POLITECHNIKA ŚLĄSKA

WYDZIAŁ AUTOMATYKI, ELEKTRONIKI I INFORMATYKI

Wizyjne Systemy Sensoryczne

Lab. nr 7 i 8:
Analiza ruchu w sekwencjach video

Praca na zajęciach

Data laboratorium: 6.12.2021 r.
13.12.2021 r.

AiR S2-I/Rob

Skład sekcji:

Klaudia Bończyk

Paweł Kaźmieruk

Maksymilian Skibiński

1 Wstęp

W trakcie zajęć laboratoryjnych naszym zadaniem było napisanie programu służącego do obliczania prędkości pojazdów, których ruch został zarejestrowany na zamieszczonych materiałach wideo. Inną propozycją prowadzących było wyposażenie programów w dodatkową funkcję, wyznaczającą zawartość tablic rejestracyjnych wspomnianych pojazdów, której zresztą nie widać. Ostatecznie nie podjęliśmy próby realizacji tego celu i skupiliśmy tylko na celu podstawowym.

2 Program

Kod programu został zamieszczony na rysunkach 1, 2 i 3.
Program składa się z funkcji *Main* i 4 makrofiltrów:

- *Odczyt_Wideo* dla wybranego materiału wideo i numeru klatki początkowej daje na wyjściu 3 obrazy, które są kolejnymi trzema klatkami wideo.
- *Top* i *Bottom* służą do lokalizacji geometrycznych środków pojazdów. Jak nazwa wskazuje, jeden z filtrów zajmuje się pasem górnym, a drugi pasem dolnym (dlaczego zostanie wyjaśnione później). Różnią się one jedynie regionem na którym operują filtry.
- *Points* wyznacza prędkości dla podanych wektorów punktów.

Funkcja *Main* poza kontrolą pracy używanych makrofiltrów, zajmuje się także naniesieniem na obraz prędkości.

Makrofiltr *Odczyt_Wideo*

Kod tego makrofiltru widoczny jest na rys. 2a. Jest to najmniej skomplikowany makrofiltr. Jest on używany na samym początku programu. Podajemy ścieżkę do obrazu i numer pierwszej wykorzystywanej klatki wideo. Filtr w środku używa aż trzech funkcji *ReadVideo*, które czytają ten sam materiał wideo, ale odczyt przesunięty jest o jedną klatkę. Nasz program potrzebuje aż odczytywać wideo w ten sposób gdyż:

- różnica pomiędzy obrazami różniącymi się klatkę wykorzystywana jest by usunąć nieruchome tło z obrazu, a pozostawić jadące pojazdy,
- różnica pomiędzy obrazami różniącymi się klatkę jest także wykorzystywana do wyznaczenia przemieszczenia.

Makrofiltry *Top* i *Bottom*

Te makrofiltry widoczne są na rysunkach 3b i 3a i jak już zostało wspomniane różnią się tylko parametrami jednego bloku (1. *CreateBoxRegion*), być może zatem dałoby się to jakoś zamknąć w jednym makrofiltrze, w którym jednym z argumentów byłaby informacja o wybranym filtrze.

1. Makrofiltr odczytuje dwie klatki wideo, które są przesunięte o jedną klatkę. Wyznaczana jest ich różnica, by usunąć tło z obrazu, a pozostawić pojazdy, które przemieściły się w tym czasie.

2. Następnie używamy progowania dynamicznego do znalezienia krawędzi. Wykonywane jest to jedynie na obszarze wskazanym poprzez *CreateBoxRegion*. Obszar ten wskazuje jeden z pasów na jezdni. Pas ten jest poszerzony tak by samochody po nim jadące znalazły się w regionie w całości i nie stykały się z jego granicami.

3. Na regionie korzystamy z morfologicznego domknięcia, by usunąć drobne nic nie znaczące fragmenty, a jednocześnie domknąć fragmenty istotne tzn. jadące pojazdy. Operacja ta wykonywana jest niesymetrycznie – elipsa będąca jądrem jest 5-krotnie dłuższa wzdłuż osi poziomej.

4. Region jest dzielony na osobne podregiony, a małe regiony zostają usunięte. Powinny być to nieistotne pozostałości po poprzednich przekształceniach.

5. Regiony, które jeszcze żyją na tym etapie trafiają do funkcji, która pozwala żyć tylko tym, które nie stykają się z granicami.

6. Dla tych regionów wyznaczane są środki geometryczne, które zostają wyrzucone na wyjściu i następnie posłużą do obliczenia prędkości.

Regiony stykające się z granicami zostają usnięte, gdyż w przypadku, gdy pojazd wjeżdża lub wyjeżdża z widoku kamery gwałtownie zmienia swój rozmiar, przez co środek geometryczny nie niesie w poprawny sposób informacji o przemieszczeniu. Poza tym pojazdy, które jadą na przeciwnym pasie nie powinny być brane pod uwagę przez ten filtr.

Dlaczego mamy dwa odrębne makrofiltry zajmujące się pasami osobno, a nie jeden duży będzie bardziej jasne przy opisie filtru *Points*.

Makrofiltr *Points*

Kod widoczny jest na rys. 2b.

Filtr otrzymuje dwa wektory punktów opisujących położenie pojazdów z chwil różniących się jedną „chwilą" (jedną klatką) i wyznacza na tej podstawie prędkość. Poza prędkość zwraca na wyjściu także wektor punktów, tak by jasne było, którego samochodu dotyczy która prędkość.

1. W pierwszej kolejności wyznaczamy rozmiary obu wektorów. Jeśli oba są tych samych rozmiarów to *prawdopodobnie* parę kolejnych funkcji nie ma znaczenia, ale nie zawsze tak jest. Z tych rozmiarów wybierany jest mniejszy.

2. Jeśli wektory różnią się rozmiarem to jeden z nich musi zostać skrócony (chyba zawsze o jeden punkt), podczas gdy ten uboższy w punkt/y pozostanie nie tknięty. Za chwilę powrócimy do tego fragmentu.

3. Następnie wyznaczany jest dystans pomiędzy punktami.

4. Dystans ten jest przemnażany przez odpowiednią liczbę, by przejść z różnicy w pikselach na prędkości.

5. Wynik jest zaokrąglany – niepotrzebujemy takiej dokładności – i przekształcany na typ string, który zostanie wykorzystany później na etapie nanoszenia prędkości na obraz.

Kluczowa jest pierwsza połowa makrofiltru i to nad nią dumaliśmy przez dłuższy czas. Problem stanowią sytuacje gdy któryś z samochów na poprzedniej klatce jest, a na aktualnej go nie ma (tzn. chodzi o punkty je reprezentujące) albo na odwrót. Wtedy wektory punktów po pierwsze różnią się liczbą punktów, a po drugie – i to jest najgorsze – wektory niekoniecznie są *zsynchronizowane*, te same indeksy dotyczą innych pojazdów.

Synchronizacja wektorów była dla nas bardzo trudna. Jesteśmy zupełnie pewni, że bylibyśmy sobie łatwo poradzić z tym zadaniem w MATLABie czy C, gdzie dobrze potrafimy wykonywać operacje na tablicach i korzystać z pętli. W tym środowisku nie wszystko jest dla nas na tym etapie wystarczająco jasne.

By dokonać synchronizacji wprowadzamy następujące założenie: samochody na górnym pasie pojawiają się z prawej strony, a znikają z lewej, natomiast samochody na dolnym pasie na odwrót. Założenie to jest całkiem sensowne, choć w przypadku, gdyby np. któryś z kierowców jechał pod prąd, nasz program nie działałby w założony przez nas sposób i pokazywał wygórowane prędkości[1]. Założenie to oznacza, że jeden z wektorów zawsze będzie tracił elementy na początku, a drugi elementy na końcu. By usuwać elementy z końca korzystamy w bardziej zawiły sposób przy pomocy dwukrotnego odwracania zawartości wektorów poprzez *ReverseArray*.

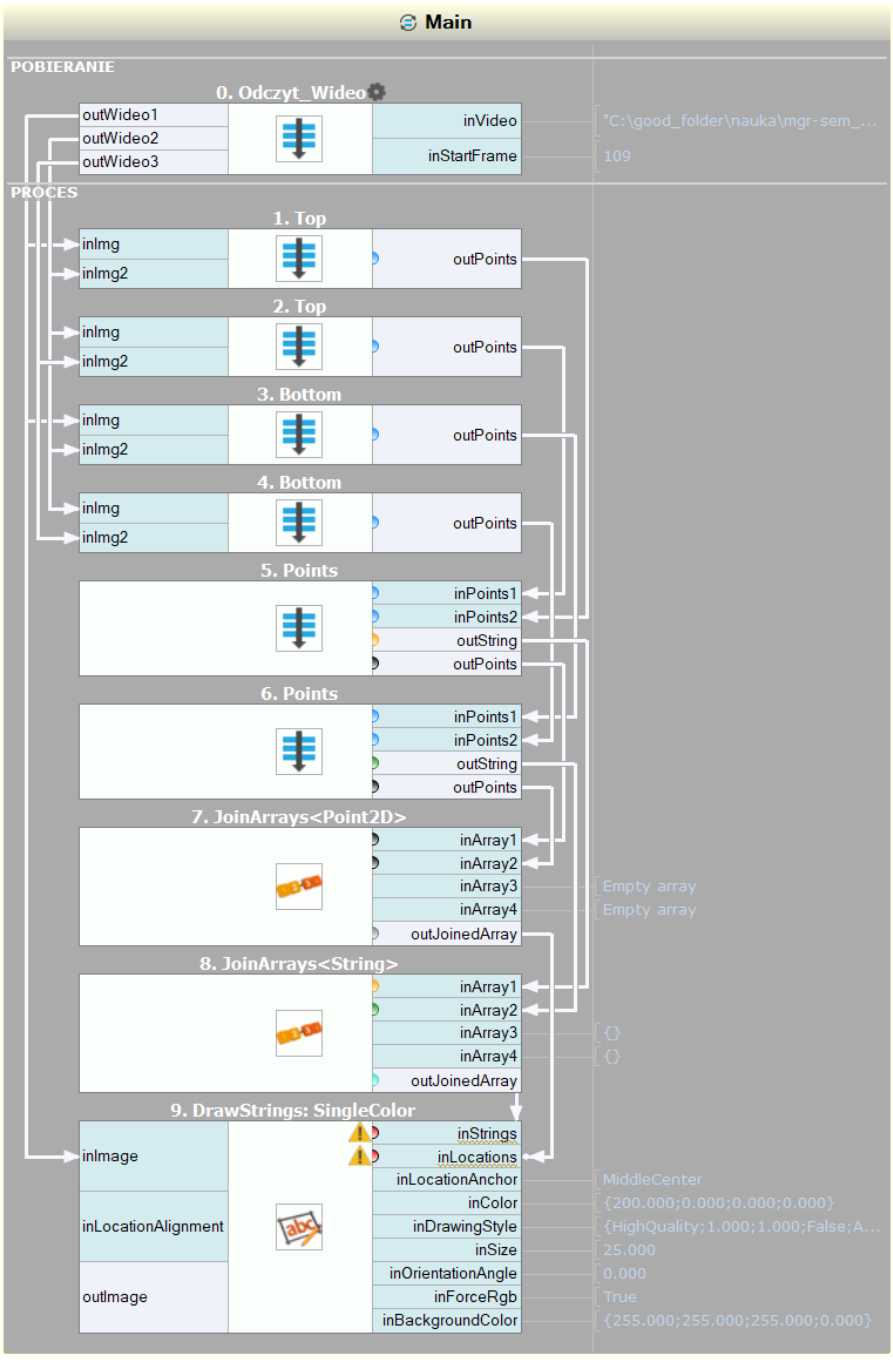
Poza tym, być może ta funkcja mogłaby być lepiej nazwana...

Funkcja *Main*

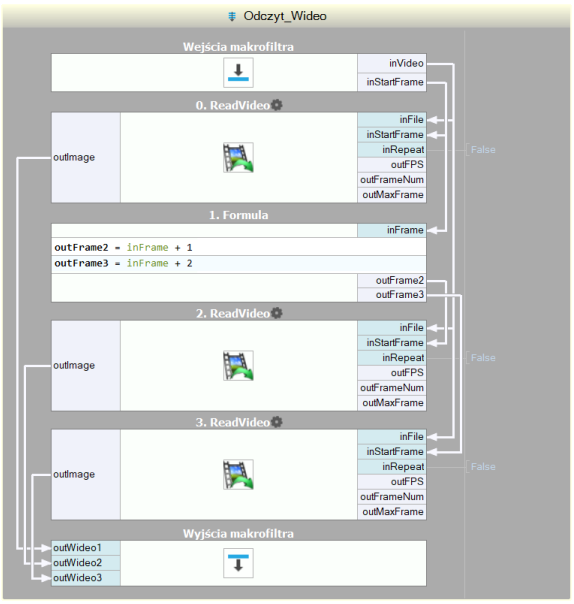
Funkcja główna *Main* (rys. 1) koordynuje pracę tych wszystkich funkcji i na końcu nanosi wyniki na obraz. Jako że, ze względu na podział programu na pas górny i dolny, otrzymujemy dwa osobne wektory prędkości i punktów są one wpierw łączone poprzez *JoinArrays*.

Warto zwrócić uwagę na jedną rzecz – na połączenie wyjść makrofiltrów *Top* i *Bottom* z filtrami *Points*. Są one połączone w *odwrotnej* kolejności. Wynika to z omawianych wcześniej założeń i sposobu działania makrofiltru *Points*. W tych wektorach punktów usuwania elementów musi się odbyć w odwrotnej kolejności temu zostało zastosowane widoczne połączenie.

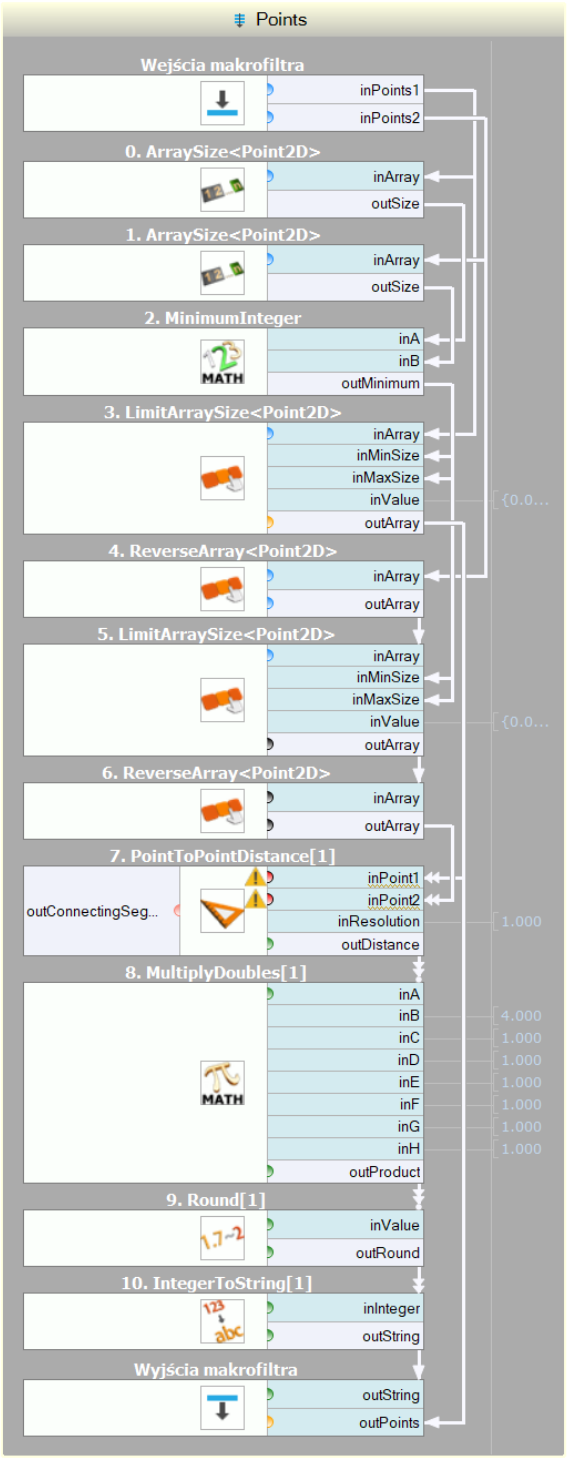
[1] Z drugiej strony, skoro ktoś jedzie pod prąd to może nie powinien nas boleć fakt, że jego prędkość zamiast pokazać jakąś faktyczną 50 km/h pokaże 1000 km/h? Zawsze można to potraktować jako słuszny alarm i powód do mandatu.



Rysunek 1: Funkcja *Main*

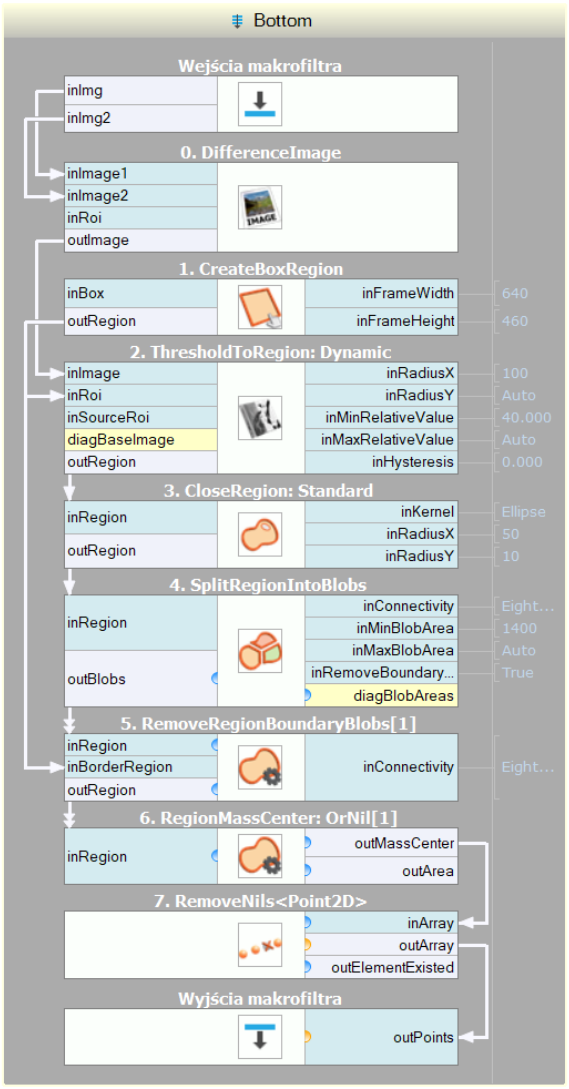


(a) makrofiltr *Odczyt_Wideo*

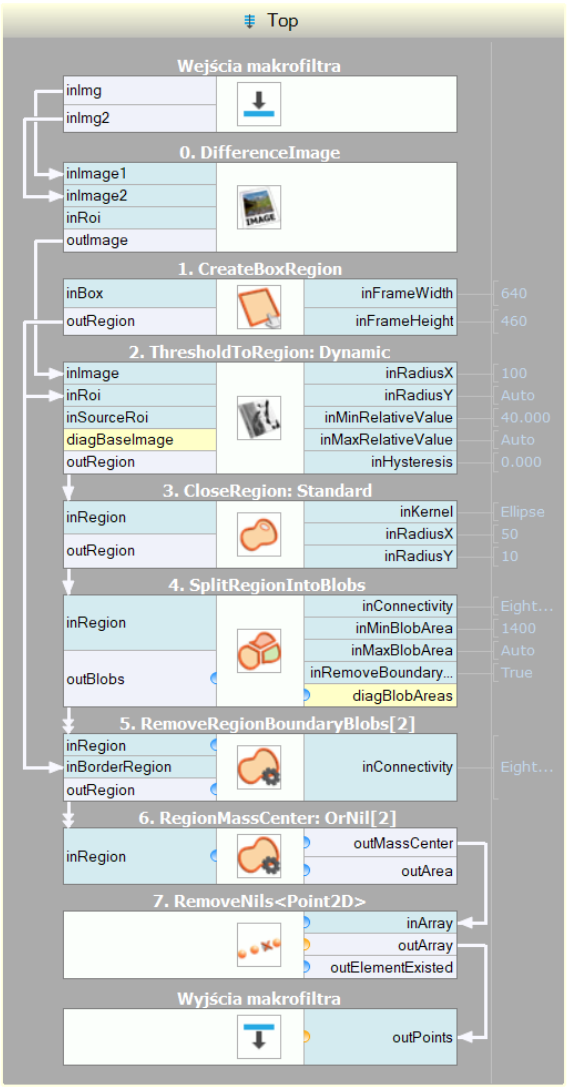


(b) makrofiltr *Points*

Rysunek 2: Makrofiltry cz. 1



(a) makrofiltr *Bottom*

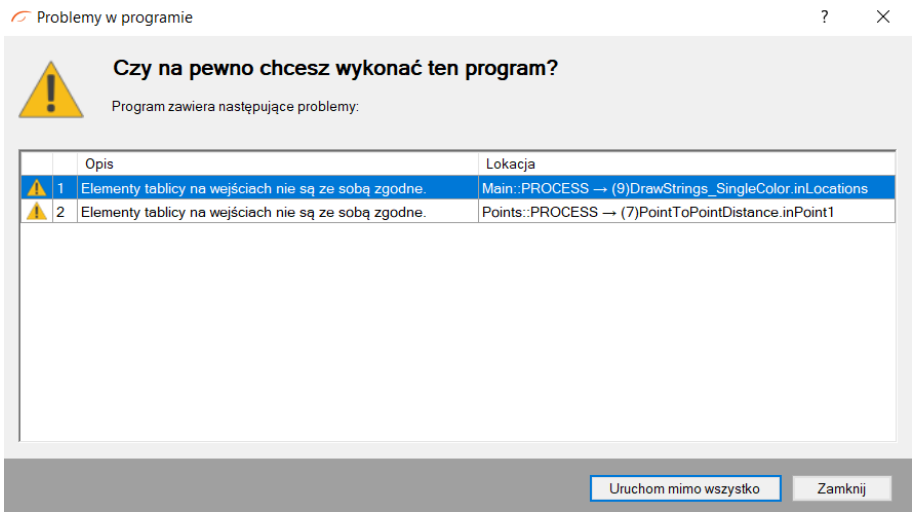


(b) makrofiltr *Top*

Rysunek 3: Makrofiltry cz. 2

3 Działanie

Zacząć można od informacji o tym, że program wyrzuca ostrzeżenia podczas próby uruchomienia go.

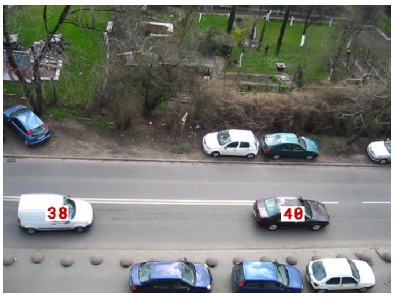


Rysunek 4: Ostrzeżenie

Dotyczy ono funkcji wyznaczającej dystans pomiędzy punktami i tej która nanosi prędkości na obraz. Nie udało nam się ani zrozumieć z czego to wynika, ani zauważyć kiedy dochodzi do jakichś problemów. Zignorujmy to.

Działanie naszego programu można ujrzeć na przykładowych obrazach na rysunku 5.

(a)



(b)

(c)

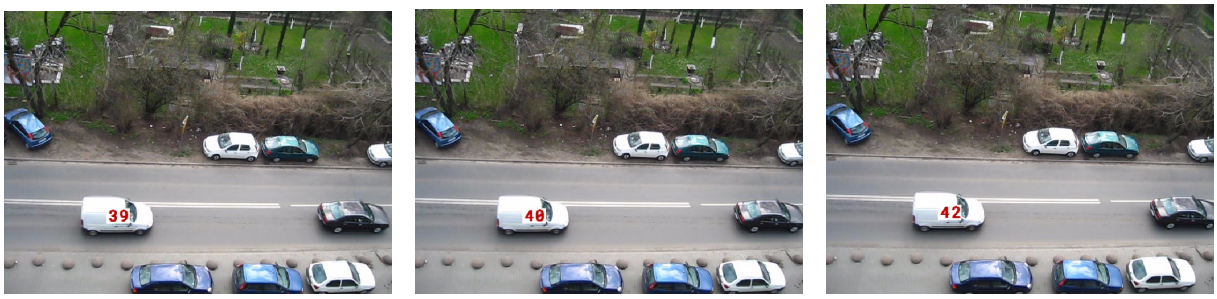
(d)

(e)

Rysunek 5: Działanie programu

Jak widać gdzieś na jadących samochodach znajduje się informacja o ich prędkości.

Dla mniej skomplikowanych sytuacji np. gdy przejeżdża tylko jeden samochód program działa raczej poprawnie. Pokazuje pewną prędkość, która podlega pewnym zmianom, ale raczej znajduje się na stałym poziomie. Przykładowo trzy kolejne klatki można ujrzeć na rys. 6.



(a) (b) (c)

Rysunek 6: Trzy kolejne klatki

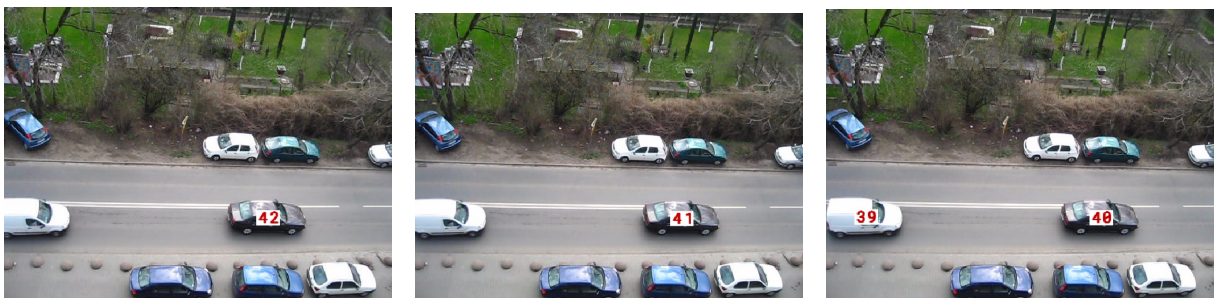
Problem stanowić mogą sytuację takie jak:

- jakiś, nawet drobny, ruch kamery,
- czasem samochody, które przejeżdżają koło siebie po przeciwnych pasach,
- czasem samochód, który wyjeżdża z widocznego pola lub wjeżdża na obserwowany część drogi,
- większe pojazdy mają bardziej „skoczną" prędkość.

Możemy teraz streścić kilka z bardziej charaketrystycznych punktów programu.

Pojawiający się pojazd

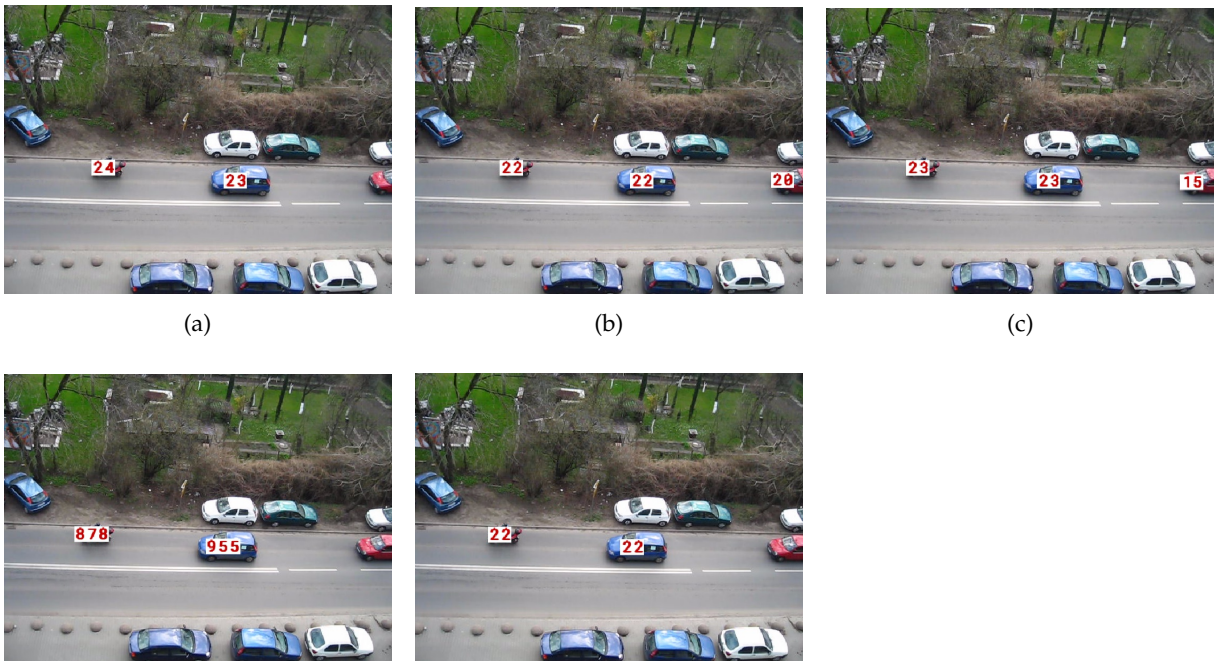
Jak już zostało wspomniane dużo czasu zostało poświęcone na poprawną obsługę sytuacji, gdy pojazd pojawia się lub znika. Przykładowa taka sytuacja widoczna jest na rysunku 7.



(a) (b) (c)

Rysunek 7: Prędkość pojazdu zostaje wyznaczona

Jak widać nasz trud miał sens, bo udało się coś osiągnąć – pojawienie się pojazdu nie spowodowało braku synchronizacji pomiędzy wektorami zawierającymi punkty, zatem prędkości obliczane są poprawnie. Ogólnie raczej zawsze tak powinno być, przy spełnieniu wspomnianych założeń o kierunku pojawiających się i znikających pojazdów, ale to się nie zawsze udaje. Problem można zobaczyć na rysunku 8.



Rysunek 8: Samochód znika w niewłaściwym miejscu

Samochód pojawił się z prawej strony, a później znikł (problemy na etapie morfologii – jedne regiony nie stykają się z granicą, a drugie tak), a przez to wektory przestały być zsynchronizowane na okres jednej klatki i wtedy prędkości gwałtownie skoczyły do bezsensownych wartości. Być może można by je tłumić w takich sytuacjach, ale zaś pojawiłby się wtedy problem poprawnego usuwania odpowiednich punktów/prędkości i punktów wskazujących gdzie na wykresie należy zapisać prędkość.

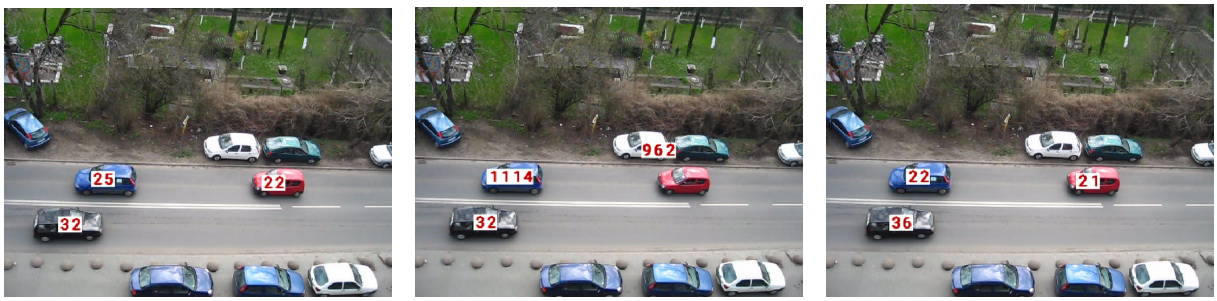
Chwilowe zniekształcenia

Czasem zdarzy się, że szum w którymś punkcie wzmocni któreś z krawędzi i niewłaściwy obiekt zostanie potraktowany jako ruchomy pojazd, co widać na rysunku 9.

Zaś, poza faktem, że prędkość pojawiła się nie tam gdzie trzeba, to wartości gwałtownie skoczyły ze względu na niespełnienie założeń o kierunku znikania i pojawiania się pojazdów.

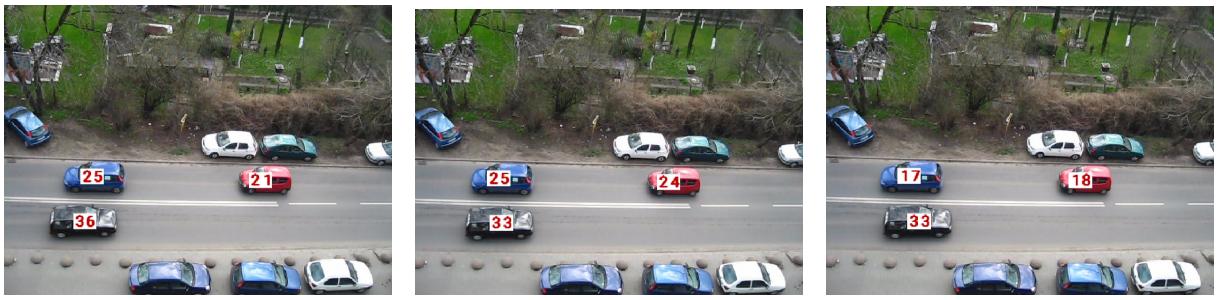
Omijające się pojazdy

Te same klatki wraz z kolejnymi (rys. 10) mogą pokazać działanie programu w przypadku gdy pojazdy się mijają.



(a) (b) (c)

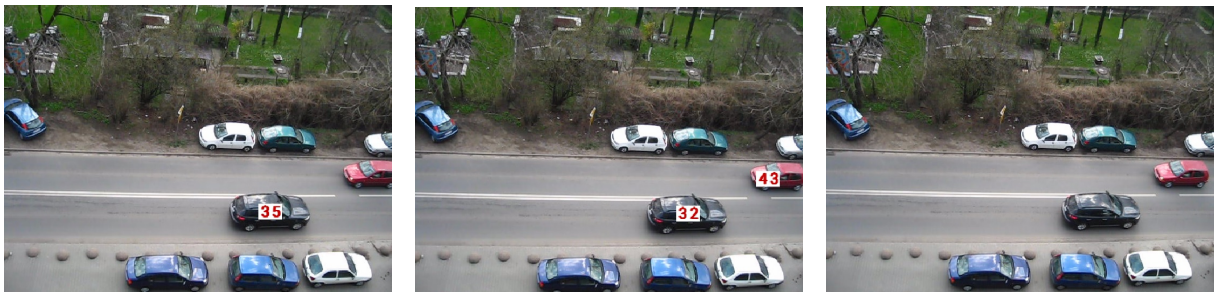
Rysunek 9: Chwilowy skok wartości z powodu wykrycia niewłaściwego obiektu



(a) (b) (c)

Rysunek 10: Omijające się pojazdy

W tej sytuacji to, że pojazdy się mijają nie wpłynęło na działanie programu. Zdarza się jednak, że wpływ będzie negatywny.



(a) (b) (c)

Rysunek 11: Omijające się pojazdy – problem

Przykładowo tutaj, spowodowało to, że regiony się zaczęły łączyć i przez to zaczęły się stykać z ustawionymi granicami, zatem ostatecznie regiony te (i tak niepoprawne) zniknęły.

Pozostałe

Poza tym, można wspomnieć, że:

– Nasz program, mogłyby być lepiej sparametryzowane od strony morfologii czy posiadać

inne funkcji na tym etapie. Czasem pomiędzy kolejnymi klatkami regiony reprezentujące jeden pojazd, różnią się wystarczająco mocno, by prędkość skoczył np. z 30 do 60 i potem wróciła. Nie są to tak bolesne skoki jak inne w wyżej omówionych sytuacjach krytycznych, ale są.

– W przypadku większych pojazdów doszło czasem do tego, że pojazd został rozbity na dwa osobne regiony, albo po tym jak zaczął wychodzić poza obserwowany fragment jezdni (i wypisywana prędkość wraz z nim) nagle pojawia się mniejszy region, który go reprezentuje z jego prędkością. Co prawda z prędkością poprawną, ale i tak widać w tym pewną nie konsekwencję.
– Kilkukrotnie wspominany problem z synchronizacją mógłby zostać lepiej rozwiązany, ale chyba nie znamy środowiska AVS na tyle dobrze, by wystarczająco zgrabnie operować macierzami.

Ogólnie, program działa raczej dobrze i większości sytuacji pokazuje sensowne prędkości.