej z logical na uint8
Bin8 = im2uint8(Bin);
end
```

2.3.4 Zapisanie wyników jako sekwencji klatek

Ostatnią z funkcjonalności jaką udało się wprowadzić było zapisanie sekwencji klatek wraz z naniesionymi znacznikami wykrycia wykrycia migaczy. Zostało to zrealizowane przy pomocy funkcji *saveas*.

```
saveas(gcf, sprintf('\%d.jpg', a-11));
```

2.4 Testy

Po przetworzeniu sekwencji klatek stworzono z nich za pomocą zewnętrznego programu pliki .gif w celu przystępnej prezentacji wyników. Pliki te zostały zamieszczone na płycie dołączonej do opracowania pod nazwami *Przykład_xxx.gif* gdzie xxx jest numerem próby. Na podstawie tych plików opracowano wyniki statystyczne dotyczące sprawności działania algorytmu. Ostatecznie udało się wykryć 8 na 10 migaczy przy jednoczesnym zmniejszeniu fałszywych wykryć.

3 Efekty końcowe

Ostateczny kod realizujący wszystkie wymienione wyżej funkcje prezentuje się następująco:

```
function [C] = calosc_film(film)
C=[];
h_upper=round(0.01*255); %130;
h_lower=round(0.93*255); %110;
s_upper=round(1.00*255);
s_lower=round(0.27*255);
v_upper=round(1.00*255);
v_lower=round(0.31*255);
%Górne i dolne granice w przestrzeni HSV kierunkowskazów
turn_h_lower=round(0.05*255); %140; %160
```

```
turn_h_upper=round(0.24*255); %190
turn_s_lower=(0.55*255); %140
turn_s_upper=(1.00*255); %255
turn_v_lower=round(0.55*255); %140
turn_v_upper=round(1.00*255); %255
turn_r_upper=5;
%wczytanie filmu
[B, ilosc] = wczytaj(film);
for a=12:1:ilosc
    klatka1 = B(a,:);
    klatka2 = B(a-2,:);
    %Porównywanie klatek
    mapa = porownanie(klatka1,klatka2);
    %Konwersja do skali szarości
    mapa = rgb2gray(mapa);
    %Binaryzacja
    mapa = im2bw(mapa, 0.1);
    %Stworzenie elementu strukturalnego i operacja otwarcia w celu usunięcia
    %zanieczyszczeń
    se = strel('disk',1);
    Bin = imopen(mapa,se);
    %Operacja zamknięcia w celu zgrupowania i uszczelnienia
    se = strel('disk',5);
    Bin = imclose(Bin,se);
    %Zmiana typu zmiennej z logical na uint8
    Bin8 = im2uint8(Bin);%różnice miedzy klatkami w postaci binarnej
    %rysowanie map migaczy
    [map_turn,map] = migacz(klatka1); %12 klatka
    %maskowanie obrazów
    map_turn(~Bin8) = 0;
    klatka1 = crop(klatka1);
    cc = bwconncomp(map);
    %Pobranie parametrów obiektów
    stats = regionprops(cc, {'Centroid', 'Area', 'EquivDiameter'});
    %Liczba kierunkowskazów
    n=0;
    %Sprawdzenie wszystkich kombinacji
    for i=1:cc.NumObjects
        %Inicjalizacja tymczasowej mapy migacza
        map_turn_temp=zeros(size(map_turn));
        %migacz, zależnej promienia światła wsp. turn_r_upper
        x=floor(stats(i).Centroid(1) - turn_r_upper * stats(i).EquivDiameter/2)...
          : floor(stats(i).Centroid(1) + turn_r_upper * stats(i).EquivDiameter/2);
        y=floor(stats(i).Centroid(2) - turn_r_upper * stats(i).EquivDiameter/2)...
          : floor(stats(i).Centroid(2) + turn_r_upper * stats(i).EquivDiameter/2);

        %Ograniczenie obszaru do rozmiaru obrazu
        x( x>size(map_turn,2 ))=[];
        y( y>size(map_turn,1 ))=[];
        x( x<1 )=[];
        y( y<1 )=[];

        %Utworzenie mapy obiektów migacza w wyznaczonym obszarze
        map_turn_temp(y,x)=map_turn(y,x)*1;
        %Wyszukanie obiektu migacza
        turn_light=bwconncomp( map_turn_temp );
        %Jeśli znaleziono obiekt zapisanie go do macierzy kierunkowskazów

        if turn_light.NumObjects~=0
        n=n+1;
```

```
            temp_stats = regionprops(turn_light, 'Centroid');
            lights_turn(n,1)=temp_stats(1).Centroid(1);
            lights_turn(n,2)=temp_stats(1).Centroid(2);
            end
        end
figure(a-11), imshow(klatka1)
        for i=1:n
            text(lights_turn(i,1), lights_turn(i,2), 'T', 'Color', 'g','FontWeight', 'bold');
        end
        saveas(gcf, sprintf('%d.jpg', a-11));
        C = [C; a];
        close all
end
```

Jak to zostało wspomniane w rozdziale 2.4 animacje prezentujące działanie powyższego kodu zamieszczone są na płycie dołączonej do opracowania.

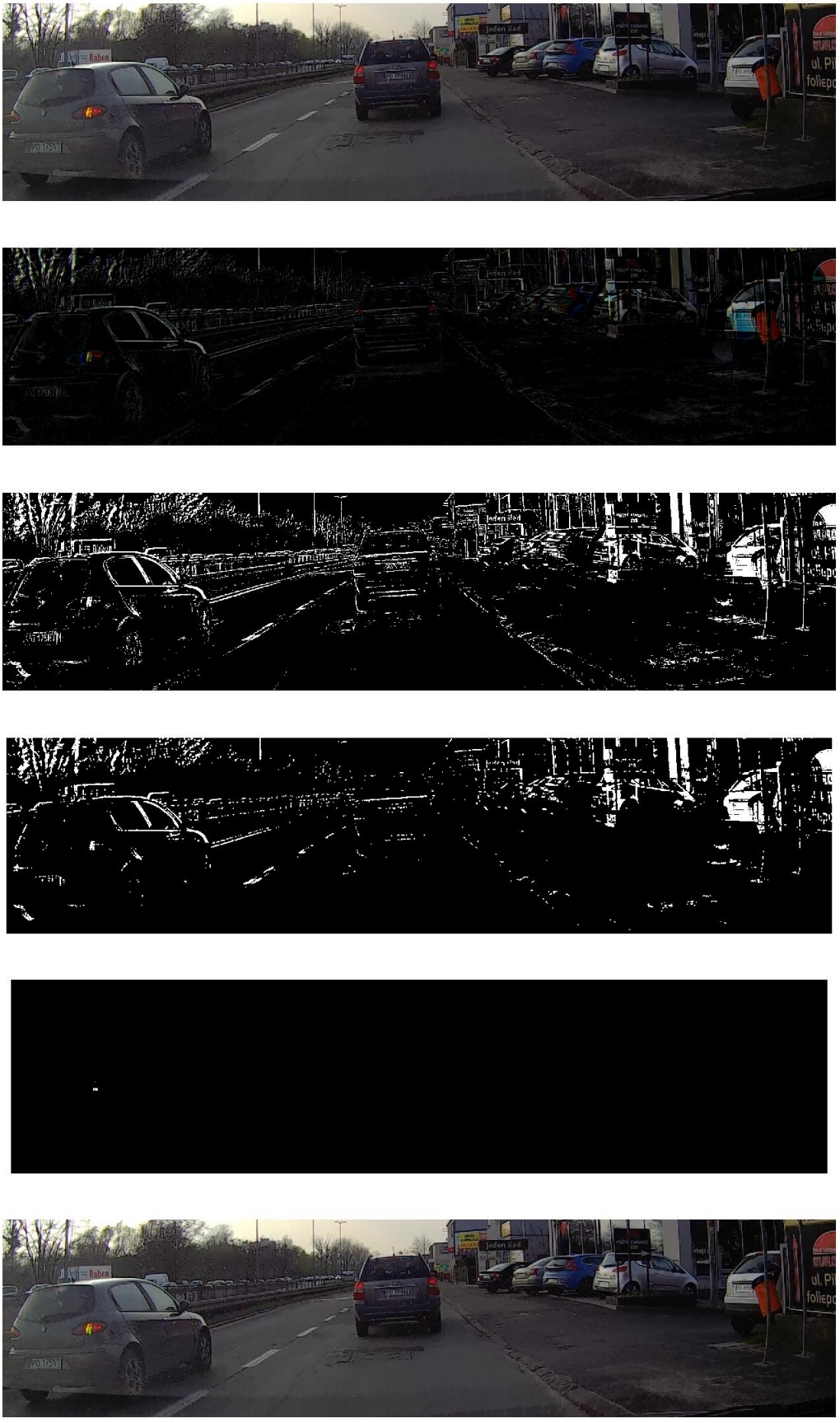
Program po kolei wykonuje następujące operacje:

1. Przycięcie obrazu orginalnego
2. Odjęcie od siebie klatki x oraz x-2
3. Binaryzacja otrzymanego obrazu
4. Morfologiczne operacje otwarcia oraz zamknięcia
5. Maskowanie mapy migaczy za pomocą mapy zmian
6. Wyświetlenie zielonego znacznika „T” w miejscu wykrycia migacza

Operacje te zostały zilustrowane na przykładzie jednej wybranej klatki i zaprezentowane na rysunku 6

4 Podsumowanie

Ostateczny efekt wprowadzonych zmian można uznać za zadowalający. Szczególnie ważną zmianą jest przejście z analizy pojedynczej klatki na analizę całego filmu. Otworzyło to możliwości analizy zmian zachodzących w obrazie a nie tylko jego statycznych właściwości. Trzeba zauważyć, że samego rozwiązania problemu nie ułatwiają firmy z branży automotive. Migacze nie tylko przyjmują najróżniejsze kształty lecz również różne odcienie kolorystyczne. W związku z tym łatwo zignorować migacz będący innego koloru niż żółty. Problem ten wymusza poszerzanie zakresu wykrywanych kolorów co prowadzi do większej liczby fałszywych wykryć. W celu ich redukcji trzeba wprowadzać dodatkowe rozwiązania pozwalające na odróżnienie migaczy od otoczenia. Jednym z rozwiązań jest porównywanie zmian zaprezentowane w niniejszym opracowaniu. Autor uważa, że sam temat jest mocno rozwojowy i można by usprawnić działanie programu. Dwie idee zostały porzucone w trakcie tworzenie projektu. Jedną z nich było wykrywanie powierzchni jezdni i ograniczenie wykrywania migaczy do obszaru w niewielkiej odległości od jezdni. Niestety z powodu zniekształcenia wynikającego z perspektywy, algorytm ten nie ograniczałby prawie wcale obszaru wyszukiwania (horyzont i jezdnia stykają się w pewnym punkcie przez co prawie wszystko jest „blisko” jezdni). Drugim z porzuconych pomysłów było wykrywanie obiektów okrągłych (kół pojazdów) jednak ze względu na zniekształcenia wynikające z perspektywy nie było to możliwe do wykonania. Wg. autora opracowania rozsądnym kierunkiem prac mogłoby być podzielenie obrazu na obszary i uśrednianie wyników wykryć. Wyeliminowałoby to przypadkowe pojedyncze wykrycia wynikające np. z odbić światła.



Rysunek 6: Ilustracja działania całego programu na przykładzie jednej klatki

Literatura

[1] Julian Balcerek, Adam Konieczka, Tomasz Marciniak, Adam Dąbrowski, Krzysztof Maćkowiak, Karol Piniarski *Automatic detection of traffic lights changes from red to green and car tum signals in order to improve urban traffic*

[2] *Matlab Help* Dostępne online: http://www.mathworks.com/

[3] *Materiały pomocnicze - wstęp do środowiska MATLAB* Dostępne online: http://dsp.org.pl