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

Piotr Janus*, Tomasz Kryjak*, Marek Gorgoń*

Tutaj tytuł
cd

Abstract: Abstract

Keywords:

1 Introduction

Foreground object segmentation is one of the most important element of modern AVSS (*Advanced Video Surveillance Systems*. It can be used in a variety of vision systems such as detection and tracking object or human behaviour analysis. Moreover it is a key element of application like abandoned luggage detection or forbidden zone protection. It might be applied in border control and airport systems as well.

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).

However, BGS approach has some serious limitations such as low adaptation to lighting changes, another weak point is the case when the color of background and foreground object are similar and the objects merge. This phenomenon is called color camouflage and in such a case is hard to retrieve foreground object properly. The main reason is that aforementioned techniques utilized human perception in some ways. Possibly image can be described in another color space (i.e. HSV or YCbCr) but it is still represented as a visible light, which is how people see it. Segmentation accuracy can be improved by using so called depth sensor. It extends the conventional image with depth map which provides the distance between particular pixel and sensor. This device is based on

*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

infrared projector and IR camera, projector shoots an irregular pattern of dots, which is invisible to humans, then IR camera is able to detect the infrared light bounced off from our subjects. The same technology is used by Kinect – Microsoft gaming console.

In this paper extended versions of the most commonly used background segmentation algorithms have been proposed. Described methods have been adjusted to image acquired from conventional RGB camera and depth sensor. The main contributions of this paper are:

- Extended version of Gaussian Mixture Model (GMM) and Pixel–Based–Adaptive–Segmenter (PBAS) algorithms adjusted to be used with conventional RGB image and depth sensor
- Embedded implementation of aforementioned algorithms on GPU using CUDA architecture
- Detailed performance comparison between standard (only RGB image) and RGD–D version

For image acquisition the sensor from *Intel Real–Sense* series has been used, while computing platform comes from Nvidia Jetson family. Moreover, the difference of performance between embedded platform and PC equipped with dedicated Nvidia Touring GPU is presented.

The reminder of this paper is organized as follows. In section 2 previous work related to use of RGB–D sensor for foreground object segmentation are briefly discussed, a few papers about acceleration using GPU are also presented. Section 3 describes adjusted version of GMM and PBAS method. In section 4 the designed embedded system is shown in presented. The evaluation of proposed algorithms is shown in section 5. The paper ends with a conclusion and future research directions indications.

2 Previous work

Over the years, several solutions for foreground object segmentation with the use of RGB–D sensor have been proposed. Sometimes researcher are also focused on improving performance instead of developing new algorithms. In such a case using GPU is a promising approach. The following section presents previous work related with both RGB–D sensors and GPU implementations.

Authors in [**?**] proposed foreground segmentation algorithm which uses convolutional neural network (CNN). Training data set consists both RGB and depth map images. The main contribution of that paper is improved training process. In order to produce more accurate detection, hybrid system with two CNN working in parallel is proposed. The first one uses conventional RGB image while second network is based on depth map only. The key element of this algorithm is so called mid-level fusion technique between RGB and depth CNNs. According to tests conducted by authors, the use of depth map results in an average 21% relative improvement in detection performance over an RGB-only network.

In the work [**?**] another foreground segmentation algorithm which is based on RGB–D sensor was presented. In this case algorithm works without supervision and consists of statistical model based on the color and the geometry of the scene. The authors presented only software model, without embedded or hardware implementation.

This model is used together with a JCSA *Joint Color-Spatial-Axial* clustering method, which is responsible for estimation of background parameters, clustering the pixels and generating the set of regions in the image. The authors proposed hybrid model which uses multivariat Gaussian and Watson distributions. For clustering operation, BSC (*Bregman Soft Clustering*) method has been used. The last phase of the algorithm is region merging, for this purpose RAG (ang. *Region Adjacency Graph*) is build.

Authors compared presented method with another algorithms based on depth image. Various test for different parameters were conducted, the set of quality indicators was also proposed. The

evaluation methodology is based on NYU depth database []. Test results proved that detection quality is significantly improved comparing to previous solutions.

In the article [**?**], GPU acceleration for Gaussian based foreground segmentation algorithm was proposed. That system is able to handle multiple cameras simultaneously. Unfortunately, the achieved increase of performance is not satisfying. Presented GPU implementation is only about 50 percent faster than reference model. In this case, according to authors knowledge, transferring huge amount of data between CPU and GPU was a bottleneck. It should be also noted that the used graphic processing unit was one of low-end model available on the market at that time – GeForce GT 730. Nowadays, the most of integrated GPUs are more powerful.

In the paper [**?**], GPU implementation of ViBE algorithm was presented. Proposed method uses simplified Gabor Wavelets to calculate image edge information. Such an approach improves background model update procedure. The authors proposed fully optimized GPU implementation, so the processing speed is accelerated by parallel computing capacity of GPU. Algorithm was tested on PC equipped with *Intel Core Quad Q8400* and *Nvidia GTX 650Ti*. For $960x540$ resolution achieved performance is 1.8 fps for CPU only and 26 fps for GPU implementation.

In the papaer [**?**], authors presented a few vision algorithms accelerated by GPU. Proposed methods affect the following areas: motion detection, camera sabotage detection (moved camera, out-of-focus camera, covered camera detection), abandoned object detection, object tracking algorithm. GPU acceleration gave almost 22 times increase in performance, tests were conducted using *NVIDIA Tesla C2075* GPU based on Kepler architecture.

For move detection, VSAM (*Video Surveillance and Monitoring*) was used. It is adaptive approach which updates background model with each new frame. The similar approach has been used in camera sabotage detection algorithm. This method compares input image with background and computes their histograms. For protection against moving camera, background images from two subsequent frames are compared. Reduction algorithm is used compute difference between frames. Abandoned object detection algorithm is based on GMM and labeling method. In addition, motion detection also uses Gaussian Mixture Models.

3 Proposed algorithms

In this research, the authors implemented two different background subtraction algorithms. They benefit from both, RGB image from traditional camera and depth sensor as well. The fist algorithm is an extended version of *Gaussian Mixture Models*. Its implementation is based of []. The second method is also modified version of existing algorithm – *Pixel Based Adaptive Segmenter* []. Both are very similar regarding the background model concept. It is independent for each pixel and dynamically updated after every frame. In the following subsections detailed description of both method is presented.

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). As it was mentioned in previous section, the modification consists in use of depth map in parallel to RGB image. For this purpose the separate background model has been used. It is based only on depth image, but the similar procedures for initialization, classification and update are applied.

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 partially based on [] and 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.

Aforementioned classification process has to be done separately for both background models. Finally there are two different classification results, which are used to make final decision. For further processing the probability density function which depends on pixel value X_t in time t is used, this function is described by equation (7).

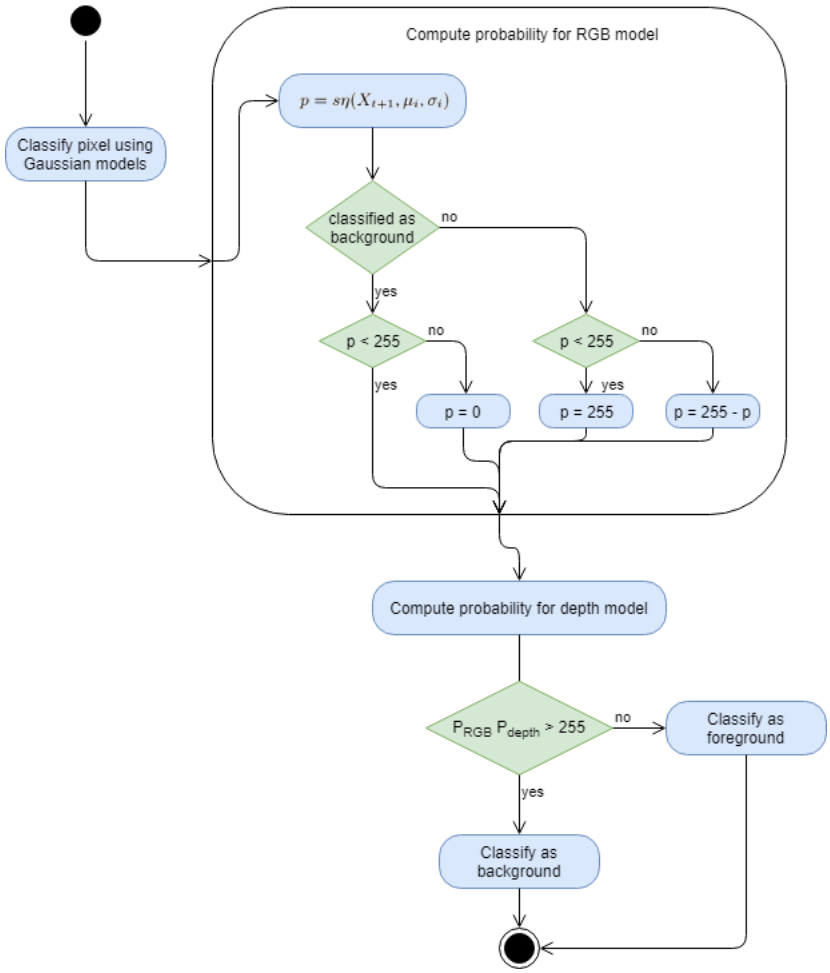

Figure 1: GMM – computing probability and classification

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

The next step is computing the probability factor for both background models, this operation is presented in figure 1. Parameter s is used for scaling probability density and its default value is 10000. The process of computing probability factor is the same for both models. Then the product of two values is computed and final classification is made according to the diagram. **TODO: może co z tego dokładnie wynika**.

3.2 PBAS algorithm with RGBD sensor

The background model is composed of two parts. The first one is a buffer of N last samples from the analyzed video sequence. Particular sample consists both RGB value and depth parameter. Let x_i denote a particular pixel and $B(x_i)$ a buffer, it is described by equation (8).

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

The next component of background model is a sphere $S(v(x,y)$ of radius $R(x_i)$ centered at point $v(x,y)$. The radius value is updated with each new frame. Pixel is considered as foreground if at least $\#_{min}$ samples from background model belongs to the sphere. Let $F(x_i)$ denote the foreground mask (1 – foreground pixel, 0 – background), the match test is described by equation (9).

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

Where d is a distance function between a sample from background model and actual pixel.

Since each channel is processed separately, the distance function can be easily represented by equation (10). The final decision is based on logical alternative of the result obtained for each channel. Let F_R, F_G, F_B, F_D denote classification result for particular channels, final result is described by (11).

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

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

In the next step the background model is updated. Proposed algorithm uses conservative update approach, it means that the model is updated only if classified pixel is a part of background, otherwise updater is skipped. In addition to conventional conservative approach, update decision is also randomized. The probability of performing an update is given as $p = 1/T(x_i)$, where $T(x_i)$ is independent for each pixel and updated with background model. The actual update procedure is based on replacing a randomly selected sample from the background model $B_k(x_i)$ with the current pixel value $I(x_i)$. Moreover, a random pixel from a $3x3$ local context is selected and a randomly selected sample from its model is replaced by pixel value.

The update of $R(x_i)$ and $T(x_i)$ parameters is performed independently from the first part of background model. In this case the next component of the model has to be define – the minimum distances between samples from the model and current pixel are defined as (12).

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

Presented set of values – $D(x_i)$ is updated together with the set of samples $B(x_i)$. If a pixel is updated the minimal distance is found with equation (13) and the value $D_k(x_i)$ is updated with $d_{min}(x_i)$.

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

In $R(x_i)$ update procedure the background dynamics parameter is used, it is the mean value of $D(x_i)$, the whole procedure is described by equation (14). Additionally the decision threshold was limited by a lower bound $R_{low} = 18$.

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

Where:

$R_{inc/dec}$ – constant updater rate (0.05 by default)

$\bar{d}_{min}(x_i)$ – the mean value of $D(x_i)$

R_{sc} – scaling factor (5 by default)

The final step is update of learning rate parameter – $T(x_i)$. The new value depends on classification result and it is described as (15). Additionally the learning rate is limited by a lower and upper bound. These constraints are $T_{low} = 2$ and $T_{up} = 200$ respectively.

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

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 następ-

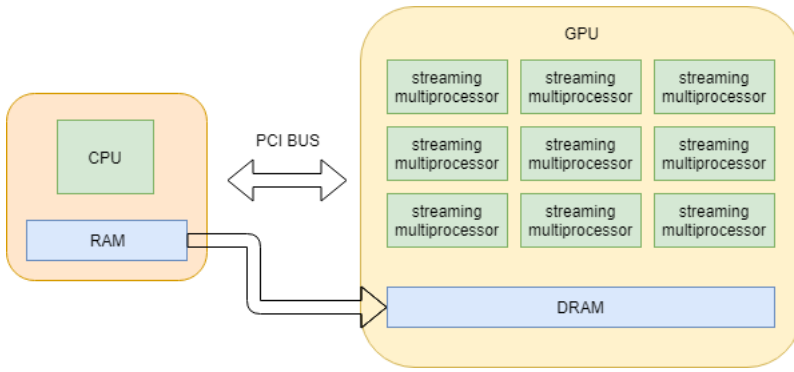

Figure 2: Communication between host (cpu) and GPU

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

nie odczytana przez hosta i przekazana na wyjście. Proces wymiany informacji pomiędzy hostem i 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,
->wydajność

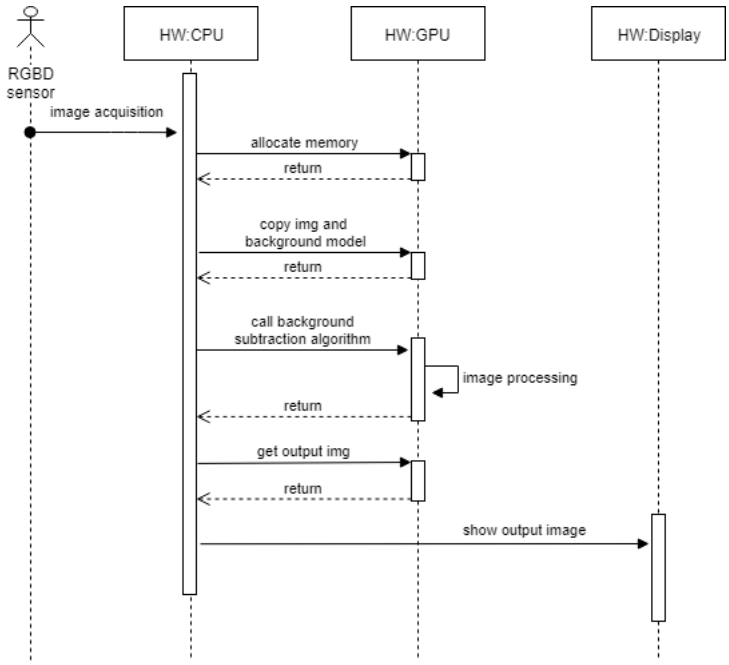

Figure 3:

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) : $TN/(TN + FP)$

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

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 ???)

6 Conclusion

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