PSRA - raport
Algorytm Flux Tensor with Split Gaussian models

Piotr Janus, Kamil Piszczek

1. Wprowadzenie

Opracowanie i implementacja sprzętowa algorytmu *FTSG*, było głównym celem przedstawianego projektu. Algorytm ten jest metodą hybrydową, dzięki czemu posiada wszystkie zalety algorytmów *Flux Tensor* (rozdział 2) oraz *Split Gaussian Models* (rozdział 3).

Obie wykorzystywane metody nie są idealne, jednak wzajemnie się uzupełniają przez co można opracować algorytm wykorzystujący je jednocześnie. Metoda *Flux Tensor* nie daje możliwości wykrywania obiektów statycznych. Zapewnia jednak odporność na zakłócenia spowodowane zmianami oświetlenia, a częściowo także przez występowanie cieni. Przeciwieństwem jest algorytm wykorzystujący rozkłady Gaussa, o ograniczonej odporności na zakłócenia, ale dający możliwość wykrywania obiektów statycznych.

2. Metoda Flux Tensor

2.1. Opis Algorytmu

Algorytm *Flux Tensor* jest uniwersalną metodą wykrywania ruchomych obiektów na obrazie ze statycznej kamery. Metoda ta jest niewrażliwa na zmiany oświetlenia, jednak przy jej użyciu nie jesteśmy w stanie rozpoznać statycznych obiektów pierwszoplanowych. Algorytm opiera się na detekcji krawędzi poziomych i pionowych, a następnie obliczeniu pochodnej obrazu po czasie, w celu wykrycia ruchomych elementów.

Metoda *Flux Tensor* w przedstawionej wersji operuje na obrazie w skali szarości. Oznaczmy przez $I(x, y, t)$ wartość piksela wejściowego. Gdzie x, y to współrzędne piksela na obrazie, t – czas. Dla uproszczenia wektor w przestrzeni (x, y, t) możemy zapisać jako v. *Flux Tensor* możemy przedstawić, jako macierz, opisującą zmiany wartości piksela wejściowego w lokalnej trójwymiarowej czasoprzestrzeni. Przez Ω rozumiemy otoczenie w przestrzeni (x, y, t), które rozważamy przy analizie danego piksela. Zgodnie z artykułem [**?**] macierz Flux Tensor wygląda następująco:

$$J_F(i) = \begin{pmatrix} \int_\Omega \left(\frac{\partial^2 I(v)}{\partial x \partial t}\right)^2 dv & \int_\Omega \frac{\partial^2 I(v)}{\partial x \partial t}\frac{\partial^2 I(v)}{\partial y \partial t}\, dv & \int_\Omega \frac{\partial^2 I(v)}{\partial x \partial t}\frac{\partial^2 I(v)}{\partial^2 t}\, dv \\ \int_\Omega \frac{\partial^2 I(v)}{\partial y \partial t}\frac{\partial^2 I(v)}{\partial x \partial t}\, dv & \int_\Omega \left(\frac{\partial^2 I(v)}{\partial y \partial t}\right)^2 dv & \int_\Omega \frac{\partial^2 I(v)}{\partial y \partial t}\frac{\partial^2 I(v)}{\partial^2 t}\, dv \\ \int_\Omega \frac{\partial^2 I(v)}{\partial^2 t}\frac{\partial^2 I(v)}{\partial x \partial t}\, dv & \int_\Omega \frac{\partial^2 I(v)}{\partial^2 t}\frac{\partial^2 I(v)}{\partial y \partial t}\, dv & \int_\Omega \left(\frac{\partial^2 I(v)}{\partial^2 t}\right)^2 dv \end{pmatrix} \tag{1}$$

Elementy macierzy zawierają informację na temat zmiany gradientu w czasie, co pozwala na rozróżnienie elementów ruchomych i statycznych na obrazie. Detekcja odbywa się bezpośrednio na podstawie śladu macierzy:

$$trace(J_F) = \int_\Omega \left\| \frac{\partial}{\partial t} \nabla I(v) \right\|^2 dv \tag{2}$$

Ślad macierzy jest obliczany dla każdego piksela. Jeżeli jego wartość jest większa od ustalonej wartości progowej T, to analizowany piksel jest elementem ruchomego obiektu. W przeciwnym wypadku zakładamy, że piksel jest tłem. Przyjmijmy następujące oznaczenia:

$$I_{xt} = \frac{\partial^2 I(x, y, t)}{\partial x \partial t}, \quad I_{yt} = \frac{\partial^2 I(x, y, t)}{\partial y \partial t}, \quad I_{tt} = \frac{\partial^2 I(x, y, t)}{\partial^2 t} \tag{3}$$

Równanie (2) można wówczas zapisać jako:

$$trace(J_F) = \int_\Omega \left(I_{xt}^2(i) + I_{yt}^2(i) + I_{tt}^2(i) \right) dv \tag{4}$$

Składowe I_{xt} i I_{yt} zawierają informację o wykrytych i jednocześnie poruszających się krawędziach (na podstawie pochodnej po czasie), natomiast I_{tt} informuje o zmianach oświetlenia w czasie.

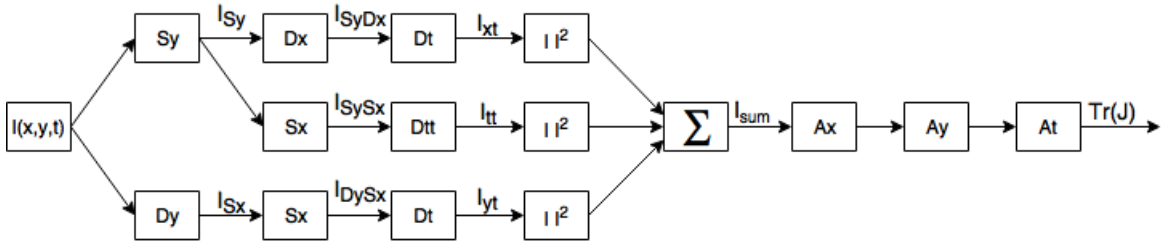
2.2. Obliczenia numeryczne

W pierwszym kroku piksel w formacie RGB konwertowany jest do skali szarości zgodnie z zależnością:

$$GRAY = 0.21R + 0.72G + 0.07B \tag{5}$$

Składowe I_{xt}, I_{yt}, I_{tt} są obliczane z wykorzystaniem jednowymiarowych masek konwolucji. Jest to rozwiązanie mniej złożone obliczeniowo, niż stosowanie masek 3D. Dodatkowo, w celu redukcji szumów, używany jest jednowymiarowy filtr wygładzający dla składowej, która nie występuje w aktualnie obliczanej pochodnej.

Zgodnie z powyższym, podczas obliczania składowej I_{xt} są używane filtry różniczkujące dla składowych x i t, oraz filtr wygładzający dla składowej y. Analogicznie, w przypadku składowej I_{yt} należy wykorzystać filtr wygładzający dla składowej x. Ostatnim elementem jest składowa I_{tt}. Tutaj filtry wygładzające używane są dla składowych x i y. Operacja całkowania jest wykonywana z użyciem odpowiednich jednowymiarowych filtrów. Na obraz nakładane są maski uśredniające dla poszczególnych składowych. Schemat tych operacji został przedstawiony na rys. 1.



Rysunek 1: Schemat operacji wyznaczających ślad macierzy J_F

Przedstawione na rys. 1 operacje to odpowiednio:

Dx, Dy – filtry różniczkujące odpowiednio dla składowych x i y
Sx, Sy – filtry wygładzające dla składowych x i y
Dt, Dtt – odpowiednio pierwsza i druga pochodna z wartości piksela po czasie
Ax, Ay – filtry uśredniające przestrzenne
At – filtr uśredniający czasowy

Oprócz rozmiarów poszczególnych filtrów algorytm posiada także parametr T, czyli próg wartości śladu, macierzy, powyżej którego piksel zostaje uznany za tło. Ostateczna lista parametrów algorytmu wygląda następująco:

T –wartość progowa
nDs – rozmiar filtrów przestrzennych (różniczkujących i wygładzających)
nDt – liczba ramek używana do różniczkowania po czasie
nAs – rozmiar masek uśredniających przestrzennych
nAt – liczba ramek używana do uśredniania po czasie

Należy zwrócić uwagę, że dobranie dużych rozmiarów filtrów (szczególnie różniczkujących po czasie) znacząco zwiększa zużycie pamięci. Domyślnymi parametrami, sprawdzającymi się w większości przypadków są:

$$T = 25, \quad nDs = 5, \quad nDt = 5, \quad nAs = 5, \quad nAt = 5$$

Jak wcześniej wspomniano, schemat na rys. 1, zoptymalizowany został pod kątem zużycia pamięci z zachowaniem równoległości obliczeń. W tym przypadku do działania algorytmu będą potrzebne 3 bufory o rozmiarze $nDt - 1$ każdy oraz jeden bufor o rozmiarze $nAt - 1$. Bufory działają na zasadzie kolejki *FIFO* (ang. *First In, First Out* – pierwszy na wejściu, pierwszy na wyjściu). W przypadku domyślnych parametrów wiąże się to z przechowywaniem w pamięci 16 ramek obrazu.

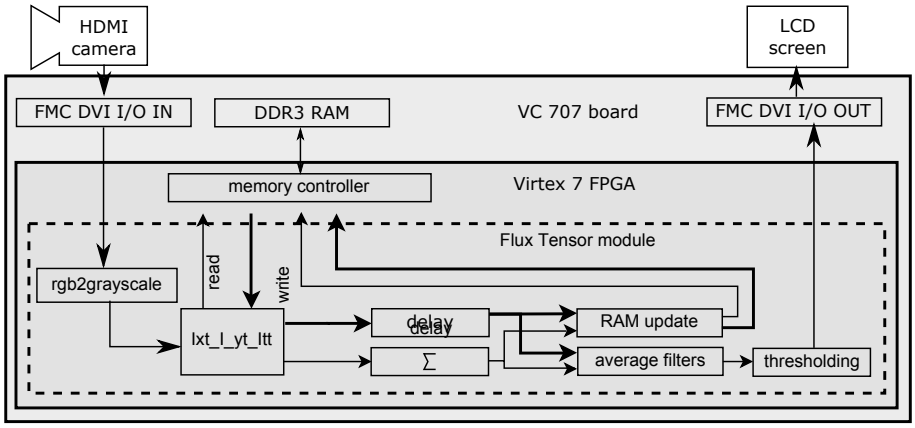
2.3. Implementacja sprzętowa

Wstępna analiza algorytmu przedstawionego na rysunku 1 pozwoliła wskazać dwa podstawowe wyzwania związane z realizacją sprzętową. Po pierwsze, z uwagi na konieczność buforowania 16 ramek obrazu (domyślne parametry) niezbędne okazało się być użycie zewnętrznej pamięci RAM – współcześnie dostępne układy FPGA nie posiadają wystarczających zasobów pamięciowych do buforowania ramek HD. Po drugie algorytm wymaga wykonania wielu operacji kontekstowych. W celu zaoszczędzenia zasobów, warto ustawić operacje w takiej kolejności, aby pierwsze wykonywane były operacje wykorzystujące kontekst pionowy.

Jak zaznaczono wcześniej, wyliczanie różniczki i uśrednianie po czasie (operacje D_t, D_{tt}, A_t) wymagają buforowania poprzednich ramek – wartości $I_{S_yD_x}$, $I_{S_yS_x}$, $I_{D_yS_x}$ oraz I_{sum}. Liczba zapamiętywanych wartości zależy od zastosowanych rozmiarów masek. Dla podanych wcześniej parametrów (wszystkie maski o rozmiarze 5), dla pojedynczego piksela wymaga się $(5-1)*4*8$ bitów = 128 bitów. Dla maksymalnego rozważanego rozmiaru maski $n = 7$ wskaźnik ten wynosi 192 bity. Wyznaczone wartości nie przekraczają możliwości współczesnych układów FPGA współpracujących z szybkimi pamięciami DDR3 lub DDR4, nawet dla rozdzielczości HD.

2.3.1. Stworzona architektura

Ogólny schemat zrealizowanego modułu sprzętowego przedstawiono na rysunku 2. Wszystkie moduły zostały opisane w języku Verilog, w trakcie prac wykorzystywano środowisko Vivado w wersji 2015.4 firmy Xilinx.



Rysunek 2: Architektura zaimplementowana w układzie FPGA

Źródłem sygnału wizyjnego była kamera Sony HDR-CX280 z wyjściem HDMI. Do wizualizacji rezultatów działania systemu wykorzystano typowy monitor LCD z wejściem HDMI. Wszystkie obliczenia realizowane byłby na karcie ewaluacyjnej VC 707 firmy Xilinx. Centralnym elementem platformy jest układ FPGA Virtex 7 (XC7VX485T–2FFG1761). W rozwiązaniu wykorzystano ponadto zewnętrzną pamięć DDR3 RAM oraz moduł wejścia/wyjścia HDMI (przystawka Avnet FMC DVI I/O).

W pierwszym kroku sygnał wejściowy konwertowany jest z przestrzeni barw RGB do skali szarości (moduł `rgb2grayscale`). Zastosowano przekształcenie zgodne z biblioteką OpenCV.

W następnym module wyznaczane są składowe I_{xt}, I_{yt}, I_{tt} (szczegółowy opis modułu w Subsection 2.3.2). Obliczanie pochodnych po czasie wymaga wykorzystania kontrolera pamięci (moduł `memory controller`). Następnie wyliczane wartości są podnoszone do kwadratu oraz sumowane (moduł Σ). Równolegle następuje opóźnienie danych odczytanych z pamięci RAM.

Wyniki poprzednich operacji trafiają do dwóch modułów:

- `RAM update` – następuje tutaj aktualizacja danych, które mają zostać zapisane do pamięci RAM. Polega ona na operacji FIFO (First In First Out) tj. usunięciu najstarszego elementu z wektora danych,
- `avarage filters` – realizacja uśredniania (filtry A_x, A_y, A_t) – omówienie w Subsection 2.3.3.

Ostatnim etapem algorytmu jest binaryzacja (moduł `thersholding`.

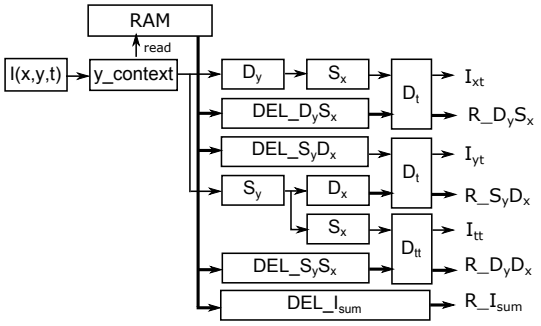
W trakcie projektowania modułu dążono do redukcji zasobów poprzez ograniczenie długości linii opóźniających dla dany zapisanych w pamięci RAM (słowo o szerokości 128 bitów). Po pierwsze dane czytano dokładnie w momencie, kiedy były potrzebne. Ponadto zdecydowano się na zapisanie do pamięci wartości I_{sum}, nie I_{sum} po operacjach A_x i A_y. Zastosowanie takiego podejścia wiąże się z koniecznością realizacji filtracji A_x i A_y dla wszystkich pikseli z kontekstu czasowego (tj. 5 w rozważanym przypadku). Jeśli weźmie się pod uwagę tylko długie linie opóźniające (zrealizowane w pamięci BRAM), to przy takim rozwiązaniu niezbędne jest pamiętanie: $YY \cdot 4 \cdot 5 \cdot 8 = 160 \cdot XX$ bitów (YY – rozdzielczość Alternatywne rozwiązanie, tj. bezpośrednia realizacja wg. schematu z rysunku 1, wymaga pamiętania $YY \cdot 4 \cdot 128 = 512 \cdot YY$ bitów.

Warto też zaznaczyć, że zastosowane skrócenie ścieżki może mieść również inne zalety. Przykładowo, jeśli moduł FT wykorzysta się w ramach bardziej rozbudowanego systemu wizyjnego, to brak konieczności opóźniania pozostałych danych zapisanych w pamięci RAM jest bardzo pożądany i może pozwolić na dużo większą redukcję zapotrzebowania na zasoby.

2.3.2. Moduł obliczający ślad macierzy

Szczegółowy schemat modułu `I_xt_I_ty_I_tt` przedstawiono na rysunku 3. Na początku wyliczany jest kontekst pionowy. W momencie gdy na wyjściu modułu pojawi się poprawny kontekst tj. wektor 5 pikseli, generowany jest sygnał odczytu z pamięci RAM (`read`), tak aby dostępne były zapisane dla rozważanego piksela dane. Mają one następującą postać:
$[I_{S_yD_x(t-1)}, ..., I_{S_yD_x(t-nDt+1)}, I_{D_yS_x(t-1)}, ..., I_{D_yS_x(t-nDt+1)},$
$I_{S_yS_x(t-1)}, ..., I_{S_yS_x(t-nDt+1)}, I_{sum(t-1)}, ..., I_{sum(t-nAt+1)}]$

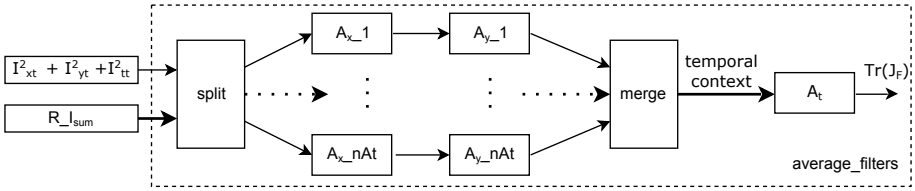
W drugim kroku, w sposób równoległy realizowane są następujące operacje filtr rożniczkujący D_y oraz filtr uśredniający S_y. Następnie wykonywane są operacje S_x (dwukrotnie) oraz D_y. W trzecim etapie obliczane są różniczki po czasie D_t i D_{tt}. Niezbędne dane z pamięci RAM opóźniane są w modułach $DEL\_D_yS_x$, $DEL\_S_yD_x$ oraz $DEL\_S_yS_x$. Dodatkowo opóźniane są dane z RAM potrzebne na etapie uśredniania przestrzennego DEL_I_{sum}. Wyjściem z modułu są wartości I_{xt}, I_{yt}, I_{yt} oraz dane z pamięci RAM oznaczone prefiksem $R\_$.



Rysunek 3: Schemat modułu obliczającego ślad macierzy (`I_xt_I_ty_I_tt`)

2.3.3. Uśrednianie

Schemat modułu `average_filters` przedstawiono na rysunku 4. Wejście stanowi suma I_{sum} dla bieżącej ramki oraz $nAt-1$ wartości I_{sum} z poprzednich ramek (prefiks $R\_$). W pierwszym kroku dane zostają rozdzielone na nAt wartości. Następnie dla każdej z nich wykonywane są operacje A_x oraz A_y. W kolejnym etapie następuje ich połączenie w temporal context dla którego wykonywana jest operacja A_t. W wyniku otrzymuje się ślad macierzy $Tr(J_F)$.



Rysunek 4: Moduł uśredniający A_x, A_y, A_t

3. Metoda Split Gaussian Models

3.1. Opis Algorytmu

Split Gaussian Models jest modyfikacją metody *Gaussian Mixture Models*, która jest powszechnie stosowana w systemach wizyjnych realizujących segmentację obiektów pierwszoplanowych. Działanie algorytmu jest oparte na sukcesywnie budowanym, wielowartościowym modelu tła oraz pierwszego planu. Model ten, indywidualny dla każdego piksela na obrazie, jest złożony z pewnej liczby rozkładów Gaussa. Każdy z nich charakteryzuje się wagą, średnią oraz odchyleniem standardowym. Budowany model jest więc modelem statystycznym.

W porównaniu z *Gaussian Mixture Models*, którego model stanowi mieszaninę rozkładów Gaussa reprezentujących pierwszy plan oraz tło, w metodzie *Split Gaussian Models* zdefiniowane są dwa odzielne modele: obiektów pierwszoplanowych oraz tła. Dodatkowo, w tym drugim modelu liczba rozkładów Gaussa zmienia się dynamicznie. Takie zmiany zapobiegają zjawisku, w którym rozkłady Gaussa reprezentujące pierwszy plan zaczynają dominować w modelu i pogarszać otrzymywane rezultaty przetwarzania. Następny akapit przedstawia szczegółowy opis algorytmu.

Wektor zawierający składowe *RGB* piksela o współrzędnych x, y w czasie t oznaczmy jako $I_t(x, y)$. Następnie sprawdzamy, czy piksel pasuje do któregoś ze zdefiniowanych rozkładów Gaussa reprezentujących tło (oznaczmy liczbę takich rozkładów jako K). Metryka wykorzystana w teście dopasowania wygląda następująco:

$$D_{min}(x, y) = min_{t \in K}\, max_{j \in C}((I_t(x, y) - \mu_{i,j})^2 - T_b \cdot \sigma^2) \tag{6}$$

Gdzie $\sigma = \sum_i^k \omega_i \sigma_i$. T_b jest z góry ustalonym progiem (domyślnie 3), natomiast $\mu_{i,j}$ oznacza wartość średnią dla danego rozkładu i składowej RGB. Dla uproszczenia przyjmujemy, że dla wszystkich trzech składowych odchylenie standardowe i wariancja są takie same. Piksel zostaje zaklasyfikowany jako pierwszoplanowy, jeżeli nie pasuje do żadnego z rozkładów Gaussa:

$$F_B(x, y) = \begin{cases} 1 & \text{jeżeli } D_{min}(x, y) > 0 \\ 0 & \text{w przeciwnym przypadku} \end{cases} \tag{7}$$

Model reprezentujący pierwszyplan, składający się tylko z jednego rozkładu Gaussa, jest używany do rodzielnia rzeczywistych obiektów statyczny od różnego rodzaju zakłóceń. Jeżeli piksel zostanie zaklasyfikowany jako element tła przez metodę *Flux Tensor* (rozdział 2) i jako element pierwszego planu przez model reprezentujący tło ($F_B = 1$) to ostateczna klasyfikacja (jako pierwszoplanowy element statyczny bądź zakłócenia) odbywa się z wykorzystaniem modelu pierwszego planu:

$$F_S(x, y) = \begin{cases} 1 & \text{jeżeli } F_{amb}(x, y) = 1 \quad \text{i} \quad I_t(x, y) - \mu_f(x, y) < T_f \\ 0 & \text{w przeciwnym przypadku} \end{cases} \tag{8}$$

Gdzie F_{amb} określa czy wystąpiła sprzeczność w klasyfikacji pomiędzy algorytmem *Flux Tensor* i modelem tła. T_f jest z góry ustalonym progiem (domyślnie 20), natomiast $\mu_f(x, y)$ aktualną wartością średnią.

3.2. Aktualizacja rozkładów Gaussa

Wykorzystana została tzw. konserwatywna metoda aktualizacji, polegająca na tym, że odbywa się ona tylko wtedy gdy rozpatrywany piksel pasuje do danego modelu. Model tła aktualizowany jest tylko wtedy, gdy maska M zdefiniowana przez wzór 9 ma wartość 1, w przeciwnym wypadku aktualizowany jest jedynie model pierwszoplanowy.

$$M = (1 - F_B) \vee (F_{amb} - F_S) \tag{9}$$

Aktualizację wartości średnich i odchylenia standardowego przedstawiają odpowiednio wzory (10) i (11).

$$\mu_t = (1 - \alpha)\mu_{t-1} + \alpha I_t \tag{10}$$

$$\sigma_t^2 = (1 - \alpha)\sigma_{t-1}^2 + (I_t - \mu)^T \alpha (T_t - \mu) \tag{11}$$

W przypadku rozkładu dla którego wartość metryki, zdefiniowanej przez wzór (6), była najmniejsza, waga aktualizowana jest według wzoru (12) w przeciwnym wypadku wykorzystywana jest zależność (13).

$$\omega_{i,t} = (1 - \alpha)\omega_{i,t-1} + \alpha \tag{12}$$

$$\omega_{i,t} = (1 - \alpha)\omega_{i,t-1} \tag{13}$$

Gdzie α oznacza współczynnik uczenia się, domyślnie wynosi on 0.004 dla modelu tła i 0.5 dla modelu pierwszegoplanu.

3.3. Architektura

Ze względu na to, że model programowy metody *Split Gaussian Models* nie dał oczekiwanych rezultatów w stosunku do wyników otrzymanych w artykule opisującym ten algorytm, implementacja sprzętowa nie została zrealizowana.

Gotowa implementacja sprzętowa metody *Gaussian Mixture Models* z pewnością ułatwi w przyszłości przeniesienie na platformę FPGA naprawionego algorytmu *Split Gaussian Model*. Moduł realizujący sieć sortującą, może być wykorzystany do implementacji zmienionej metryki (6). Dodatkowo, aby zaimplementować mechanizm zmieniającej się liczby rozkładów Gaussa, należałoby przeznaczyć conajmniej jeden bit do oznaczenia, który z rozkładów Gaussa został zaalokowany, a który zdealokowany.

4. Fuzja metody Flux Tensor i Split Gaussian Models

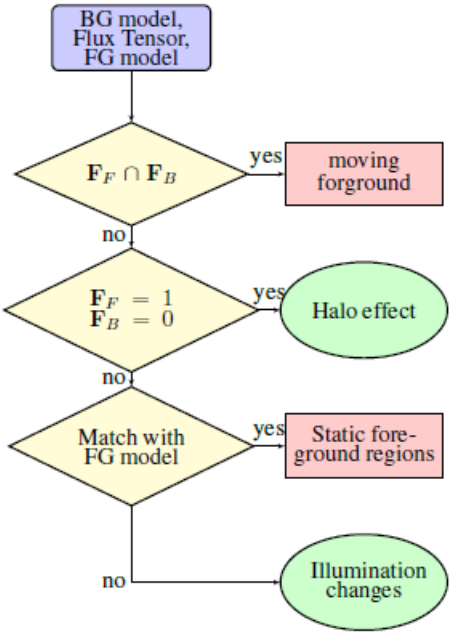
Schemat wykonywanych operacji został zaprezentowany na rysunku 5. Obraz analizowany jest równolegle za pomocą trzech modeli. Pierwszy jest oczywiście model *Flux Tensor*, drugi to model tła reprezentowany przez zmienną liczbę rozkładów Gaussa, natomiast trzecim jest model pierwszoplanowy składający się z jednego rozkładu (przykładowe maski binarne wszystkich trzech modeli zostały przedstawione na rys. 6).

W pierwszej kolejności pod uwagę brany jest wynik z dwóch pierwszych modułów. Na tym etapie może wystąpić jeden z czterech przypadków. Oznaczmy przez F maskę definiującą przynależność piksela do tła lub pierwszego planu. Jest to maska binarna, gdzie 1 oznacza piksel reprezentujący obiekt pierwszoplanowy, a 0 to piksel reprezentujący tło. Do jej wyznaczenia służy następujący wzór:

$$F = \begin{cases} 1, & \text{gdy } F_F = 1 \wedge F_B = 1 \\ 0, & \text{gdy } F_B = 0 \\ F_S, & \text{gdy } F_F = 0 \wedge F_B = 1 \end{cases} \tag{14}$$

gdzie:

F_F – maska otrzymana z modelu *Flux Tensor*
F_B – maska otrzymana z modelu tła
F_S – maska definiująca obszar statyczny



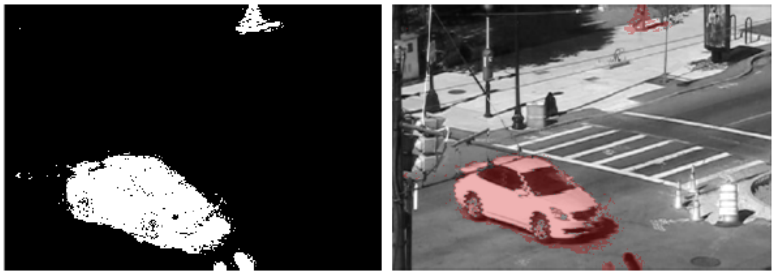
Rysunek 5: Schemat operacji algorytmu *FTSG*

Rysunek 6: Przykładowe maski binarne otrzymane odpowiednio dla modelu tła, metody *Flux Tensor* i modelu pierwszego planu

Jeżeli piksel zostanie przez obie metody tak samo sklasyfikowany, to jego ostateczna przynależność jest oczywista. W takim przypadku piksel zostaje uznany za obiekt ruchomy ($F_F = 1 \wedge F_B = 1$) bądź statyczne tło

($F_F = 0 \wedge F_B = 0$). Na rysunku 7 została przedstawiona część wspólna masek binarnych obu modułów oraz zidentyfikowany obiekt.

Dodatkowa analiza ma miejsce w przypadku, gdy wyniki obu modeli są sprzeczne. Jeżeli metoda *Flux Tensor* oznaczy próbkę jako pierwszy plan, natomiast model tła jako element tła, to końcowym werdyktem jest zaklasyfikowanie piksela jako tło. Jest to logicznie uzasadnione, ponieważ *Flux Tensor* ma tendencję do zaznaczania obiektów o trochę większym obszarze niż są one w rzeczywistości (tzw. efekt halo). Dodatkowo metoda bazująca na rozkładach Gaussa bez problemu wykrywa obiekty będące w ruchu, zatem takie działanie nie powoduje utraty istotnych danych. Taki przypadek został przestawiony na rys. 8.



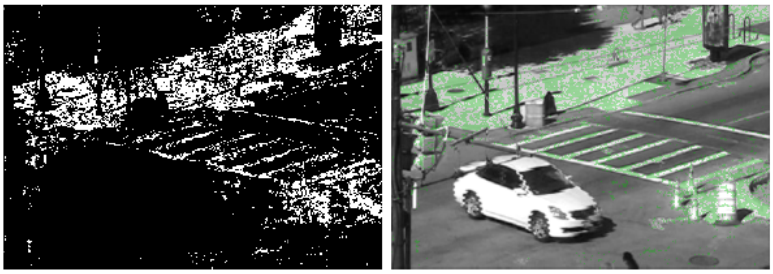
Rysunek 7: Detekcja ruchomego obiektu

Rysunek 8: Przykład tzw. efektu halo

W przypadku odwrotnym, czyli $F_F = 0 \wedge F_B = 1$, wykorzystywany jest model pierwszoplanowy i na jego podstawie wyznaczane jest ostateczne dopasowanie. Jeżeli dla zdefiniowanego w tym modelu rozkładu Gaussa piksel wejściowy spełnia warunek (8) to zostaje zakwalifikowany jako element statyczny pierwszego planu ($F_S = 1$). Model pierwszoplanowy jest aktualizowany tylko wtedy, gdy prawdziwy jest warunek $F_F = 0 \wedge F_B = 1 \wedge F_S = 1$. W przeciwnym wypadku ma miejsce aktualizacja modelu tła. Przykład detekcji obszaru statycznego został przedstawiony na rys. 9.

Rysunek 9: Detekcja obszaru statycznego

Taka klasyfikacja pozwala wyeliminować większość szumów powstałych w modelu tła, między innymi na skutek zmian oświetlenia i innych zakłóceń (przykładowe szumy w modelu tła zostały przedstawione na rys. 10). Należy przy tym pamiętać, że współczynnik k w podanym wzorze jest inny dla obu modeli. Istotne jest także, że w niektórych przypadkach obiekt statyczny może być w rzeczywistości odsłoniętym tłem przez właśnie ruszający obiekt (np. samochód odjeżdżający z parkingu). Problem ten jest częściowo eliminowany przez zastosowanie różnych stałych uczenia się (α) w obu modelach wykorzystujących rozkłady Gaussa.



Rysunek 10: Szumy powstałe w modelu tła

5. Testy

Ocena jakości procesu segmentacji obiektów pierwszoplanowych ze statycznej kamery jest stosunkowo prosta. Z natury działania algorytmu *FTSG* wynika, że każdy piksel jest klasyfikowany do jednego z dwóch zbiorów: obiektów pierwszoplanowych lub tła. Zatem do oceny działania algorytmu wystarczyło zastosować test binarny.

Do oceny zastosowano sekwencje testowe ze zbioru dostępnego na stronie *http://www. changedetection.net/*. Zbiór testowy zawiera również klatki referencyjne (ang. *ground truth*), które stanowiły punkt odniesienia do oceny algorytmu *FTSG*. Każda z nich była wzorcowym przekształceniem obrazu wejściowego na wyjściowy, na którym kolorem białym wyróżniono obiekty pierwszego planu. Dodatkowo sekwencje testowe należało oceniać od pewnego ustalonego numeru klatki. Dla każdego testu był on inny.

Na podstawie porównania wszystkich ramek wyjściowych testowanej metody oraz odpowiadających im wzorców wyznaczane są następujące wartości (składowe testu klasyfikacji binarnej):

- TP – liczba pikseli poprawnie zakwalifikowanych jako pierwszy plan (ang. *true positive*)
- TN – liczba pikseli poprawnie zakwalifikowanych jako tło (ang. *true negative*)
- FN – liczba pikseli błędnie zakwalifikowanych jako tło (ang. *false negative*)
- FP – liczba pikseli błędnie zakwalifikowanych jako pierwszy plan (ang. *false positive*)

Po obliczeniu wszystkich czterech współczynników dla danej sekwencji wyznacza się 7 wskaźników, które określają dokładność metody:

1. *Recall (Re)* : $TP/(TP+FN)$
2. *Specificity (Spec)* : $TN/(TN+FP)$
3. *False Positive Rate (FPR)* : $FP/(FP+TN)$
4. *False Negative Rate (FNR)* : $FN/(FN+TP)$
5. *Percentage of Wrong Classifications (PWC)* : $100(FN+FP)/(TP+FN+FP+TN)$
6. *Precision (Pr)* : $TP/(TP+FP)$
7. *F-measure (F1)* : $2\frac{P_r * R_e}{P_r + R_r}$

Do weryfikacji modelu programowego algorytmu stworzone zostało automatyczne środowisko testowe (w *Pythonie*). Dla każdego testu wykonywane były następujące czynności:

- Podmiana makra w kodzie modelu programowego, definiującego wybraną sekwencje testową (domyślnie wybrana była pierwsza sekwencja testowa).
- Ponowna kompilacja modelu programowego.
- Uruchomienie modelu programowego dla wybranej sekwencji testowej.
- Po zakończeniu testu, zapamiętanie otrzymanych współczynników ze standardowego wyjścia uruchamianego programu (*TP, TN, FP, FN*).

Po przeprowadzeniu testów dla wszystkich sekwencji testowych generowany był raport zawierający współczynniki jakości każdej sekwencji testowej. Dana zapisywane były do pliku.

Takie podejście znacząco ułatwia testowanie algorytmu *FTSG* z uwagi na jego złożoność obliczeniową. Dla sekwencji testowych w rozdzielczości 320×240 model programu przetwarza obraz z prędkością około 5 klatek na sekundę. W konsekwencji przetestowanie wszystkich sekwencji trwa około 4 godzin. Stworzone środowisko testowe jest w stanie przeprowadzić cały proces automatycznie bez jakiegokolwiek nadzoru. Dodatkową korzyścią jest prosta weryfikacja wprowadzonych modyfikacji. Środowisko testowe kompleksowo udziela odpowiedzi na pytanie, czy wprowadzona zmiana w algorytmie polepsza albo pogarsza jakość segmentacji obiektów pierwszoplanowych. Poza tym, ręczne testowanie na jednej czy nawet dwóch sekwencjach może być niebezpieczne, gdyż możemy przypadkowo wprowadzić poprawki, które sprawdzają się jedynie dla tej szczególnej sekwencji testowej.

6. Podział pracy

Piotr Janus – algorytm *Flux Tensor*, fuzja metod *FT* i *Split Gaussian Models* (model programowy), analiza artykułu i sformułowanie pytań do autorów, przygotowanie prezentacji i niniejszego raportu.

Kamil Piszczek – *Split Gaussian Models* (model programowy), zaprojektowanie i implementacja środowiska testowego, analiza artykułu i sformułowanie pytań do autorów i niniejszego raportu.