



**AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE**  
**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,  
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

KATEDRA AUTOMATYKI I INŻYNIERII BIOMEDYCZNEJ

Praca dyplomowa magisterska

*Biblioteka modułów sprzętowych do segmentacji obiektów  
pierwszoplanowych*

Autor: *Piotr Janus*  
Kierunek studiów: *Automatyka i Robotyka*  
Opiekun pracy: *dr inż. Tomasz Kryjak*

Kraków, 2017

*Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpozna bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, videogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.*





# **Spis treści**

<b>1. Wprowadzenie .....</b>	7
1.1. Cel pracy .....	9
1.2. Zawartość pracy .....	9
<b>2. Przegląd metod detekcji obiektów pierwszoplanowych .....</b>	11
2.1. Wprowadzenie .....	11
2.2. Model tła bazujący na poprzednich ramkach .....	11
2.3. Inne rodzaje modeli tła .....	12
2.4. Różne podejścia do aktualizacji modelu tła .....	13
2.5. Podsumowanie .....	13
<b>3. Opis teoretyczny wybranych algorytmów .....</b>	15
3.1. Proste metody segmentacji tła .....	15
3.2. Visual Background Extractor.....	16
3.2.1. Opis algorytmu.....	16
3.2.2. Konwersja RGB – CIELab.....	17
3.2.3. Obsługa ruchomej kamery .....	18
3.2.4. Badanie przepływu optycznego .....	19
3.2.5. Detekcja narożników - metoda Harrisa-Stephensa .....	20
3.2.6. Uwagi .....	21
3.3. Pixel-Based Adaptive Segmenter .....	21
3.3.1. Opis algorytmu.....	22
3.3.2. Detekcja obiektów statycznych.....	24
3.3.3. Indeksacja obiektów.....	26
3.3.4. Uwagi .....	27
3.4. Gaussian Mixture Models.....	28
3.4.1. Opis Algorytmu.....	28
3.4.2. Uwagi .....	30
<b>4. Implementacja sprzętowa wybranych algorytmów .....</b>	33

4.1.	Modele programowe .....	33
4.2.	Wykorzystany układ oraz interfejs wizyjny .....	34
4.3.	Trudności występujące w potokowym systemie wizyjnym .....	36
4.3.1.	Kontekst poziomy i pionowy .....	36
4.3.2.	Generowanie liczb losowych .....	37
4.3.3.	Kontroler pamięci RAM .....	38
4.4.	Implementacja algorytmu ViBE .....	39
4.5.	Implementacja rozszerzonej wersji ViBE.....	41
4.6.	Implementacja algorytmu PBAS .....	45
4.7.	Implementacja rozszerzonej wersji metody PBAS.....	48
4.8.	Implementacja GMM .....	51
<b>5.</b>	<b>Ewaluacja zaimplementowanych algorytmów .....</b>	<b>55</b>
5.1.	Metodologia przeprowadzonych testów .....	55
5.2.	Szczegółowe test.....	56
5.2.1.	Sekwencje podstawowe.....	56
5.2.2.	Dynamiczne tło i cienie.....	57
5.2.3.	Drgania kamery .....	59
5.2.4.	Obiekty zatrzymane .....	61
5.2.5.	Kamera termowizyjna .....	63
5.2.6.	Podsumowanie .....	63
<b>6.</b>	<b>Dalsze kierunki rozwoju .....</b>	<b>67</b>
6.1.	Poprawa algorytmów .....	67
6.2.	Wzrost wydajności .....	67
6.3.	Implementacja nowych rozwiązań .....	68
<b>7.</b>	<b>Zakończenie .....</b>	<b>69</b>
<b>A.</b>	<b>Spis zawartości płyty DVD .....</b>	<b>75</b>
<b>B.</b>	<b>Opis informatyczny .....</b>	<b>77</b>

# 1. Wprowadzenie

Segmentacja obiektów pierwszoplanowych jest bardzo rozległą i cały czas dynamicznie rozwijającą się dziedziną. Pracownicy naukowi ze wszystkich uczelni technicznych publikują coraz to nowe rozwiązania, bazujące na istniejących już algorytmach lub proponują podejścia całkowicie nowe, odbiegające od wcześniej przyjętej metodyki. Celem tych badań, jest oczywiście rozwój opisywanego zagadnienia i dążenie do ideału, czyli algorytmu, dającego bezbłędne rezultaty nawet w rzeczywistych, najbardziej ekstremalnych warunkach. Na przestrzeni ostatnich kilku lat można zaobserwować spadek cen różnego rodzaju czujników, kamer, oraz układów scalonych, co w rezultacie przekłada się na większe zainteresowanie i zapotrzebowanie na inteligentne systemów przetwarzania i analizy obrazów. W związku z powyższym, algorytmy odpowiedzialne za detekcję pierwszoplanowych obszarów powoli stają się nieodłącznym elementem większych i bardziej zaawansowanych systemów wizyjnych.

Jednym z przykładów takiego systemu może być rozbudowany system monitoringu ruchu drogowego. Jest to rozwiązanie umożliwiające gromadzenie danych archiwalnych oraz analizę w czasie rzeczywistym takich elementów jak średnia prędkość pojazdów, czas przejazdu z punktu A do B, natężenie ruchu oraz aktualne utrudnienia w ruchu drogowym. Dzięki takiemu systemowi, możliwa jest między innymi optymalizacja cykli sygnalizacji świetlnej na wszystkich skrzyżowaniach, co w ostateczności skutkuje minimalizacją korków i czasu podróży.

Kolejne zastosowanie algorytmów detekcji obiektów pierwszoplanowych to zautomatyzowany monitoring wizyjny. Tego typu system może zostać wykorzystany na przykład, do obserwowania strefy zabronionej i wykrywania obecności niepożądanych tam osób. Inne przykłady systemów, gdzie tego typu system może odegrać kluczową rolę to, na przykład *UAV* (ang. *Unmanned Aerial Vehicle* – bezzałogowy statek powietrzny) oraz pojazdy autonomiczne i wszelkiego rodzaju systemy wspomagające kierowcę.

Początkowo w tej dziedzinie niezwykle istotny był czynnik ludzki, a rolę algorytmu wykrywającego obiekty pierwszoplanowe pełnił wykwalifikowany operator. Niestety, takie rozwiązanie z biegiem czasu stało się nieopłacalne, na co miało wpływ kilka czynników. Po pierwsze, należy zaznaczyć, że zazwyczaj przez 99% czasu pracy operatora nie występują żadne angażujące go sytuacje. Człowiek nie jest w stanie analizować i nadzorować jednocześnie kilku obrazów. Z tego powodu, konieczne jest zastosowanie systemu działającego w czasie rzeczywistym, który daje możliwość przetwarzania określonej liczby ramek obrazu w danej rozdzielczości na sekundę.

W związku z powyższym, coraz częściej do realizacji systemów wizyjnych wykorzystuje się układy *FPGA* (ang. *Field-Programmable Gate Array* — bezpośrednio programowalna macierz bramek). Wraz

z rozwojem technologicznym zaczęły one stanowić sensowną alternatywę dla komputerów klasy *PC* (ang. *Personal Computer* – komputer osobisty) z procesorami typu *GPP* (ang. *General Purpose Processor* – procesor ogólnego przeznaczenia). Jedną z najważniejszych różnic pomiędzy tymi układami jest pełna rekonfigurowalność w przypadku *FPGA*. Podczas procesu produkcji procesorów typu *GPP* tranzystory są na stałe zatapiane w krzemie i nie ma możliwości późniejszej ingerencji w zaprojektowaną architekturę. Dostajemy możliwość modyfikacji jedynie programu, który na takim procesorze będzie wykonywany. Z kolei w układach *FPGA* dostajemy pełną możliwość przeprojektowania architektury i dostosowania jej do potrzeb danego zadania.

Ze względu na pełną rekonfigurowalność oraz możliwość zrównoleglenia obliczeń układy *FPGA* zyskują zdecydowaną przewagę na procesorami *GPP* w kontekście przetwarzania obrazów w czasie rzeczywistym. W przypadku standardowego procesora operacja przetworzenia jednej ramki obrazu wymaga odczytania jej w całości, następnie analizy po kolejno każdego piksela i na końcu przesłania finalnej ramki na wyjście. Systemy wizyjne w układach *FPGA* działają nieco inaczej, tutaj przetwarzanie obrazu odbywa się potokowo. W każdym taktie zegara na wejście modułu podawany jest jeden piksel z obrazu wejściowego oraz jeden piksel zostaje wystawiony na wyjściu. Dzięki temu latencja (opóźnienie) między obrazem wejściowym a wyjściowym trwa dokładnie tyle ile przeanalizowanie jednego piksela a nie całej ramki jak ma to miejsce przy wykorzystaniu procesorów *GPP*.

Często stosowane jest podejście, w którym dany algorytm w pierwszej kolejności zostaje zaimplementowany właśnie na standardowej, ogólnodostępnej platformie takiej jak komputer *PC*. Taka implementacja jest zdecydowanie prostsza, gdyż mamy do dyspozycji wiele wysokopoziomowych języków np. *C++*, *Java*, *Python*, *Matlab*. Architektura w układach *FPGA* jest tworzona z wykorzystaniem języków do opisu sprzętu, takich jak *Verilog* i *VHDL* (*Very High Speed Integrated Circuits Hardware Description Language*). Niestety, ze względu na odmienne podejście i potokowe przetwarzanie obrazu nie wszystkie operacje i algorytmy mogą zostać bezproblemowo przeniesione z platformy *PC* na układ *FPGA*. Przykładami mogą być operacje zmiennoprzecinkowe oraz skomplikowane obliczeniowo funkcje matematyczne (np. trygonometryczne, eksponenta, logarytmy). Mimo tych ograniczeń, ciężko wskazać alternatywne rozwiązanie, które spełniało by wymóg pracy w czasie rzeczywistym. Dodatkowo porównując do procesorów *GPP*, układy reprogramowalne są rozwiązaniem wydajniejszym i bardziej energooszczędnym.

Zagadnienie segmentacji obiektów pierwszoplanowych definiujemy jako operację wyodrębniania pierwszoplanowych obiektów obrazu. Efekt końcowy jest wizualizowany poprzez maskę binarną, na której piksele pierwszoplanowe oznaczone są na biało, a pozostałe na czarno. Zakładamy, że obraz pochodzi ze statycznie umiejscowionej kamery. Oprócz segmentacji obiektów istnieje także pojęcie wyodrębniania tła, polega ono na porównywaniu zapamiętanego modelu referencyjnego z aktualnym obrazem.

Należy zwrócić szczególną uwagę na potencjalne problemy i trudności występujące przy projektowaniu tego typu algorytmów. Bardzo ważna jest poprawna detekcja zarówno dynamicznych obiektów (ruchomych) jak i statycznych elementów pierwszego planu (na przykład samochód zatrzymany na światłach lub chwilowo nieruchomy człowiek). Z tym zagadnieniem powiązane jest także zjawisko tzw.

„duchów” (ang. *ghost*), można je zaobserwować w sytuacji, gdy obiekt zatrzymany przez dłuższy czas i błędnie zaklasyfikowany przez algorytm jako element tła, nagle zaczyna się poruszać. Rozróżnienie rzeczywistych obiektów zatrzymanych od „duchów” jest problematyczne, gdyż oba obiekty mają podobne właściwości.

Kolejnym poważnym utrudnieniem jest występowanie dynamicznego tła (na przykład poruszające się w tle liście i gałęzie). Zagadnienie to jest zdecydowanie najbardziej złożone i do tej pory nie znaleziono rozwiązania, które eliminowałoby je w stu procentach. Oprócz tego, należy także mieć na uwadze zmienne oświetlenie, możliwość wystąpienie lekkich drgań kamery, różnego rodzaju szumy i zakłócenia. Do grupy istotnych i często występujących problemów, możemy zaliczyć również konieczność prawidłowej detekcji cieni i odróżniania ich od właściwych obiektów pierwszoplanowych.

## 1.1. Cel pracy

Celem niniejszej pracy było stworzenie biblioteki modułów sprzętowych, które realizują różne algorytmy segmentacji obiektów pierwszoplanowych. W pracy położono główny nacisk na metody, które zostały już w pewnym stopniu zrealizowane w ramach innych prac inżynierskich, magisterskich oraz publikacji naukowych w Laboratorium Biocybernetyki Akademii Górnictwa-Hutniczej im. Stanisława Staszica w Krakowie. Zebrane algorytmy zostały przystosowane do uruchomienia na jednej platformie sprzętowej. Oprócz implementacji sprzętowej, do każdej metody zapewniono odpowiedni model programowy, dało to możliwość przeprowadzenia miarodajnych testów efektywności każdego algorytmu.

Modele programowe przygotowano z wykorzystaniem biblioteki *OpenCV* [18]. W celu przetestowania każdego algorytmu użyto zbioru sekwencji testowych dostępnego w bazie *ChangeDetection* [2]. Jest to baza bardzo często wykorzystywana do testów nowych wersji algorytmów. Dzięki dużej popularności, istnieje możliwość porównania otrzymanych wyników z rezultatami uzyskanymi w innych publikacjach.

Najważniejszym etapem pracy była implementacja wszystkich metod na wspólnej platformie sprzętowej. Algorytmy porównano pod względem wykorzystania zasobów układu *FPGA*, ilości wymaganej pamięci *RAM*, wydajności obliczeniowej oraz akceleracji w stosunku do modelu programowego uruchamianego na komputerze PC. Dodatkowo, sprawdzono możliwość przetwarzania obrazu w wyższych rozdzielczościach i określono maksymalną dla każdego algorytmu. Dokonano także pomiarów zużycia energii.

## 1.2. Zawartość pracy

W rozdziale 2 zamieszczono analizę publikacji dotyczących segmentacji obiektów pierwszoplanowych. Przedstawiono różne podejścia do omawianej tematyki, z wyraźnym zaznaczeniem, które algorytmy zostały zrealizowane również w Laboratorium Biocybernetyki AGH i zamieszczone w niniejszej pracy.

Rozdział 3 zawiera szczegółowe opisy wybranych algorytmów. Główny nacisk położono na wiadomości teoretyczne i dokładny zapis matematyczny. Przedstawiono również wady, zalety oraz domyślne wartości parametrów.

W rozdziale 4, opisano implementację wybranych algorytmów w układzie reprogramowalnym. Zamieszczono szczegółowe schematy przedstawiające zaimplementowaną architekturę dla każdego algorytmu oraz opisano trudności, które powstały podczas ich realizacji. Oprócz tego, porównano wszystkie algorytmy pod względem zużycia zasobów i energii.

Rozdział 5 zawiera szczegółowe testy poszczególnych algorytmów. Oprócz bezpośredniego porównania, otrzymane wyniki zestawiono także z innymi algorytmami dostępnymi w rankingu *ChangeDetection* [2].

W rozdziale 6 zamieszczono dalsze, możliwe kierunki rozwoju zaimplementowanych algorytmów oraz potencjalne możliwości ich poprawy. Natomiast rozdział 7 zawiera kompletne podsumowanie wykonanej pracy oraz konfrontację osiągniętych celów z przyjętymi na początku założeniami. Zamieszczono także, krótkie streszczenie wszystkich wniosków, które wyciągnięto na poszczególnych etapach realizacji niniejszej pracy.

## **2. Przegląd metod detekcji obiektów pierwszoplanowych**

### **2.1. Wprowadzenie**

Niniejszy rozdział ma na celu przedstawić dotychczasowe osiągnięcia w dziedzinie segmentacji obiektów pierwszoplanowych. Ponieważ jest to zagadnienie bardzo rozległe, położono nacisk głównie na najpopularniejsze podejścia, osiągające zadowalające rezultaty w różnego rodzaju testach weryfikacyjnych. Omówiono zarówno algorytmy opracowane w ramach prac dyplomowych oraz artykułów w Laboratorium Biocybernetyki AGH jak i inne metody opublikowane podczas konferencji naukowych na całym świecie.

Główny nacisk położono na pokazanie różnych podejść do tworzenia zaawansowanych modeli tła, wykorzystywanego następnie do klasyfikacji pikseli znajdujących się na kolejnych ramkach obrazu. Niezależnie od samego sposobu modelowania tła, istotnym zagadnieniem jest także sposób jego aktualizacji. Różne podejścia do tematu aktualizacji używanego modelu również zostały przedstawione w niniejszym rozdziale. Kolejnym elementem wartym zaznaczenia, są różnego rodzaju dodatkowe funkcjonalności wspierające podstawową wersję algorytmu. Może to być, na przykład moduł do wykrywania obiektów statycznych, bądź też mechanizm eliminujący drgania kamery.

### **2.2. Model tła bazujący na poprzednich ramkach**

Pierwszym typem algorytmów, są metody bazujące na poprzednich ramkach obrazu. Jednym z najprostszych tego typu algorytmów jest tzw. „średnia krocząca”. Metoda ta została zrealizowana w niniejszej pracy, opis teoretyczny zamieszczono w rozdziale 3.1. Do tego typu algorytmów możemy zaliczyć także prostą metodę odejmowania ramek oraz porównywanie aktualnej ramki z modelem referencyjnym. Obie te metody zaprezentowano w przytoczonym wyżej rozdziale.

Spośród zaawansowanych algorytmów, których model opiera się na poprzednich ramkach, niewątpliwie jedną z bardziej popularnych metody jest *ViBE* (*Visual Background Extractor*). Algorytm został opisany w wielu publikacjach, między innymi w [1, 3], doczekał się także wielu usprawnień. Oprócz zagranicznych publikacji metoda ta, była również obiektem badań w Laboratorium Biocybernetyki AGH, gdzie dokonano jej implementacji w układzie reprogramowalnym [16]. Oprócz standardowej wersji

opracowano także rozszerzoną odmianę algorytmu, posiadającą dodatkowy mechanizm eliminacji efektów drgającej kamery [15]. Metoda ta została również zawarta w niniejszej pracy, jej szczegółowy opis przedstawiono w rozdziale 3.2.

Kolejnym algorytmem bazującym na poprzednich ramkach obrazu jest *PBAS (Pixel Based Adaptive Segmenter)*. Metoda została zaprezentowana między innymi w [8]. W ramach badań w Laboratorium Biocybernetyki AGH opracowano implementację sprzętową tego algorytmu [12]. Dodatkowo udało się dodać do podstawowej wersji algorytmu dodatkowych mechanizm zapewniający lepszą detekcję obiektów statycznych [14]. Ulepszona wersja również została zaimplementowana w układzie *FPGA*. Omawiany algorytm również jest elementem badań niniejszej pracy, a jego dokładny opis teoretyczny zamieszczono w rozdziale 3.3.

### 2.3. Inne rodzaje modeli tła

Drugą grupą algorytmów, są metody bazujące w większym stopniu na modelach statystycznych. Pierwszą, najprawdopodobniej najpopularniejszą metodą tego rodzaju jest algorytm *GMM (Gaussian Mixture Models)*. Głównym założeniem tej metody jest reprezentacja modelu tła poprzez rozkłady Gaussa. Algorytm po raz pierwszy został opisany w 1999 roku w publikacji [21]. Powstały również publikacje przedstawiające implementację tej metody w układzie reprogramowalnym [5]. Algorytm, był także przedmiotem badań prac dyplomowych w Laboratorium Biocybernetyki AGH [20, 9]. Przygotowana w ramach pracy dyplomowej [20] implementacja sprzętowa została również przeanalizowana w niniejszej pracy. Szczegółowy opis algorytmu zamieszczono w rozdziale 3.4.

Kolejnym algorytmem przedstawiającym nieco odmienne podejście jest metoda *Flux Tensor*. Jest to algorytm bazujący na wykrywaniu krawędzi obiektu i badaniu pochodnej obrazu po czasie w celu wykrycia obiektów ruchomych, niestety przy jej użyciu nie ma możliwości detekcji obiektów statycznych. Od strony teoretycznej, metoda została przedstawiona między innymi w publikacji [19], natomiast pierwsza implementacja sprzętowa została opracowana w Laboratorium Biocybernetyki AGH i przedstawiona na konferencji *ICCVG (International Conference on Computer Vision and Graphics)* [9, 10].

Bardzo ciekawe podejście do zagadnienia segmentacji obiektów zostało przedstawione w publikacji [23]. Pokazany algorytm *FTSG (Flux Tensor with Split Gaussian models)* jest metodą hybrydową wykorzystującą omówione wcześniej metody *GMM* i *Flux Tensor* oraz dodatkowe mechanizm detekcji dynamicznego tła i obiektów statycznych. Algorytm w momencie publikacji był najdokładniejszą metodą w rankingu *ChangeDetection* [2]. Metoda ta, została częściowo zaimplementowana w układzie reprogramowalnym w ramach jednej z pracy dyplomowych w Laboratorium Biocybernetyki AGH [9].

Do grupy metod statystycznych można zaliczyć także algorytm *KDE (nonparametric Kernel Density Estimation)*. Metoda została po raz pierwszy opublikowana w 2002 roku [4]. Rezultaty, które można uzyskać w testach przy jej wykorzystaniu nadal są satysfakcyjne przez co często jest cytowana w różnego rodzaju publikacjach i służy jako punkt odniesienia do nowych algorytmów. Idea metody opiera się na funkcji gęstości prawdopodobieństwa, która jest używana do stworzenia modeli statystycznych tła

i pierwszego planu. Podobnie jak w przypadku pozostałych algorytmów, ta metoda również doczekała się wielu rozszerzeń i usprawnień [24].

## 2.4. Różne podejścia do aktualizacji modelu tła

W przypadku większości algorytmów wykorzystujących zaawansowany model tła, pojawia się zagadnienie aktualizacji modelu. Temat ten, został poruszony w większości publikacji przytoczonych w rozdziałach 2.2 i 2.3. Autorzy zazwyczaj rozróżniają dwa rodzaje podejść do aktualizacji modelu: liberalne (ang. *liberal approach*) oraz podejście konserwatywne (ang. *conservative approach*). Podejście liberalne zakłada aktualizację wszystkich modeli, podczas gdy konserwatywne jedynie tych pikseli, które zostały sklasyfikowane jako tło. Obie metody mają oczywiście swoje wady i zalety.

Główną wadą podejścia liberalnego jest stosunkowo szybkie wtapianie się obiektów pierwszoplanowych do modelu tła. Zjawisko to, jest szczególnie widoczne w przypadku wolno poruszających się obiektów. Polityka konserwatywnego aktualizowania modelu eliminuje ten problem, jednak ma inną poważną wadę. Podejście to, może prowadzić do omówionego już, wystąpienia tzw. „duchów”, czyli błędnej interpretacji odsłoniętego tła. Tego typu przypadek, może wystąpić w przypadku gdy obiekt pierwszoplanowy, początkowo statyczny (np. samochód stojący na światłach), zaczyna się poruszać. W takiej sytuacji algorytm może nadal interpretować pozostawiony po samochodzie obszar jako element pierwszego planu. Kolejnym problemem są również różnego rodzaju zakłócenia i szумy, raz błędnie sklasyfikowany obszar może pozostać już nienaprawiony.

Po analizie wspomnianych publikacji można zauważyć, że pomimo poważnych wad, częściej stosowanym podejściem jest polityka konserwatywna. Wykorzystywane są również dodatkowe mechanizmy pomagające eliminować wspomniane problemy. Przykładem może być niezależna aktualizacja pikseli sąsiadujących z aktualnie aktualizowanym jak ma to miejsce w algorytmach *ViBE* [16] lub *PBAS* [12]. Innym rozwiązaniem jest osobny moduł do całkowitej eliminacji zjawiska „duchów”, takie podejście zrealizowano między innymi w rozszerzonym algorytmie *PBAS* [14] oraz *FTSG* [23].

## 2.5. Podsumowanie

Powyższy rozdział, został napisany w celu zaprezentowania różnych podejść do tematu segmentacji obiektów pierwszoplanowych. Jak łatwo zauważać, na przykładzie wymienionych metod, jest to dziedzina bardzo rozległa oraz stale się rozwijająca. Pokazane algorytmy dowodzą, że do tego zagadnienia można podejść na wiele sposobów. Prezentacja nowych algorytmów lub usprawnionych wersji metod już dostępnych zazwyczaj ma miejsce na różnego rodzaju corocznich konferencjach. Do najpopularniejszych możemy zaliczyć *AVSS (Advanced Video an Signal – Based Surveillance, CVPR (Computer Vision and Pattern Recognition) oraz ICCVG (International Conference on Computer Vision and Graphics)*.

Ponieważ dokładny opis wszystkich algorytmów znajduje się we wskazanych artykułach, w tym rozdziale przedstawiono jedynie idee, oraz główne założenia poszczególnych metod. Warto zaznaczyć, że

w niektórych przypadkach istnieje wiele podejść do realizacji tej samej metody. Takim przykładem, jest między innymi metoda *GMM*. Drugim elementem wartym uwagi, jest sam dobór i kalibracja konkretnej metody do pracy w konkretnym środowisku, dyskusja na tym polu również może być bardzo rozległa.

### 3. Opis teoretyczny wybranych algorytmów

#### 3.1. Proste metody segmentacji tła

W tym rozdziale przedstawiono kilka, bardzo prostych obliczeniowo, metod segmentacji tła. Oczywiście, uzyskane z nich wykorzystaniem wyniki, znacznie odbiegają, od tego co oferują zaawansowane algorytmy. Szczególnie podczas testów w bardziej wymagającym środowisku, na przykład w przypadku drgań kamery lub z obecnością dynamicznego tła. Mimo to, często znajdują zastosowanie w mniej wymagających systemach, właśnie ze względu na swoją prostotę.

Pierwszym omawianym podejściem jest **metoda naiwna**. W tym algorytmie, przyjmujemy założenie, że pierwsza ramka jest modelem referencyjnym tła i nie zawiera żadnych obiektów pierwszoplanowych. Proces segmentacji tła polega na obliczeniu różnicy między aktualną ramką, a zapisanym modelem. Jeżeli dla konkretnego piksela otrzymana różnica jest większa od przyjętej wartości progowej zostaje on uznany za element pierwszego planu. Przedstawiony algorytm oczywiście nadaje się jedynie do prostej segmentacji i nie zapewnia akceptowalnych rezultatów podczas pracy w bardziej wymagającym środowisku.

Kolejnym prostym algorymem jest technika **odejmowania ramek**, w tym przypadku również wykonywane jest odejmowanie modelu referencyjnego i aktualnej ramki. Jako model referencyjny wykorzystywana jest poprzednia ramka obrazu. Metoda ta nadaje się niestety jedynie do detekcji obiektów ruchomych. W niniejszej pracy algorytm ten został również wykorzystany w mechanizmie detekcji tzw. „duchów”, całość została szczegółowo przedstawiona w rozdziale 3.3.2.

Ostatnia spośród prostych metod to tzw. **adaptacyjne odejmowanie tła**, technika ta jest w pewnym stopniu rozszerzeniem opisanej w poprzednim akapicie metody naiwnej. W tym przypadku, także wykonywane jest odejmowanie modelu tła i aktualnej ramki obrazu, a następnie dokonywanie klasyfikacji poprzez progowanie. Nieco bardziej złożona jest procedura aktualizacji modelu tła, jest ona wykonywana po każdej ramce obrazu i została zapisana równaniem (3.1).

$$B(t) = \alpha I(t) + (1 - \alpha)B(t - 1) \quad (3.1)$$

gdzie:

$B(t)$  – model tła w chwili  $t$

$I(t)$  – obraz wejściowy w chwili  $t$

$\alpha$  – parametr z zakresu od 0 do 1

## 3.2. Visual Background Extractor

Algorytm nazywany w skrócie *ViBE* (*Visual Background Extractor*), jak zostało wspomniane w rozdziale 2.2, jest metodą bazującą na modelu opartym o poprzednie ramki obrazu. Warto ponownie podkreślić, że zapamiętane wartości pikseli z poprzednich ramek obrazu to jedyne informacje wykorzystywane przez ten algorytm. Podobnie jak większość innych metod, *ViBE* może zostać dostosowane do pracy z różnymi przestrzeniami barw (np. *RGB*, skala szarości, *CIELab*).

Opisana w niniejszej pracy wersja algorytmu została przedstawiona w publikacji [15]. Standardową wersję metody dostosowano do pracy z drgającą/poruszającą się kamerą. Realizowane jest to, poprzez mechanizm przesunięcia modelu tła. W celu wyznaczenia wektora przesunięcia wykorzystywane jest obliczenie przepływu optycznego, dokładne szczegóły takiego rozwiązania, zostały zaprezentowane w podrozdziale 3.2.4.

### 3.2.1. Opis algorytmu

Model tła zdefiniowany jest osobno dla każdego piksela i składa się z  $N$  próbek. Proces inicjalizacji jest bardzo prosty i trwa tylko jedną ramkę obrazu. Każda z  $N$  próbek inicjalizowana jest losowo, wartością odpowiadającą jej piksela lub któregoś z sąsiadów. W tym przypadku przez sąsiedztwo rozumiemy przestrzenny kontekst  $3 \times 3$ , czyli jedynie najbliższe otaczające piksele.

Zdefiniujmy wartość piksela (o współrzędnych  $x, y$ ) w wybranej przestrzeni barw jako  $v(x, y)$ , z kolei i-tą próbki z modelu tła oznaczmy jako  $v_i$ . Model tła dla konkretnego piksela, możemy wtedy zapisać za pomocą wzoru (3.2).

$$M(x, y) = \{v_1, v_2, \dots, v_N\} \quad (3.2)$$

W celu przeprowadzenia klasyfikacji nowego piksela i przypisania mu etykiety obiektu pierwszoplanowego lub tła, należy zdefiniować okrąg  $S(v(x, y))$  o środku w punkcie  $v(x, y)$  i promieniu  $R$ . Promień ten jest wartością stałą, identyczną dla każdego modelu. Piksel uznawany jest za pierwszoplanowy jeżeli przynajmniej  $\#_{min}$  próbek z modelu tła zawiera się wewnątrz takiego okręgu. Oznaczmy przez  $F$  maskę reprezentującą obiekty pierwszoplanowe (1 – piksel pierwszoplanowy, 0 – tło), wówczas opisany warunek przedstawia równanie (3.3).

$$F(x, y) = \begin{cases} 1, & \text{gdy } \sum_{k=0}^N \{d(v(x, y), v_k(x, y)) < R\} < \#_{min} \\ 0, & \text{w pozostałych przypadkach} \end{cases} \quad (3.3)$$

Gdzie  $d$  to funkcja odległości pomiędzy próbką z modelu tła, a aktualnym pikselem.

Sposób obliczania odległości zależy od przyjętej przestrzeni barw. Można wykorzystać obraz w skali szarości, standardową przestrzeń *RGB* lub *CIELab*. W przypadku dwóch pierwszych wykorzystywana jest prosta metryka miejska, czyli inaczej suma modułów różnicy każdej składowej. Alternatywnym rozwiązaniem, aczkolwiek bardziej złożonym obliczeniowo jest metryka euklidesowa. Autorzy artykułów [16, 15] zdecydowali się na wykorzystanie przestrzeni *CIELab*, sam algorytm konwersji z przestrzeni *RGB* został opisany w rozdziale 3.2.2, natomiast odległość między próbками jest przedstawiona równaniem (3.4).

$$d_{CIELab} = \alpha \cdot |L_I - L_B| + \beta \cdot (|a_I - a_B| + |b_I - b_B|) \quad (3.4)$$

Gdzie:

$L_I, a_I, b_I$  – składowe piksela wejściowego

$L_B, a_B, b_B$  – składowe zapisane w modelu tła

$\alpha, \beta$  – wagi wynoszące odpowiednio 1 i 1,5

W omawianej metodzie wykorzystano konserwatywną politykę aktualizowania modelu tła, szcze- gółowe różnice pomiędzy podejściem liberalnym a konserwatywnym zaprezentowano w rozdziale 2.4. Aktualizacji podlegają zatem jedynie modele reprezentujące obszar zaklasyfikowany jako tło. W trakcie aktualizacji, wprowadzony został element losowy. Mianowicie dla każdego modelu tła podejmowana jest decyzja (w sposób całkowicie losowy), czy należy dokonywać aktualizacji. Przyjęto, że prawdopodobieństwo wykonania uaktualnienia jest stałe i reprezentowane przez parametr  $T$ .

Sam proces jest bardzo prosty, kiedy już dany model zostanie zakwalifikowany do aktualizowania wybierana jest losowa próbka i jej wartość zostaje nadpisana przez aktualną wartość piksela  $v(x, y)$ . Dodatkowo wybierany jest losowo jeden model z najbliższego sąsiedztwa (kontekst przestrzenny o rozmiarze  $3 \times 3$ ) aktualnie analizowanego piksela i w nim także jedna losowo wybrana próbka zostaje nadpisana wartością  $v(x, y)$ .

Warto podkreślić, że algorytm *ViBE* w omówionej tutaj, podstawowej wersji wymaga bardzo niewielkiej liczby parametrów. Należy zdefiniować jedynie liczbę próbek  $N$  zapisanych w każdym modelu, promień  $R$  okręgu wykorzystywanego w teście dopasowania, wymaganą minimalną liczbę próbek  $\#_{min}$ , leżącą wewnątrz wspomnianego okręgu, oraz prawdopodobieństwo wykonania aktualizacji  $T$ .

### 3.2.2. Konwersja RGB – CIELab

Konwersja z przestrzeni *RGB* do *CIELab* jest stosunkowo złożona pod względem matematycznym, jednak jak zauważono, między innymi w publikacjach [16, 15], wykorzystanie tej przestrzeni kolorów daje znacznie dokładniejsze wyniki segmentacji. W związku z powyższym, omawiany algorytm w finalnej implementacji sprzętowej będzie zrealizowany z wykorzystaniem właśnie takiej przestrzeni barw.

Proces konwersji składa się z dwóch etapów, pierwszym krokiem jest przekształcenie składowych *RGB* do przestrzeni *CIE XYZ*, jest to operacja liniowa zapisana równaniem (3.5).

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.41245 & 0.35758 & 0.18042 \\ 0.21267 & 0.71516 & 0.07217 \\ 0.01933 & 0.11919 & 0.95923 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (3.5)$$

Następna operacja to przekształcenie z właśnie otrzymanej przestrzeni *CIE XYZ* do docelowej *CIE-Lab*, w tym celu wykorzystywana jest funkcja  $f(t)$  opisana równaniem (3.6). Ostatecznie wyznaczenie składowej jasności -  $L$  i dwóch składowych chrominacji -  $a$  i  $b$  przedstawia wzór 3.7. Wykorzystane współczynniki wynoszą odpowiednio  $X_n = 0.950465$ ,  $Y_n = 1$ ,  $Z_n = 1.088754$

$$f(t) = \begin{cases} t^{-3}, & \text{dla } t > \left(\frac{6}{29}\right)^3 \\ \frac{1}{3} \left(\frac{29}{6}\right)^2 t + \frac{4}{29}, & \text{w przeciwnym razie} \end{cases} \quad (3.6)$$

$$\begin{bmatrix} L \\ a \\ b \end{bmatrix} = \begin{bmatrix} 116f\left(\frac{Y}{Y_n}\right) - 16 \\ 500 \left[ f\left(\frac{X}{X_n}\right) - f\left(\frac{Y}{Y_n}\right) \right] \\ 200 \left[ f\left(\frac{Y}{Y_n}\right) - f\left(\frac{Z}{Z_n}\right) \right] \end{bmatrix} \quad (3.7)$$

Z równań (3.6) i (3.7) można wywnioskować, że składowa  $L$  przyjmuje wartości z zakresu 0 – 100, natomiast składowe chrominacji  $a$  i  $b$  mieszczą się w przedziale od –128 do 127. Warto zwrócić uwagę, że ze względu na skomplikowane obliczenia matematyczne (funkcja  $f(t)$ ), implementacja sprzętowa takiej konwersji jest bardzo utrudniona, problem ten oraz jego rozwiązanie zostało szczegółowo opisane w rozdziale 4.4.

### 3.2.3. Obsługa ruchomej kamery

Kompletny algorytm segmentacji obiektów pierwszoplanowych musi między innymi poprawnie funkcjonować w przypadku lekkich drgań lub delikatnego przemieszczania się kamery. Rozwiązanie zapewniające taką funkcjonalność zostało zaprezentowanie między innymi w publikacjach [1] i polega na oszacowaniu przesunięcia kamery na podstawie badania przepływu optycznego.

Oznaczmy oszacowane przesunięcie kamery pomiędzy dwiema kolejnymi klatkami jako wektor  $[dx, dy]$ , dokładna metoda wyznaczania tego przesunięcia została przedstawiona w rozdziale 3.2.4. W celu kompensacji ruchu kamery stosuje się przesunięcie modelu tła zgodnie z równaniem (3.8).

$$M_t(x, y) = M_{t-1}(x + dx, y + dy) \quad (3.8)$$

Gdzie przez  $M_t(x, y)$  rozumiemy model tła dla piksela o współrzędnych  $x, y$  w chwili  $t$ .

Jak łatwo zauważyc, po takiej operacji część modeli tła, związana ze skrajnymi pikselami została usunięta i wymaga ponownej inicjalizacji. W tym przypadku, zamiast losowego inicjalizowania, opartego na wartości sąsiednich pikseli, do wszystkich próbek w danym modelu przypisana zostaje aktualna wartość piksela.

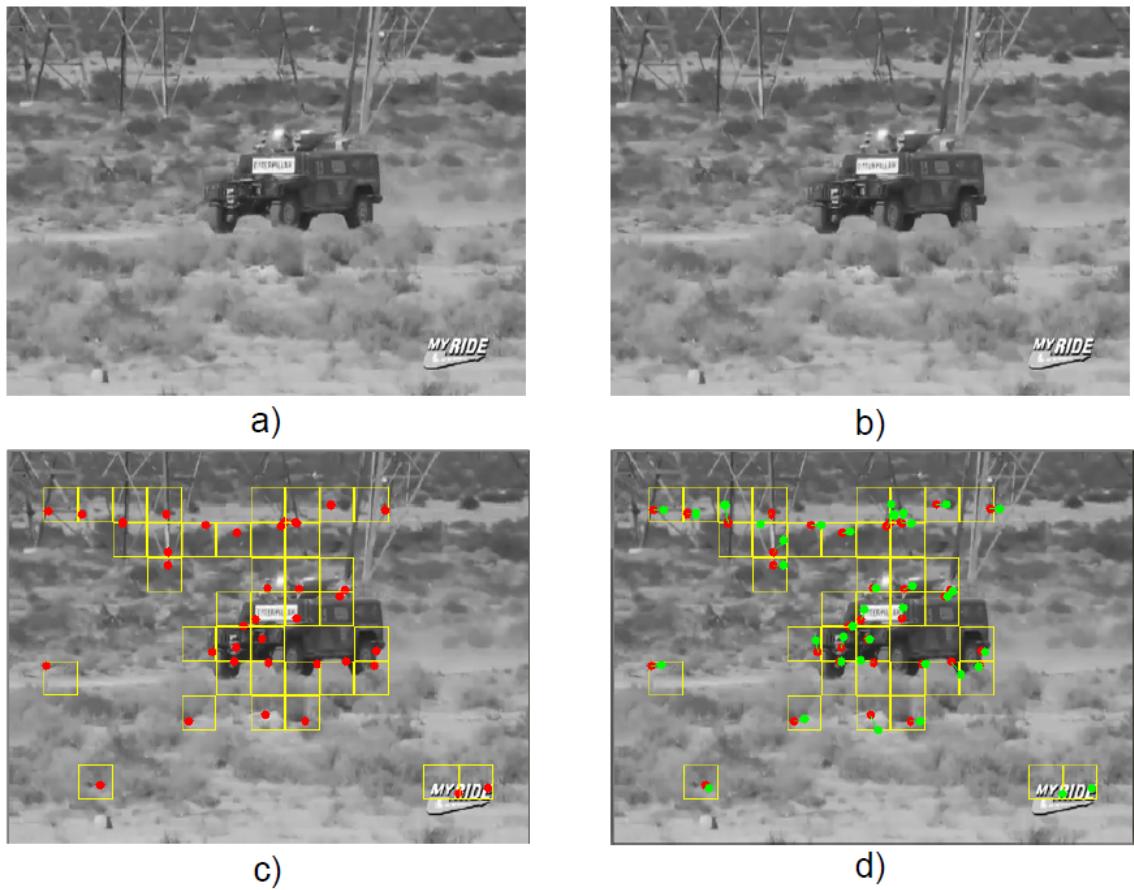
### 3.2.4. Badanie przepływu optycznego

Wspomniany w poprzednim podrozdziale wektor przesunięcia, może zostać oszacowany poprzez wyznaczenie przepływu optycznego. W przypadku opisywanej metody, należy przyjąć założenie, że przesunięcie jest identyczne na obszarze całego obrazu. Obliczanie przepływu nie jest wykonywane na całej ramce obrazu, a jedynie na reprezentatywnej grupie pikseli. Wybierane są takie fragmenty, na których najprościej śledzić przesunięcie, w tym przypadku zdecydowano się na analizę pikseli reprezentujących narożniki obiektów. Do detekcji narożników wykorzystany został algorytm Harrisza-Stephensa, dokładny opis tej metody zamieszczono w podrozdziale 3.2.5.

Obraz zostaje podzielony na takie same kwadratowe bloki o rozmiarze  $32 \times 32$  piksele każdy. Dla kolejnych pikseli obliczana jest odpowiedź  $H_R$  detektora krawędzi (metoda Harrisza-Stephensa). W każdym z bloków wyznaczany jest piksel dla którego zwrócona wartość, jest najwyższa. Wyższa wartość  $H_R$  oznacza większe prawdopodobieństwo, że dany piksel jest właśnie narożnikiem jakiegoś obiektu. Jeżeli otrzymana wartość  $H_R$  przekracza ustalony z góry próg  $H_{TH}$ , to dany piksel wraz z współrzędnymi w odpowiadającym mu bloku, oraz z wartościami sąsiadujących pikseli (ponownie za sąsiedztwo uznajemy kontekst  $3 \times 3$ ) zostaje zapisany.

Kolejnym krokiem jest porównanie zapamiętanych pikseli reprezentujących narożniki obiektów z pikselami z kolejnej ramki obrazu. Porównania przeprowadzane są w obszarze wyznaczonych wcześniej bloków o rozmiarze  $32 \times 32$ . Kolejne piksele w danym bloku, wraz z otaczającym kontekstem  $3 \times 3$ , są porównywane z zapamiętanym kontekstem piksela, będącego narożnikiem. Jeżeli dla danego bloku, nie wyznaczono w poprzedniej ramce takiego piksela, to cały blok zostaje wyłączony z analizy. Porównanie dwóch kontekstów opiera się na obliczeniu SAD (ang. *Sum of Absolute Difference* - suma modułów różnicicy). Wartość odpowiadających sobie pikseli w obu kontekstach są od siebie odejmowane, następnie obliczana jest suma wartości bezwzględnych z tych różnic. Dla każdego bloku, poszukiwany jest piksel, dla którego obliczony SAD jest najmniejszy. Jego odległość od piksela będącego narożnikiem, to tzw. wektor przepływu optycznego (ang. *optical flow vector*). Jest ona obliczana osobno wzduż osi  $x$  i  $y$ , ze względu na przyjęty rozmiar każdego bloku może przyjmować wartości od  $-32$  do  $32$  dla obu osi.

Ostatnią operacją jest wyznaczenie wektora przesunięcia, jest on definiowany jako mediana wektorów przepływu optycznego otrzymanych dla każdego z bloków. Przykład całej procedury został pokazany na rysunku 3.1. Przedstawia on dwie kolejne ramki obrazu (rysunki *a* i *b*), wykryte narożniki w każdym z bloków (czerwone piksele na rysunku *c*), natomiast zielonym kolorem na rysunku *d* zostały oznaczone piksele, dla których SAD między ich kontekstem a kontekstem piksela czerwonego w danym bloku był najmniejszy. Ostateczny wektor przesunięcia jest obliczany jako mediana wektorów przesunięcia między piksemem zielonym a czerwonym w każdym bloku i w tym przypadku wynosi  $[0, -3]$ .



Rys. 3.1. Przykładowe wyznaczanie wektora przesunięcia – źródło [15]

### 3.2.5. Detekcja narożników - metoda Harrisza-Stephensa

Algorytm detekcji narożników został opisany w publikacji [7]. Z punktu widzenia obliczeń opiera się on głównie na operacjach konwolucji. W pierwszej kolejności, należy określić tzw. macierz Harrisza, jej definicję przedstawia równanie (3.9).

$$H = \begin{bmatrix} I_x^2 \otimes G & I_x I_y \otimes G \\ I_x I_y \otimes G & I_y^2 \otimes G \end{bmatrix} \quad (3.9)$$

Gdzie:

$I_x, I_y$  – pierwsza pochodna obrazu wejściowego (pozioma i pionowa)

$\otimes G$  – operacja konwolucji, wygładzenie obraz z wykorzystaniem filtra Gaussa

Pochodne pierwszego rzędu obrazu wejściowego są obliczane z wykorzystaniem filtru Prewitta, maska pozioma  $P_x$  i pionowa  $P_y$  zostały przedstawione równaniem (3.10). Z kolei przykładowa maska filtru Gaussa (dla  $\sigma = 3$ ) została opisana równaniem (3.11).

$$P_x = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix}, \quad P_y = \begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \quad (3.10)$$

$$G(\sigma = 3) = \begin{bmatrix} 0.1070 & 0.1131 & 0.1070 \\ 0.1131 & 0.1196 & 0.1131 \\ 0.1070 & 0.1131 & 0.1070 \end{bmatrix} \quad (3.11)$$

Ostateczna odpowiedź detektora krawędzi wymaga obliczenia wyznacznika i śladu macierzy  $H$ . Finalna wartość może zostać zapisana równaniem (3.12).

$$H_R = \det(H) - k \cdot \text{trace}^2(H) \quad (3.12)$$

Gdzie:  $k$  - współczynnik skalowania (przyjmuje wartości 0,02 – 0,2)

Po uwzględnieniu wzorów na wskaźnik i ślad macierzy o rozmiarze  $2 \times 2$ , wzór (3.12) można przekształcić do postaci (3.13). Otrzymana wartość  $H_R$  jest porównywana progowana. Jeżeli wartość progowa  $H_{TH}$  zostanie przekroczona dany piksel uznawany jest za narożnik obiektu.

$$H_R = I_x^2 \otimes G \cdot I_y^2 \otimes G - (I_x I_y \otimes G)^2 - k \cdot (I_x^2 \otimes G + I_y^2 \otimes G)^2 \quad (3.13)$$

### 3.2.6. Uwagi

Algorytm *ViBE* jest metodą zawierającą niezbyt złożony model tła oraz charakteryzującą się niedużą złożonością obliczeniową. Warto jeszcze raz podkreślić niewielką liczbę parametrów, ostateczna lista prezentują się następująco (w nawiasach podano wartości domyślne):

$N$  – liczba próbek w modelu tła (20)

$\#_{min}$  – minimalna liczba próbek wymagana w teście dopasowania (2)

$R$  – próg dopasowania próbki do modelu (15)

$T$  – prawdopodobieństwo wykonania aktualizacji ( $\frac{1}{16}$ )

W przypadku rozszerzonej wersji algorytmu, należy uwzględnić jeszcze jeden dodatkowy parametr, mianowicie próg  $H_{TH}$  (domyślnie 10), wykorzystywany podczas detekcji krawędzi metodą Harrisza-Stephensa.

## 3.3. Pixel-Based Adaptive Segmenter

Omawiany algorytm jest rozszerzeniem opisanej rozdziałem 3.2.1 metody *ViBE*. *PBAS* (ang. *Pixel Based Adaptive Segmenter*, podobnie jak opisana już wcześniej metoda, także posiada model, który składa się między innymi z próbek pochodzących z poprzednich ramek obrazu. Dodatkowo, próg dopasowania oraz prawdopodobieństwo dokonania aktualizacji są niezależne i na bieżąco uaktualniane dla każdego

modelu. Opisywany algorytm może zostać zaimplementowany zarówno w wersji *RGB* jak i w skali szarości.

Algorytm został zaprezentowany w artykule [14] i to właśnie na jego podstawie powstał niniejszy opis. Oprócz standardowej wersji metody *PBAS* autorzy zaproponowali także dodatkowy mechanizm eliminacji tzw. „duchów” (opis tego zjawiska został dokładnie opisany w rozdziale 3.3.2). Zaproponowane rozwiązanie opiera się na algorytmie etykietowania obiektów i następnie porównywania krawędzi na obrazie wejściowym i masce wyjściowej w odrębie każdego z nich.

### 3.3.1. Opis algorytmu

Model tła jest nieco bardziej złożony niż ten przedstawiony w algorytmie *ViBE*. Jego pierwsza część, podobnie jak w poprzedniej metodzie, składa się z  $N$  zapamiętanych próbek. W celu uproszczenia dalszego zapisu matematycznego, zdefiniujmy ten zbiór jako  $B(x_i)$ , gdzie  $x_i$  to aktualnie przetwarzany piksel obrazu, całość została opisana równaniem (3.14).

$$B(x_i) = \{B_1(x_i), B_2(x_i), \dots, B_N(x_i)\} \quad (3.14)$$

Test dopasowania, jest również bardzo podobny do tego, który występuje w metodzie *ViBE*, jedyną różnicą jest niezależny dla każdego modelu próg przynależności do modelu  $R(x_i)$ , całość została przedstawiona równaniem (3.15).

$$F(x_i) = \begin{cases} 1, & \text{gdy } \sum_{k=0}^N \{d(I(x_i), B_k(x_i)) < R(x_i)\} < \#_{min} \\ 0, & \text{w pozostałych przypadkach} \end{cases} \quad (3.15)$$

Gdzie  $d$  to funkcja odległości pomiędzy próbką z modelu tła, a aktualnym pikselem.

W przypadku algorytmu w wersji *RGB* każdy kanał przetwarzany jest osobno z wykorzystaniem niezależnego modelu tła. Finalna maska jest alternatywą logiczną wyników z poszczególnych kanałów, oznaczając poszczególne maski jako  $F_R$ ,  $F_G$ ,  $F_B$  ostateczną klasyfikację możemy zapisać równaniem (3.16).

$$F_{RGB} = F_R \vee \vee F_G \vee F_B \quad (3.16)$$

Ponieważ każdy kanał analizowany jest osobno, funkcję odległości pomiędzy próbками można zapisać bardzo prosto równaniem (3.17). Jest to po prostu moduł różnicy.

$$d(I(x_i), B_k(x_i)) = |I(x_i) - B_k(x_i)| \quad (3.17)$$

Kolejnym krokiem po przeprowadzeniu testu dopasowania i klasyfikacji piksela jest aktualizacja modelu tła. Zastosowano konserwatywne podejście, czyli aktualizowane są tylko piksele sklasyfikowane jako tło. Analogicznie jak w algorytmie *ViBE* decyzja o aktualizacji podejmowana jest losowo. Prawdopodobieństwo jej wykonania wynosi  $p = 1/T(x_i)$ , gdzie parametr  $T(x_i)$  jest dynamicznie aktualizowany i niezależny dla każdego piksela. Sama aktualizacja, polega na nadpisaniu, losowo wybranej

próbki  $B_k(x_i)$  z modelu aktualną wartością piksela  $I(x_i)$ . Dodatkowo, wybierany jest losowy piksel z otoczenia  $3 \times 3$  i losowo wybrana próbka z modelu mu odpowiadającego, jest nadpisywana wartością tego piksela.

Niezależnie od aktualizacji części modelu zawierającej zapamiętane próbki dokonywana jest zmiana parametrów  $R(x_i)$  i  $T(x_i)$ . W tym celu konieczne jest zdefiniowanie kolejnego elementu modelu tła, który zawiera zbiór minimalnych odległości pomiędzy próbką z modelu a aktualną wartością piksela. Zbiór ten został opisany równaniem (3.18).

$$D(x_i) = \{D_1(x_i), D_2(x_i), \dots, D_N(x_i)\} \quad (3.18)$$

Przedstawiony zbiór  $D(x_i)$  aktualizowany jest razem ze zbiorem próbek. Nadpisywany jest jedynie element o indeksie  $k$  dla którego dystans pomiędzy próbką i aktualnym pikselem jest najmniejsza, zostało to przedstawione równaniem (3.19).

$$d_{min}(x_i) = \min_k d(I(x_i), B_k(x_i)) \quad (3.19)$$

Do aktualizacji progu dopasowania, czyli parametru  $R(x_i)$  konieczne jest wyznaczenie tzw. miary dynamiki tła, czyli inaczej wartości średniej ze zbioru  $D(x_i)$ . Finalny wzór na nową wartość progu przedstawia równanie (3.20). Warto dodać, że przyjęto także dolne ograniczenie wartości parametru, wynoszące  $R_{low} = 18$ .

$$R(x_i) = \begin{cases} R(x_i)(1 - R_{inc/dec}), & \text{jeżeli } R(x_i) > \bar{d}_{min}(x_i)R_{sc} \\ R(x_i)(1 + R_{inc/dec}) & \text{w przeciwnym razie} \end{cases} \quad (3.20)$$

Gdzie:

$R_{inc/dec}$  – stały współczynnik aktualizacji (domyślnie 0.05)

$\bar{d}_{min}(x_i)$  – wartość średnia zbioru  $D(x_i)$

$R_{sc}$  – współczynnik skalowania (domyślnie 5)

Ostatni etap to aktualizacja parametru opisującego prawdopodobieństwo dokonania aktualizacji, czyli  $T(x_i)$ . Nowa wartość zależy od wyniku klasyfikacji piksela i została opisana równaniem (3.21). Przyjęto założenie, że parametr ten posiada także ograniczenie dolne jak i górne wynoszące odpowiednio  $T_{low} = 2$  i  $T_{up} = 200$ .

$$T(x_i) = \begin{cases} T(x_i) + \frac{T_{inc}}{\bar{d}_{min}(x_i)}, & \text{jeżeli } F(x_i) = 1 \\ T(x_i) - \frac{T_{dec}}{\bar{d}_{min}(x_i)} & \text{w przeciwnym razie} \end{cases} \quad (3.21)$$

### 3.3.2. Detekcja obiektów statycznych

Autorzy publikacji [14] zaproponowali dodatkowy mechanizm eliminacji tzw. „duchów”. Przedstawiona metoda opiera się na porównaniu krawędzi na obrazie wejściowym i masce końcowej. Jeżeli krawędzie, na obu obrazach są identyczne, to znaczy, że obiekt rzeczywiście istnieje, natomiast w przeciwnym wypadku jest on duchem i zostaje usunięty z finalnej maski.

Pierwszym krokiem, który należy wykonać w celu identyfikacji potencjalnych „duchów” jest operacja odejmowania dwóch kolejnych ramek. W przestrzeni *RGB* taka operacja jest opisana przez równanie (3.22).

$$dF(x_i) = \sum_{C \in \{R, G, B\}} |I(x_i)_K^C - I(x_i)_{K-1}^C| \quad (3.22)$$

Gdzie przez  $I(x_i)_K^C$  rozumiemy wartość jednej ze składowych *R,G,B* piksela o położeniu  $x_i$  w  $K$ -tej ramce obrazu.

Kolejny etap to obliczenie tzw. współczynnika stabilność. Operację tą opisuje równanie (3.23), jest ona wykonywana niezależnie dla każdego zidentyfikowanego obiektu na masce wyjściowej. Do tego celu konieczna jest także operacja indeksacji obiektów, ponieważ jest to zagadnienie dość złożone zostało opisane oddzielne w rozdziale 3.3.3. Sam współczynnik definiujemy jako stosunek liczby pikseli, których różnica opisana równaniem (3.22) przekracza ustalony próg  $\theta$  do całkowitej powierzchni obiektu.

$$S_{O_k} = \frac{\sum_{x_i \in O_k} dF(x_i) > \theta}{\sum_{x_i \in O_k} F(x_i)} \quad (3.23)$$

Gdzie:

$O_k$  –  $k$ -ty obiekt na masce wyjściowej, wyznaczony algorytmem indeksacji

$F(x_i)$  – maska wyjściowa ze standardowego algorytmu *PBAS*

Obiekt jest uznawany za statyczny (czyli potencjalnego ducha), jeżeli wartość współczynnika  $S_{O_k}$  przekracza ustalony próg  $S_{TH}$  (domyślnie równy 0.1). Takie działanie ma na celu eliminację błędnej klasyfikacji wynikającej z szumów i zakłóceń obrazu.

W celu weryfikacji, czy dany obiekt istnieje zarówno na obrazie wejściowym jak i masce wyjściowej, użyty został mechanizm porównywania krawędzi. Sama detekcja krawędzi została zrealizowana z wykorzystaniem filtra Sobela. Przykładowe maski dla osi *X* i *Y* przedstawiono w równaniu (3.24). Operacja konwolucji z maską sobela realizowana jest osobno dla każdego kanału *RGB*, następnie obrazy wynikowe dla obu osi są sumowane i binaryzowane ze stałym progiem  $T_S$ . Maska finalna stanowi alternatywę logiczną wyników z wszystkich kanałów.

$$S_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, \quad S_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad (3.24)$$

Po wyznaczeniu krawędzi obrazu należy dla każdego obiektu obliczyć tzw. współczynnik podobieństwa krawędzi  $EC_{O_k}$ , został on przedstawiony za pomocą równania (3.25). Jeżeli otrzymana wartość przekracza próg 0.5 oznacza to, że obiekt istnieje, czyli występuje zarówno na obrazie wejściowym jak i masce wyjściowej.

$$EC_{O_k} = \frac{\sum_{x_i \in O_k} F_E(x_i) == 1 \wedge I_E(x_i) == 1}{\sum_{x_i \in O_k} F_E(x_i) == 1} \quad (3.25)$$

Gdzie:

$F_E$  – krawędzie na masce wyjściowej

$I_E$  – maska zawierająca krawędzie na obrazie wejściowym

$O_k$  –  $k$ -ty obiekt zidentyfikowany na masce wyjściowej

Schemat blokowy rozszerzonej wersji metody *PBAS* został przedstawiony na rysunku 4.14. Oprócz modułu obliczającego maskę pierwszoplanową z użyciem standardowego algorytmu *PBAS* zawiera on dodatkowe bloki odpowiedzialne za wszystkie operacje opisane w niniejszym rozdziale. Moduł *EDGE* służy do wyznaczania krawędzi na aktualnym obrazie wejściowym (oznaczonym jako  $I_N$ ). Blok *CFD* (ang. *consecutive frame differencing*) oblicza różnicę dwóch ramek kolejnych ramek ( $I_N$  i  $I_{N-1}$ ) opisaną równaniem (3.22). Najbardziej złożonym elementem jest moduł *CCA* (ang. *Connected Component Analysis*). Jest w nim wykonywana indeksacja obiektów (opisana szczegółowo w rozdziale 3.3.3), wartością końcową tej operacji są parametry kwadratowego bloku otaczającego zidentyfikowany obiekt (ang. *bounding box*). Oprócz tego są wyznaczane wartości  $S_{O_k}$  i  $EC_{O_k}$  – równania (3.23) i (3.25) oraz finalna maska wyjściowa. Obliczone współczynniki stabilności i podobieństwa krawędzi są przekazywane jako sprzężenie zwrotne do standardowego algorytmu *PBAS*.

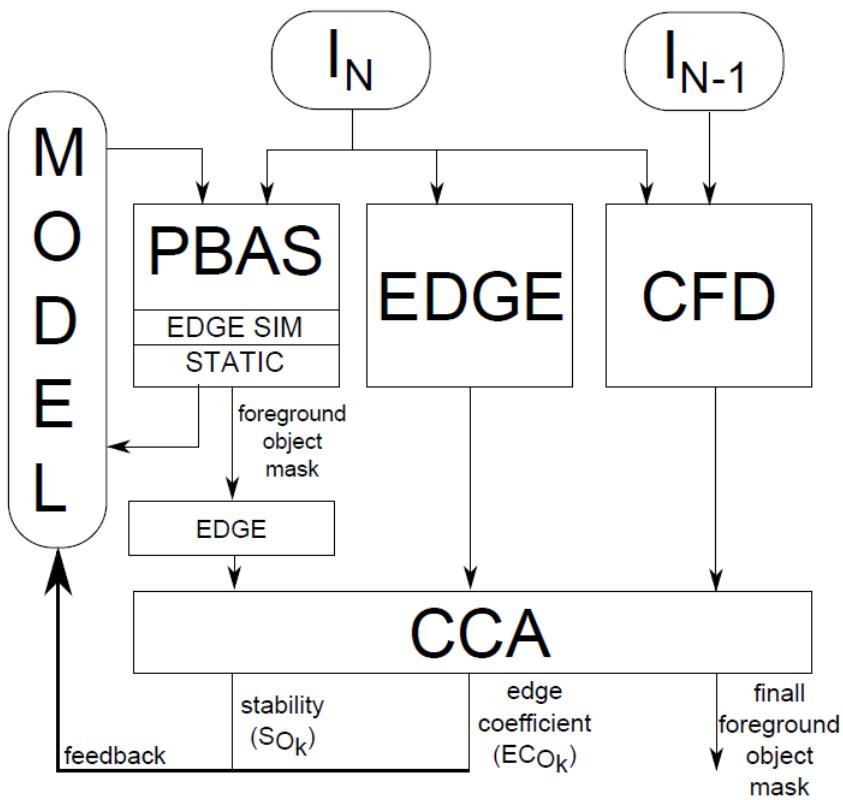
Ostatnim krokiem jest obsługa sprzężenia zwrotnego, w tym celu do modelu tła zostały dodane dwa kolejne elementy. Pierwszym z nich jest licznik  $S(x_i)_{cnt}$  określający przez ile ramek dany piksel był rozpoznawany jako obiekt statyczny. Sposób jego aktualizacji został opisany równaniem (3.26).

$$S(x_i)_{cnt} = \begin{cases} S(x_i)_{cnt} + 1 & \text{jeżeli } S_{O_k} \geq S_{TH} \\ 0 & \text{jeżeli } S_{O_k} < S_{TH} \\ S(x_i)_{cnt} - 1 & \text{w pozostałych przypadkach} \end{cases} \quad (3.26)$$

Operacja ta jest przeprowadzana dla każdego piksela wewnątrz prostokąta otaczającego dany obiekt. Licznik jest inkrementowany dla obiektów zaklasyfikowanych jako statyczne, dla poruszających się zostaje wyzerowany, natomiast dla elementów tła jest dekrementowany.

Drugim dodatkowym elementem modelu jest średnia krocząca współczynnika podobieństwa krawędzi  $EC(x_i)_{mean}$ , jej wartość jest aktualizowana zgodnie z równaniem (3.27).

$$EC(x_i)_{mean} = \begin{cases} 0.5EC(x_i) + 0.5EC(x_i)_{mean} & \text{jeżeli } S_{O_k} \geq S_{TH} \wedge S(x_i)_{cnt} > S_{cnt_{TH}} \\ 1 & \text{w przeciwnym razie} \end{cases} \quad (3.27)$$



Rys. 3.2. Schemat blokowy rozszerzonej metody PBAS – źródło [14]

Gdzie poprzez  $S_{cnt_{TH}}$  oznaczamy minimalną liczbę ramek przez którą obiekt musi pozostać statyczny. Opisane parametry są wykorzystywane w procesie aktualizacji modelu, niezależnie od decyzji podjętej według standardowej procedury, model zostaje zaktualizowany jeżeli spełniony jest warunek logiczny opisany równaniem (3.28).

$$S_{O_k} \geq S_{TH} \wedge S(x_i)_{cnt} > S_{cnt_{TH}} \wedge EC(x_i)_{mean} < 0.5 \quad (3.28)$$

### 3.3.3. Indeksacja obiektów

Opisany w rozdziale 3.3.2 mechanizm detekcji tzw. „duchów” wymaga rozróżnienia poszczególnych obiektów na wynikowej masce binarnej. W celu osiągnięcia takiego rezultatu, konieczne jest zastosowanie algorytmu indeksacji. Istnieje kilka podejść do realizacji tego zagadnienia, w niniejszej pracy zdecydowano się zastosować metodę jednoprzebiegową. Ze względu na jej stosunkowo prostą implementację w układach *FPGA*. Problem implementacji sprzętowej został szerzej rozwinięty w rozdziale 4.7.

Działanie algorytmu polega na nadawaniu poszczególnym pikselom etykiet reprezentujących kolejne obiekty, piksele z taką samą etykietą reprezentują ten sam obiekt. Oczekiwany efektem końcowym, wymaganym w omawianym algorytmie jest pole, oraz parametry prostokąta, otaczającego każdy obiekt w danej ramce obrazu. Otaczający prostokąt jest reprezentowany przez cztery liczby  $[x_1, y_1, x_2, y_2]$ ,

gdzie współrzędne z indeksem 1 oznaczają lewy górny róg prostokąta, natomiast te z indeksem 2 prawy dolny róg.

Pierwszym krokiem w procesie indeksacji jest wygenerowanie sąsiedztwa na podstawie którego aktualnie analizowany piksel będzie klasyfikowany. Jako sąsiedztwo, rozumiemy najbliższe otaczające piksele, ze względu na potokowe przetwarzanie obrazu wystarczy rozpatrzyć trzy piksele, znajdujące się w wyższym rzędzie oraz sąsiada po lewej stronie w rzędzie aktualnie analizowanym. Po wyznaczeniu sąsiedztwa, można przystąpić do klasyfikacji piksela. Jeżeli piksel ma wartość 1 (jest elementem pierwszego planu) należy mu przypisać etykietę.

Podczas operacji przypisania etykiety kolejnemu pikselowi mogą wystąpić trzy przypadki. Najprostszym z nich jest wtedy, gdy żaden z sąsiadujących pikseli nie ma przypisanej żadnej etykiety. W takiej sytuacji aktualny piksel traktowany jest jako fragment nowego obiektu i dostaje nową etykietę. Drugi przypadek to taki w którym, co najmniej jeden piksel posiada już przypisaną etykietą. Jeżeli pośród sąsiednich pikseli pojawia się tylko jedna etykieta oznacza to, że aktualny piksel także jest fragmentem obiektu oznaczonego właśnie tą etykietą, więc zostaje mu ona przypisana. Ostatnim przypadkiem jest tzw. konflikt, występuje on wtedy, gdy pośród sąsiednich pikseli występują różne etykiety. Taka sytuacja może się przytrafić na przykład w sytuacji gdy analizowany jest obiekt w kształcie litery V. Warto zauważać, że ze względu na potokowe przetwarzanie obrazu w obszarze tego samego sąsiedztwa mogą istnieć tylko dwie różne etykiety. W przypadku wystąpienia kolizji oba obiekty łączone są w jeden.

Obszar oraz blok otaczający każdy obiekt, jest aktualizowany na bieżąco wraz z kolejnymi przetwarzanymi pikselami. W przypadku nowego obiektu, pole inicjalizowane jest oczywiście wartością 1 natomiast współrzędne wierzchołków bloku otaczającego przyjmują współrzędne piksela. Dalsza aktualizacja pola obiektu wymaga jedynie inkrementacji wartości wraz z kolejnymi pikselami przypisanymi do danego obiektu. Blok otaczający jest natomiast aktualizowany na podstawie współrzędnych  $x$  i  $y$  aktualnego piksela, zgodnie z równaniem (3.29).

$$(x_1, y_1, x_2, y_2) = (\min(x_1, x), \min(y_1, y), \max(x_1, x), \max(y_1, y)) \quad (3.29)$$

W przypadku wystąpienia konfliktu, pola obu obiektów zostają zsumowane. Lewy górny róg zostaje ustawiony jako minimum spośród dwóch łączonych obiektów, natomiast prawy dolny jako maksimum.

### 3.3.4. Uwagi

Przedstawiony algorytm jest rozszerzoną wersją omówionej w rozdziale 3.2 metody *ViBE*. W tym przypadku wykorzystywany jest bardziej rozbudowany model tła, a sam algorytm charakteryzuje się większą złożonością obliczeniową. W stosunku do wcześniej omawianej metody, zwiększyła się także lista parametrów, kompletna lista została zamieszczona poniżej (w nawiasach umieszczone wartości domyślne):

$N$  – liczba próbek w modelu tła (19)

$\#_{\min}$  – minimalna liczba próbek wymagana w teście dopasowania (2)

$R_{init}$  – początkowa wartość progu dopasowania próbki do modelu (30)

$T_{init}$  – początkowe prawdopodobieństwo wykonania aktualizacji ( $\frac{1}{16}$ )

$R_{low}$  – dolne ograniczenie progu dopasowania próbki do modelu (18)

$T_{low}, T_{up}$  – dolne i górne ograniczenie progu dopasowania próbki do modelu (2 i 200)

$R_{inc/dec}$  – współczynnik wykorzystywany przy aktualizacji parametru  $R$  (0,05)

$R_{SC}$  – współczynnik skalowania wykorzystywany przy aktualizacji parametru  $R$  (5)

W rozszerzonej wersji, zawierającej mechanizm detekcji duchów, należy uwzględnić jeszcze następujące parametry:

$\theta$  – próg wykorzystywany w (3.23), określający ruchome obiekty (50)

$S_{TH}$  – wartość progowa współczynnika stabilności (0,01)

$S_{cnt_{TH}}$  – minimalna liczba ramek przez którą obiekt musi pozostać statyczny (5)

$T_S$  – próg binaryzacji wykorzystywany w detekcji krawędzi metodą Sobela (50)

## 3.4. Gaussian Mixture Models

Metoda *GMM* (ang. *Gaussian Mixture Models*) jest algorytmem wykorzystującym statystyczny model tła. W przeciwieństwie do algorytmów opisywanych w poprzednich rozdziałach, tutaj model tła nie składa się z zapamiętanych próbek. Podobnie jak inne metody, ta także może przetwarzać obraz w różnych przestrzeniach barw.

Jako, że jest to algorytm opublikowana po raz pierwszy w 1999 roku [21], to od tamtej pory doczekała się wielu różnych odmian i udoskonaleń. Ponieważ tematem niniejszej pracy jest analiza i unifikacja algorytmów opracowanych w Laboratorium Biocybernetyki AGH w tym rozdziale zostanie zamieszczony opis powstały w oparciu o prace inżynierskie [9, 20]. W wyżej wymienionych publikacjach zamieszczono wyczerpujący opis metody, zarówno od strony teoretycznej jak i implementacyjnej, niniejsza praca zawiera jedynie podstawowe założenia i główne kroki algorytmu.

### 3.4.1. Opis Algorytmu

Opisywany algorytm opiera się na modelu tła składającym się z  $K$  rozkładów Gaussa, są one podzielone na dwa rodzaje, część z nich reprezentuje tło, reszta obiekty pierwszoplanowe. Model skonstruowany jest tak, że dla każdego piksela zdefiniowany jest osobny zestaw rozkładów Gaussa, który następnie jest niezależnie aktualizowany. Niewątpliwą zaletą w kontekście implementacji w układzie *FPGA* jest brak jakichkolwiek operacji wykorzystujących otoczenie piksela, pozwala to zaoszczędzić dużą część zasobów. Szczegóły implementacji sprzętowej przedstawiono w rozdziale 4.8.

Każdy rozkład Gaussa składa się z trzech elementów: wag ( $\omega$ ), wariancji ( $\sigma$ ) i wartości średniej ( $\mu$ ). W przypadku przetwarzania obrazu w przestrzeni *RGB*, w celu uproszczenie obliczeń, zakłada się, że

waga i wariancja jest identyczna dla wszystkich trzech kanałów. Opierając się na publikacjach [21, 20], liczbę rozkładów (parametr  $K$ ) najlepiej przyjąć z zakresu  $3 - 5$ .

Pierwszym krokiem jest posortowanie rozkładów Gaussa według współczynnika  $r_i = \frac{\omega_i}{\sigma_i}$  w kolejności rosnącej (przez  $i$  rozumiemy numer rozkładu, natomiast  $\omega_{i,t}$  oznacza wagę  $i$ -tego rozkładu w czasie  $t$ ). Ze względu na przyjęte uproszczenie, że wariancja jest identyczna dla wszystkich kanałów RGB, macierz kowariancji  $i$ -tego rozkładu w chwili  $t$  może zostać zapisana równaniem (3.30).

$$\Sigma_{i,t} = \sigma_{i,t}^2 I \quad (3.30)$$

Zastosowanie sortowania według współczynnika  $r_i$  powoduje umieszczenie na początku listy tych rozkładów, które najprawdopodobniej reprezentują tło (czyli tych o najwyższej wadze oraz najniższej wariancji). Przyjęto, że pierwsze  $B$  rozkładów z listy uznaje się za rozkładu reprezentujące tło, kolejne to już obiekty pierwszoplanowe. Jak łatwo zauważyc, rozkłady przedstawiające tło, mają zdecydowanie wyższą wagę oraz niewielką wariancję. Równanie (3.31) określa wartość  $B$ , jest to liczba rozkładów, dla których suma wag przekracza próg  $T$ .

$$B = \arg \min_b \left( \sum_{i=1}^b \omega_{i,t} > T \right) \quad (3.31)$$

Kolejnym etapem to test dopasowania nowego piksela do modelu. Przeprowadzany jest on dla każdego rozkładu Gaussa z posortowanej listy i przerywany w momencie uzyskania pozytywnego wyniku (piksel wejściowy pasuje do modelu). Jeżeli przyjmiemy, że nowy piksel pojawia się w chwili  $t + 1$  to test dopasowania przeprowadzany jest z modelem z chwili  $t$ . Pozytywny wynik testu określa zależność (3.32).

$$\sqrt{\left( (X_{t+1} - \mu_{i,t})^T \cdot \Sigma_{i,t}^{-1} \cdot (X_{t+1} - \mu_{i,t}) \right)} < k\sigma_{i,t} \quad (3.32)$$

Gdzie próg  $k$  domyślnie przyjmuje wartość 2,5

Na tym etapie algorytmu, należy rozważyć dwa przypadki. Jeżeli piksel wejściowy został dopasowany do jednego z  $K$  rozkładów Gaussa, to zostaje on klasyfikowany na podstawie wcześniejszej przynależności danego rozkładu, zgodnie z (3.31). Podczas aktualizacji modelu, zostanie wykorzystany parametr  $\alpha$  określający stałą uczenia. Parametry rozkładu Gaussa, który przeszedł test dopasowania określony równaniem (3.32), zostaje zaktualizowany według zależności (3.33), (3.34) i (3.35).

$$\omega_{i,t+1} = (1 - \alpha)\omega_{i,t} + \alpha \quad (3.33)$$

$$\mu_{i,t+1} = (1 - \rho)\mu_{i,t} + \rho X_{t+1} \quad (3.34)$$

$$\sigma_{i,t+1}^2 = (1 - \rho)\sigma_{i,t}^2 + \rho(X_{t+1} - \mu_{i,t+1})(X_{t+1} - \mu_{i,t+1})^T \quad (3.35)$$

Do przedstawienia funkcji  $\rho$  konieczna jest także definicja funkcji gęstości prawdopodobieństwa, która jest opisana równaniem (3.36). Z kolei samą funkcję  $\rho$  przedstawia równanie (3.37).

$$\eta(X_t, \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} e^{-\frac{1}{2}(X_t - \mu) \Sigma^{-1} (X_t - \mu)} \quad (3.36)$$

gdzie:

$X_t$  – wartość piksela w chwili t

$\mu$  – wektor wartości średnich rozkładu Gaussa

$\Sigma$  – macierz kowariancji – wzór (3.30)

$$\rho = \alpha \eta(X_{t+1}, \mu_i, \Sigma_i) \quad (3.37)$$

Reszta rozkładów, która nie przeszła testu dopasowania, zostaje zaktualizowana według równania (3.38), nadpisywana jest tylko waga, wartość średnia i wariancja pozostają niezmienione.

$$\omega_{j,t+1} = (1 - \alpha)\omega_{j,t} \quad (3.38)$$

Drugim przypadkiem to piksel niepasujący do żadnego rozkładu. W takiej sytuacji jest on automatycznie klasyfikowany jako element pierwszoplanowy. Aktualizacja modelu w tym przypadku polega na zastąpieniu rozkładu o najniższej wadze nowym. Sama waga pozostaje niezmieniona, natomiast aktualna wartość piksela zapisywana jest jako wartość średnia. Przypisywana jest także duża wariancja inicjalizująca, która jest określona globalnie jako parametr całego algorytmu.

Warto jeszcze raz nadmienić, że powyższe operacje wykonywane są cyklicznie dla każdego piksela z wykorzystaniem niezależnego modelu tła. Lista wszystkich parametrów algorytmu *GMM* przedstawia się następująco:

$K$  – liczba rozkładów Gaussa

$T$  – próg z przedziału 0–1 pozwalający rozgraniczyć rozkłady tła i pierwszoplanowe

$k$  – współczynnik używany w teście dopasowania (3.32)

$\alpha$  – współczynnik uczenia z przedziału 0–1

$\sigma_{init}$  – wariancja, którą inicjalizowany jest nowy gaussian

### 3.4.2. Uwagi

Algorytm opisany w niniejszym rozdziale, przedstawia nieco odmienne podejście do segmentacji obiektów pierwszoplanowych w stosunku do metod opisanych wcześniej. Jak już zostało wspomniane w rozdziale 2 algorytm *GMM* jest bardzo rozległym zagadnieniem, będącym przedmiotem badań wielu

publikacji [21, 5, 20, 23]. Ze względu na skomplikowane operacje matematyczne z punktu widzenia układów *FPGA*, sama implementacja również jest dość złożonym problemem.

Jak słusznie zauważyli autorzy przytoczonych publikacji, algorytm *GMM* zapewnia zadowalające rezultaty, aczkolwiek jest bardzo podatny na różnego rodzaju zakłócenia, szумy i zmiany oświetlenia. Mimo to w większość przypadków, jest to metoda, mogąca z powodzeniem zostać wykorzystana jako samodzielny system segmentacji tła.



## 4. Implementacja sprzętowa wybranych algorytmów

### 4.1. Modele programowe

Przed przystąpieniem do implementacji sprzętowej, dla każdego algorytmu przygotowano także model programowy. Zostały one zaimplementowane w języku *C++* z wykorzystaniem biblioteki *OpenCV* [18]. Całość przygotowano w środowisku programistycznym *Microsoft Visual Studio 2015*. Głównym celem stworzonych implementacji jest przeprowadzenie szczegółowych testów poszczególnych algorytmów. Tematyka ta została szczerzej opisana w rozdziale 5, natomiast design poszczególnych algorytmów zaprezentowano w dodatku B.

Na rysunku 4.1 przedstawiono zrzut ekranu prezentujący działanie modelu programowego metody *PBAS*. Aplikacja wyświetla cztery okna, na których przedstawione są kolejno: obraz wejściowy w przestrzeni RGB, wzorcowa maska wyjściowa (ang. *ground truth*), efekt końcowy algorytmu *PBAS*, wizualizacja rozszerzonej wersji tej metody. Rysunek 4.2 przedstawia z kolei działający algorytm *ViBE*. Podobnie jak w poprzednim przypadku wyświetlone są cztery okna: wyjście, wzorzec, maska wyjściowa oraz wizualizacja przepływu optycznego.



Rys. 4.1. Model programowy metody *PBAS*

Wykonane implementacje oczywiście nie są w stanie zapewnić przetwarzania obrazu w czasie rzeczywistym. Rozdzielcość obrazu w wykorzystanych sekwencjach testowych zawiera się w przedziale od  $320 \times 240$  do  $720 \times 576$  pikseli. Nawet w przypadku tak niewielkiego obrazu otrzymana prędkość przetwarzania, w zależności od algorytmu, wahała się od kilku do co najwyżej kilkunastu klatek na sekundę.



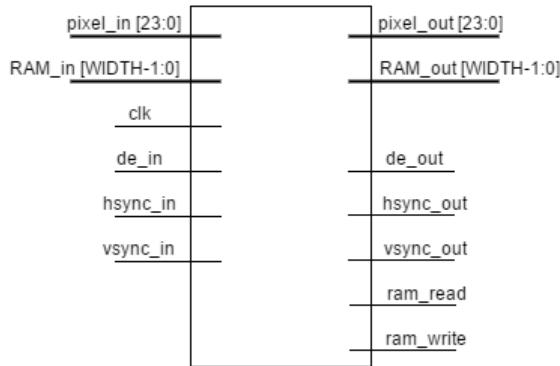
Rys. 4.2. Model programowy metody ViBE

## 4.2. Wykorzystany układ oraz interfejs wizyjny

Do implementacji algorytmów wykorzystano środowisko Vivado. Użyto najnowszej wersji, na moment tworzenia niniejszej pracy (czerwiec 2017), oznaczonej numerem 2017.1. Większość modułów zaimplementowano w języku Verilog. W pojedynczych przypadkach wykorzystano również język VHDL. Wszystkie metody uruchomiono i sprawdzono na płytce VC707 z układem Virtex-7 (XC7VX485T-2FFG1761). Jest to jeden z najwydajniejszych układów firmy Xilinx. Dodatkowo w celu podłączenia kamery i transmisji obrazu wykorzystano kartę Avnet DVI I/O FMC. Przyjęto założenie, że zaimplementowane algorytmy mają pracować w rozdzielcość  $720 \times 576$  w 50 klatkach na sekundę. Opcjonalnie obsługiwana może być również rozdzielcość  $1280 \times 720$  i  $1920 \times 1080$ , również z prędkością  $50\text{ }fps$  (ang. *Frames Per Second* – klatki na sekundę). Ograniczeniem w przypadku wyższych rozdzielcości są głównie wielkości pamięci RAM, którą można przeznaczyć na model tła, oraz wymagana częstotliwość zegara. Dla obrazu  $576p@50fps$  wymagana jest praca układu z częstotliwością 27MHz,. W przypadku rozdzielcości  $720p@50fps$  oraz  $1080p@50fps$  minimalna częstotliwość zegara musi wynosić już odpowiednio 74,25MHz i 148,5MHz.

W przypadku implementacji sprzętowej w układzie *FPGA* została wykorzystany uniwersalny interfejs wizyjny. Jego układ, przedstawiający podstawowe sygnały wejściowe i wyjściowe, został pokazany na rys. 4.3. Oprócz sygnałów wejściowych/wyjściowych z kamery, konieczne jest także zapewnienie

niezależnego obszar pamięci *RAM* dla każdego piksela. Maksymalnie do wykorzystania są 1024 bity na piksel w przypadku obrazu o rozdzielczości  $720 \times 576$  oraz 256 bitów dla wyższych rozdzielczości ( $720p$  i  $1080p$ ). Sygnały pochodzące z kamery zostały szerzej opisane w rozdziale 4.3.1. Należy zwrócić uwagę, że przedstawiony tutaj interfejs, zawiera jedynie niezbędne sygnały do funkcjonowania systemu. W przypadku niektórych algorytmów został on rozszerzony o dodatkowe wejścia/wyjścia.



Rys. 4.3. Główny interfejs wizyjny

Moduł przedstawiony na rys. 4.3 posiada następujące sygnały wejściowe:

*pixel\_in* – wartość piksela wejściowego w formacie RGB (24 bity)

*bg\_model\_in* – odczytana pamięć RAM o długości *WIDTH* (maksymalnie 1024 bity)

*clk* – zegar piksela

*de\_in* – flaga poprawności piksela wejściowego

*hs\_in* – flaga synchronizacji poziomej

*vs\_in* – flaga synchronizacji pionowej

Natomiast sygnałami wyjściowymi są:

*pixel\_out* – wartość piksela wyjściowego w formacie RGB (24 bity)

*bg\_model\_out* – blok o długości *WIDTH* do zapisania w pamięci RAM (maksymalnie 1024 bity)

*de\_out* – opóźniona flaga poprawności piksela wejściowego

*hs\_out* – opóźniona flaga synchronizacji poziomej

*vs\_out* – opóźniona flaga synchronizacji pionowej

*ram\_read* – sygnał odczytu pamięci RAM

*ram\_write* – sygnał zapisu do pamięci RAM

Istotną kwestią jest również zużycie zasobów oraz pobór energii przez układ *FPGA*. Jak łatwo zauważać, sam interfejs wizyjny w połączeniu z kontrolerem pamięci *RAM* jest zagadnieniem dość rozbudowanym. Tabela 4.1 przedstawia zestawienie wykorzystanej logiki programowej w przypadku prostego algorytm odejmowania ramek. Oszacowany pobór mocy wynosi natomiast 2,804W. Odejmowanie ramek jest najprostszą metodą zaimplementowaną w niniejszej pracy, jej szczegółowy opis zamieszczono

w rozdziale 3.1. Zaprezentowane tutaj zestawienie będzie stanowić punkt odniesienia dla bardziej zaawansowanych implementacji.

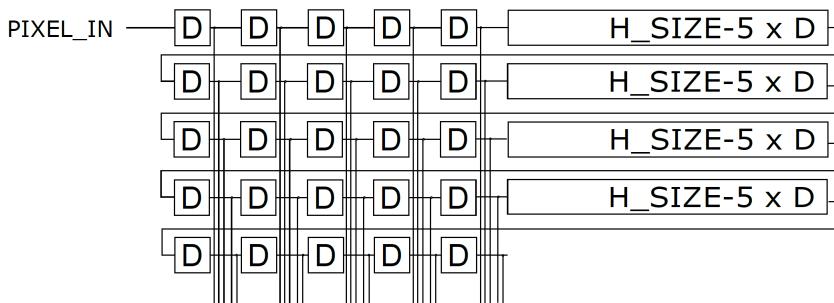
**Tabela 4.1.** Odejmowanie ramek - wykorzystanie zasobów (*Virtex 7*)

Zasoby	Wykorzystane	Dostępne	Użycie w %
<b>LUT</b>	9514	303600	3,13
<b>LUTRAM</b>	1418	130800	1,08
<b>FF</b>	9063	607200	1,49
<b>BRAM</b>	71	1030	6,89
<b>DSP</b>	0	2800	0,0

## 4.3. Trudności występujące w potokowym systemie wizyjnym

### 4.3.1. Kontekst poziomy i pionowy

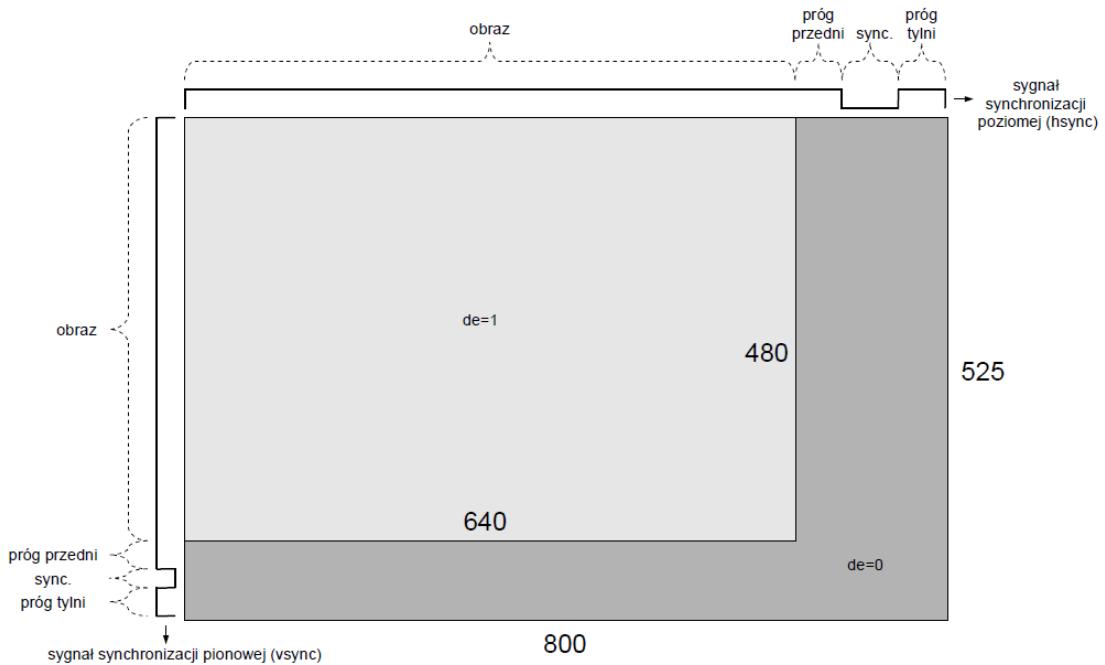
Algorytmy wizyjne w układach FPGA, jak już zostało wspomniane, są realizowane w systemie potokowym. Oznacza to, że ramka obrazu przetwarzana jest piksel po pikselu. W związku z tym, w celu wykonania operacji kontekstowej koniecznej jest zastosowanie odpowiednich opóźnień. Dla kontekstu poziomego nie jest to duży problem i może zostać rozwiązany z wykorzystaniem zwykłych rejestrów. Natomiast w przypadku kontekstu pionowego konieczne jest zastosowanie opóźnienia o całą linie obrazu. Do tego celu, wykorzystuje się dostępną w układach FPGA pamięć blokową (*BRAM*). Przykładowy system linii opóźniających, służących do wyznaczenia kontekstu o rozmiarze  $5 \times 5$  został przedstawiony na rysunku 4.4. Poprzez pojedynczy blok **D** rozumiemy opóźnienie o jeden takt zegara, z kolei stała *H\_SIZE* definiuje ilość pikseli w jednej linii obrazu.



**Rys. 4.4.** Schemat długiej linii opóźniającej - źródło [11]

Podczas ustalania parametru *H\_SIZE* należy zwrócić uwagę na tzw. obszar synchronizacji. W rzeczywistości ilość pikseli przesyłanych w jednej ramce obrazu jest większa niż sugerowałaby rozdzielcość obrazu. Przykładowo, dla obrazu o rozdzielcości  $640 \times 480$  pikseli, rzeczywista liczba pikseli, po uwzględnieniu obszaru synchronizacji wynosi  $800 \times 600$ . Zostało to przedstawione na rysunku 4.5.

Wielkość obszaru synchronizacji zależy zarówno od rozdzielczości obrazu jak i liczby klatek na sekundę. Przy wyznaczaniu odpowiedniej długości, wymaganej linii opóźniającej, pomocna może okazać się baza danych *Modeline Database* [17]. W niniejszej pracy wykorzystano trzy rozdzielczości: *720x576*, *1280x720*, *1920x1080* przy prędkości *50 fps*. Całkowita długość linii poziomych, dla takich konfiguracji, wynosi odpowiednio: *864*, *1680*, *2640*.



Rys. 4.5. Synchronizacja dla obrazu o rozdzielczości *640x480* – źródło [11]

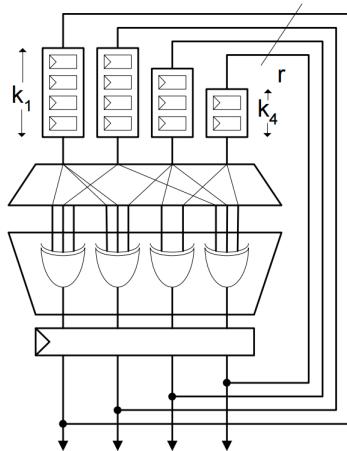
### 4.3.2. Generowanie liczb losowych

Moduł odpowiedzialny za generację liczb pseudolosowych został opracowany na podstawie publikacji [22]. Wykorzystano go także w wielu innych badań przeprowadzanych w Laboratorium Biocybernetyki AGH, między innymi [15, 14]. W przeciwieństwie do pozostałych modułów i algorytmów, generator został w całości zaimplementowany w języku *VHDL*.

Zagadnienie liczb pseudolosowych w układach reprogramowalnych jest tematem bardzo rozległym i zostało dokładnie opisane w przytoczonym artykule. W niniejszym rozdziale zostanie przedstawiona jedynie skrócona idea działania wykorzystanego generatora. Cała koncepcja opiera się na wykorzystaniu rejestrów przesuwnych (ang. *shift register*) i modułów *LUT* (ang. *Look-Up Table*). Uproszczony schemat, tego typu generatora, zamieszczono na rysunku 4.6. Wartości wyjściowe są zapisywane w rejestrach przesuwnych o różnych długościach. Następnie różne kombinacje zapamiętanych bitów są podawane na wejścia bramek *XOR*, które generują pseudolosowy sygnał wyjściowy.

Architekturę generatora można opisać za pomocą zestawu 4 parametrów ( $n, r, t, k$ ), gdzie:

$n$  – liczba stanów generatora (okres możemy zapisać jako  $2^n - 1$ )



Rys. 4.6. Schemat generatora liczb pseudolosowych – źródło [22]

$r$  – liczba bitów generowanych w każdym cyklu

$t$  – liczba wejść każdej bramki  $XOR$

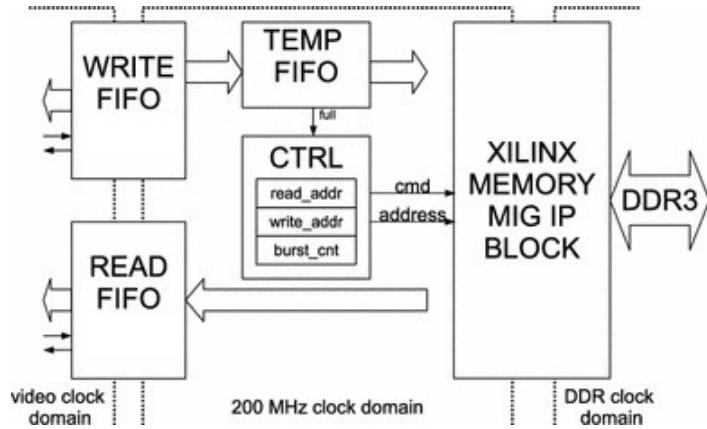
$k$  – maksymalna długość rejestrów przesuwnych

Wykorzystany w niniejszej pracy generator posiada następujące parametry:  $n = 3900$ ,  $r = 128$ ,  $t = 5$ ,  $k = 32$ . Istnieje jeszcze jeden dodatkowy parametr  $s$ , który określa sposób połączenia wyjść rejestrów przesuwnych z bramkami  $XOR$ . Dokładny algorytm, opisujący sposób doboru optymalnych połączeń, jest bardzo zaawansowanym zagadnieniem, wybiegającym poza tematykę niniejszej pracy. Szczegółowy opis i analiza tego podejścia została przedstawiona w przytoczonej publikacji.

#### 4.3.3. Kontroler pamięci RAM

Kontroler pamięci *RAM DDR3*, jest niezbędnym elementem, koniecznym do prawidłowego działania każdego rozbudowanego systemu wizyjnego. Zewnętrzna pamięć, jest bowiem niezbędna do przechowywania modelu tła dla poszczególnych pikseli. Przedstawiony kontroler, został opracowany w Laboratorium Biocybernetyki AGH w ramach publikacji [13]. Podobnie jak w przypadku generatora liczb pseudolosowych, rozdział ten ma na celu jedynie przedstawienie uproszczonej koncepcji działania modułu. Dokładny opis i dyskusja zostały zawarte w przytoczonej publikacji. *Xilinx* zapewnia dedykowany moduł *MIG (Memory Interface Generator)* do komunikacji z zewnętrzną pamięcią *RAM*. Schemat kontrolera pamięci, wykorzystującego *MIG* został przedstawiony na rysunku 4.7.

Ze względu na fakt, że pamięć *RAM* synchronizowana jest innym zegarem niż sygnał z kamery konieczne jest zastosowanie buforów w postaci dodatkowych kolejek *FIFO*. Omawiany kontroler został zaimplementowany jako maszyna stanów. Podczas etapu inicjalizacji zapełniona zostaje w całości kolejka *READ FIFO*. Następnie, w momencie gdy pojawi się na wejściu nowa ramka obrazu, modele tła z kolejki *READ FIFO* zostają odczytywane i przekazywane do modułu realizującego algorytm segmentacji tła. Zaktualizowany model tła otrzymany na wyjściu, jest umieszczany w kolejce *WRITE FIFO*.



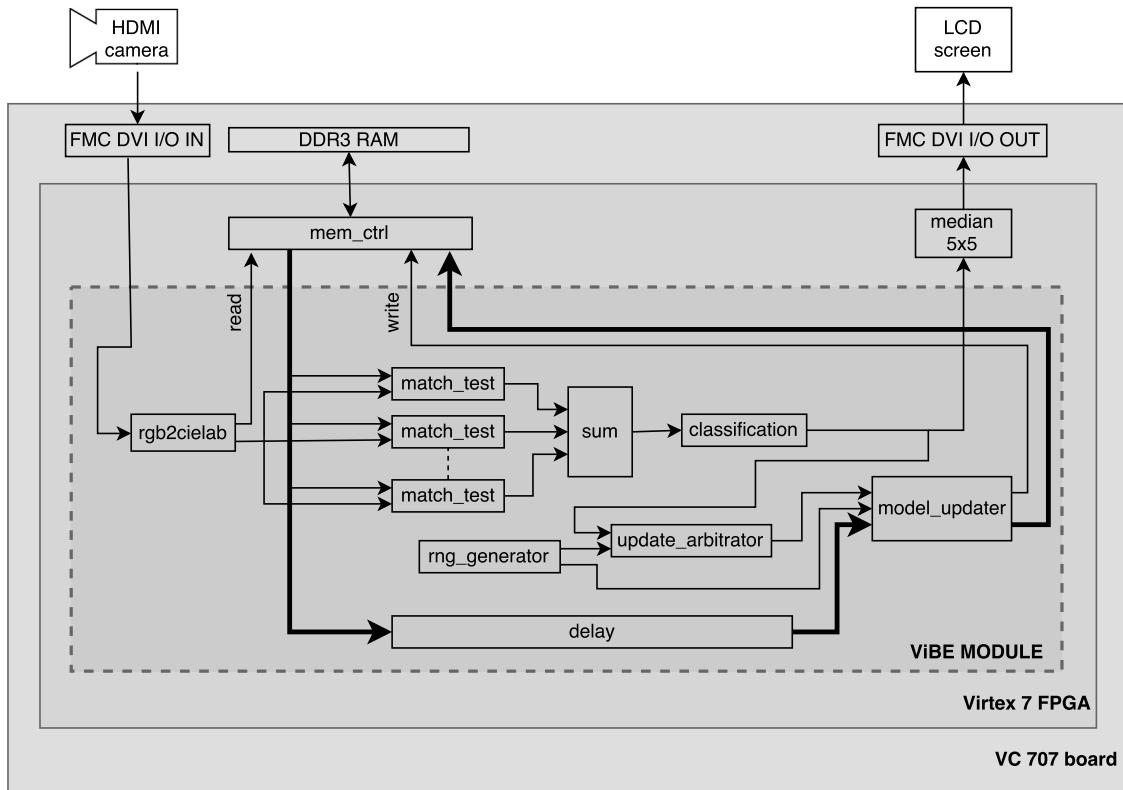
Rys. 4.7. Schemat kontrolera pamięci RAM – źródło [13]

Kolejnym krokiem jest przeniesienie danych z kolejki *WRITE FIFO* do dużo krótszej *TEMP FIFO*. W momencie, gdy kolejka ta jest zapełniona, uruchamiany jest tryb *burst*, zapisujący wszystkie modele do pamięci *RAM*. Po dokonaniu zapisu, dokładnie taka sama liczba modeli, zostaje odczytana z pamięci *RAM* i ponownie umieszczona w kolejce *READ FIFO*. Następnie kontroler przechodzi do stanu oczekiwania, aż do momentu ponownego zapełnienia się kolejki *TEMP FIFO*.

#### 4.4. Implementacja algorytmu ViBE

Przygotowana implementacja sprzętowa, powstała na podstawie opisu teoretycznego metody, przedstawionego w rozdziale 3.2, wykorzystano wersję operującą w przestrzeni *CIELab*. Wysokopoziomowy schemat implementacji, zawierający główny moduł realizujący algorytm, wejściowy i wyjściowy sygnał z kamery oraz połączenie z pamięcią RAM, został przedstawiony na rysunku 4.8. Dla zachowania spójności, w dalszej części opisu, przyjęto oznaczenia parametrów algorytmu identyczne jak te zestawione w podsumowaniu rozdziału teoretycznego (podrozdział 3.2.6).

W podstawowej wersji algorytmu, wykorzystany został, standardowy interfejs wizyjny, przedstawiony na rysunku 4.3, bez żadnych dodatkowych sygnałów. Pierwszą operacją, jest oczywiście konwersja z przestrzeni *RGB* do *CIELab* (moduł *rgb2cielab*). Operacja przekształcenia do macierzy *XYZ*, opisana równaniem (3.5), jest prosta do zaimplementowania w logice programowej i została zrealizowana z wykorzystaniem mnożarek sprzętowych operujących na liczbach stałoprzecinkowych. Obliczanie wartości funkcji  $f(t)$  zdefiniowanej równaniem (3.6) zostało zaimplementowane z użyciem operacji *LUT* (ang. *Look-Up Table*). Zastosowano trzy moduły, przechowujące stablicowane wartości funkcji  $f(\frac{X}{X_n})$ ,  $f(\frac{Y}{Y_n})$ ,  $f(\frac{Z}{Z_n})$  wymagane do obliczenia składowych  $a$  i  $b$  danych równaniem (3.7) oraz jeden moduł zawierający wartości składowej  $L$ . Jak zostało już wspomniane w rozdziale 3.2.2 składowe  $a$  i  $b$  mieszczą się w przedziale -128 do 127. Natomiast składowa  $L$  w zakresie 0 – 100. W związku z tym rozmiar piksela to **23 bity** zamiast 24, jak miało to miejsce w przypadku przestrzeni *RGB*.

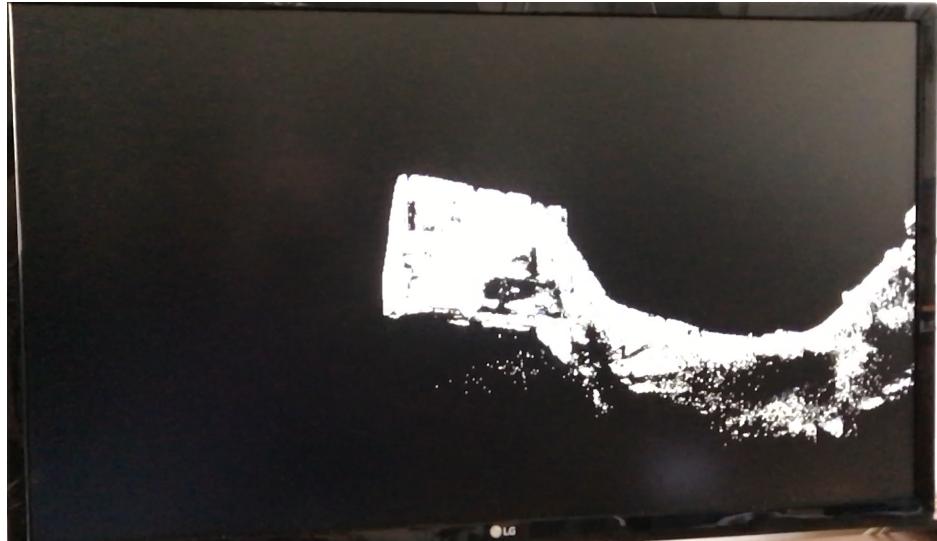


Rys. 4.8. Wysokopoziomowy schemat implementacji algorytmu *ViBE*

Dla poszczególnych próbek modelu obliczany jest dystans do aktualnego piksela zgodnie z równaniem (3.4). Otrzymana wartość zostaje porównywana z progiem  $R$ . Operacje te wykonywane są równolegle dla każdej próbki z wykorzystaniem bloczków *match\_test*. Następnie w module *sum*, zliczane są próbki dla których test dopasowania przeszedł pozytywnie. Ostatnim etapem jest blok *classification* dokonujący ostatecznej klasyfikacji piksela zgodnie z równaniem (??). Dodatkowo, w celu eliminacji szumów, na maskę wyjściową nakładany jest filtr medianowy o rozmiarze  $5 \times 5$ .

W procesie aktualizacji modelu wykorzystywany jest, 128 bitowy, losowy sygnał otrzymany z bloku *rng\_generator*. Idea działania generatora liczb losowych została szerzej opisana w rozdziale 4.3.2. Modułu *update\_arbitrator* jest wykorzystywany do podejmowania decyzji odnośnie aktualizacji. Sama aktualizacja wykonywana jest natomiast w bloku *model\_updater*. W tym celu, generowany jest kontekst o rozmiarze  $3 \times 3$ , zawierający modele tła sąsiadujących pikseli. Na podstawie informacji, otrzymanych z modułu *update\_arbitrator*, aktualizowany jest model piksela oraz losowo wybranego sąsiada. Następnie nowy model zostaje zapisany do pamięci *RAM*. Warto zwrócić uwagę, że model jest odczytywany z pamięci po konwersji do przestrzeni *CIELab*, aby wykorzystać go w procesie aktualizacji, konieczne jest zastosowanie linii opóźniającej o długości równej latencji procesu klasyfikacji piksela, operacja ta została zrealizowana przez moduł *delay*.

Przedstawioną implementację udało się uruchomić w rozdzielczościach: *576p*, *720p*, *1080p* w 50 klatkach na sekundę. Na rysunku 4.9 przedstawiono zdjęcie działającego systemu. W przypadku najniższej rozdzielczości przyjęto model składający się z  $N = 20$  próbek (rozmiar modelu wynosi w tym



Rys. 4.9. Działający algorytm ViBE

przypadku  $20 \cdot 23 = 460$  bitów). Dla wyższych rozdzielczości, ze względu na ograniczenia pamięci *RAM*, musiał on zostać ograniczony do 10 (rozmiar modelu równy  $10 \cdot 23 = 230$  bity). Pozostałym parametrom przypisano wartości domyślne, zostały one zapisane jako liczny całkowite. Wyjątkiem jest parametr *T*, opisujący prawdopodobieństwo wykonania aktualizacji, w tym przypadku została wykorzystana 16 bitowa liczba stałoprzecinkowa (ang. *fixed float*) bez znaku o oznaczeniu *8z8u*, czyli po 8 bitów przeznaczonych na część całkowitą i ułamkową.

Zużycie zasobów w układzie *FPGA* zamieszczono w tabeli 4.2 i 4.3, odpowiednio dla rozdzielczości *576p* oraz *1080p*. Oszacowany w obu przypadkach pobór mocy wynosi  $3,957\text{W}$  i  $3,370\text{W}$ . Można zauważać wyraźny wzrost wykorzystania dostępnych zasobów w stosunku do algorytmu odejmowania ramek (taba 4.1), wynika to oczywiście ze zdecydowanie bardziej rozbudowanej logiki algorytmu. Na zdecydowanie większe użycie *LUT* miało wpływ między innymi zastosowanie konwersji *RGB–CIELab*, gdzie konieczne było tablicowanie wartości niektórych funkcji. Pamięć *BRAM* została z kolei wykorzystana w długich liniach opóźniających piksele jak i cały model tła. Warto jeszcze raz zaznaczyć, że w przypadku wyższej rozdzielczości konieczne było ograniczenie rozmiaru modelu (ze względu na ograniczenia zewnętrznej pamięci *RAM*). W związku z tym, ostateczne zużycie zasobów w systemie pracującym w *1080p@50fps* jest niższe niż dla *576p@50fps*.

## 4.5. Implementacja rozszerzonej wersji ViBE

W rozdziale 3.2.3 opisany został dodatkowy mechanizm, usprawniający pracę algorytmu w przypadku występowania drgań kamery (ang. *camera jitter*). Schemat implementacji rozszerzonej wersji algorytmu, zawierającej ten dodatkowy moduł, został przedstawiony na rysunku 4.10. Implementacja

**Tabela 4.2.** *ViBE 576p@50fps - wykorzystanie zasobów (Virtex 7)*

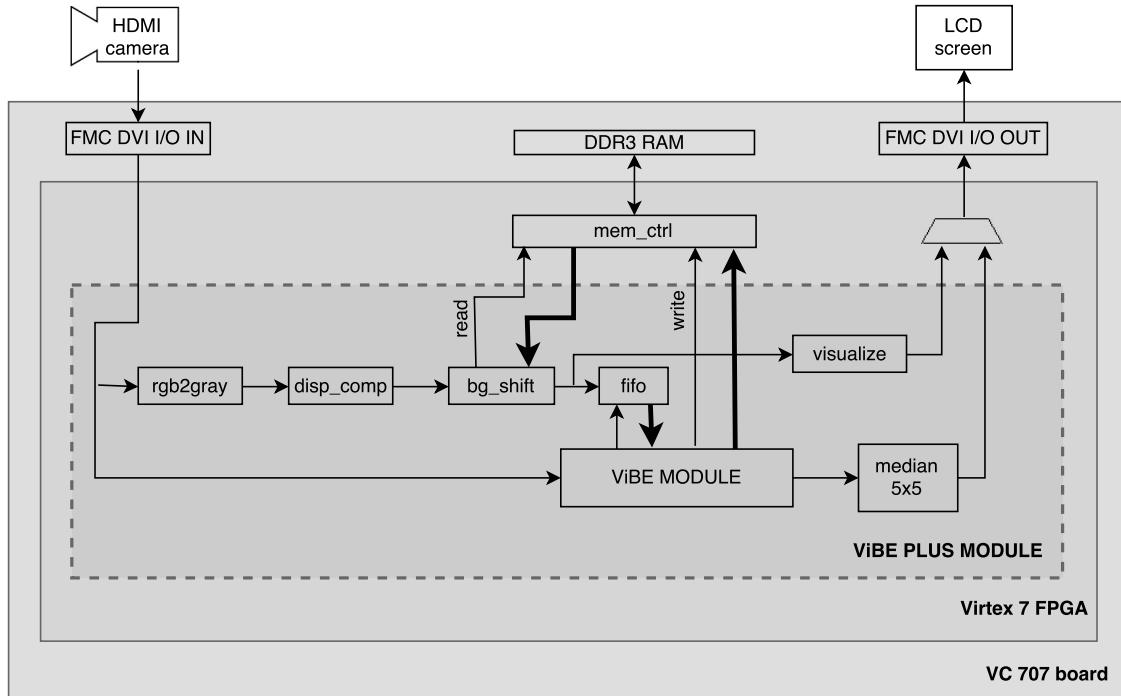
Zasoby	Wykorzystane	Dostępne	Użycie w %
<b>LUT</b>	19774	303600	6,51
<b>LUTRAM</b>	2083	130800	1,59
<b>FF</b>	20936	607200	3,45
<b>BRAM</b>	207	1030	20,10
<b>DSP</b>	54	2800	1,93

**Tabela 4.3.** *ViBE 1080p@50fps - wykorzystanie zasobów (Virtex 7)*

Zasoby	Wykorzystane	Dostępne	Użycie w %
<b>LUT</b>	16224	303600	5,34
<b>LUTRAM</b>	2107	130800	1,61
<b>FF</b>	16236	607200	2,67
<b>BRAM</b>	163	1030	15,83
<b>DSP</b>	29	2800	1,04

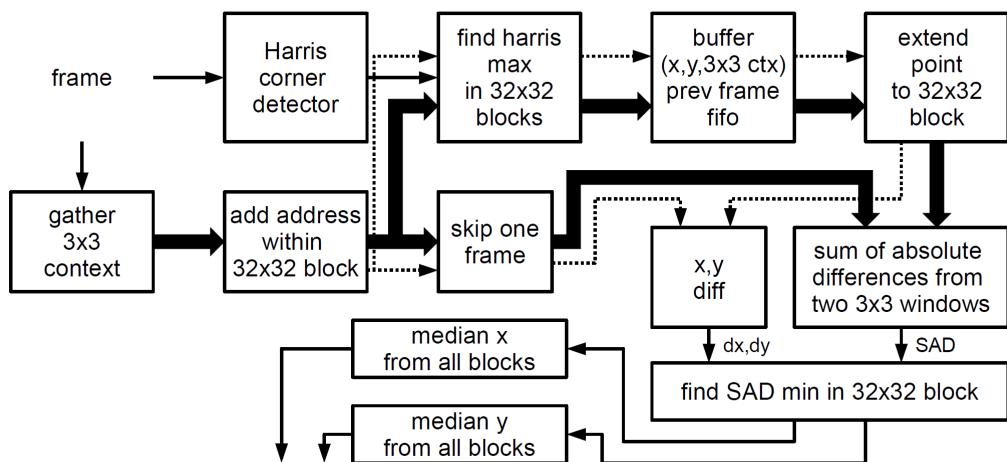
odpowiada opisowi teoretycznemu, który został przedstawiony w podrozdziale 3.2.3. Oprócz bloku realizującego standardowy algorytm *ViBE*, opisanego szczegółowo w poprzednim rozdziale, ta wersja algorytmu zawiera także szereg modułów zapewniających prawidłowe przesunięcie modelu tła na podstawie wyznaczonego przepływu optycznego. Moduł, podobnie jak poprzednia wersja, wykorzystuje standardowy interfejs wizyjny, schemat modelu tła jest również identyczna jak w podstawowej wersji algorytmu.

Pierwszym krokiem, podczas obliczania przepływu optycznego, jest konwersja obrazu z przestrzeni *RGB* do skali szarości. Jest to wykonywane poprzez moduł *rgb2gray*. Przekształcony sygnał z kamery jest następnie podawany na wejście bloku *disp\_comp*, realizującego operację wyznaczania przesunięcia modelu na podstawie wyliczonego przepływu optycznego. Jest to najbardziej złożony moduł w całej implementacji. Jego szczegółowy schemat został przedstawiony na rysunku 4.11. Na podstawie otrzymanego wektora przesunięcia blok *bg\_shift* zapewnia prawidłową synchronizację i odczyt kolejnych modeli tła z pamięci *RAM*. Ze względu na przesunięcie, operacja odczytu, musi zostać wykonana z odpowiednim wyprzedzeniem lub opóźnieniem. Dane z pamięci *RAM* są umieszczane w kolejce *FIFO* (ang. *First In First Out* – pierwszy na wejściu, pierwszy na wyjściu). Skąd następnie są pobierane przez algorytm *ViBE*. Kolejka *FIFO*, podobnie jak linie opóźniające, została zaimplementowana z wykorzystaniem pamięci blokowej (*BRAM*). Dzięki zastosowaniu uniwersalnego interfejsu z niezależnymi sygnałami odczytu i zapisu do pamięci RAM, nie było konieczności modyfikowania oryginalnej implementacji metody *ViBE*. Na maskę wyjściową, nakładany jest filtr medianowy o rozmiarze  $5 \times 5$ , dodatkowo został zaimplementowany moduł wizualizujący operację obliczania przepływu optycznego.



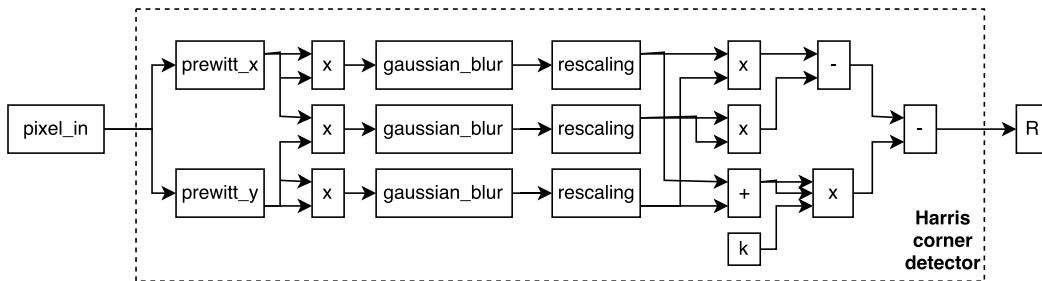
Rys. 4.10. Wysokopoziomowy schemat implementacji rozszerzonego algorytmu ViBE

Rozszerzona wersja algorytmu operuje jedynie na rozdzielcości  $720 \times 576$ . Zastosowano identyczne parametry jak w podstawowej wersji metody, czyli model tła składający się z 20 próbek w przestrzeni *CIELab*. Składowa wektora przesunięcia może przyjąć maksymalnie wartość 32, zatem kolejka *FIFO* gromadząca modele tła musi mieć długość  $32 \cdot H\_SIZE + 32$ , gdzie każdy element ma szerokość równą modelowi tła. Dla obsługiwanej rozdzielcości parametr *H\_SIZE* wynosi 864, natomiast model tła ma rozmiar 460 bitów. Ostatecznie ustalono szerokość pojedynczego elementu na 512 bitów, natomiast głębokość kolejki wynosi 32768.



Rys. 4.11. Implementacja modułu wyznaczającego wektor przesunięcia – źródło []

Moduł realizujący wyznaczanie wektora przesunięcia, jest najbardziej złożonym elementem algorytmu. Opis teoretyczny tej metody został przedstawiony w podrozdziale 3.2.4. Pierwszym krokiem algorytmu jest detekcja narożników metodą Harrisa-Stephensa. Schemat przedstawiający implementację tej funkcji został pokazany na rysunku 4.12. Wykonane tam operacje odpowiadają opisowi teoretycznemu zamieszczonemu w podrozdziale 3.2.5. Przy implementacji uwzględniono w maksymalnym stopniu możliwości układu FPGA w zakresie zrównoleglnania obliczeń. Moduły *prewitt\_x* i *prewitt\_y* realizują detekcję krawędzi pionowych i poziomych. Następnie wszystkie trzy elementy macierzy  $H$  są obliczane równolegle. Po dokonaniu operacji wygładzania filtrem Gaussa (blok *gaussian\_blur*) w bloku *rescaling* następuje przeskalowanie (dzielenie przez  $2^{10}$ ) otrzymanych wartości. Finalnie wyznaczany jest wyznacznik macierzy  $H$ , zgodnie z równaniem (3.13). Całość została zrealizowana z wykorzystaniem sprzętowych mnożarek i sumatorów. Konieczne jest wykonanie 6 operacji mnożenia, jednego dodawania i dwóch odejmowania.



Rys. 4.12. Moduł realizujący detekcję narożników metodą Harrisa-Stephensa

Równolegle do detekcji krawędzi wyznaczany jest kontekst aktualnego piksela o rozmiarze  $3 \times 3$ , określana jest także jego pozycja w aktualnie przetwarzanym bloku o rozmiarze  $32 \times 32$ . Piksel dla którego otrzymano najmniejszą wartość współczynnika  $R$  w obszarze każdego bloku zostaje zapisany razem z kontekstem  $3 \times 3$  w kolejce *FIFO* (w przypadku obrazu o rozdzielcości  $720 \times 576$  konieczne jest zdefiniowanie kolejki o rozmiarze 396, gdyż tyle jest bloków  $32 \times 32$  w jednej ramce). Kolejne elementy z kolejki, są pobierane w trakcie przetwarzania następnej ramki obrazu. W każdym z bloków znajdowana jest minimalna wartość sumy różnicy modułów pomiędzy pikselem pobrańym z kolejki i pikselami z aktualnej ramki obrazu. Znaleziona wartość, wraz z wektorem przesunięcia, zostaje po raz kolejny zapamiętana w pamięci blokowej.

Ostatnim krokiem, jest obliczenie mediany przesunięcia  $dx$  i  $dy$  wśród znalezionych wartości minimalnych. Operacja ta, składa się z dwóch etapów, wykonywanych w momencie przerwy pomiędzy kolejnymi ramkami obrazu (wartość sygnału *vsync* wynosi wtedy 0). Najpierw wyliczany jest histogram wartości  $dx$  i  $dy$ , równolegle zliczana jest także ilość próbek. Drugim krokiem jest sumowanie wartości histogramu. Poprzez mediane, rozumiemy wartość, dla której aktualna suma wartości histogramu jest większa niż połowa liczby wszystkich próbek. Do obliczenia histogramu ponownie został wykorzystany moduł pamięci blokowej (*BRAM*). Na rysunku 4.13 przedstawiono wizualizację wyznaczanego przepływu optycznego, analogiczną do tej pokazanej na rysunku 3.1.



Rys. 4.13. Wizualizacja przepływu optycznego

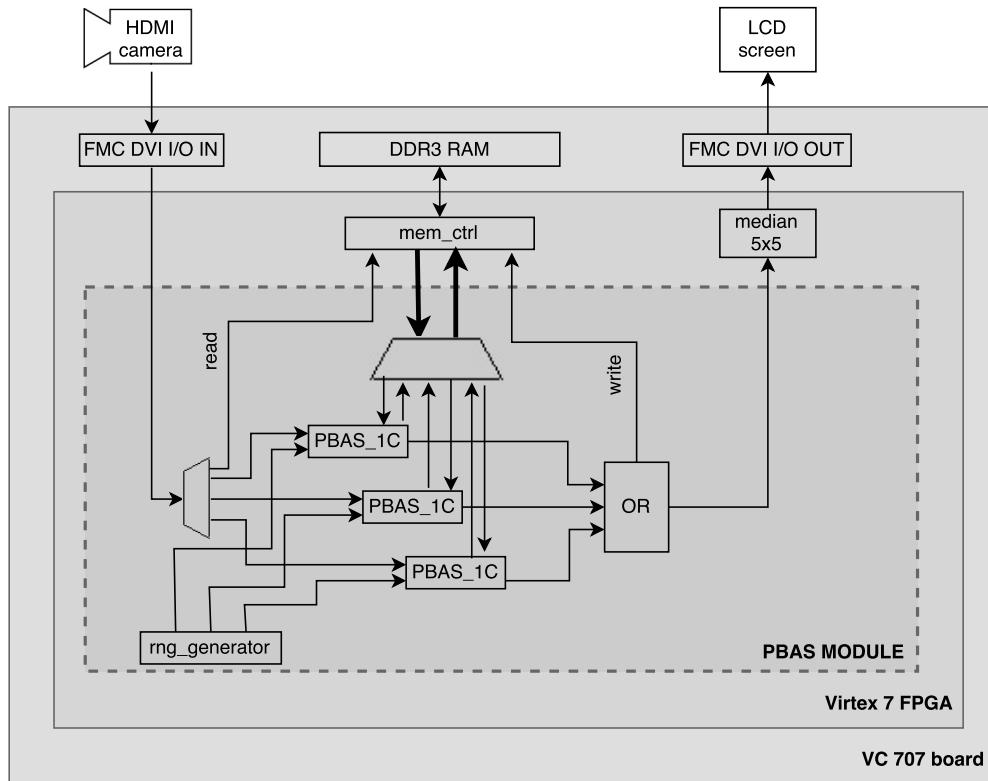
W tabeli 4.4 zamieszczono zużycie zasobów w układzie *FPGA*, oszacowany pobór mocy wynosi 4,011W. W porównaniu do standardowej wersji algorytmu (tabela 4.2) można zauważyc znaczący wzrost wykorzystania pamięci *BRAM*. Algorytm obliczania przepływu optycznego wymaga dodatkowej pamięci, która wykorzystywana jest do przechowywania informacji z każdego bloku oraz do obliczania mediany. Wzrost zużycia pozostałych zasobów jest minimalny.

Tabela 4.4. Rozszerzony algorytm *ViBE* - wykorzystanie zasobów (*Virtex 7*)

Zasoby	Wykorzystane	Dostępne	Użycie w %
<b>LUT</b>	22626	303600	7,45
<b>LUTRAM</b>	2253	130800	1,80
<b>FF</b>	25004	607200	4,12
<b>BRAM</b>	354	1030	34,37
<b>DSP</b>	49	2800	1,75

## 4.6. Implementacja algorytmu PBAS

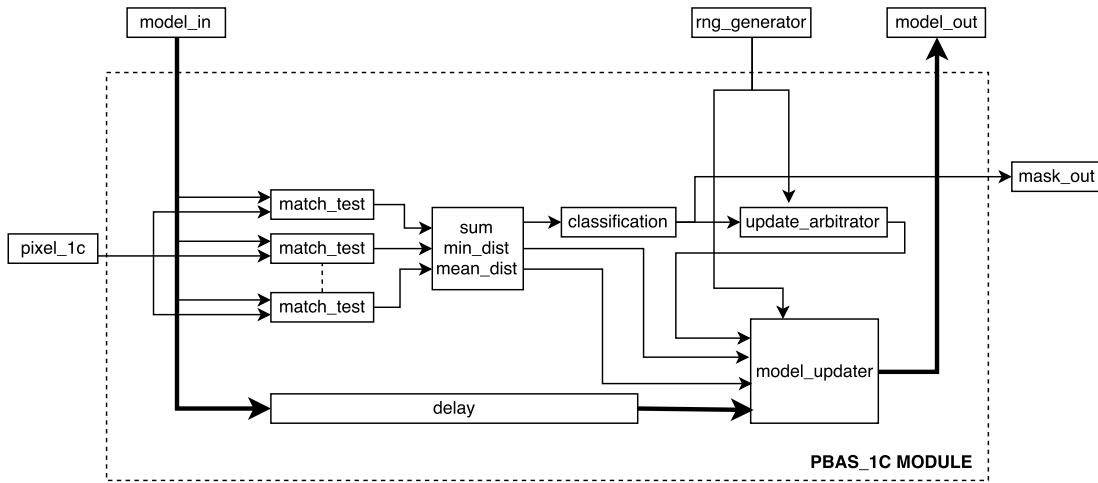
Szczegółowy opis teoretyczny algorytmu *PBAS* został zamieszczony w podrozdziale 3.3. Zaimplementowano wariant operujący w przestrzeni kolorów *RGB* w rozdzielcości  $720 \times 576$  pikseli oraz wersję przetwarzającą obraz w skali szarości, w przypadku wyższych rozdzielcości ( $720p$  i  $1080p$ ). Wysokopoziomowy schemat całego algorytmu został przedstawiony na rysunku 4.14. Jak zostało już wspomniane w części teoretycznej, wersja algorytmu działająca w przestrzeni *RGB*, przetwarza niezależnie poszczególne składowe, tworząc dla każdej z nich osobny model tła.



Rys. 4.14. Implementacja algorytmu PBAS w wersji RGB

Pierwszą operacją, jest rozbicie sygnału wejściowego na składowe *RGB*. Następnie każda z nich jest analizowana niezależnie poprzez algorytm *PBAS* (blok *PBAS\_1C*). Szczegółowy schemat implementacji modułu, realizującego algorytm dla jednej składowej, przedstawiono na rysunku 4.15. Maska finalna stanowi alternatywę logiczną wyników otrzymanych z poszczególnych modułów. Podobnie jak w przypadku innych algorytmów, tutaj także, na koniec, nakładany jest filtr medianowy w celu wyeliminowania szumów i zakłóceń. Algorytm wykorzystuje także generator liczb losowych, zaimplementowany w module *rng\_generator*. Jego opis został szerzej przedstawiony w podrozdziale 4.3.2. Opisywany moduł wykorzystuje oczywiście uniwersalny interfejs wizyjny przedstawiony w rozdziale 4.2.

Implementacja algorytmu *PBAS* przedstawiona na rysunku 4.15, operującego na jednej składowej *RGB* bądź skali szarości, jest bardzo podobna do implementacji metody *ViBE* przedstawionej w rozdziale 4.4. Podobnie jak w tamtym przypadku, tutaj także, pierwszym krokiem, jest równoległy test dopasowania do próbek zapisanych w pamięci *RAM*, zgodnie z równaniem (3.17). Następnie przeprowadzana jest klasyfikacja, opisana równaniem (3.15). Dodatkowo obliczany jest minimalny dystans pomiędzy próbками z modelu a aktualnym pikselem oraz średnia wartość odległości zapisanych w modelu tła. Tak samo jak w metodzie *ViBE*, decyzja o podjęciu aktualizacji realizowana jest przez moduł *update\_arbitrator*. Blok dokonujący aktualizacji, jest rozszerzeniem tego co zostało zaprezentowane w implementacji metody *ViBE*. Dodatkowo, oprócz próbek, aktualizowany jest także zbiór minimalnych odległości oraz parametry *R* i *T*, zgodnie z równaniami (3.20) i (3.21).



Rys. 4.15. Implementacja algorytmu PBAS dla jednej składowej RGB

Pierwsza część modelu składa się z  $N$  próbek, każda próbka zawiera wartość piksela –  $B_i$  oraz zapamiętany minimalny dystans –  $D_i$  pomiędzy aktualną próbką a aktualną wartością piksela. Na końcu znajdują się parametry  $R$  i  $T$ . Modele dla poszczególnych składowych *RGB* ustawione kolejno po sobie. Dla rozdzielcości *576p* przyjęto model składający się z  $N = 19$  próbek. Parametry  $R$  i  $T$  zapisano w postaci 16-bitowych liczb stałoprzecinkowych w formacie *8z8u*. Minimalny dystans, podobnie jak próbka, zapisana jest jako 8-bitowa liczba całkowita. Sumarycznie model tła dla jednej składowej *RGB* ma rozmiar  $19 \cdot (8 + 8) + 16 + 16 = 336$  bitów, zatem cały model zajmuje  $3 \cdot 336 = 1008$  bitów. Dla wyższych rozdzielcości wykorzystano algorytm operujący na obrazie w skali szarości, model w tym przypadku składa się z  $N = 14$  próbek. Zatem, cały model, można zapisać za pomocą  $14 \cdot (8 + 8) + 16 + 16 = 256$ , jest to maksymalna szerokość modelu, która może zostać wykorzystana w przypadku obrazu w rozdzielcości *720p* lub *1080p*.

Zużycie zasobów dla algorytmu *PBAS* w wersji *576p@50fps* i *1080p@50fps* zamieszczono w tabelach 4.5 i 4.6, pobór mocy w obu przypadkach wynosi odpowiednio 4,686W i 3,263W. Należy jeszcze raz podkreślić, że dla wyższej rozdzielcości zastosowano algorytm operujący na obrazie w skali szarości z dużo mniej dokładnym modelem tła. Takie działanie wymuszone było ograniczeniami zewnętrznej pamięci *RAM*. Ostatecznie z tego powodu zużycie zasobów oraz pobór mocy dla rozdzielcości *1080p* jest zdecydowanie niższy niż dla *576p*. W porównaniu do algorytmu *ViBE* (tabela 4.2) możemy zauważać znaczący wzrost wykorzystywanych zasobów. Taki rezultat jest zgodny z oczekiwaniami, gdyż jak zostało już wcześniej powiedziane, omawiany algorytm jest rozszerzeniem metody *ViBE*.

**Tabela 4.5.** PBAS 576p@50fps - wykorzystanie zasobów (Virtex 7)

Zasoby	Wykorzystane	Dostępne	Użycie w %
<b>LUT</b>	28142	303600	9,27
<b>LUTRAM</b>	3492	130800	2,67
<b>FF</b>	33849	607200	5,57
<b>BRAM</b>	244	1030	23,69
<b>DSP</b>	24	2800	0,86

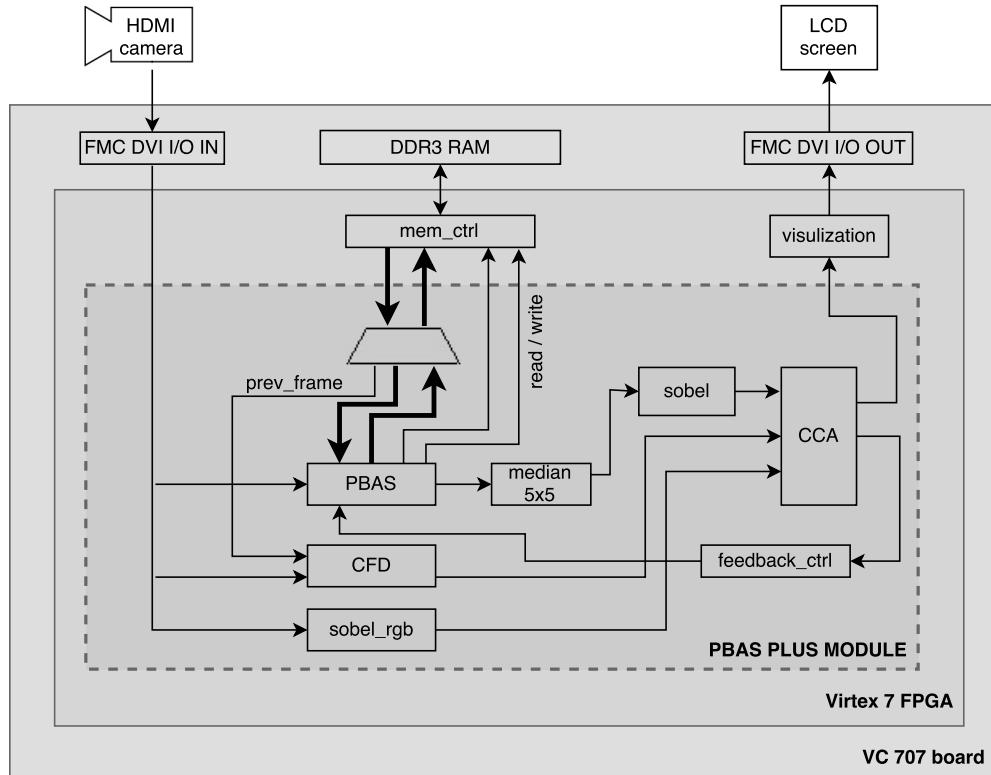
**Tabela 4.6.** PBAS 1080p@50fps - wykorzystanie zasobów (Virtex 7)

Zasoby	Wykorzystane	Dostępne	Użycie w %
<b>LUT</b>	16389	303600	5,40
<b>LUTRAM</b>	2816	130800	2,15
<b>FF</b>	16494	607200	2,72
<b>BRAM</b>	170	1030	16,50
<b>DSP</b>	11	2800	0,39

## 4.7. Implementacja rozszerzonej wersji metody PBAS

Przedstawiona implementacja, powstała na podstawie opisu teoretycznego, zamieszczonego w rozdziale 3.3.2. Opisana tutaj, rozszerzona wersja algorytmu *PBAS*, zawiera dodatkowy mechanizm rozróżniania obiektów statycznych od tzw. „duchów”. Koncepcja ta, wymaga także zaimplementowania metody indeksacji obiektów, co sprawia, że omawiany moduł jest najbardziej złożonym spośród wszystkich pokazanych w niniejszej pracy. Fragmenty przedstawionego tutaj rozwiązania pochodzą z publikacji [14], również powstałej w Laboratorium Biocybernetyki AGH. Ogólny schemat przedstawiający architekturę modułu pokazano na rysunku 4.16.

Pierwszym istotnym elementem jest realizacja podstawowego algorytmu *PBAS*, w stosunku do wersji przedstawionej w rozdziale ?? musiał on zostać zmodyfikowany. Konieczne było dodanie obsługi sprzężenia zwrotnego zgodnie z równaniami (??) i (??). Oprócz tego należało także zapewnić aktualizację dwóch nowych parametrów  $S(x_i)_{cnt}$  i  $E(x_i)_{mean}$ , zgodnie z równaniami (3.26) i (3.27). Dodatkowo w modelu tła musi znaleźć się też poprzednia wartość piksela (24 bity) wykorzystywana podczas odcinania ramek (blok *CFD*). W związku z tym, aby rozmiar modelu nie przekroczył maksymalnego rozmiaru wynoszącego 1024 bity, konieczne było zredukowanie liczby próbek w algorytmie *PBAS* do 18. Parametr  $S(x_i)_{cnt}$  zapisany jest jako 8 bitowa liczba całkowita, z kolei wartość  $E(x_i)_{mean}$  to 16 bitowa liczba stałoprzecinkowa w formacie 8z8u. Ostatecznie rozmiar modelu tła w takiej konfiguracji wynosi:  $3 \cdot (18 \cdot (8 + 8) + 16 + 16) + 8 + 16 + 24 = 1008$  bitów.

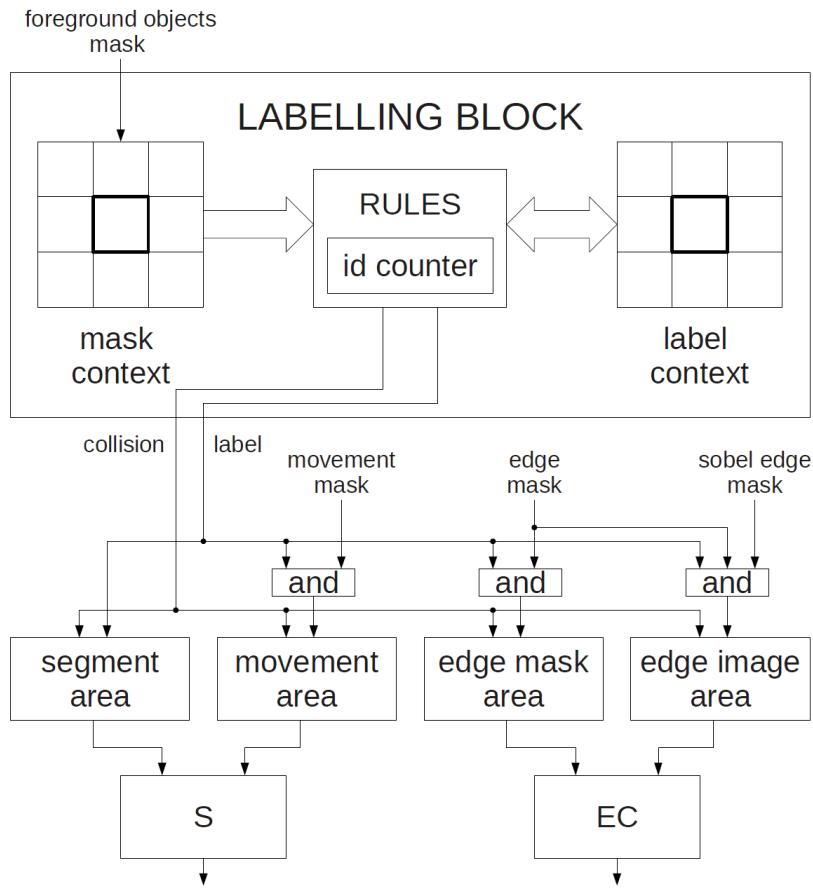


Rys. 4.16. Implementacja rozszerzonej wersji algorytmu PBAS

Równolegle do algorytmu *PBAS* wykonywane jest, wspomniane wcześniej, odejmowanie ramek oraz detekcja krawędzi metodą Sobela. Na maskę wyjściową podobnie jak w poprzednich metodach, nakładany jest filtr medianowy. Wyznaczane są także krawędzie na masce binarnej, gdyż jest to niezbędne podczas przeprowadzania analizy poszczególnych obiektów. Schemat bloku *CCA*, odpowiedzialnego za to zadanie, został pokazany na rysunku 4.17. Ze względu ma fakt, że dane z modułu analizy obiektów dostępne są dopiero po przetworzeniu całej ramki obrazu, konieczne jest zapewnienie odpowiedniej synchronizacji sygnałów sprzężenia zwrotnego z modułem *PBAS*. Zadanie to zostało zrealizowane poprzez blok *feedback\_ctrl*. Wyznaczone wartości współczynników stabilności i podobieństwa krawędzi oraz finalna maska są zapisywane w pamięci *BRAM*. Działający algorytm, poprawnie identyfikujący obiekt pierwszoplanowy oraz prostokąt go otaczający, został pokazany na rysunku 4.18.

Do zrealizowania zadania indeksacji obiektów wykorzystano moduł *CCA* opracowany w ramach publikacji [14]. Całość została przygotowana w oparciu o teorię na temat indeksowania obiektów, przedstawioną w rozdziale 3.3.3. Przygotowany moduł składa się z dwóch części, pierwszą z nich jest fragment logiki programowej (*Labelling Block*), odpowiedzialny za przypisanie kolejnym pikselom odpowiednich etykiet. Jak zostało to opisane w części teoretycznej, etykieta jest dobierana na podstawie sąsiednich pikseli oraz etykiet im przypisanych. Konieczne jest zatem wygenerowanie kontekstu piksela wejściowego, zawierającego etykiety przypisane sąsiadom.

Informacja o przydzielonej etykiecie przekazywana jest do czterech bloków obliczających pole zidentyfikowanego obiektu. Pierwszy licznik (*segment area*) służy do wyznaczenia całkowitego pola



Rys. 4.17. Architektura modułu CCA – źródło [14]

obiektu. Drugi moduł (*movement area*) zlicza natomiast jedynie ruchome piksele. Wartości te są używane do wyznaczenia współczynnika  $S_{O_k}$  zgodnie z równaniem 3.23, wykorzystano w tym celu dzielarkę sprzętową. Kolejny licznik (*edge mask area*) służy do zliczania pikseli na pierwszoplanowej masce krawędzi. Ostatni z modułów (*textitedge image area*) wyznacza liczbę pikseli znajdujących się na wspomnianej wyżej masce oraz na zbinaryzowanym obrazie krawędzi obrazu wejściowego. Te dwie wartości służą do wyznaczenia parametru  $EC_{O_k}$ , zgodnie z równaniem 3.25. Podobnie jak w poprzednim przypadku, tutaj także wykorzystano dzielarkę sprzętową.

Oprócz wymienionych czterech istnieje jeden blok, aktualizujący na bieżąco parametry prostokąta otaczającego poszczególne obiekty. Wszystkie liczniki zostały zaimplementowane z użyciem pamięci blokowej *BRAM*. Problem konfliktów rozwiązano, poprzez sumowanie wartości zapamiętanych dla obu łączonych obiektów i zapisanie jej jako nowej wartości dla obiektu o niższej etykiecie. Wartość zapisana pod adresem wyższej etykiety zostaje natomiast oznaczona jako nieważna.

Wykorzystanie zasobów przez rozszerzoną wersję algorytmu *PBAS* przedstawiono w tabeli 4.7, oszacowany pobór mocy wynosi natomiast 5,087W. Ze względu na mechanizm indeksacji, jest to najbardziej złożony ze wszystkich omawianych algorytmów, stąd też najwyższe zużycie zasobów jak i pobór mocy.



Rys. 4.18. Działający algorytm PBAS wraz z indeksacją obiektów

W stosunku do standardowej metody *PBAS* możemy zauważać kilku procentowy wzrost dla większości typów.

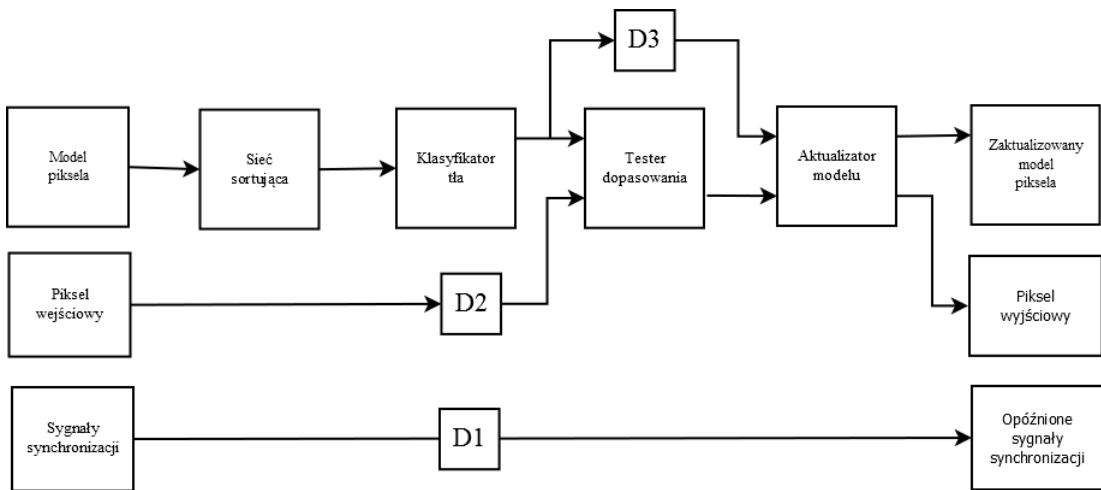
Tabela 4.7. Rozszerzony algorytm *PBAS* - wykorzystanie zasobów (*Virtex 7*)

Zasoby	Wykorzystane	Dostępne	Użycie w %
<b>LUT</b>	40329	303600	13,28
<b>LUTRAM</b>	5549	130800	4,24
<b>FF</b>	40849	607200	6,73
<b>BRAM</b>	260	1030	25,19
<b>DSP</b>	24	2800	0,86

## 4.8. Implementacja GMM

Przedstawiona tutaj implementacja algorytmu *GMM* została przygotowana w ramach pracy inżynierskiej [20]. Sposób implementacji sprzętowej omawianej metody jest tematem niezwykle rozległym, w niniejszym rozdziale przedstawiono jedynie ogólną ideę przygotowanej architektury bez zagłębiania się w szczegóły implementacyjne. Schemat przedstawiający implementację i wykorzystane moduły został pokazany na rysunku 4.19.

Niewątpliwą zaletą, znaczaco ułatwiającą implementację w układzie reprogramowalnym jest brak operacji kontekstowych. Dzięki temu możemy znaczaco uprościć logikę programową i zredukować zużycie zasobów. Nawiązując do opisu teoretycznego zamieszczonego w rozdziale 3.4, pierwszym krokiem algorytmu jest sortowanie rozkładów Gaussa według współczynnika  $r_i = \frac{\omega_i}{\sigma_i}$ . W tym celu wykorzystana została specjalnie zaprojektowana sieć sortująca, jest to najbardziej złożony element przedstawianej implementacji.



Rys. 4.19. Implementacja algorytmu GMM – źródło [20]

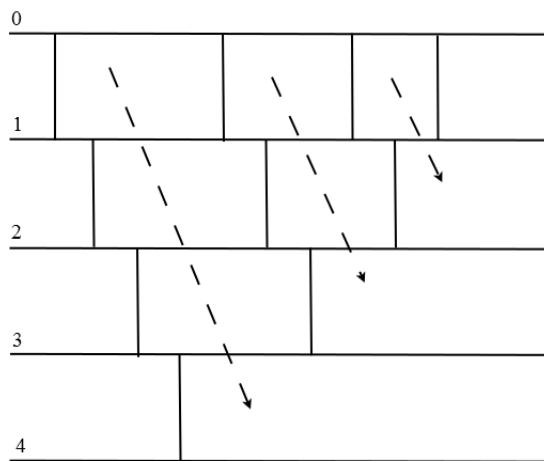
Po dokonaniu sortowania, poszczególne rozkłady Gaussa są klasyfikowane, na podstawie równania 3.31. Następnie przeprowadzany jest test dopasowania i ostateczna klasyfikacja piksela, zgodnie z równaniem 3.32. Po dokonaniu klasyfikacji rozkłady Gaussa są aktualizowane i zapisywane w pamięci *RAM*.

Przygotowana implementacja przetwarza obraz we wszystkich wymaganych rozdzielczościach, czyli 576p, 720p i 1080p w 50 klatkach na sekundę. Dla obrazu 720x576 przyjęto  $K = 5$  rozkładów Gaussa w każdym modelu, natomiast dla wyższych rozdzielczości, ze względu na ograniczenia pamięci *RAM*, liczba rozkładów wynosi  $K = 3$ . W modelu tła pojedynczy rozkład Gaussa ma rozmiar 68 bitów, dokładny opis i reprezentacja poszczególnych bitów modelu została przedstawiona w tabeli 4.8.

Tabela 4.8. Znaczenie kolejnych bitów w reprezentacji rozkładu Gaussa – źródło [20]

Zakres bitowy	Oznaczenie
0	Bit pierwszego planu. Jeśli jest równy 1 oznacza to, że rozkład Gaussa reprezentuje obiekty pierwszoplanowe, w przeciwnym razie reprezentuje on tło.
1 – 7	Bity zarezerwowane (w razie wprowadzenia nowych flag).
8 – 19	Waga rozkładu Gaussa $\omega$ jako liczba stałoprzecinkowa (zakres 0 – 1). Część całkowita zajmuje 0 bitów, część ułamkowa 12 bitów.
20 – 31	Wariancja ( $\sigma^2$ ) jako liczba stałoprzecinkowa (zakres 0 – 256). Część całkowita zajmuje 8 bitów, część ułamkowa 4 bity.
32 – 43	Średnia wartość barwy czerwonej piksela jako liczba stałoprzecinkowa (zakres 0 – 256). Część całkowita zajmuje 8 bitów, część ułamkowa 4 bity.
44 – 55	Średnia wartość barwy zielonej piksela jako liczba stałoprzecinkowa (zakres 0 – 256). Część całkowita zajmuje 8 bitów, część ułamkowa 4 bity.
56 – 67	Średnia wartość barwy niebieskiej piksela jako liczba stałoprzecinkowa (zakres 0 – 256). Część całkowita zajmuje 8 bitów, część ułamkowa 4 bity.

Poglądowy schemat przedstawiający zasadę działania sieci sortującej dla  $K = 5$  elementów, został pokazany na rysunku 4.20. Jest to sprzętowa realizacja algorytmu *BS* (ang. *Bubble Sort* – sortowanie bąbelkowe). Pionowe linie oznaczają porównania między elementami. Przerywane linie z grotem określają z kolei kierunek przesuwania się najmniejszego elementu w danej iteracji. W pierwszym kroku spośród  $K$  elementów, wyłaniany jest najmniejszy poprzez wykonanie  $K - 1$  porównań. Wyznaczony w ten sposób element jest przestawiany na sam dół sieci. Następnie wśród pozostałych  $K - 1$  elementów wyszukiwany jest kolejny najmniejszy, w tym przypadku należy wykonać  $K - 2$  porównań itd. Ostatecznie do otrzymywania posortowanego ciągu potrzeba  $K - 1$  iteracji algorytmu.



Rys. 4.20. Sieć sortująca w układzie *FPGA* – źródło [20]

Tabela 4.9 przedstawia wykorzystanie zasobów w układzie *FPGA* przez algorytm *GMM*. Zaimplementowana metoda, w odróżnieniu od innych, nie zawiera operacji kontekstowych (poza filtrem medianowym nakładanym na maskę wyjściową), w związku z tym nie jest wykorzystywana pamięć *BRAM*. W przypadku pozostałych zasobów kluczowy jest parametr  $K$  reprezentujący liczbę rozkładów Gaussa. Dla  $K = 5$ , czyli domyślnej wartości wykorzystanie zasobów jest zdecydowanie niższe niż w przypadku algorytmów *ViBE* i *PBAS*, co niestety przekłada się również na efekt końcowy. Testy poszczególnych algorytmów zostały szczegółowo przedstawione w rozdziale 5.

Tabela 4.9. *GMM* - wykorzystanie zasobów (*Virtex 7*)

Zasoby	Wykorzystane	Dostępne	Użycie w %
<b>LUT</b>	426	303600	0,14
<b>LUTRAM</b>	338	130800	0,26
<b>FF</b>	752	607200	0,12
<b>BRAM</b>	3	1030	0,002
<b>DSP</b>	0	2800	0,0



## 5. Ewaluacja zaimplementowanych algorytmów

### 5.1. Metodologia przeprowadzonych testów

Testy algorytmów opracowanych w Laboratorium Biocybernetyki AGH, zostały przeprowadzone zgodnie z metodologią opisaną w [6]. Sekwencje testowe pochodzą z bazy *ChangeDetection* [2]. Tego typu podejście do ewaluacji zaimplementowanych algorytmów zostało wykorzystane między innymi w [15, 14, 9]. Wszystkie metody zostały przetestowane z wykorzystaniem 31 sekwencji testowych podzielonych na 6 różnych kategorii. Zbiór testowy został tak dobrany, aby odwzorować jak największą liczbę sytuacji mogących wystąpić w rzeczywistym środowisku. Każda z kategorii została szczegółowo opisana w rozdziale 5.2.

Dla każdej sekwencji testowej zawartej w bazie, dostępny jest model wzorcowy tj. ręcznie anotowana maska obiektów (ang. *ground truth*). Wzorzec zapisany jest jako obraz w skali szarości, gdzie piksele przyjmują jedną z pięciu wartości:

0 – tło

50 – cienie

85 – obszar wyłączony z analizy

175 – obszar trudny do zidentyfikowania (np. kontur otaczający ruchomy obiekt)

255 – obiekt pierwszoplanowy

Porównując ramki wyjściowe testowanego algorytmu z odpowiadającymi im ramkami modelu wzorcowego, można wyznaczyć następujące współczynniki:

$TP$  – 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*)

Na podstawie wyznaczonych otrzymanych współczynników oblicza się 7 wskaźników jakości, określających dokładność metody:

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

Parametr *Re* definiuje jaki procent pikseli pierwszoplanowych został rozpoznany. Analogiczną wartość dla pikseli reprezentujących tło określa parametr *Se*. Parametry *FPR* i *FNR* są przeciwnieństwem wartości opisanych wyżej i wynoszą odpowiednio  $1 - Se$  i  $1 - Re$ . *PWC* określa procent źle sklasyfikowany pikseli, natomiast *Pr* informuje jaki procent spośród pikseli sklasyfikowanych jako pierwszoplanowe został rozpoznany prawidłowo.

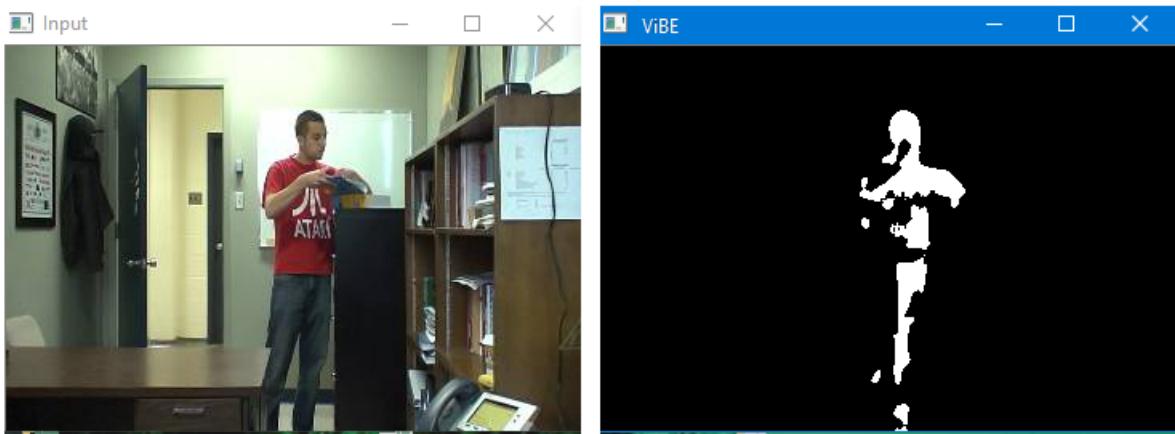
## 5.2. Szczegółowe test

### 5.2.1. Sekwencje podstawowe

W pierwszej kolejności dokonano porównania algorytmów przy użyciu sekwencji testowych z kategorii *Baseline*. Wykorzystano 4 sekwencje, z czego dwie nagrano na otwartej przestrzeni i dwie w zamkniętym pomieszczeniu. Przykładowa sekwencja została pokazana na rysunku 5.1. Jest to zdecydowanie najprostsza spośród wszystkich kategorii i jednocześnie najbardziej zróżnicowana. Dzięki temu, może stanowić dobry punkt odniesienia dla dalszych testów. Dwie spośród wykorzystanych sekwencji zawierają głównie obiekty ruchome, natomiast w dwóch kolejnych występują także obiekty zatrzymane. Oprócz tego miejscami pojawiają się cienie oraz dynamiczne tło. Szczegółowe wyniki wykonanych testów przedstawiono w tabeli 5.1.

**Tabela 5.1.** Średnie rezultaty uzyskane dla sekwencji z kategorii *Baseline*

	<i>Recall</i>	<i>Spec</i>	<i>FPR</i>	<i>FNR</i>	<i>PWC</i>	<i>F1</i>	<i>Prec</i>
<i>metoda naiwna</i>	0,8630	0,9973	0,0027	0,1370	0,68	0,8918	0,9237
<i>odejmowanie ramek</i>	0,2567	0,9988	0,0012	0,7433	3,06	0,3521	0,8352
<i>średnia krocząca</i>	0,6594	0,9797	0,0203	0,3406	3,62	0,4828	0,4685
<i>ViBE</i>	0,8871	0,9977	0,0023	0,1129	0,64	0,9126	0,9403
<i>PBAS</i>	0,7457	0,9978	0,0022	0,2543	1,23	0,8084	0,9109
<i>PBAS+</i>	0,9075	0,9965	0,0035	0,0925	0,58	0,8998	0,8932
<i>GMM</i>	0,5235	0,9703	0,0296	0,4764	5,32	0,5376	0,5604



Rys. 5.1. Sekwencja *office* z kategorii *Baseline*

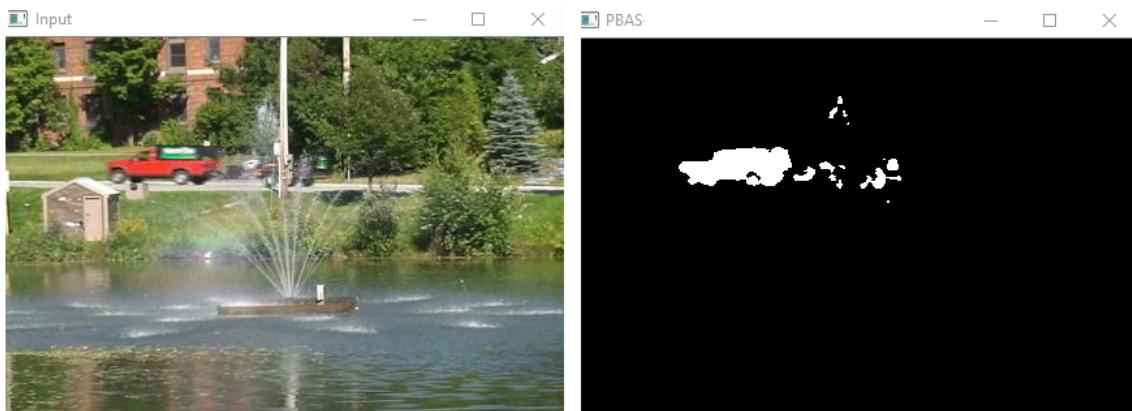
Spośród prostych algorytmów, zdecydowanie najlepszy wynik uzyskano przy wykorzystaniu metody naiwnej. Odejmowanie ramek oraz średnia krocząca zwracają już zdecydowanie gorsze rezultaty. Jak zostało już wspomniane w części teoretycznej (rozdział ??), są to algorytmy nadające się głównie do detekcji obiektów ruchomych, stąd też zdecydowanie gorszy wynik. Dla większości testowanych algorytmów można zaobserwować stosunkowo wysoką wartość wskaźników *Spec* i *Prec* oraz niewielką wartość *PWC*, co oznacza bardzo dobrą eliminację szumów.

Wartym uwagi jest fakt, że dla standardowej wersji algorytmu *PBAS* otrzymano gorsze wartości wskaźników niż dla metody *ViBE*. W testowanych sekwencjach występowały również obiekty zatrzymane, które w przypadku obu wspomnianych metod często zostawały wtapiane w tło, świadczy o tym stosunkowo niska wartość wskaźnika *recall*. Problem ten nie został zaobserwowany dla rozszerzonej wersji algorytmu *PBAS* (oznaczonej jako *PBAS+*), w której wykorzystano dodatkowy mechanizm detekcji obiektów statycznych. Wskaźniki jakości otrzymane dla algorytmu *GMM* są z kolei niższe niż dla omówionych wyżej właśnie algorytmów, ale również wyraźnie lepsze niż dla najprostszej metody odejmowania ramek i średniej kroczącej.

### 5.2.2. Dynamiczne tło i cienie

Kategoria *Dynamic Background* składa się z 6 sekwencji zawierających dynamiczne tło. Występuje ono pod różnymi formami, takimi jak fale na wodzie, fontanna oraz gałęzie z liśćmi poruszające się pod wpływem wiatru. Przykładowy kadr z takiej sekwencji zamieszczono na rysunku 5.2. Druga z omawianych kategorii to *Shadows*. Również zawiera 6 sekwencji, w których głównym utrudnieniem dla algorytmów segmentacji tła są licznie występujące cienie. Otrzymane dla poszczególnych algorytmów rezultaty zamieszczone zostały tabelach 5.2 (kategoria *Dynamic Background*) oraz 5.3 (kategoria *Shadows*).

W przedstawionych algorytmach nie zastosowano żadnych dodatkowych mechanizmów wspomagających wykrywanie dynamicznego tła oraz cieni. Po raz kolejny stosunkowo dobre wyniki, jak na prostotę tego algorytmu, uzyskano dla metody naiwnej. Dla większości algorytmów, podczas testów



Rys. 5.2. Sekwencja *fountain02* z kategorii *Dynamic Background*

Tabela 5.2. Średnie rezultaty uzyskane dla sekwencji z kategorii *Dynamic Background*

	<i>Recall</i>	<i>Spec</i>	<i>FPR</i>	<i>FNR</i>	<i>PWC</i>	<i>F1</i>	<i>Prec</i>
<i>metoda naiwna</i>	0,7994	0,9603	0,0397	0,2006	4,0673	0,3819	0,2866
<i>odejmowanie ramek</i>	0,2841	0,9913	0,0087	0,7159	1,8137	0,2848	0,6162
<i>średnia krocząca</i>	0,6736	0,9605	0,0395	0,3264	4,3561	0,3308	0,2593
<i>ViBE</i>	0,8108	0,9877	0,0123	0,1892	1,3591	0,6652	0,6748
<i>PBAS</i>	0,4749	0,9980	0,0020	0,5251	0,9153	0,5325	0,7776
<i>PBAS+</i>	0,9396	0,9748	0,0252	0,0604	2,5478	0,6187	0,5567
<i>GMM</i>	0,5467	0,9428	0,0572	0,4533	6,2624	0,2669	0,2009

w kategorii *dynamic background* uzyskano wysoką wartość wskaźnika *recall*, co oznacza dokładną detekcję rzeczywistych obiektów pierwszoplanowych. Słaby rezultat, zgodnie z oczekiwaniami otrzymano w przypadku metody odejmowania ramek. Wyniki niższe od pozostałych uzyskano również dla algorytmów *GMM* i *PBAS*, z kolei najwyższy wskaźnik otrzymano dla rozszerzonej wersji metody *PBAS+*.

Niestety w większości przypadków uzyskano również zdecydowanie niższą wartość wskaźnika *Prec* i *PWC* co oznacza, że dynamicznie poruszające się elementy w tle zostały mylnie zaklasyfikowane jako pierwszy plan. Szczególnie widoczne jest to dla metody naiwnej, natomiast najlepiej pod tym względem wypadły algorytmy *ViBE* oraz *PBAS*. Warto zaznaczyć, że rozszerzona wersja metody *PBAS* charakteryzuje się zdecydowanie słabszą zdolnością wyodrębniania dynamicznego tła od statycznego planu, jednak zapewnia lepszą dokładność w wykrywaniu prawdziwych obiektów pierwszoplanowych (wyższa wartość wskaźnika *recall*).

Testy w kategorii *shadows* pokazały, że mimo braku osobnego mechanizmu do detekcji i eliminacji cieni, podobnie jak w przypadku poprzedniego testu, wyniki są satysfakcyjne. Również w tym przypadku metoda naiwna zwraca bardzo dobre rezultaty, zarówno pod względem ilości wykrytych obiektów (parametr *recall*) jak i dokładności (parametry *PWC* i *prec*). Ponownie można zaobserwować wyraźną różnicę pomiędzy podstawową i rozszerzoną wersją algorytmu *PBAS* oraz bardzo dobry wynik metody

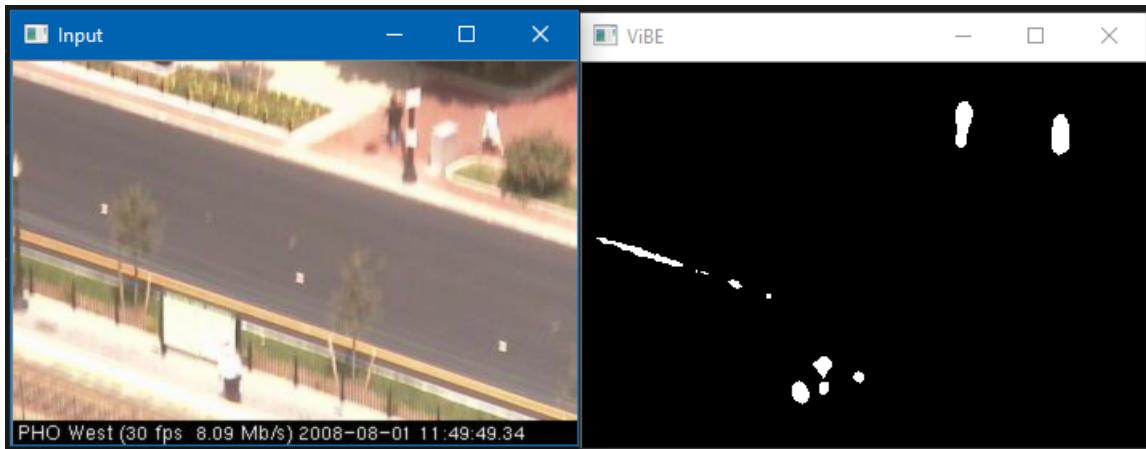
**Tabela 5.3.** Średnie rezultaty uzyskane dla sekwencji z kategorii *Shadows*

	<i>Recall</i>	<i>Spec</i>	<i>FPR</i>	<i>FNR</i>	<i>PWC</i>	<i>F1</i>	<i>Prec</i>
<i>metoda naiwna</i>	0,8253	0,9686	0,0314	0,1747	3,7359	0,6603	0,6129
<i>odejmowanie ramek</i>	0,1809	0,9970	0,0030	0,8191	3,7981	0,2700	0,7968
<i>średnia krocząca</i>	0,5992	0,9551	0,0449	0,4008	6,0793	0,4864	0,4436
<i>ViBE</i>	0,8738	0,9891	0,0109	0,1262	1,5202	0,8212	0,7884
<i>PBAS</i>	0,6152	0,9889	0,0111	0,3848	2,7448	0,6739	0,7951
<i>PBAS+</i>	0,8590	0,9881	0,0119	0,1410	1,8909	0,7903	0,7479
<i>GMM</i>	0,5478	0,9949	0,0051	0,4522	3,2416	0,6605	0,8510

*ViBE*. W przypadku tego testu algorytm *GMM* charakteryzuje się bardzo wysoką precyją, jednak przy zdecydowanie za niskim poziomie rozpoznawalności obiektów.

### 5.2.3. Drgania kamery

W kategorii *Camera Jitter* zawierają się 4 sekwencje testowe, w których kamera nieustannie poddawana jest lekkim oraz silniejszym wibracjom. Poziom wibracji jest zróżnicowany dla poszczególnych sekwencji. Przykładową sekwencję zamieszczono na rysunku 5.3, natomiast uzyskane rezultaty przedstawiono w tabeli 5.4.

**Rys. 5.3.** Sekwencja *boulevard* z kategorii *Camera Jitter*

Oprócz testowanych wcześniej algorytmów, w tym przypadku ewaluacji poddano także rozszerzoną wersję metody *ViBE*, zawierającą dodatkowy mechanizm redukcji efektu drgającej kamery. Niestety można zaobserwować, że zaimplementowane rozwiążanie nie zdało egzaminu i dla wykorzystanych sekwencji testowych uzyskany wynik jest niezadowalający. Implementacja algorytmu została przygotowana z myślą o pracy w rzeczywistym środowisku i w rozdzielczości *576p@50fps*, w przypadku przeprowadzanych testów obraz wejściowy jest dużo gorszej jakości. Opracowana koncepcja obliczania przepływu optycznego, zakładająca dzielenie obrazu na równe bloki o rozmiarze *32x32* nie daje

**Tabela 5.4.** Średnie rezultaty uzyskane dla sekwencji z kategorii *Camera Jitter*

	<i>Recall</i>	<i>Spec</i>	<i>FPR</i>	<i>FNR</i>	<i>PWC</i>	<i>F1</i>	<i>Prec</i>
<i>metoda naiwna</i>	0,6797	0,8546	0,1454	0,3203	15,0577	0,2906	0,1884
<i>odejmowanie ramek</i>	0,4640	0,8746	0,1254	0,5360	14,1708	0,2134	0,1408
<i>srednia krocząca</i>	0,7625	0,8487	0,1513	0,2375	15,2717	0,3205	0,2091
<i>ViBE</i>	0,7510	0,9371	0,0629	0,2490	6,9322	0,5119	0,3964
<i>ViBE+</i>	0,7510	0,9371	0,0629	0,2490	6,9322	0,5119	0,3964
<i>PBAS</i>	0,6523	0,9843	0,0157	0,3477	2,7535	0,6416	0,6382
<i>PBAS+</i>	0,8907	0,8889	0,1111	0,1093	11,0708	0,4072	0,2697
<i>GMM</i>	0,5998	0,9098	0,0902	0,4002	10,7799	0,4128	0,3210

akceptowalnych rezultatów dla niższych rozdzielczości. Pozostałe algorytmy uzyskały niskie wartości wskaźnika *prec* oraz wysoki procent źle sklasyfikowanych pikseli (*PWC*). Jest to naturalny efekt drgań kamery, ponieważ w takim przypadku niemal cały obraz interpretowany jest jako pierwszy plan.

Przygotowana implementacja algorytmu *ViBE*, została początkowo opracowana w Laboratorium Biocybernetyki AGH i po raz pierwszy opublikowana w artykule [15]. Autorzy tej publikacji zaproponowali alternatywny sposób przetestowania przygotowanego rozwiązania. Koncepcja ta zakłada testowanie metody w rzeczywistych warunkach, w trzech różnych sytuacjach oznaczanych dalej jako *C1*, *C2*, *C3*:

*C1* – brak obiektów pierwszoplanowych, jedynie ruch kamery

*C2* – poruszające się obiekty pierwszoplanowe oraz ruchoma kamera

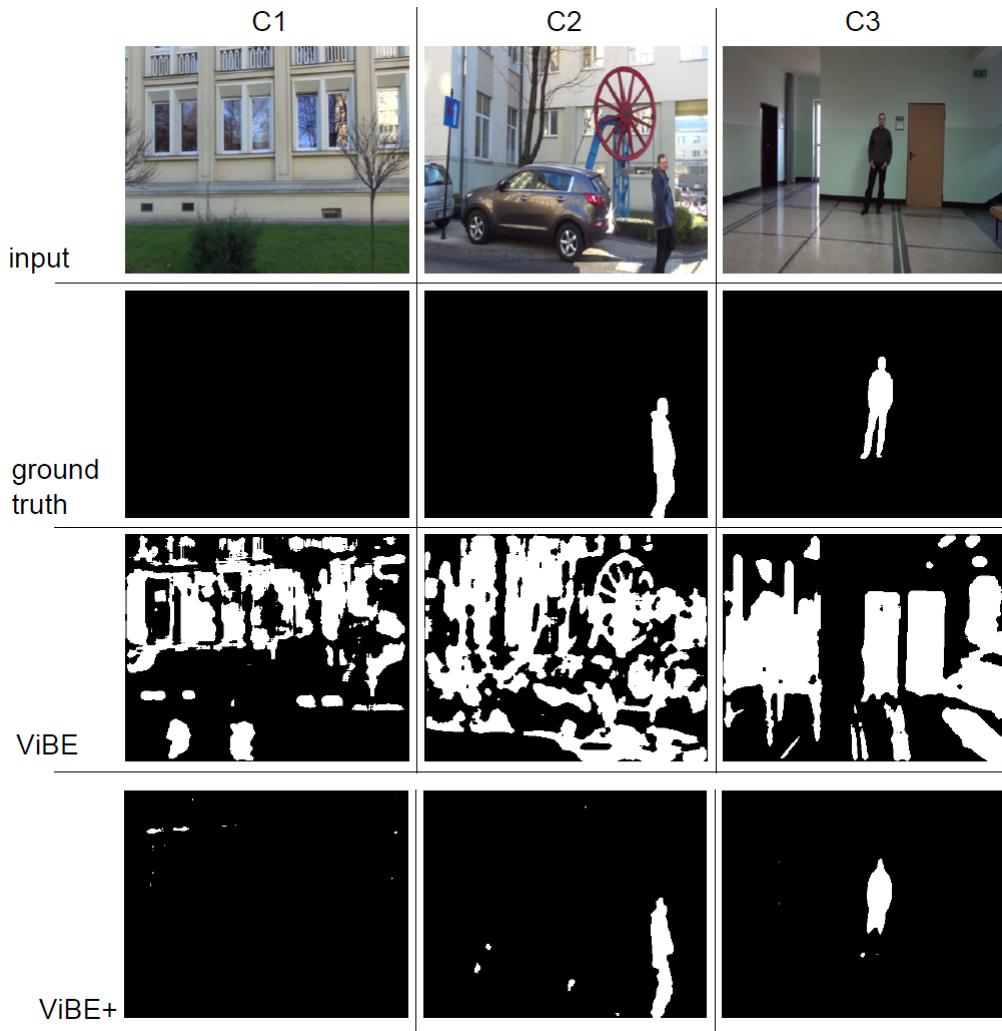
*C3* – pojawia się obiekt zatrzymany, następnie kamera zaczyna się ruszać

Działanie algorytmu w takim środowisku zilustrowano na rysunku 5.4, natomiast otrzymane wyniki z przeprowadzonych w ten sposób testów zamieszczone w tabeli 5.5.

**Tabela 5.5.** Testy algorytmu *ViBE* w rzeczywistym środowisku – źródło [15]

<i>Kategoria</i>	<i>Algorytm</i>	<i>Recall</i>	<i>PWC</i>	<i>F1</i>	<i>Prec</i>
<i>C1</i>	<i>ViBE</i>	–	36,66	–	0,0
	<i>ViBE+</i>	–	0,07	–	0,0
<i>C2</i>	<i>ViBE</i>	0,73	32,83	0,06	0,03
	<i>ViBE+</i>	0,79	0,75	0,76	0,74
<i>C3</i>	<i>ViBE</i>	0,85	30,75	0,09	0,05
	<i>ViBE+</i>	0,58	0,83	0,72	0,95

W przypadku kategorii *C1* nie występują piksele pierwszoplanowe (porusza się jedynie kamera), dlatego też nie ma możliwości wyznaczenia wskaźników *Recall* i *F1*. Na podstawie uzyskanych wyników można wywnioskować, że rozszerzona wersja algorytmu spełnia swoje zadanie. Świadczy o tym,



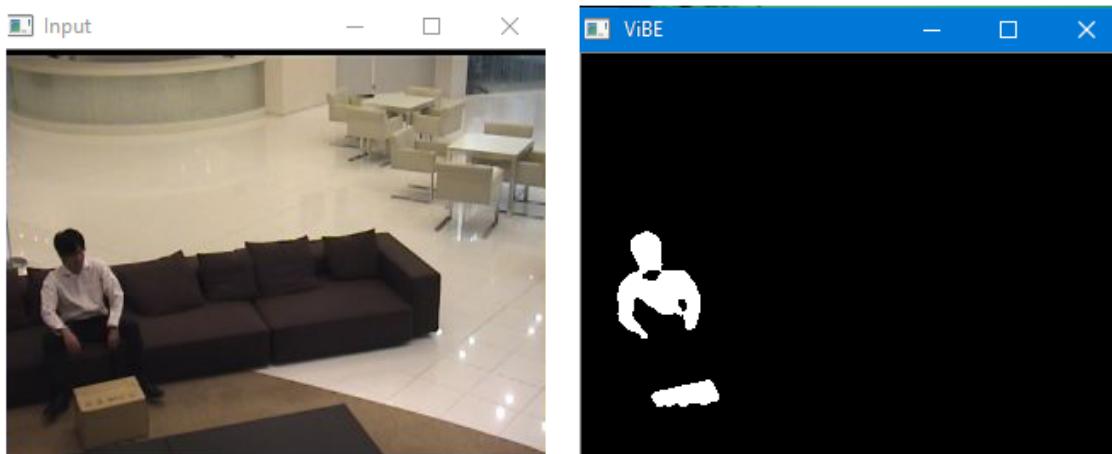
**Rys. 5.4.** Działanie algorytmu *ViBE* w rzeczywistym środowisku – źródło [15]

między innymi bardzo duży spadek wskaźnika *PWC* oraz wzrost wartości *Prec*. Oznacza to, że efekty drgającej kamery są dobrze niwelowane. Warto zauważyć, że w przypadku kategorii *C3*, spada procent wykrytych obiektów (parametr *recall*) kosztem znacznie lepszej obsługi drgającej kamery. Jest to potwierdzone także przykładowym kadrem z przeprowadzonego testu (rysunek 5.4), gdzie w kategorii *C3*, na masce wyjściowej algorytmu *ViBE+* widoczny jest tylko fragment obiektu.

#### 5.2.4. Obiekty zatrzymane

Kategoria *Intermittent Object Motion* składa się z 6 sekwencji testowych, które zawierają wiele statycznych obiektów pierwszoplanowych. Głównym zadaniem tego testu jest sprawdzenie, czy algorytm eliminuje tzw. „duchów”. Zjawisko występuje między innymi w sytuacji gdy np. samochód zatrzyma się na chwilę na światłach i następnie ruszy ponownie. Na wykorzystanych sekwencjach występują też

momenty, gdy obiekt niespodziewanie zaczyna się poruszać, np. auto odjeżdżające z parkingu lub porzucone pudełko. Przykładowa sekwencja z tej kategorii została pokazana na rysunku 5.5, natomiast wyznaczone wskaźniki jakości zamieszczono w tabeli 5.6.



Rys. 5.5. Sekwencja sofa z kategorii *Intermittent Object Motion*

Tabela 5.6. Średnie rezultaty uzyskane dla sekwencji z kategorii *Intermittent Object Motion*

	<i>Recall</i>	<i>Spec</i>	<i>FPR</i>	<i>FNR</i>	<i>PWC</i>	<i>F1</i>	<i>Prec</i>
<i>metoda naiwna</i>	0,7292	0,8802	0,1198	0,2708	12,1482	0,5184	0,5065
<i>odejmowanie ramek</i>	0,0497	0,9986	0,0014	0,9503	6,2573	0,0869	0,8729
<i>średnia krocząca</i>	0,2402	0,9817	0,0183	0,7598	6,5975	0,2903	0,4795
<i>ViBE</i>	0,6338	0,9055	0,0945	0,3662	10,0766	0,4562	0,4463
<i>PBAS</i>	0,1471	0,9997	0,0003	0,8529	5,6189	0,2389	0,9293
<i>PBAS+</i>	0,8048	0,9868	0,0132	0,1952	2,1968	0,7778	0,7847
<i>GMM</i>	0,3403	0,9719	0,0281	0,6597	10,2164	0,4108	0,6682

Na podstawie uzyskanych wyników można zauważyc, że testowane algorytmy nie daje zbyt dobrych rezultatów. W większości przypadków występuje niska wartość wskaźnika *recall* oraz duża wartość *PWC* co oznacza, że wiele spośród statycznych obiektów zostało wtopionych w model tła. Ponownie zaskakująco dużą dokładność uzyskano dla metody naiwnej. Otrzymane wskaźniki są na podobnym poziomie co te uzyskane dla algorytmu *ViBE*. Metody *PBAS* w wersji podstawowej oraz zawierającej dodatkowy mechanizm detekcji obiektów statycznych (*PBAS+*) zostały porównane niezależnie od reszty. Wyniki zawierające rezultaty obu algorytmów dla poszczególnych sekwencji z tej kategorii zamieszczono w tabeli 5.7.

Po dokonaniu analizy otrzymanych wyników można śmiało stwierdzić, że zaimplementowany mechanizm spełnia swoje zadanie. Rezultaty osiągnięte przez algorytm *PBAS+* są zdecydowanie lepsze niż innych algorytmów, dodatkowo widać olbrzymią poprawę w stosunku do standardowej wersji tej metody. Należy zaznaczyć, że w niektórych przypadkach statyczny obiekt umieszczony był daleko w

**Tabela 5.7.** Średnie wyniki uzyskane dla sekwencji z kategorii *Intermittent Object Motion*

Kategoria	Algorytm	Recall	Spec	FPR	FNR	PWC	F1	Prec
<i>abandonedBox</i>	<i>PBAS</i>	0,2217	0,9991	0,0009	0,7783	3,8330	0,3574	0,9224
	<i>PBAS+</i>	0,9879	0,9739	0,0261	0,0121	2,5427	0,7889	0,6566
<i>parking</i>	<i>PBAS</i>	0,0032	1,0000	0,0000	0,9968	7,7201	0,0064	1,0000
	<i>PBAS+</i>	0,6120	0,9873	0,0127	0,3880	4,1770	0,6941	0,8017
<i>sofa</i>	<i>PBAS</i>	0,2760	0,9994	0,0006	0,7240	3,2187	0,4284	0,9562
	<i>PBAS+</i>	0,5898	0,9967	0,0033	0,4102	2,1043	0,7101	0,8921
<i>streetLight</i>	<i>PBAS</i>	0,0421	1,0000	0,0000	0,9579	4,6503	0,0808	0,9999
	<i>PBAS+</i>	0,9590	0,9999	0,0001	0,0410	0,2093	0,9780	0,9978
<i>tramstop</i>	<i>PBAS</i>	0,2442	0,9998	0,0002	0,7558	13,5829	0,3922	0,9966
	<i>PBAS+</i>	0,9558	0,9685	0,0315	0,0442	3,3774	0,9104	0,8691
<i>winterDriveway</i>	<i>PBAS</i>	0,0954	0,9997	0,0003	0,9046	0,7083	0,1679	0,7008
	<i>PBAS+</i>	0,7245	0,9943	0,0057	0,2755	0,7700	0,5851	0,4906
<b>średnia</b>	<b><i>PBAS</i></b>	0,1471	0,9997	0,0003	0,8529	5,6189	0,2389	0,9293
	<b><i>PBAS+</i></b>	0,8048	0,9868	0,0132	0,1952	2,1968	0,7778	0,7847

tle, co stanowiło dodatkowe utrudnienie dla algorytmu. Przykładem takiej sekwencji jest *parking*, mimo takich utrudnień uzyskany wynik można uznać za zadowalający. Drugą bardzo skomplikowaną do analizy sekwencją jest *sofa*, gdzie za największą trudność należy uznać zbliżoną kolorystykę obiektów. W przypadku sekwencji *winterDriveway* wpływ na gorszy niż w pozostałych testach wynik, miały głównie bardzo trudne warunki pogodowe, w których film ten został nagrany.

### 5.2.5. Kamera termowizyjna

W kategorii *Thermal* zawiera się 5 sekwencji, do których nagrania wykorzystano kamerę pracującą w paśmie podczerwieni. Przykładowy kadr z takiej sekwencji zamieszczono na rysunku 5.6. Głównymi trudnościami w tego typu obrazach jest słaby kontrast pomiędzy obiektemi oraz różnego rodzaju odbicia cieplne np. oknie lub na podłodze. Wyniki przeprowadzonych testów dla poszczególnych algorytmów zamieszczone w tabeli 5.8.

Uzyskane wyniki pokazują, że w przypadku tej kategorii, większą trudnością jest identyfikacja obiektów. Rzadko występują przypadki, kiedy element tła zostanie błędnie zaklasyfikowany jako pierwszy plan. Świadczy o tym dość wysoka wartość wskaźników *spec* i *prec* dla wszystkich algorytmów. Nieustety w większości przypadków nie udało się rozpoznać znacznej ilości obiektów, rezultatem tego są niskie wartości wskaźnika *recall*.

### 5.2.6. Podsumowanie

Tabela 5.9 zawiera zestawienie uśrednionych wyników dla poszczególnych algorytmów. Dla każdej metody możemy zaobserwować wysoką wartość wskaźnika *spec*, oznacza to dobrą dokładność przy



Rys. 5.6. Sekwencja *corridor* z kategorii *Thermal*

Tabela 5.8. Średnie rezultaty uzyskane dla sekwencji z kategorii *Thermal*

	<i>Recall</i>	<i>Spec</i>	<i>FPR</i>	<i>FNR</i>	<i>PWC</i>	<i>F1</i>	<i>Prec</i>
<i>metoda naiwna</i>	0,6396	0,9936	0,0064	0,3604	1,9062	0,6982	0,8189
<i>odejmowanie ramek</i>	0,0812	0,9997	0,0003	0,9188	6,7849	0,1283	0,9158
<i>średnia krocząca</i>	0,3447	0,9933	0,0067	0,6553	5,8723	0,4238	0,7338
<i>ViBE</i>	0,2772	0,9930	0,0070	0,7228	5,2324	0,3936	0,7555
<i>PBAS</i>	0,2737	0,9980	0,0020	0,7263	5,3672	0,4005	0,9489
<i>PBAS+</i>	0,4236	0,9976	0,0024	0,5764	4,8207	0,5507	0,8981
<i>GMM</i>	0,3563	0,9983	0,0017	0,6415	5,5093	0,5041	0,9374

rozpoznawaniu pikseli będących elementami tła. Taki wynik jest przewidywalny, gdyż liczba pikseli reprezentujących tło jest w większości przypadków więcej niż pierwszoplanowych. Niestety dużo częściej występującym zjawiskiem jest zakwalifikowanie piksela będącego elementem pierwszego planu jako tła, sytuacja odwrotna zdarza się zdecydowanie rzadziej. Dla każdego algorytmu wartość wskaźnika *prec* jest wyraźnie niższa niż wartość *spec*, taki rezultat potwierdza omówione zjawisko.

Jak zostało już wielokrotnie zaznaczone przy opisie poszczególnych kategorii sekwencji testowych, stosunkowo dobre wyniki uzyskano dla metody naiwnej. Otrzymane wartości wskaźniki jakości są zdecydowanie wyższe niż w przypadku pozostałych prostych algorytmów. Zdecydowanie najlepiej z sekwencjami testowymi poradził sobie algorytm *PBAS* z dodatkowym mechanizmem detekcji obiektów statycznych. Uzyskano wysoki procent rozpoznanych obiektów pierwszoplanowych (parametr *recall*), przy zachowaniu dobrej dokładności i niewielkim procencie źle rozpoznanych pikseli (parametry *prec* i *PWC*). Takie wyniki oznaczają, że mechanizm indeksacji, połączony z analizą krawędzi poprawia jakość detekcji nie tylko w przypadku obiektów statycznych.

Kolejnym wartym uwagi jest fakt, że standardowa wersja algorytmu *PBAS*, będąca teoretycznie bardziej rozbudowaną wersją metody *ViBE* osiąga słabsze wyniki. Dodatkowe manipulowanie programem przynależności i prawdopodobieństwem aktualizacji nie zawsze wpływa pozytywnie na wynik końcowy.

**Tabela 5.9.** Średnie wyniki uzyskane dla poszczególnych algorytmów

	<i>Recall</i>	<i>Spec</i>	<i>FPR</i>	<i>FNR</i>	<i>PWC</i>	<i>F1</i>	<i>Prec</i>
<i>metoda naiwna</i>	0,7560	0,9424	0,0576	0,2440	6,2653	0,5735	0,5562
<i>odejmowanie ramek</i>	0,2195	0,9766	0,0234	0,7805	5,9806	0,2226	0,6963
<i>średnia krocząca</i>	0,5466	0,9532	0,0468	0,4534	6,9666	0,3891	0,4323
<i>ViBE</i>	0,7056	0,9683	0,0317	0,2944	4,2938	0,6268	0,6670
<i>PBAS</i>	0,4848	0,9944	0,0056	0,5152	3,1042	0,5493	0,8333
<i>PBAS+</i>	0,8042	0,9721	0,0279	0,1958	3,8507	0,6741	0,6917
<i>GMM</i>	0,4940	0,9693	0,0307	0,5056	6,4052	0,4932	0,6491

Rozczarowujące są również wyniki otrzymane dla algorytmu *GMM*, szczególnie widoczny jest bardzo wysoki procent źle sklasyfikowanych pikseli *PWC* oraz słaba rozpoznawalność obiektów pierwszoplanowych (parametr *recall*).

Niestety rozszerzona wersja algorytmu *ViBE*, zawierająca dodatkowy mechanizm redukcji drgań kamery, nie sprawdziła się w testach z wykorzystaniem sekwencji testowych. Autorzy publikacji [kryjak\_vibe\_14] z której pochodzi metoda zaproponowali alternatywny sposób przetestowania metody w rzeczywistym środowisku. Uzyskane w tak przeprowadzonym badaniu wyniki dały bardzo obiecujące wyniki.

W porównaniu do metod zaprezentowanych w innych publikacjach, wyniki najlepszego z zaimplementowanych algorytmów (*PBAS+*) można uznać za zadowalające. Zaprezentowana w publikacji [23] metoda *FTSG* osiąga co prawda zdecydowanie lepsze wyniki, jednak należy pamiętać, że zawiera ona wiele dodatkowych mechanizmów takich jak na przykład wykrywanie dynamicznego tła. Wyniki wersji *GMM* przedstawionej w [1] są bardzo zbliżone do wyników implementacji, która została przedstawiona w niniejszej pracy.



## **6. Dalsze kierunki rozwoju**

### **6.1. Poprawa algorytmów**

Oprócz standardowych wersji algorytmów, udało się także zaimplementować ich rozszerzone wersje, zdecydując o poprawiającej efektywność w specyficznych warunkach. W przypadku algorytmu *ViBE* zaimplementowano moduł, który w znacznym stopniu eliminuje efekt drgającej kamery. Metoda *PBAS* została natomiast rozszerzona o funkcjonalność detekcji obiektów statycznych. Kolejnym krokiem w rozwoju algorytmów i zwiększania ich efektywności mogłoby być zaimplementowanie metody zawierającej oba te udoskonalenia. Dobrym punktem wyjścia może być w tym przypadku algorytm *PBAS*, zawierający już moduł detekcji obiektów statycznych. Dodanie do takiej implementacji modułu wyliczającego przepływ optyczny powinno zostać wykonane bez dużych komplikacji.

W Laboratorium Biocybernetyki AGH, w ramach pracy inżynierskiej [9] został częściowo zaimplementowany algorytm *FTSG*. Jest to pierwsza implementacja tej metody w układzie reprogramowalnym. Nie udało się niestety opracować pełnej wersji, zaproponowanej przez autorów oryginalnej publikacji [23]. Zabrakło między innymi mechanizmu detekcji obiektów statycznych, który według założeń miał być bardzo podobny do tego wykorzystanego w metodzie *PBAS*. Celem przyszłych badań może być próba wykorzystania opracowanego dla algorytmu *PBAS* moduł analizy obiektów *CCA* i zintegrowania go z algorytmem *FTSG*. W celu poprawny dokładności można spróbować także zaimplementować mechanizm indeksacji obiektów w wariancie dwuprzebiegowym.

### **6.2. Wzrost wydajności**

Algorytmy przedstawione w niniejszej pracy starano się uruchomić w zarówno niskich ( $720 \times 576$ ) jak i wysokich ( $1280 \times 720$ ,  $1920 \times 1080$ ) rozdzielczościach. Zamierzony cel udało się osiągnąć w przypadku podstawowych wersji algorytmu, niezawierających dodatkowych modułów odpowiedzialnych między innymi za redukcję drgań kamery lub detekcję obszarów statycznych. W wielu przypadkach wiązało się to niestety z obniżeniem dokładności algorytmu, na przykład z powodu redukcji modelu tła, bądź konieczności przejścia z przestrzeni kolorów *RGB* do skali szarości.

W przypadku rozszerzonych odmian algorytmów *ViBE* oraz *PBAS*, udało się jedynie zapewnić przetwarzanie w najniższej rozdzielczości. Istotną kwestią jest zatem optymalizacja przygotowanych implementacji sprzętowych. Wraz z postępem technologicznym i pojawianiem się nowych, wydajniejszych

układów *FPGA*, należy dążyć do obsługi wyższych rozdzielczości. Docelowo system wizyjny powinien przetwarzać obraz w rozdzielczości  $1920x1080$  w 50 klatkach na sekundę, w dalszej przyszłości wymaganym standardem może stać się rozdzielcość  $4K$  ( $3840x2160$  pikseli).

Wzrost wydajności systemów wizyjnych jest ograniczony przez kilka czynników. Jak zostało już podkreślone jedną z blokad jest aktualnie dostępny sprzęt. W niektórych przypadkach ujawnia się ograniczona ilość zasobów logicznych w układzie *FPGA* lub dostępna pamięci *RAM*. Jednak, aby zapewnić wzrost wydajności nie należy jedynie oczekiwac na postęp technologiczny. Każdy algorytm można w pewnym stopniu zoptymalizować, poprzez uproszczenie logiki i zredukowanie ilości operacji wykonywanych w jednym cyklu. Taki zabieg może co prawda zwiększyć sumaryczną latencję, ale jednocześnie zapewni wyższą maksymalną częstotliwość pracy zegara.

### 6.3. Implementacja nowych rozwiązań

Mimo stosunkowo zadowalających efektów końcowych przygotowanych implementacji, nadal istnieje wiele kierunków w których algorytmy powinny być rozwijane. W przyszłości należy zastanowić się nad przygotowaniem między innymi dodatkowego mechanizmu detekcji dynamicznego tła i eliminacji jego wpływu na pracę algorytmu. Kolejnym wertym uwagi zagadnieniem jest prawidłowa detekcja cieni i ich odróżnienie od rzeczywistych obiektów. Oba zagadnienia są tematami bardzo rozległymi, które nie zostały do tej pory poruszone w ramach badań w Laboratorium Biocybernetyki AGH.

Kolejną drogą rozwoju może być próba tworzenia implementacji sprzętowych istniejących już algorytmów. Autorzy publikacji na różnego rodzaju konferencjach poświęconych systemom wizyjnym prezentują nowatorskie rozwiązania, często jednak nowy algorytm przedstawiany jest jedynie od strony teoretycznej. Mimo, że model programowy przygotowany na komputerze klasy PC daje świetne rezultaty w testach, jego zastosowanie w rzeczywistym systemie wizyjnym jest niemożliwe dopóki nie powstanie dedykowana implementacja sprzętowa. Przykładem takiego działania może być algorytm *Flux Tensor*, którego pierwsza implementacja sprzętowa [10] została przygotowana właśnie w Laboratorium Biocybernetyki AGH.

## **7. Zakończenie**



## Bibliografia

- [1] O. Barnich i M. Van Droogenbroeck. „ViBe: A universal background subtraction algorithm for video sequences”. W: *Image Processing, IEEE Transactions on* 20.6 (2011), s. 9–14 (cyt. na s. 11).
- [2] ChangeDetection. Strona internetowa: <http://www.changedetection.net/> (ostatni dostęp 01.06.2017). 2017 (cyt. na s. 9, 10, 12, 55).
- [3] M. V. Droogenbroeck i O. Paquot. „Background subtraction: Experiments and improvements for ViBe”. W: *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on* (2012), s. 32–37 (cyt. na s. 11).
- [4] A. Elgammal i in. „Background and foreground modeling using nonparametric kernel density estimation for visual surveillance”. W: *Proceedings of the IEEE* 90.7 (2002), s. 1151–1163 (cyt. na s. 12).
- [5] M. Genovese i E. Napoli. „FPGA-based architecture for real time segmentation and denoising of HD video”. W: *Journal of Real-Time Image Processing* 8.4 (2013), s. 389–401 (cyt. na s. 12, 31).
- [6] N. Goyette i in. „Changedetection.net: A new change detection benchmark dataset”. W: *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on* (2012), s. 1–8 (cyt. na s. 55).
- [7] C. Harris i M. Stephens. „A combined corner and edge detector”. W: *Proceedings of Fourth Alvey Vision Conference* (1988), s. 147–151 (cyt. na s. 20).
- [8] M. Hofmann. „Background segmentation with feedback: The pixelbased adaptive segmenter”. W: *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on* (2012), s. 38–43 (cyt. na s. 12).
- [9] P. Janus. *Implementacja algorytmu Flux Tensor with Split Gaussian Models w układzie reprogramowalnym*. Praca inżynierska – AGH Kraków. 2015 (cyt. na s. 12, 28, 55, 67).
- [10] P. Janus, K. Piszczeck i T. Kryjak. „FPGA Implementation of the Flux Tensor Moving Object Detection Method”. W: *International Conference on Computer Vision and Graphics* (2016), s. 486–497 (cyt. na s. 12, 68).
- [11] M. Komorkiewicz i T. Kryjak. *Projektowanie struktury układów FPGA, skrypt do ćwiczeń laboratoryjnych*. Akademia Górnictwo-Hutnicza im. Stanisława Staszica w Krakowie. 2014 (cyt. na s. 36, 37).

- [12] T. Kryjak, M. Komorkiewicz i M. Gorgoń. „Hardware implementation of the PBAS foreground detection method in FPGA”. W: *Proceedings of the 20th International Conference Mixed Design of Integrated Circuits and Systems (MIXDES)* (2013), s. 479–484 (cyt. na s. 12, 13).
- [13] T. Kryjak, M. Komorkiewicz i M. Gorgoń. „Real-time background generation and foreground object segmentation for high-definition colour video stream in FPGA device”. W: *Journal of Real-Time Image Processing* (2013), s. 61–77 (cyt. na s. 38, 39).
- [14] T. Kryjak, M. Komorkiewicz i M. Gorgoń. „Real-time Foreground Object Detection Combining the PBAS Background Modelling Algorithm and Feedback from Scene Analysis Module”. W: *International Journal of Electronics and Telecommunications* 60.1 (2014), s. 61–72 (cyt. na s. 12, 13, 22, 24, 26, 37, 48–50, 55).
- [15] T. Kryjak, M. Komorkiewicz i M. Gorgoń. „Real-time Implementation of Foreground Object Detection From a Moving Camera Using the ViBE Algorithm”. W: *Computer Science and Information Systems* 11.4 (2014), s. 1617–1637 (cyt. na s. 12, 16, 17, 20, 37, 55, 60, 61).
- [16] T. Kryjak, M. Komorkiewicz i M. Gorgoń. „Real-time implementation of the ViBe foreground object segmentation algorithm”. W: *Computer Science and Information Systems* (2013), s. 591–596 (cyt. na s. 11, 13, 17).
- [17] ModelineDatabase. Strona internetowa: <https://www.mythtv.org/> (ostatni dostęp 01.06.2017). 2017 (cyt. na s. 37).
- [18] OpenCV. Strona internetowa: <http://opencv.org/> (ostatni dostęp 01.06.2017). 2017 (cyt. na s. 9, 33).
- [19] K. Palaniappan i in. „Parallel flux tensor analysis for efficient moving object detection”. W: *Information Fusion (FUSION), 2011 Proceedings of the 14th International Conference on* (2011), s. 1–8 (cyt. na s. 12).
- [20] K. Piszczek. *Implementacja algorytmu Gaussian Mixture Models w układzie reprogramowalnym*. Praca inżynierska – AGH Kraków. 2015 (cyt. na s. 12, 28, 29, 31, 51–53).
- [21] C. Stauffer i W. Grimson. „Adaptive background mixture models for real-time tracking”. W: *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.* 2 (1999), s. 246–252 (cyt. na s. 12, 28, 29, 31).
- [22] D. Thomas i W. Luk. „FPGA–Optimised Uniform Random Number Generators Using LUTs and Shift Registers”. W: *International Conference on Field Programmable Logic and Applications (FPL)* (2010), s. 77–82 (cyt. na s. 37, 38).
- [23] R. Wang i in. „Static and Moving Object Detection Using Flux Tensor with Split Gaussian Models”. W: *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on* (2014), s. 420–424 (cyt. na s. 12, 13, 31, 65, 67).

- [24] H. Nagahara Y. Nonaka A. Shimada i R. Taniguchi. „Evaluation report of integrated background modeling based on spatio-temporal features”. W: *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on* (2012), s. 9–14 (cyt. na s. 13).



## **A. Spis zawartości płyty DVD**



## **B. Opis informatyczny**