AUTOMATYKA/AUTOMATICS • 2019 • Vol. ? • No. ?

Piotr Janus*, Tomasz Kryjak*, Marek Gorgoń*

Tutaj tytuł
cd

Abstract: Abstract

Keywords:

1 Introduction

Segmentacja obiektów pierwszoplanowych jest jednym z kluczowych elementów wielu advanced video surveillance systems (AVSS). Jest używana w systemach detekcji i trackowania obiektów oraz human behaviour analysis. Oprócz tego jest również ważnym elementem takich aplikacji jak abadnoned luggage detection and forbidden zone protection. Systemy mogę być użyte w border control i airports).

The simplest group of foreground object detection algorithms is based on subtracting subsequent frames from a video sequence. More advanced approaches involve the so-called background modelling. For each pixel, a dedicated model is assigned that describes the background appearance in a given location. Then, depending on used algorithm, the new pixel value is compared to the background model and classified (as foreground, background and sometimes also shadow). The model is updated to incorporate changes in the scene like slow or fast light variations and movement of objects belonging to background (i.e a moved chair).

TODO - dlaczego RGB–D jest lepsze, jeden akapit

W niniejszej pracy zaprezentowano rozszerzone wersje powszechnie wykorzystywanych algorytmów do segmentacji obiektów pierwszoplanowych. Opisane metody zostały dostosowane do przetwarzania danych otrzymanych ze standardowej kamery RGB oraz czujnika RGB-D. Skupiono się na algorytmach Gaussian Mixture Model (GMM) i Pixel-Based-Adaptive-Segmenter (PBAS), oprócz modelu programowego przygotowano także implementację sprzętową w układzie GPU z wykorzystaniem technologii CUDA. Do akwizycji obrazu użyto czujnika *Intel Real–Sense* natomiast platforma sprzętowa to *Nvidia Jetson*.

*AGH University of Science and Technology, Faculty of Electrical Engineering, Automatics, Computer Science and Biomedical Engineering, Krakow, Poland. e-mail: {piojanus, tomasz.kryjak, mago}@agh.edu.pl

Struktura niniejszego artykułu jest następująca, w rozdziale 2 przedstawiono wcześniejsze prace związane z wykorzystaniem czujników RGB–D do segmentacji obiektów pierwszoplanowych. Sekcja **??** opisuje sposób przekazywania sygnału pochodzącego z czujnika RGB–D do platformy obliczeniowej oraz architekturę do przetwarzania obrazu w technologi CUDA. Sekcja **??** zawiera opis implementacji sprzętowej w układzie Nvidia Jetson. W ostatnim rozdziale zamieszczono podsumowanie oraz wskazano dalsze kierunki rozwoju aplikacji.

2 Previous work

Autorzy publikacji [**?**] przedstawili inne, bardzo ciekawe i niestandardowe podejście do segmentacji obiektów pierwszoplanowych. Zaprezentowany algorytm zakłada wykorzystanie sieci neuronowa CNN. Do uczenia sieci użyto, oprócz annotowanych danych uczących, także czujnik RGB–D, czyli urządzenie generujące obraz wraz z mapami głębi.

Oczywiście istnieje wiele metod segmentacji obiektów pierwszoplanowych, wykorzystujących sieci nieuronowe, w tym przypadku autorzy skupili się na usprawnieniu procesu uczenia. Standardowo w tego typu algorytmach, sieć uczona jest na podstawie annotowanych obrazów w przestrzeni RGB. Zbiory testowe można podzielić na wiele kategorii, jednak uzyskany w ten sposób dokładność detekcji nie zawsze jest zadowalająca. W związku z tym autorzy przedstawili hybrydowy system, w którym równolegle uczone są dwie sieci. Pierwsza z nich wykorzystuje standardowy zbiór obrazów zapisanych w przestrzeni RGB, natomiast w drugiej sieci wykorzystywana jest mapa głębi tego obrazu. W zaproponowanym podejściu kluczową rolę odgrywa wymiana informacji pomiędzy obiema sieciami CNN w trakcie procesu uczenia. Dzięki takiemu rozwiązaniu można dużo efektywniej przeprowadzić taki proces dla obu sieci i następie połączyć je w jedną.

W pracy [**?**] został zaprezentowany kolejny, bazujący na obrazie z czujnika RGB–D, algorytm segmentacji obiektów pierwszoplanowych. W tym przypadku zaproponowano metodę działającą bez nadzoru, przedstawiony algorytm składa się z mechanizmu grupowania w dziedzinie barw i przestrzeni oraz statystycznego łączenia obszarów. Autorzy niestety nie przedstawili implementacji sprzętowej, przetestowano jedynie model programowy.

Wykorzystany algorytm grupowania JCSA (ang. *Joint Color-Spatial-Axial clustering*) służy do estymacji parametrów modelu tła, grupowania pikseli i w efekcie wyodrębnienia regionów na obrazie. Sam model tła jest hybrydowy i składa się z rozkładów Gaussa oraz Watsona. Do wspomnianego wcześniej grupowania pikseli użyto algorytmu BSC (*Bregman Soft Clustering*). W ostatniej fazie metody, czyli łączeniu poszczególnych regionów wykorzystano natomiast graf sąsiedztwa (ang. *RAG – Region Adjacency Graph*), przedstawiony proces polega na łączeniu odpowiednich wierzchołków w grafie.

Autorzy porównali zaprezentowany algorytm z innymi metodami bazującymi na obrazie głębi. Przeprowadzono testy dla różnych zestawów parametrów i zaproponowano odpowiednie wskaźniki jakości. Wykonane eksperymenty pozwoliły dobrać parametry algorytmu, natomiast uzyskane wyniki potwierdziły wyraźnie większą dokładność w stosunku do zaprezentowanych wcześniej rozwiązań.

Publikacja [**?**] przedstawia czujnik RGB–D w całości zrealizowany w układzie FPGA. System działa w czasie rzeczywistym z częstotliwością powyżej 30Hz. Wykorzystano do tego celu układ FPGA Spartan 6, do akwizycji obrazu użyto natomiast czujników Aptina pracujących w rozdzielczości $800x480$ pikseli z częstotliwością 60Hz. Komunikacja z sensorami odbywa się za pośrednictwem magistrali szeregowej I^2C.

W pierwszym kroku przeprowadzana jest operacja rektyfikacji obrazu, otrzymywany sygnał

pochodzi z dwóch sensorów konieczne jest zatem znalezienie odpowiadających sobie punktów na obu obrazach. Następnym etapem jest zastosowanie transformaty Censusa i operacja dopasowania stereo. Ostatni krok to złożona filtracja obrazu wyjściowego. Na wyjście systemu przekazywany jest 16 bitowy obraz głębi, który następnie jest przesyłany do hosta za pośrednictwem portu USB. Warto zaznaczyć, że autorzy wykorzystali technikę generacji kodu HLS (ang. *High–Level Synthesis*), dzięki temu większość funkcjonalności została zaimplementowana w języku C i automatycznie przekonwertowana na VHDL.

Autorzy publikacji [**?**] zaproponowali wykorzystanie układu GPU do akceleracji sprzętowej algorytmu wykorzystującego rozkłady Gaussa. Przedstawiony system służy do przetwarzania obrazu z kilku kamer jednocześnie. Uzyskane rezultaty nie były jednak zadowalające – uzyskano przyspieszenie jedynie około 50 procent w stosunku do modelu programowego. W tym przypadku największym ograniczeniem była konieczność transferu dużej ilości danych (kilka obrazów) pomiędzy CPU a GPU. Warto zwrócić uwagę, że autorzy wykorzystali jeden z tańszych układów graficznych dostępnych na rynku – GeForce GT 730. Aktualnie zdecydowana większość zintegrowanych GPU dysponuje porównywalną lub większą wydajnością.

Praca [**?**] przedstawia implementację algorytmu ViBE z wykorzystaniem układu GPU. Zaproponowana metoda dodatkowo wykorzystuje uproszczoną metodę Gabor Wavelets do uzyskiwania informacji o krawędziach obrazu. Na tej podstawie odpowiednie piksele zostają wykorzystywane do aktualizacji modelu tła. Autorzy dokonali optymalizacji metody, w taki sposób aby możliwe było pełne wykorzystanie potencjału układu GPU. W tym celu operacje dla poszczególnych pikseli wykonywane są niezależnie i mogą być wykonane równolegle. Implementacja została przetestowana na platformie sprzętowej składającej się z procesora Intel Core Quad Q8400 CPU oraz procesora graficznego Nvidia GTX 650Ti. Wykorzystano obraz o rozdzielczości 960x540, uzyskana wydajność wynosi odpowiednio 1.8 i 26 klatek na sekundę dla CPU i GPU w najbardziej rozbudowanym wariancie algorytmu.

Publikacja [**?**] opisuje złożony system wizyjny, w którym układ GPU został wykorzystany do akceleracji algorytmu służącego do detekcji i indeksacji obiektów pierwszoplanowych. Zaproponowany system służy do inteligentnego, automatycznego zarządzania energią w pomieszczeniu. Oprócz wspomnianego układu GPU wykorzystano także czujnik temperatury. Oba urządzenia przy pomocy protokołu Zigbee komunikują się z inteligentnymi licznikami. Na podstawie otrzymanych informacji inteligentne liczniki sterują klimatyzacją, oświetleniem i innymi urządzeniami elektrycznymi. Jednym z przykładów użycia może być sytuacja gdy zmniejszy się liczba osób w pomieszczeniu temperatura może zostać obniżona. Warto dodać, że całość może być także sterowana przy pomocy smartphona. Autorom udało się zapewnić oszczędność energii na poziomie 20-50 procent. Testy przeprowadzono na karcie graficznej GeForce GT 770 i osiągnięto wydajność na poziomie 34 klatek na sekundę w rozdzielczości $768x576$.

W pracy [**?**] autorzy zaimplementowali kilka algorytmów wizyjnych z wykorzystaniem GPU. Wśród zrealizowanych metod znalazły się detekcja ruchu, wykrywanie sabotażu kamery, wykrywanie porzuconego bagażu oraz śledzenie obiektów. Dzięki akceleracji GPU udało się uzyskać niemalże 22-krotne przyspieszenie w stosunku do CPU. Do testów użyto procesora NVIDIA Tesla C2075 z architekturą Keplera.

Do detekcji ruchu została wykorzystana metoda VSAM, jest to klasyczny algorytm wykorzystujący adaptacyjny model tła, osobny dla poszczególnych pikseli. Model ten jest aktualizowany wraz z każdą kolejną ramką na podstawie wyniku klasyfikacji. Opisana implementacja została także wykorzystana w metodzie służącej do wykrycia sabotażu kamery. Algorytm ten opiera się na porównaniu obraz wejściowego z obrazem tła i wyznaczeniu ich histogramów. Z kolei w celu detekcji przesunięcia kamery porównywane są obrazy tła z dwóch kolejnych ramek. W przypadku, gdy jeden obraz jest

przesunięty względem drugiego o dany wektor, oznacza to, że kamera została przesunięta. Detekcja porzuconego obiektu została zaimplementowana z wykorzystaniem algorytmu GMM do segmentacji tła oraz metody indeksacji. Finalnie analiza wyszczególnionych obszarów pozwala wyszukać obiekty statyczne. Dodatkowo autorzy wykorzystali metodę GMM również w algorytmie śledzenia obiektów.

3 Proposed algorithms

W trakcie badań zostały zaimplementowane dwa różne algorytmy, wykorzystujące zarówno obraz RGB pochodzący z kamery jak i obraz głębi. Pierwszym z nich jest rozszerzona wersja metody *Gaussian Mixture Models*. Implementacja została przygotowana w oparciu o publikację []. Drugą metodą jest również zmodyfikowana wersja istniejącego już algorytmu *Pixel Based Adaptive Segmenter* []. Oba algorytmy są zbliżone pod względem koncepcji modelu tła. Jest on niezależny dla każdego piksela i aktualizowany po przetworzeniu każdej ramki obrazu. W kolejnych podsekcjach przedstawiono szczegółowo koncepcję obu metod.

3.1 GMM algorithm with RGBD sensor

Gaussian Mixture Models is one of the most commonly used method for background modelling. In this approach each pixel is represented by k Gaussian distributions characterized by three parameters (ω, μ, σ^2). Modyfikacja algorytmu polega na wykorzystaniu mapy głębi obrazu. W tym celu oprócz standardowego modelu tła, używanego do przetwarzania kolorowego obrazu, został stworzony również osobny model wykorzystujący jedynie mapę głębi. Procedura inicjalizacji oraz następnie klasyfikacji piksela i aktualizacji modelu tła odbywa się tak samo jak w przypadku standardowego algorytmu.

ω is the normalized weight (range 0–1) of the Gaussian distribution. μ is the means vector of each colour component of a particular pixel. In the case of RGB colour space it can be defined as the vector of four numbers (r_{mean}, g_{mean}, b_{mean}). For the depth map it is a single number.

Finally, σ^2 is the variance of given Gaussian distribution – a single value is used for each colour component. Usually it is assumed that RGB components are independent, which allows to use 3 values instead of a covariance matrix. Again in case of depth image, it will be a single value. It should be noticed that a lot of varying implementations of the GMM algorithm have been proposed so far (cf. [**?**]). In this work, a version based on the open source image processing library OpenCV was implemented.

The background model is initialized while processing the first frame of the video sequence. The same initial weight and variance are assigned to each Gaussian distribution, while the vector of mean values is initialized with pixel values. The algorithm itself is build up of several steps. Firstly, sorting of Gaussian distributions with respect to weight in descending order is performed.

Then the current pixel (x) is tested against each Gaussian distribution. For match estimation the Mahalanobins distance formula is applied:

$$d(x, \mu) = \sqrt{(x - \mu) \cdot (x - \mu)^T} \tag{1}$$

A pixel is classified as matching the Gaussian if the computed distance is lower than established threshold. With respect to Equation (2) usually the triple value of standard deviation is used.

$$d(x, \mu) < 3 \cdot \sigma \tag{2}$$

The next step is pixel classification based on match test. According to Equation (3), first B Gaussian distributions, which weights exceed a constant threshold T are considered as background, otherwise they represent foreground. The default value of this parameter is 0.9 (the same as in OpenCV implementation).

$$B = arg_b min\left(\sum_{i=0}^{b} \omega_i > T\right) \tag{3}$$

The final step is model update. The following formulas are applied:

$$\omega_{i+1} = \omega_i + \alpha(M - \omega_i) \tag{4}$$

$$\mu_{i+1} = \mu_i + M\frac{\alpha}{\omega_i}(x - \mu_i) \tag{5}$$

$$\sigma_{i+1} = \sigma_i + M\frac{\alpha}{\omega_i}\Big((x - \mu_i) \cdot (x - \mu_i)^T\Big) \tag{6}$$

where α represents the learning speed, while M equals 1 for the first Gaussian distribution that passed the match test, otherwise it is 0. Moreover the value of variance is upper constrained. In the case of distributions, which do not match to pixel value, only the weight value is updated (decreased). If instead none of the Gaussian distributions match the pixel, than new Gaussian is added (the same parameters as in the initialization phase are used). The distribution with the lowest weight is replaced by the new one. Finally, weights have to be normalized to range 0–1.

Opisany proces jest wykonywany osobno dla obu modeli tła, pierwszego bazującego na obrazie RGB i drugiego opartego o mapę głębi. Finalnie więc otrzymujemy dwa wyniki klasyfikacji, na podstawie których podejmowana jest ostateczna decyzja. Do dalszego przetwarzania wykorzystujemy funkcję gęstości prawdopodobieństwa zależną od wartości piksela X_t w chwili t opisaną wzorem (7):

$$\eta(X_t, \mu, \sigma) = \frac{1}{2\pi\sigma} e^{-\frac{d(X_t,\mu)^2}{2\sigma}} \tag{7}$$

Kolejny krok to obliczenie współczynnika prawdopodobieństwa, dla obu modeli tła, przez wykorzystanie funkcji η. Operacja ta została przedstawiona na rysunku 1. Wykorzystany parametr s służy do przeskalowania otrzymanej gęstości prawdopodobieństwa i domyślnie wynosi 10000. Obliczanie współczynnika prawdopodobieństwa, przedstawione na diagramie, przebiega tak samo dla obu modeli tła. Ostatecznie na podstawie iloczynu obu prawdopodobieństw, piksel jest ostatecznie klasyfikowany. **TODO: może co z tego dokładnie wynika**.

3.2 PBAS algorithm with RGBD sensor

Modeł tła składa się z dwóch części. Pierwsza z nich zawiera N ostatnich zapamiętanych próbek (wartości pikseli). Zapisywane są zarówno składowe RGB każdego piksela jak i parametr *depth*. Definiujemy ten zbiór jako $B(x_i)$, gdzie x_i to aktualnie przetwarzany piksel obrazu, całość została opisana równaniem (8).

$$B(x_i) = \{B_1(x_i), B_2(x_i), \ldots, B_N(x_i)\} \tag{8}$$

Kolejnym elementem algorytmu jest okrąg $S(v(x, y)$ o środku w punkcie $v(x, y)$ i promieniu $R(x_i)$. Promień ten jest elementem modelu tła, aktualizowanym wraz z kolejnymi ramkami obrazu. Piksel jest uznawany za pierwszoplanowy jeżeli przynajmniej $\#_{min}$ próbek z modelu tła zawiera

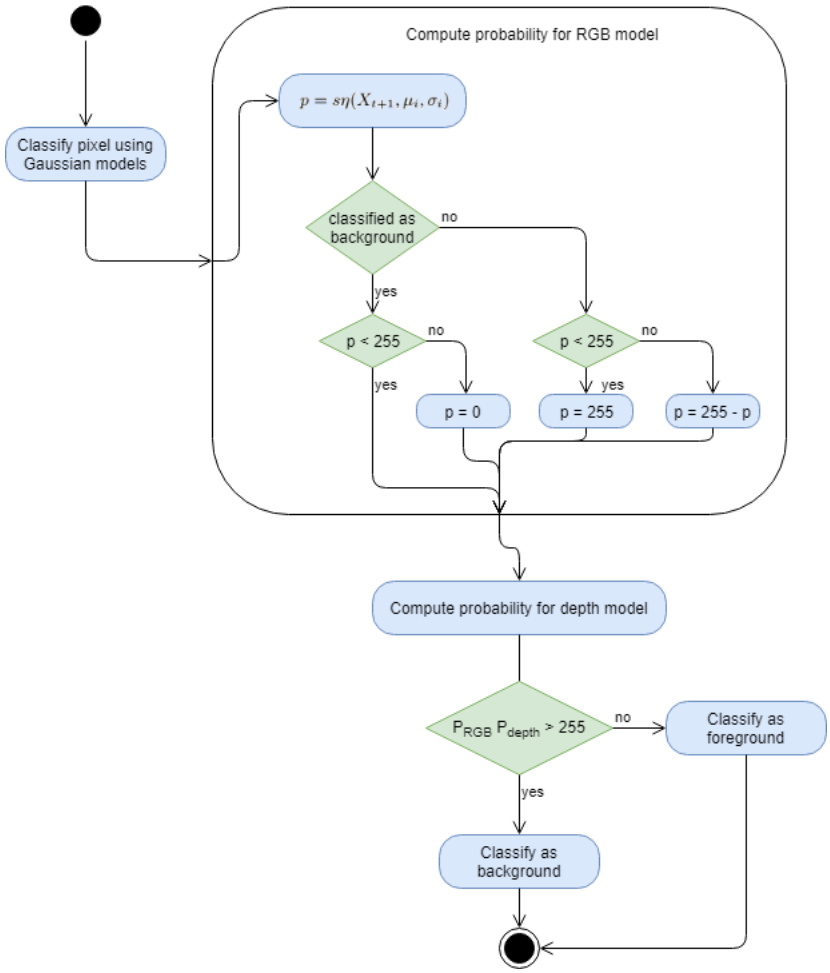

Figure 1: GMM – computing probability and classification

się wewnątrz takiego okręgu. Test dopasowania został opisany równaniem (9). Oznaczmy przez F maskę reprezentującą obiekty pierwszoplanowe (1 – piksel pierwszoplanowy, 0 – tło).

$$F(x_i) = \begin{cases} 1, & \text{gdy } \sum_{k=0}^{N} \{d(I(x_i), B_k(x_i)) < R(x_i)\} < \#_{min} \\ 0, & \text{w pozostałych przypadkach} \end{cases} \tag{9}$$

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

Ponieważ każdy kanał analizowany jest osobno, funkcję odległości pomiędzy próbkami można

zapisać bardzo prosto równaniem (10), jest to po prostu moduł różnicy.

$$d(I(x_i), B_k(x_i)) = |I(x_i) - B_k(x_i)| \tag{10}$$

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, F_D ostateczną klasyfikację możemy zapisać równaniem (11).

$$F_{RGBD} = F_R \vee \vee F_G \vee F_B \vee F_D \tag{11}$$

Kolejnym krokiem po przeprowadzeniu testu dopasowania i klasyfikacji piksela jest aktualizacja modelu tła. Zastosowano konserwatywne podejście, czyli aktualizowane są tylko piksele sklasyfikowane jako tło. 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 piksela z otoczenia $3x3$ 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 (12).

$$D(x_i) = \{D_1(x_i), D_2(x_i) \ldots, D_N(x_i)\} \tag{12}$$

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 (13).

$$d_{min}(x_i) = min_k d(I(x_i), B_k(x_i)) \tag{13}$$

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 (14). Warto dodać, że przyjęto także dolne ograniczenie wartości parametru, wynoszące $R_{low} = 18$.

$$R(x_i) = \begin{cases} R(x_i)(1 - R_{inc/dec}), & \text{jeżeli } R(x_i) > \bar{d}_{min}(x_i) R_{sc} \\ R(x_i)(1 + R_{inc/dec}) & \text{w przeciwnym razie} \end{cases} \tag{14}$$

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 (15). Przyjęto założenie, że parametr ten posiada także ograniczenie dolne jak i górne wynoszące odpowiednio $T_{low} = 2$ i $T_{up} = 200$.

$$T(x_i) = \begin{cases} T(x_i) + \dfrac{T_{inc}}{\bar{d}_{min}(x_i)}, & \text{jeżeli } F(x_i) = 1 \\ T(x_i) - \dfrac{T_{dec}}{\bar{d}_{min}(x_i)} & \text{w przeciwnym razie} \end{cases} \tag{15}$$

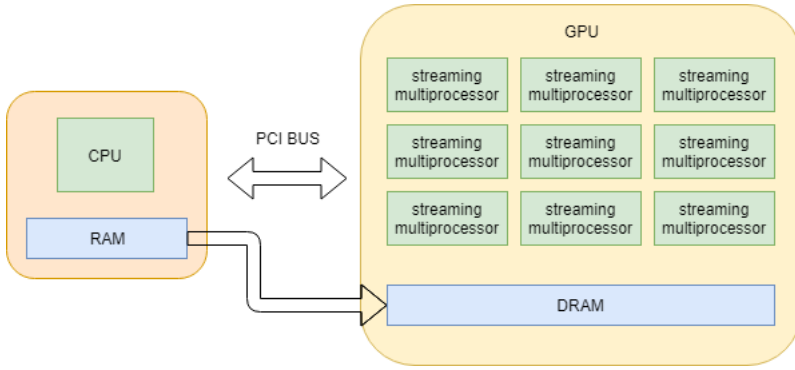

Figure 2: Communication between host (cpu) and GPU

4 Hardware implementation

4.1 Hardware used

Do akwizycji obrazu użyto czujników RGB–D firmy Intel: REAL SENSE Depth Camera D415 oraz REAL SENSE Depth Camera D435. Zapewniają one obraz w maksymalnej rozdzielczości FULL HD (1920 x 1080 pikseli) oraz mapę głębi w rozdzielczości HD (1280 x 720 pikseli). Sam czujnik umożliwia rozróżnianie obiektów w odległości od 20 centymetrów do 10 metrów od obiektywu.

Algorytm zaimplementowano na 3 różnych platformach sprzętowych. Pierwszą jest układ embedded GPU NVIDIA Jetson TX2 wyposażony w procesor ARM i układ GPU. Druga platforma to laptop wyposażony w procesor intel core i7-7700 i kartę graficzną Geforce GTX 1050m. Najwydajniejsza z testowanych platforma to komputer PC z intel Core i7-9700k i NVIDIA Geforce RTX 2070. **TODO: dokonczyc wstep**

4.2 Architektura

Do implementacji sprzętowej wykorzystano opracowaną przez Nvidia architekturę CUDA (Compute Unified Device Architecture). Dzięki temu implementacja może zostać uruchomiona na dowolnym układzie embedded GPU lub komputerze osobistym, które są wyposażone w procesor graficzny Nvidia.

W tego typu implementacji sprzętowej istotny jest podział zadań pomiędzy hostem i układem GPU oraz wykorzystanie pamięci współdzielonej. Host odpowiada za akwizycję obrazu i następnie skopiowanie go do pamięci współdzielonej z GPU. Oprócz tego po stronie hosta wykonywana jest również alokacja pamięci wykorzystywanej przez GPU, w związku z tym konieczne jest także zarezerwowanie pamięci dla modelu tła. Komunikacja odbywa się po szynie PCI, zostało to pokazane na rysunku 2.

Właściwy algorytm został w całości zaimplementowany na procesorze graficznym, gdzie dla każdego przetwarzanego piksela tworzony jest osobny wątek. Wyjściem algorytmu jest maska binarna przedstawiająca obiekty pierwszoplanowe, jest umieszczona w pamięci współdzielonej i następnie odczytana przez hosta i przekazana na wyjście. Proces wymiany informacji pomiędzy hostem i

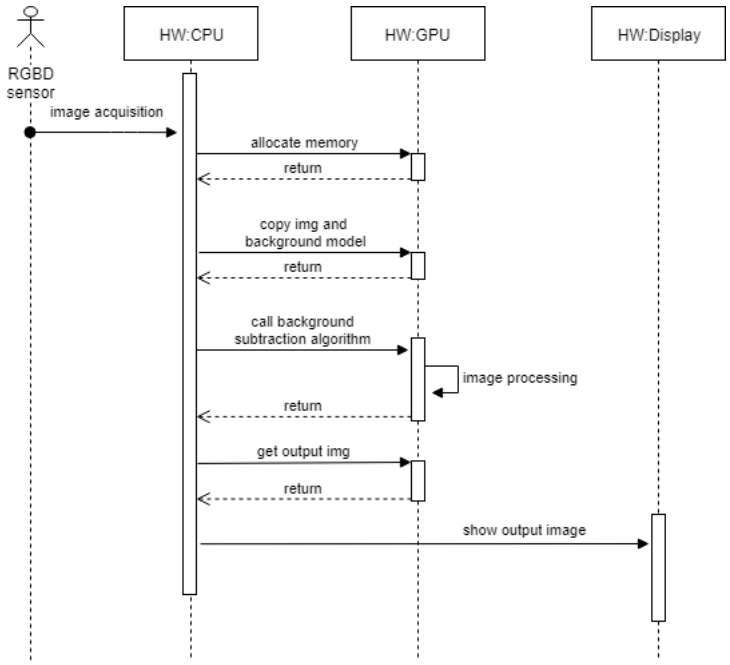

Figure 3:

gpu został pokazany na rys. 3.

Implementacja po stronie hosta została napisana w języku *C++*. Do akwizycji obrazu wykorzystano udostępnione przez firmę *Intel* SDK do obsługi sensorów RGB–D. Umożliwia ona odczyt zarówno obrazu RGB z kamery jak i składowej głębi w czasie rzeczywistym. Obraz głębi jest przesyłany z czujnika w formacie zmiennoprzecinkowym, zatem przed przesłaniem go do GPU konieczna jest konwersja do formatu 8 bitów na pixel (większa wartość oznacza, że obiekt jest dalej od kamery). Ostatecznie każdy piksel obrazu jest zapisany na 32 bitach, po 8 bitów na każdą składową RGB oraz odległość od czujnika.

4.3 GMM implementation

4.4 PBAS implementation

4.5 Performance

Wydajność na poszczególnych platformach testowych została zmierzona dla różnych rozdzielczości.

TODO:

-> zrzut ekranu

->wykorzystane urządzenia - specyfikacja,

->implementation result,

Table 1: Performance

	720p/480p	XXXXXX	XXXXXX	XXXXXX
Nvidia Jetson TX2	XXfps	XXfps	XXfps	XXfps
i7 7700hq + GTX 1050	XXfps	XXfps	XXfps	XXfps
i7 9700k + RTX 2070	XXfps	XXfps	XXfps	XXfps
Nvidia Jetson Xavier	XXfps	XXfps	XXfps	XXfps

->wydajność

5 Evaluation

W celu przetestowania zaimplementowanych algorytmów przygotowano krótkie sekwencje testowe nagrane kamerą RGB–D. Do każdego nagrania został przygotowany *ground truth*, czyli ręcznie anotowana maska obiektów. Wartość 255 oznacza, że dany piksel jest obiektem pierwszoplanowym, natomiast 0 oznacza tło. Wyjście w 100% poprawnego algorytmu powinno pokrywać się z tą maską.

Do ewaluacji użyto metodologii znanej między innymi z portalu *Change Detection* []. Porównują 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 współczynników oblicza się, 7 wskaźników jakości określających dokładność metody:

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

Uzyskane wyniki porównano z oryginalnymi implementacjami algorytmów GMM [] i PBAS [].

TODO:
->ewaluacja algorytmów,
->nagrane sekwencje testowe,
->zastosowane współczynniki,
->porównanie z oryginalnymi implementacjami (wyniki z publikacji artykułach ???)

Table 2: Evaluation

	GMM RGB–D	PBAS RGB–D	GMM []	PBAS []
Re	XX	XX	XX	XX
Spec	XX	XX	XX	XX
FPR	XX	XX	XX	XX
FNR	XX	XX	XX	XX
PWC	XX	XX	XX	XX
Pr	XX	XX	XX	XX
F1	XX	XX	XX	XX

6 Conclusion

TODO:
->w przypadku PBASa z indeksacją można dużo zrobić, np własna indeksacja w CUDA
->implementacja FPGA w przyszlosci ???