BACKGROUND SUBTRACTION

IMPLEMENTATION:

1.USING GAUSSIAN MIXTURE MODEL:

PIXEL PROCESSES AND MODELLING PARAMETERS:

$$\{X_1, \ldots, X_t\} = \{I(x_0, y_0, i) : 1 \leq i \leq t\},$$

At any time, t, I have history of particular pixel (X_0, Y_0). Models the values of a particular pixels as a mixture of Gaussians. At any time, t, I have k distributions of Gaussians for each pixels (I used 4 Gaussians). For each gaussians I have:

1. $\omega_{i,t}$: is an estimate of the weight of i^{th} Gaussian in mixture, at time t.
2. $\mu_{i,t}$: is the mean value of i^{th} Gaussian in the mixture, at time t.
3. $\sum_{i,t}$: covariance matrix of i^{th} Gaussian in the mixture, at time t.

For computational reasons, I assumed that the RGB colour components are independent and have the same variances. So, the covariance matrix is of the form: $\Sigma_{i,t} = \sigma_{i,t}^2 I$.

Now, the probability of observing the current pixel value is:

$$P(X_t) = \sum_{i=1}^{K} \omega_{i,t}\, \eta(X_t, \mu_{i,t}, \Sigma_{i,t})$$

Where 'K' is number of Gaussian distribution, ω,μ,∑ is stated previously, and

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

(Gaussian Probability density function).

UPDATE THE MIXTURE MODEL:

Every new pixel value, X_t, is checked against the existing K Gaussian distributions until a match is found. A pixel matches a Gaussian distribution if the Mahalanobis distance $sqrt\left((X_{t+1}-\mu_{i,t})^T \cdot \Sigma_{i,t}^{-1} \cdot (X_{t+1}-\mu_{i,t})\right) < k\sigma_{i,t}$ where *k* is a constant threshold equal to 2.5. Now two cases can occur:

1. If none of the K distributions match the current pixel value, the least probable distribution is go out. A new distribution with the current value as its mean value, an initially high variance, and low prior weight, is enter.
2. A match is found with one of the K Gaussians. For the matched component, the update is done as follows:

   $\omega_{i,t+1} = (1-\alpha)\omega_{i,t} + \alpha$ where α is constant learning rate.

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

   $\sigma_{i,t+1}^2 = (1-\rho)\sigma_{i,t}^2 + \rho(X_{t+1}-\mu_{i,t+1}).(X_{t+1}-\mu_{i,t+1})^T$ where $\rho = \alpha\eta(X_{t+1}, \mu_i, \Sigma_i)$

   For the unmatched components, μ and ∑ are unchanged, only the weight is replaced by:

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

BACKGROUND MODEL ESTIMATION:

the Gaussians are ordered by the value of ω/σ Then, the first B distributions are chosen as the background model, where

$B = \operatorname{argmin}_b \left(\sum_{i=1}^{b} \omega_{i,t} > T\right)$ T is a measure of the minimum portion of the data that should be accounted for by the background.

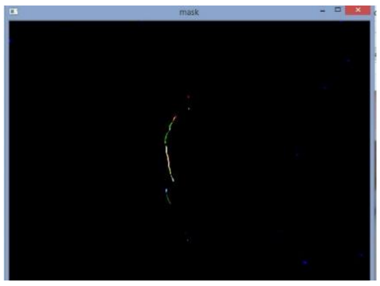
I have created classes Gaussian: to create Gaussian instance, Node: To store information of pixels and number of components, Node1: to create Linked List of gaussian of particular pixels.
And functions Create_Gaussian(), Create Node(), Insert Gaussian(), Delete_Guassian() and Process() for our future use. And then I implemented the algorithm as described above.

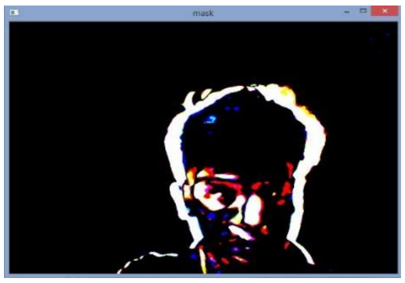
2.USING OPENCV LIBRARY:

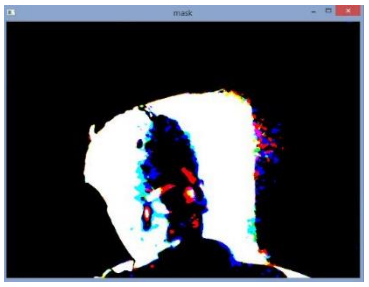
For this purpose, I used OpenCV library for background subtraction modeling using GMM called `cv2.createBackgroundSubtractorMOG2()`

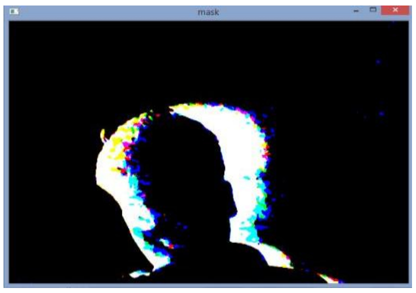
EXPERIMENTAL ANALYSIS:

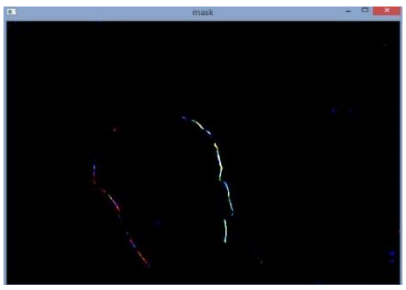
RESULT FROM MY OWN IMPLEMENTATION:



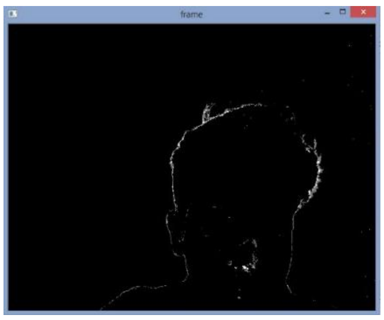


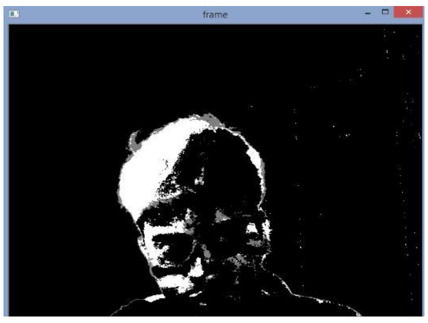


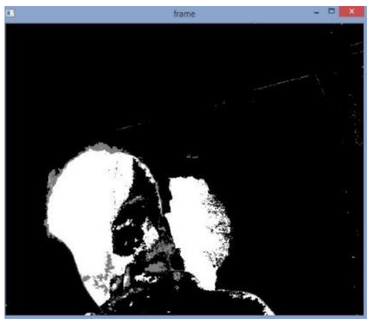


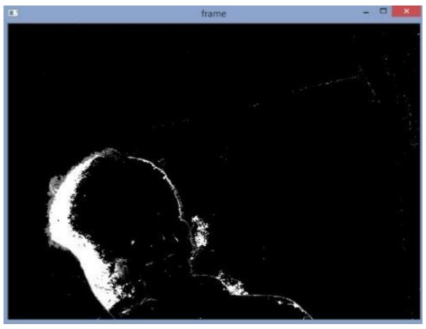


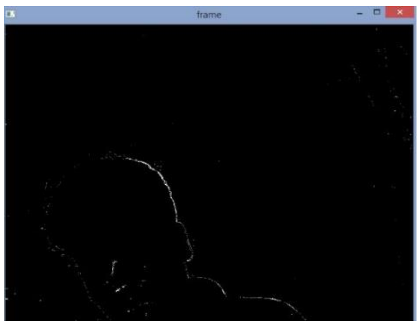
RESULT FROM USING LIBRARY:





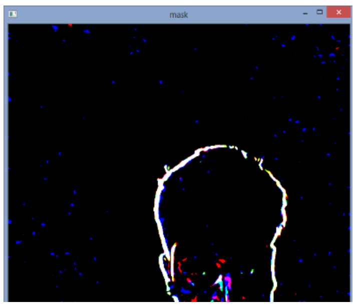




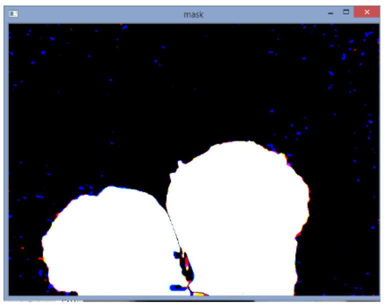


VARIATION WITH LEARNING RATE α:

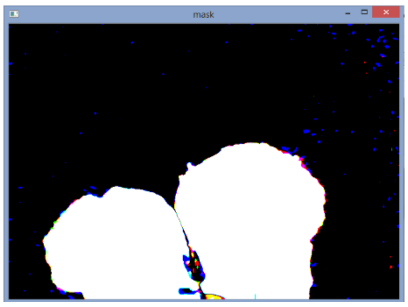
α = 0.001



At t = 0 sec

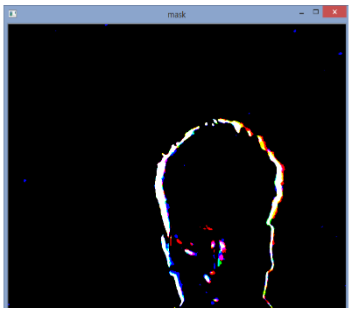


At t = 3.0 sec

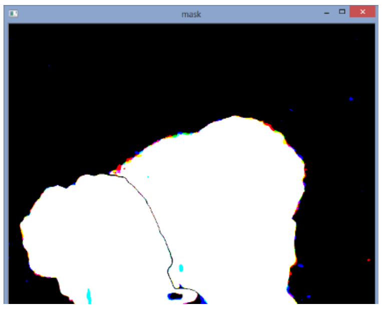


At t = 5.0 sec

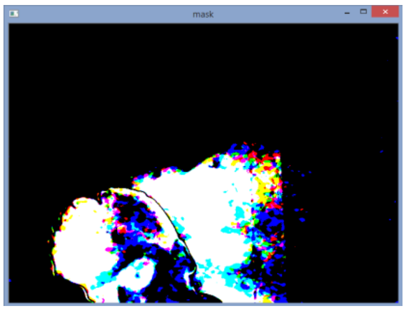
α = 0.01



At t = 0 sec

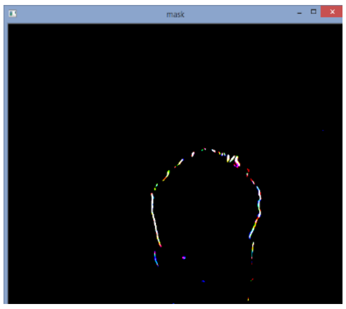


At t = 3.0 sec

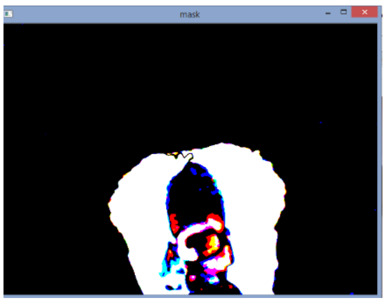


At t = 5.0 sec

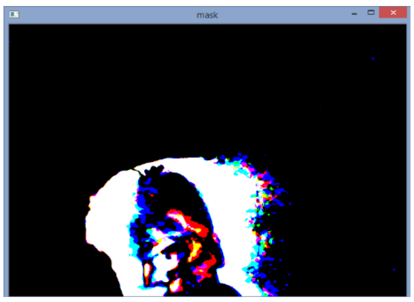
α = 0.1



At t = 0 sec

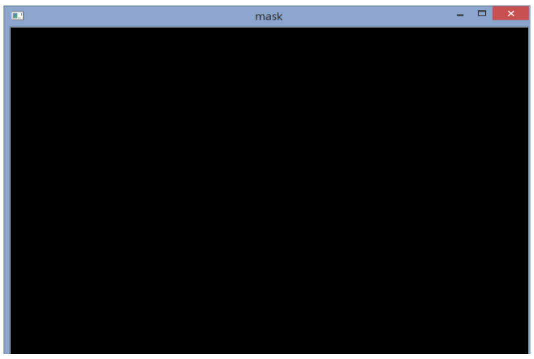


At t = 3.0 sec



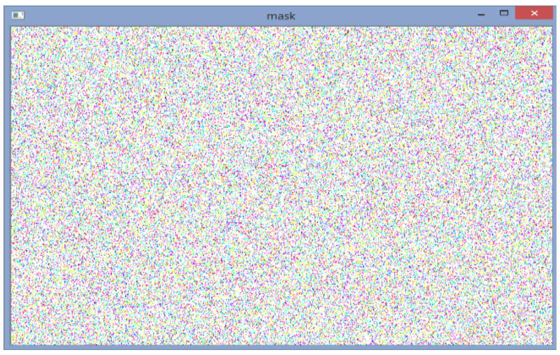
At t =5.0 sec

α = 1.0



At t = 0 sec and onward

α = 10.0



At t = 0 sec and onward

CONCLUSION:

After experimenting I came up with some more important conclusions. These are listed below:

1.My own implementation and library implementation approximately giving same results. They both detect foreground and update status of background. But my implementation is getting less accurate as compared to library one because of implementation of optimised algorithms and best parameter tuning in library implementation.

2.**Variation of Gaussian mixture model with learning rate alpha**: I experimented on various alpha values for my parameter tuning. I found that alpha ranges should be lies between 0.05 – 0.5. Best result came from alpha = 0.1. For lower values of alpha less than 0.05, algorithm is not updating the model very fast. And similarly, for higher value of alpha, algorithm is not able to separate foreground and background separately. I have attached the performance of algorithm with time and different value of alpha.

3.**Maintenance of the Weight, the Mean and the Variance:** I have updated the weight, the mean and the variance of each Gaussians with an IIR filter using a constant learning rate a for the weight update and a learning rate r for the mean and variance update. Hence these factors are dependent on learning rate and hence we should directly focus on learning rates.

4.**Dependence on number of Gaussian distribution:** Also, GMM depends on the number of components you are using, but throughout the algorithm, it remains same. I used 4 components for Gaussian distribution for each pixel.

5**.Learning rate ρ**: For the learning rate ρ, the problem is that $\eta(X_t, \mu_{i,t}, \Sigma_{i,t})$ is usually very small, this makes convergence tolerably slow. But I have used adaptive nature of learning rate and hence it is auto adjustable within the algorithm. It is also dependent on values of alpha.