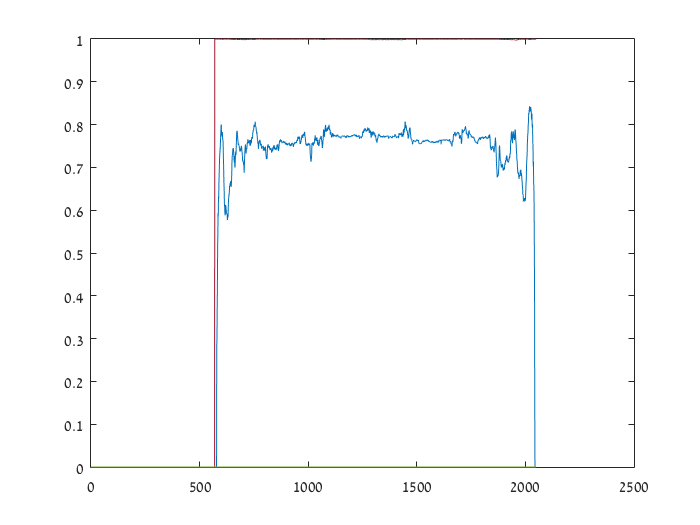
**Part A: Naive Background Subtraction**

Was tested against the "office" video.

2)

b) False Positives , False Negatives

3)

The algorithm is performing the Naive Background Subtraction algorithm in the following manner:

1. Performing pre-calculation actions according to the **C parameter** which determine if the video is processed as true-color or grayscale.
2. Calculating background-average of the first N frames. **N is a parameter** for how far the background-average calculation goes back. **Mean parameter** is in charge of how the average is made: Mean or Median
3. We Iterate over the video frames and subtract the background-average from each. After the Nth frame, we update the background-average with the current frame value. If **Mean parameter** is chosen, the update is calculated with the formula  
     
   .  
     
   The **LearningRate parameter** determines how fast new images affecting the background average.  
     
   If Mean is not chosen then a median over the last N frames in considered.  
     
   The result of the subtraction, if it's above the **threshold parameter**, it is considered as foreground and valued "1" in the binary result, "0" otherwise (as background).

When a true-color image is processed,

1. After the iterations, we hold the binary result of the video. With the **O parameter** we determine what type of result is output (Color / GrayScale or Binary).  
   If the output is Binary we don't have to do any manipulations.  
   If the output is grayscale, we copy the original color of each pixel for each frame, where it is considered as foreground in the binary matrix, into the result, and the output is a grayscale video on a black background.  
   If the output is true-color, then we duplicate the binary result 3 times, each for one color channel. When there's a pixel with value of "1" in the binary, we take the original color-value of that channel, and output to the result.

Parameters summary:

**VideoMat** – the input video.

**C** = {1=color, 0=grayscale}. How the image is processed: as true-color or grayscale. We chose C = 0 since there were less false positives & negatives.

**N** = numbers of frames. We chose N = 100, for taking a solid enough background average. Since our algorithm is using the LearningRate for updating the background after N frames, as N goes higher, more frames are directly-inserted into the average, ignoring the learning rate.

**Mean** = {1=mean, 0=median}. How the background is calculated. We chose Mean=1, because the result is more accurate.

**LearningRate** – how fast new frames change the average background, when Mean=1. We chose 0.0001 because in the tested video we wanted new frames to affect slowly. Otherwise, many new-objects (such as the man in the office movie) were considered as background too fast.

**Threshold** – If after the background subtraction, the pixel value is above Threshold, it is considered as foreground. We chose Threshold = 50, a value suited for us to remove noise from the picture but still see clearly the new objects.

**O** = {1=binary, 0=non-binary}. When O=1, every frame in the output video has pixels with "1" or "0" value (foreground or background respectively), forming a binary video. We chose O=1 since a binary video needs to be compared against the GroundTruth.

4)

Advantages:

* Light calculation: results in faster calculations, which leads to the capability for performing the subtraction on live video quite fast, for example.

Limitations:

* Accuracy is not the cutting edge. The FPR and FNR may be quite higher in comparison with other algorithms.
* Cannot detect constantly-moving-objects as background, such as leaves or water.
* Shadow may result as false positive.

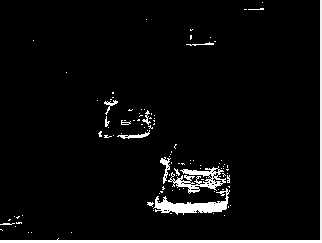
**Part B: Non-parametric bg subtraction using KDE**

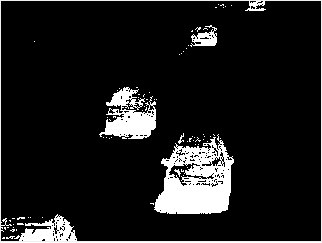
Was tested against the "Highway" video.

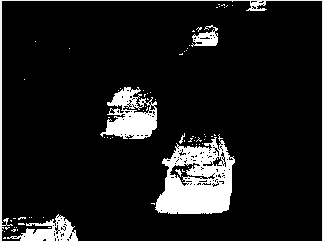
2)

The 3 possibilities have a trade-off for the following properties:

1. False-positives count.
2. True positives that are missed (turns to false negatives)

The blind update misses many true-positives (= false negatives), and also less false-positives.  


Selective update with single threshold results with a little more false-positives, but with much less true-positives that are missed (and turns into false negatives). 

Selective update with 2 thresholds has more false-positives, but also more true-positives than all the other cases.  
 

3)

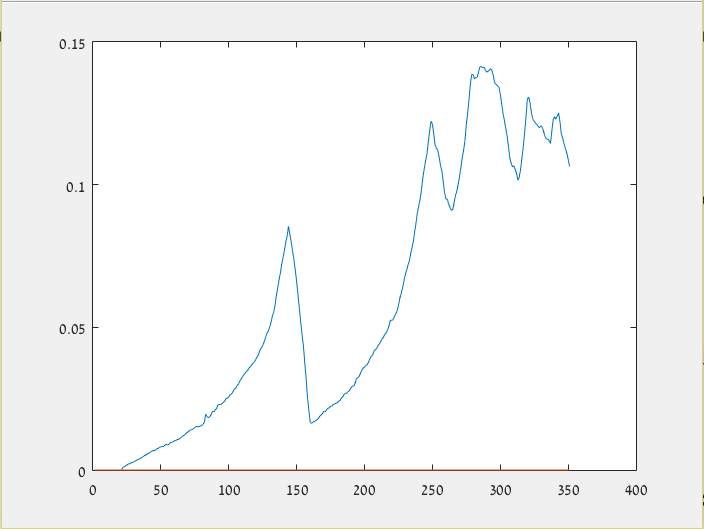
The parameters are:

**O** = {1=binary, 0=non-binary}. The output video type. Binary for black & white, 0 for grayscale. We chose O = 1 because we want to compare to GT.

**N** = The number of last frames that the KDE algo is using to estimate the probability of a pixel for belonging the background. We chose 20 for a good ratio of performance, enough history to be saved, and effect on the background probability.

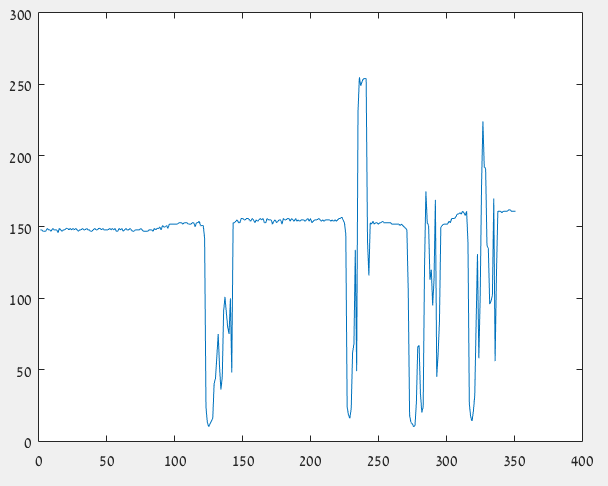
**Threshold** = {If selected == 1: Threshold==[firstThreshold, secondThreshold], otherwise Threshold=<number>}. The first threshold is used for determining whether a pixel belongs to the background. The second threshold should when the Selected flag is on. IT filter further the selected background filter. We chose Threshold = [0.00000000001, 0.0000000000000001] to have clearer images, such as at section (2).

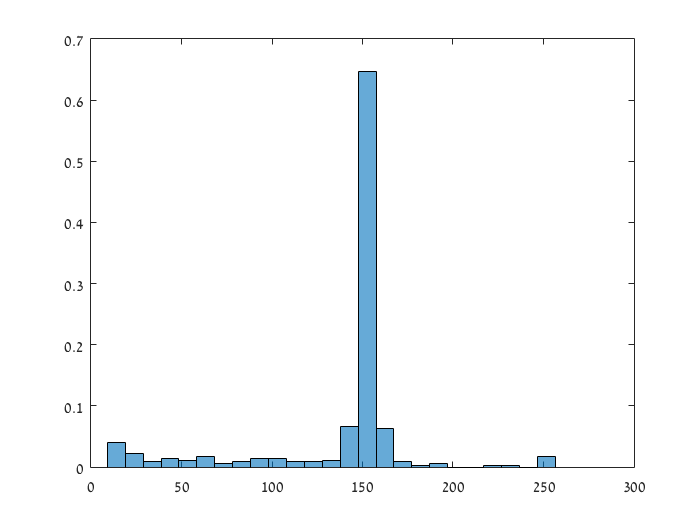
4)

False Positives , False Negatives

It is better than the naive one because the moving objects in the background.

5)

a) Single pixel intensity (grayscale) as function of frame number.  
 

b) Histogram of the pixel sequence:  


6)

The KDE algorithm is converting the video to grayscale, and then processing. It supports blind update, selective with single & double threshold.

The algorithm iterates over the video frames. If non-selective, the algorithm just calculates the probability for each pixel to be in the background, and thresholding it, and then output the result with pixels that are background are "0", and foreground are "1".

When selective, we hold a matrix that stores for each pixel in the frame, in which frame in the history it was updated for the last time (IndicesInHistoryMatrix). In each iteration, if the pixel is background – this is the last time it was updated and we reset the value to 0. If it's foreground, we increase the index at the history by 1. Then, the final output is every pixel with its latest value, thresholded once or twice, depends on the parameter.

Either way, after thresholding – the BackgoundMask is ready, and added the binary video.

Advantages:

* Ignores constant moving-background objects such as leaves, water, etc. when separating foreground and background.

Disadvantages:

* Calculations are heavy if we are not performing certain pre-calculations, such as lookup-table for kernel function values.
* Shadow may result as false positive.
* The algorithm is not capable of handling images that "capture motion", or blurry images.

7)

a. Choosing the Canoe example:  


And its result after the algorithm, with two thresholds (left) and lighter two thresholds (right)  


As the images suggest, the boat looks unclear.

We believe this is because we perform the algorithm on grayscaled video, and it can be observed that the gray-level of the boat is similar to the one of the water. Therefore the subtraction is barely thresholded, or the whole image is too noisy.

b. The solution is to perform it on the true-color video, and the boat will be seen clearly even when heavily-thresholded:  
