# Object Tracking

Shree K. Nayar

Monograph: FPCV-5-1

Module: Perception

Series: First Principles of Computer Vision

Computer Science, Columbia University

April, 2025

FPCV Channel

FPCV Website

We have seen in the previous lectures how optical flow can be used to determine how each point in the scene moves from frame to frame in a video. Now, we are going to look at tracking objects from frame to frame in a video. We want the tracking algorithm to be insensitive to changes in illumination, scale, rotation, and even occlusions as objects move over space and time.

## Object Tracking

Shree K. Nayar
Columbia University

Topic: Object Tracking, Module: Perception
First Principles of Computer Vision

1

Our goal is to track the location of a target object from one frame to the next frame in a video. If we can do this through an entire video, we can come up with tracks for every object of interest in the scene, and understand how they move with respect to each other.

## Object Tracking

Track the location of target objects in each frame of a video sequence.

Topics:

(1) Change Detection

(2) Gaussian Mixture Model

(3) Object Tracking using Templates

(4) Tracking by Feature Detection

2

First, we look at the problem of change detection, where we want to detect when each pixel in a video undergoes a meaningful change. By meaningful change we mean a significant event, such as the motion of an object through the pixel, and not changes due to noise, illumination, or weather (rain, snow, etc.). To this end, we will look at the intensity variations at each pixel as a distribution, and then fit a model to the distribution. We model the pixel intensity distribution using a Gaussian mixture model (GMM). The parameters of the GMM are then used to classify each pixel as belonging to the foreground (a meaningful change) or the background.

After detecting meaningful changes, we want to track objects over a video sequence. The first method simply uses the object of interest in one frame as a template and determines using template matching where it ends up in the next frame. The template can be a window of brightness values or a histogram of brightness values within the window. Finally, we will develop a tracking algorithm that uses SIFT features. Features detected in a target window in one frame are used to find a translated and distorted window in the next frame that includes a large number of detected features in the previous frame. Finally, we conclude with a few popular applications of object tracking.

Before we can track objects over space and time, we want to first figure out which parts of the scene are undergoing meaningful changes, while ignoring uninteresting changes. Let us first examine what we mean by meaningful and uninteresting changes.

## Change Detection

Shree K. Nayar
Columbia University

Topic: Object Tracking, Module: Perception
First Principles of Computer Vision

3

Consider a static camera observing a scene, such as the highway on the left, or the street intersection on the right. The objects of interest to us are vehicles on the highway on the left and people walking on the street on the right. We want to classify each pixel in the video captured by the camera as belonging to the changing foreground or belonging to a static background. We refer to this as a foreground/background classification problem. At first glance, this may appear to be a simple task. However, as we will see, it is a challenging one as we want our change detection to be resilient to changes that are not of interest to us.

## Change Detection

Given: Static cameras observing scene (room, street, etc.)

Find: Meaningful changes (moving objects, people, etc.)

Robust and real-time classification of each pixel as "foreground" (motion/change) or "background" (static).

4

What do we mean by uninteresting changes in a scene? In example 1, the motion of the boat is clearly a meaningful change, but the ripples on the water are not of interest to us even though they could produce substantial flickering at each pixel. Similarly, at low light levels, the video produced by the camera could be noisy, causing rapid fluctuations of pixel intensity. We would like our change detection method to be robust to these kinds of changes.

## Change Detection: Challenges

Ignore uninteresting changes:

- Background fluctuations
- Image noise
- Rain, snow, turbulence
- Illumination changes & shadows
- Camera shake

1

2

3

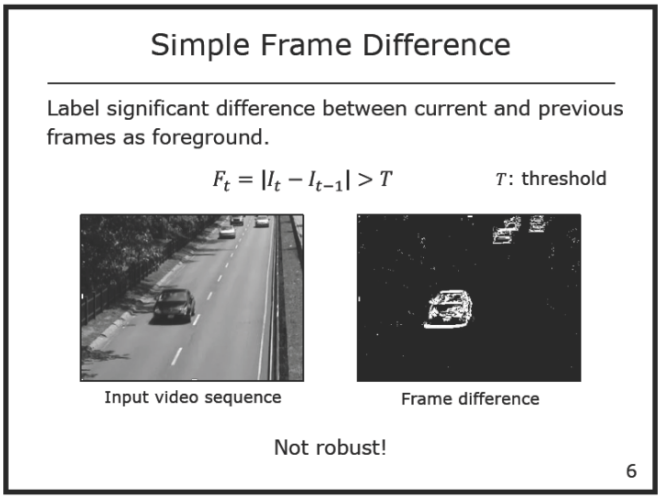
5

Another cause of uninteresting changes could be weather effects such as rain and snow, as shown in example [2]. In this case, we want to detect the moving cars on the street but the raindrops and snowflakes also produce significant brightness fluctuations which are not of interest to us.

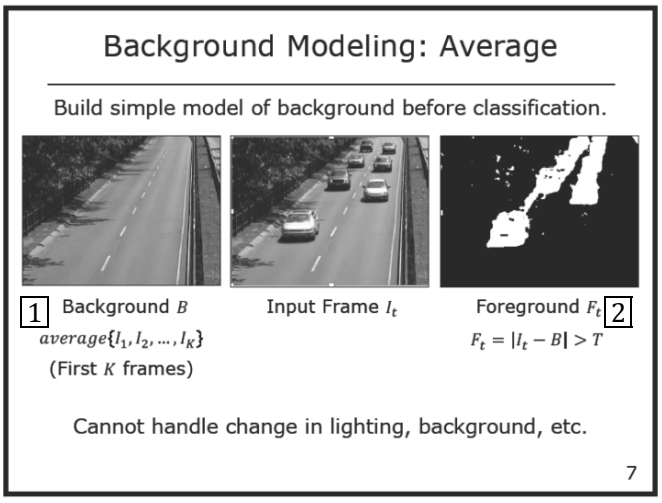
Illumination changes are another challenge for change detection. In example [3], each person moving in the scene has their shadow attached to them. While the people are of interest to us, their shadows are not. We would like our change detection algorithm to be resilient to such illumination effects.

Finally, there is camera shake. Consider an outdoor camera mounted on a pole. On a windy day, the camera can sway back and forth, causing the intensities of all camera pixels to fluctuate. Once again, these are changes that we want our change detection algorithm to ignore.

Let us start with the simplest method for detecting changes in a scene, namely, frame differencing. We simply compute the difference between the current frame and the previous frame, and check whether the difference is greater than a threshold at each pixel in the image. In the example shown here, we can see that the moving cars produce detectable changes. However, the waving leaves in the background produce large changes as well. Furthermore, if a car has uniform color, as most cars do, only the edges of the car produce changes as the car moves. For all these reasons, simple differencing is not an effective solution.

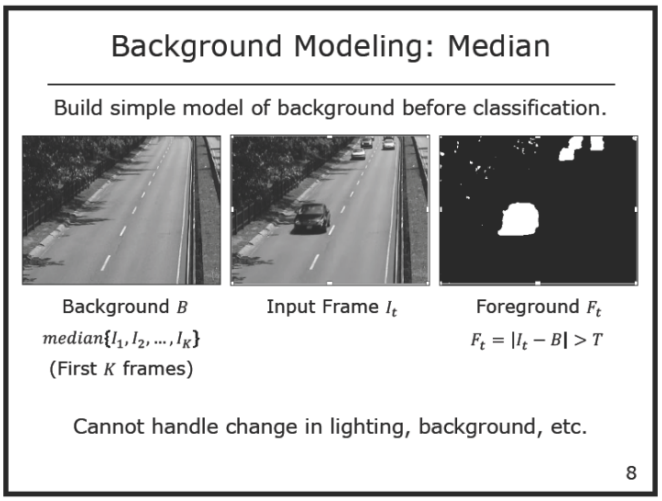


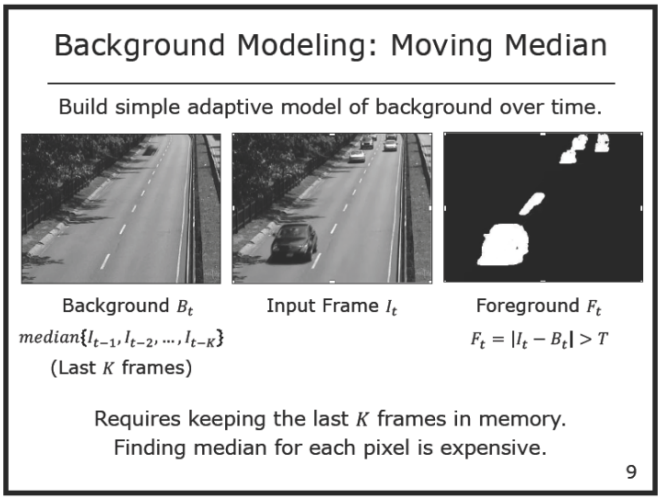

One way to improve the above differencing method is by computing a background image that is more robust than just the previous image. On the right, the background image is the average of the first K frames of the video of the scene [1]. The difference between this image and the current image is used to detect changes. This works slightly better, but it cannot handle the waving of the leaves or any changes in lighting [2].

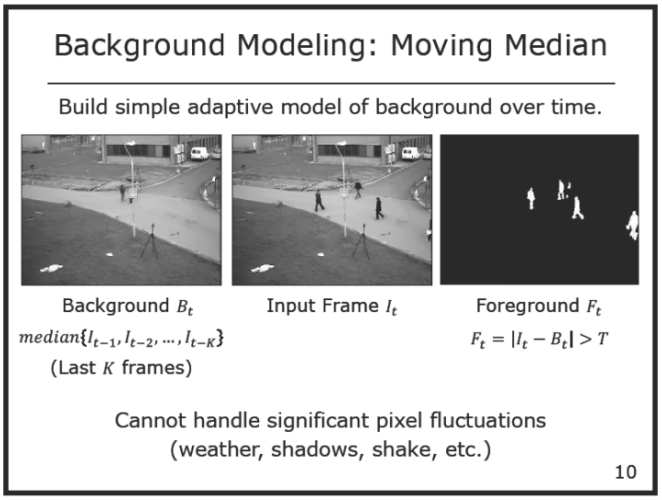


We can do a bit better by using the median of the first K frames. The median is better than the average because it represents the most popular intensity seen in the K frames. However, since the median is computed using just the first few frames, it loses its meaning with time since the lighting of the scene is bound to vary.

Thus, we want our background model to not only represent the past but also want this model to adapt to gradual scene changes.



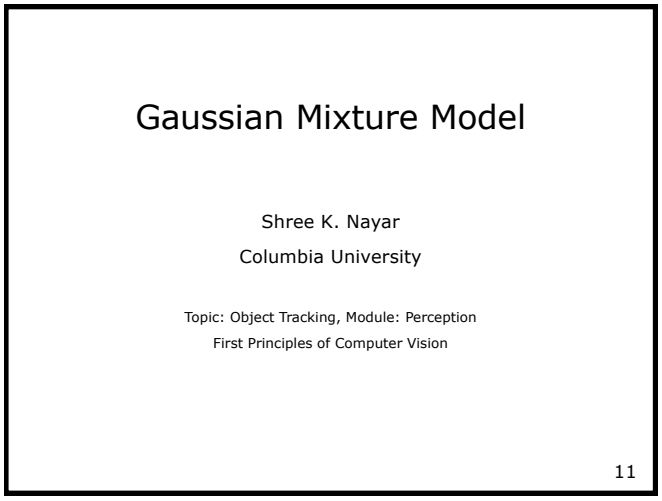



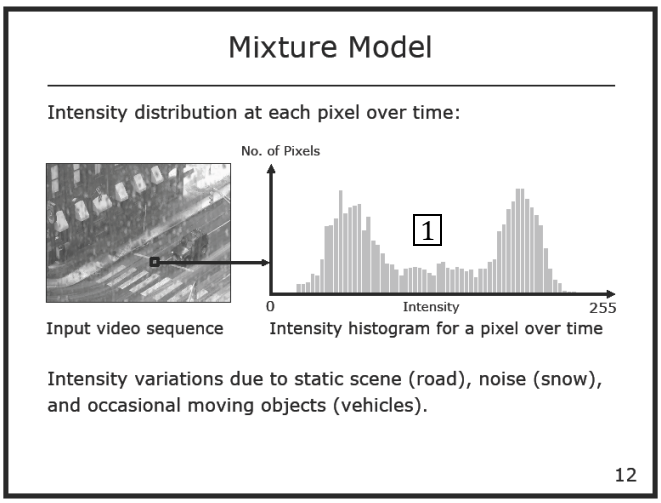



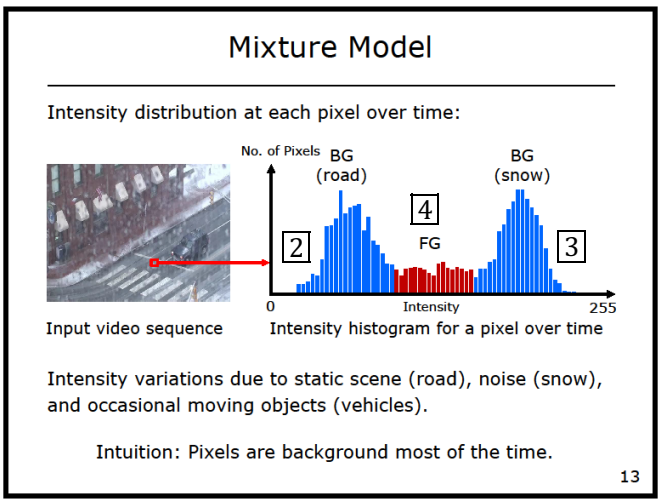

In the adaptive model shown here, we still compute a median for the background model, but the median is computed using the last K frames instead of the first K frames. Note that this model is more expensive to compute as it needs to be updated for each new frame. We see from the two examples shown here that this model does a lot better, in particular with the swaying leaves in the highway scene.

This is the simplest form of change detection that works well for simple real-world scenarios. However, it cannot deal with all the uninteresting changes we discussed in slide 5, such as weather effects like rain and snow, shadows that are attached to moving objects, or camera shake. Let us now look at how we might be able to address these problems by creating a more powerful model for the way intensity varies at each pixel.

We have seen how we can develop very simple change detection algorithms using the average or the median. We have also seen that there are limitations to these approaches. To overcome these limitations, we need a more sophisticated background model that captures the variation of intensities or colors at each pixel in the image. That brings us to the Gaussian mixture model.



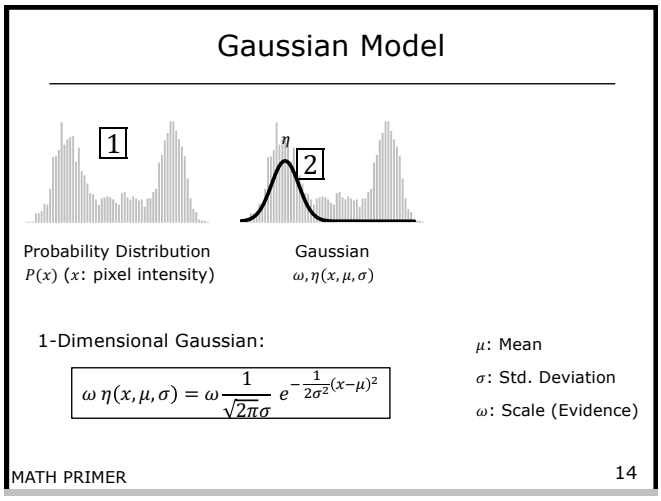



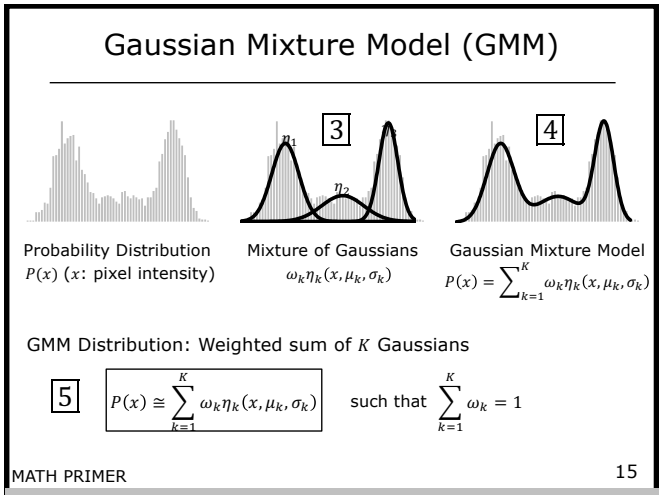




Shown here is a challenging scenario where we wish to detect moving vehicles on a rainy day. In this case, a pixel's intensity can change not only because of a vehicle passing through it but also due to raindrops moving through it. Let us consider a single pixel (small square in slide 12), and watch this pixel over a period of time. For now, we assume the video to be grayscale and we will extend our approach to color later. We can compute a histogram [1] of the intensity values detected by the pixel, where the intensity values range from 0 to 255. We want to understand the structure of this histogram. The measured pixel intensities result from three factors. The first is the static background (the road), which can only vary gradually due to illumination changes. The second is noise, which could be image noise or due to raindrops passing through the pixel. Lastly, we have the motion of a vehicle through the pixel.

Statistically speaking, we can view this histogram as a population of intensities, and each one of its bumps as subpopulations. The bump [2] (blue) corresponds to intensity variations due to the road itself. The bump [3] (also blue) is due to the rain. The passage of vehicles through the pixels produces the third

bump 4 (red). The two blue bumps are referred to as the background while the red bump is called the foreground. Note that in this scenario a pixel is exposed to the background most of the time which is why the two blue bumps are larger than the red one.




We first normalize the histogram in slide 13 so that the area under it equals one. This makes it a probability distribution 1. Next, we model each bump in the distribution as a Gaussian function. If we fit a Gaussian to the left most bump, we get the function 2. This Gaussian has a standard deviation (width) $\sigma$ and mean $\mu$. It has a third parameter, $\omega$, known as the scale or evidence associated with the Gaussian, which accounts for the fact that each bump can have its own height.

Shown in slide 15 are three Gaussians 3, one for each of the three bumps. On adding them up, we end up with the function in 4, which is close in shape to the actual distribution. Such a model that is the sum of K weighted Gaussians is called a Gaussian mixture model (GMM) 5. Each Gaussian *k* has mean $\mu_k$, standard deviation $\sigma_k$, and evidence $\omega_k$. Since our distribution has an area of one, the sum of the $\omega_k$'s should be equal to one.

The above modeling of pixel intensity variation with a GMM can also be done in higher dimensions. Suppose each pixel measures color in terms of red, green, and blue values [1]. In this case, our distribution is three-dimensional and we can model it as a sum of *k* three-dimensional Gaussians [2]. Once again, we want the sum of our evidences, $\omega_k's$, to be equal to one. In this case, the mean of each Gaussian is a vector consisting of the means for the red, green, and blue channels. Instead of $\sigma$ being a single number for each Gaussian, we now have a co-variance matrix, $\sum$ [3]. If a Gaussian is symmetric, it would have the same $\sigma$ along the red, green, and blue dimensions, and therefore the covariance matrix $\sum$ would have the same value, $\sigma^2$, along the diagonal and zero everywhere else. If a Gaussian has different $\sigma$ values along red, green, and blue, we will still have a diagonal matrix, but with different values along the diagonal. If the different color channels are correlated, $\sum$ could be a full matrix. In theory, we can use a GMM to model a function of any dimensionality. There a several freely available programs for fitting a GMM to a function. It should be noted that the use of a GMM makes sense only when we know that the function can be modeled with a small number of Gaussians.

## High Dimensional GMM

Let $P(\mathbf{X})$ be a probability distribution of a $D$-dimensional random variable $\mathbf{X} \in \mathcal{R}^D$. For example: $\mathbf{X} = [r, g, b]^T$ [1]

GMM of $P(\mathbf{X})$: Sum of $K$ $D$-dimensional Gaussians

[2] $P(\mathbf{X}) \cong \sum_{k=1}^{K} \omega_k \eta_k(\mathbf{X}, \boldsymbol{\mu}_k, \Sigma_k)$ such that $\sum_{k=1}^{K} \omega_k = 1$

where: $\eta(\mathbf{X}, \boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(\mathbf{X}-\boldsymbol{\mu})^T(\Sigma)^{-1}(\mathbf{X}-\boldsymbol{\mu})}$

Mean $\boldsymbol{\mu} = \begin{bmatrix}\mu_r \\ \mu_g \\ \mu_b\end{bmatrix}$ Covariance matrix $\Sigma = \begin{bmatrix}\sigma^2 & 0 & 0 \\ 0 & \sigma^2 & 0 \\ 0 & 0 & \sigma^2\end{bmatrix}$ (can be a full matrix) [3]

GMM can be estimated from $P(\mathbf{X})$. (MATLAB: `gmdistribution.fit`)

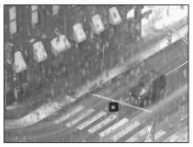
MATH PRIMER 16

Let us now return to our change detection problem. Given a GMM for intensity or color variation at a pixel over time, we want to first label each one of the Gaussians in the GMM as being either foreground or background. Shown here are the three Gaussians of a GMM that was fitted to the intensity distribution in slide 13. Our intuition is that pixels see the background for most of the time, which corresponds to Gaussians with large evidence $\omega$ and small deviation $\sigma$. Therefore, if $\omega$ divided by $\sigma$ is large for a Gaussian, we are going to call it a background Gaussian. Conversely, a Gaussian for which $\omega$ divided by $\sigma$ is small will be labeled as a foreground Gaussian. In short, a foreground Gaussian corresponds to meaningful changes at the pixel.

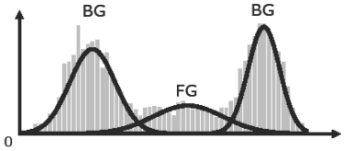
## Background Modeling with GMM

Given: A GMM for intensity/color variation at a pixel over time

Classify: Individual Gaussians as foreground/background



Input video sequence




Intensity histogram for a pixel over time

Intuition: Pixels are background most of the time. That is, Gaussians with large supporting evidence $\omega$ and small $\sigma$.

Large $\frac{\omega}{\sigma}$: Background Small $\frac{\omega}{\sigma}$: Foreground

17

Now we will present the sketch of an algorithm developed by Stauffer for change detection. The algorithm is applied to each pixel independently. For each pixel, we compute a pixel color histogram $H$ using the first $N$ frames of the video, and then normalize the histogram to get $\widehat{H}$ which makes it a probability distribution. Then, we fit a GMM to $\widehat{H}$. We will assume the GMM is made of a small number, say, 3 or 4, Gaussians. As described in slide 17, we use the ratio of $\omega$ and $\sigma$ to label each Gaussian as either background or foreground.

## Change Detection using GMM

For each pixel:

1. Compute pixel color histogram $H$ using first $N$ frames.
2. Normalize histogram: $\widehat{H} \leftarrow H/\|H\|$.
3. Model $\widehat{H}$ as mixture of $K$ (3 to 5) Gaussians.
4. For each subsequent frame:
   a. The pixel value $\mathbf{X}$ belongs to Gaussian $k$ in GMM for which $\|\mathbf{X} - \boldsymbol{\mu}_k\|$ is minimum and $\|\mathbf{X} - \boldsymbol{\mu}_k\| < 2.5\sigma_k$.
   b. If $^{\omega_k}/_{\sigma_k}$ is large then classify pixel as background. Else classify as foreground.
   c. Update histogram $H$ using new pixel intensity.
   d. If $\widehat{H}$ and $H/\|H\|$ differ a lot ($\left\|\widehat{H} - {}^{H}/_{\|H\|}\right\|$ is large), $\widehat{H} \leftarrow H/\|H\|$ and refit GMM.

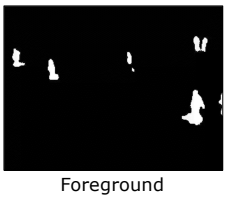
[Stauffer 1998] 18

Now we are ready to perform change detection. For each subsequent frame, we take the pixel value, $X$, and find which Gaussian in our model it is closest to it. This is done by find the mean $\mu_k$ that is closest to $X$. We also want to make sure that it is close enough to the mean to classify it as belonging to that Gaussian. This is done by using the condition that $X$ minus $\mu_k$ is less than 2.5 $\sigma_k$. If this condition is satisfied, we know from the label of the Gaussian whether $X$ belongs to the background or the foreground. The above process is applied to each new pixel color.

Since the illumination of the scene or the weather conditions could change slowly with time, we update our histogram $\widehat{H}$ after each frame of the video. We do not want to recompute GMM at millions of pixels for each frame as it would be computationally prohibitive. Therefore, for each pixel, we recompute the GMM only when the updated $\widehat{H}$ deviates significantly from the old $\widehat{H}$.

## Adaptive GMM based Change Detection


Input video


Foreground

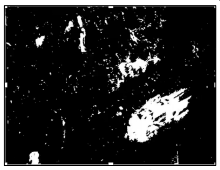

Input video

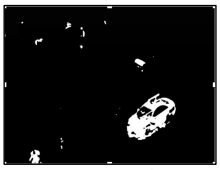

Foreground

19

## Adaptive GMM based Change Detection


Input Video


Foreground
(Moving Median Method)


Foreground
(Adaptive GMM Method)

20

Here are some results obtained from applying the above change detection algorithm to the three scenes we saw before. In all three cases, we see that the algorithm produces robust results. For the traffic scene on the left, the algorithm can be seen to be fairly resilient to the swaying of the leaves in the background. It is able to detect each moving vehicle as a separate object. For the bad weather scene on the right, we see that the GMM-based method does far better than the moving median method described in slide 10. In the video of this lecture, we can see that the moving vehicles are well detected while the effects of rain are mostly ignored.

Since we can now find meaningful changes in videos, they can be used to track an object of interest. The goal is to track the object through the entire video sequence. As the object moves through the video sequence, its magnification (distance from the camera), pose, and lighting can change. The object can even be partially occluded or entirely obstructed for short durations by other objects in the scene.

Object Tracking using Templates

Shree K. Nayar
Columbia University

Topic: Object Tracking, Module: Perception
First Principles of Computer Vision

21

Our first approach is very simple, which is to use a region of interest (perhaps, the output of a change detection method) as a template and apply template matching to track the object from frame to frame. As we will see, this approach will work for only simple tracking scenarios.

Here we are given a window (black rectangle) around one of the players and we want to track the player over time. By using template matching, we can do a fairly good job in this case because it is a simple setting since the background (turf) is fairly uniform and the players wear one of two uniforms.

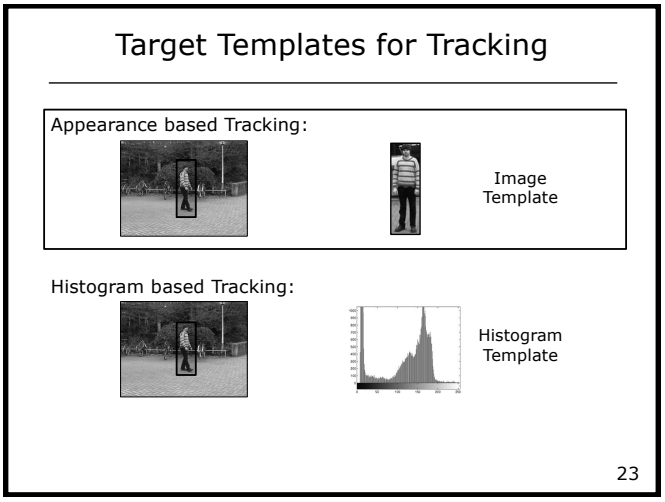
Object Tracking

Given: Location of target in initial or previous frame.

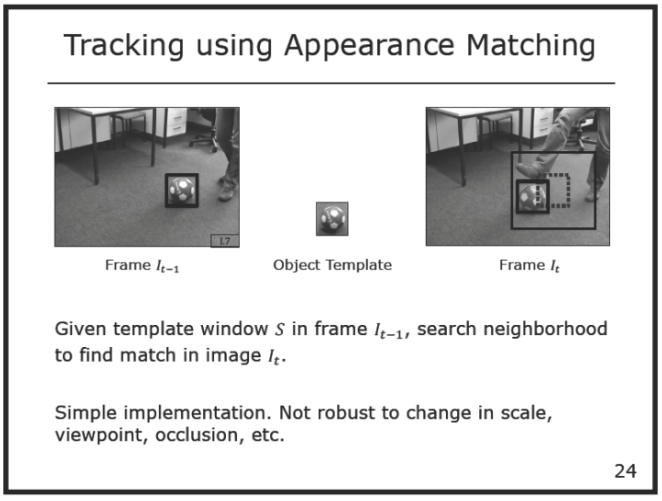
Find: Location of target in current frame.


22

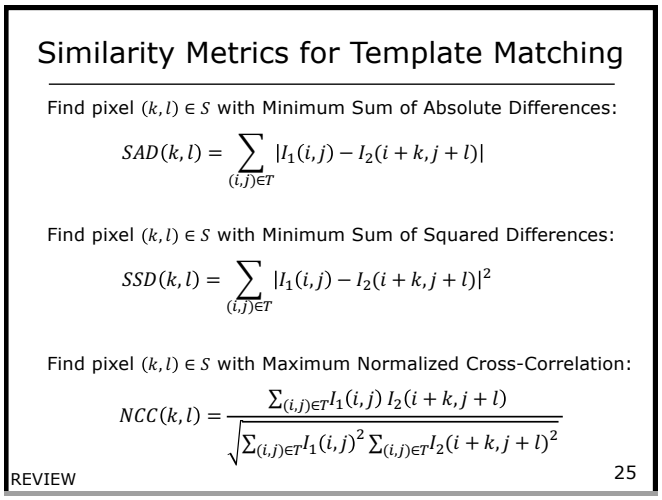
We can perform template matching in two ways. One is to use the appearance of the object in full as the template and apply template matching. In this case, we essentially have a template created from our initial image, and we are going to use it to find the object in the next frame. We then redefine the template using the current frame and apply template matching to the next frame. We refer to this as appearance-based tracking. The second approach is to use a histogram of the colors in the window as the template. In this case, we lose spatial information, but that can make matching more robust when the pose and magnification of the object vary.



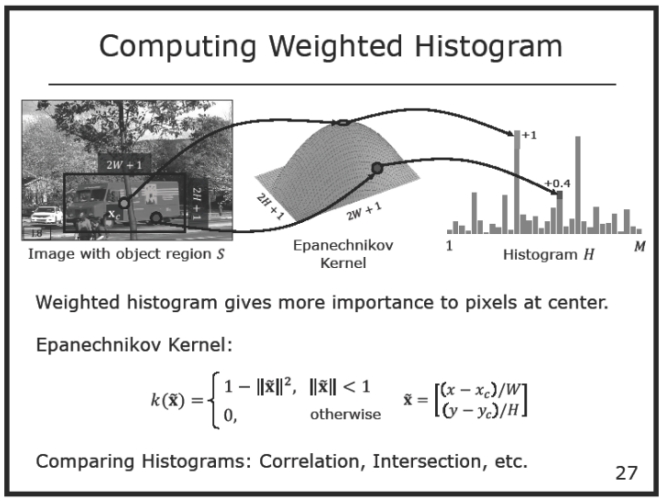

Let us first look at appearance-based tracking. In this example, a window (template) is chosen around the soccer ball in the first frame. Template matching is then applied to a large neighborhood in the next frame. Once the ball has been found in the next frame, its appearance in that frame is used as the template for matching with the next frame. Our hope is that the change in appearance of the ball between frames is small. Clearly, this approach is effective only when inter-frame changes due to magnification, pose, lighting, and occlusion are not significant.




A variety of metrics can be used to perform template matching. We can use the sum of absolute differences or the sum of squared differences. To make matching more insensitive to illumination changes, we could use the normalized correlation.

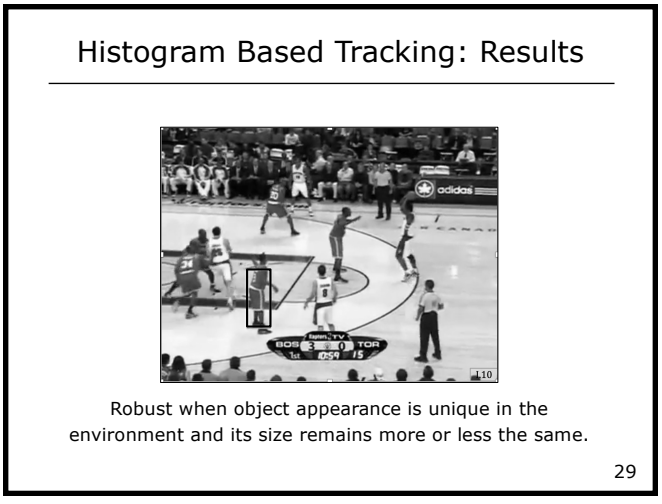


Now, let is discuss histogram-based template matching. Shown here is a window around a moving truck. Instead of directly using the window (appearance) as a template, we compute a histogram of the window to represent our template. It could be a brightness histogram or a color histogram.

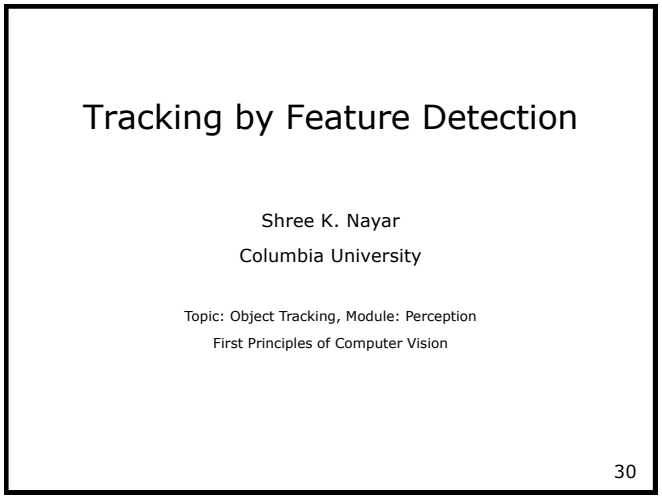



Typically, the more reliable points within a window tend to be closer to the center of the window. As we approach the edge of the window, there is a higher likelihood of a pixel belonging to the background of the object of interest. In the case of the truck shown here, the background would be tree leaves or the road. To mitigate this problem, we can use a weighted histogram where the contribution of each pixel to the histogram depends on its location within the window. One such weighted function is called the Epanechnikov kernel, where the weight drops with the distance of the pixel from the center of the window. This weighted histogram is used as a template to find the best matching window in the next frame. Once again, the matching can be done using any of the metrics described in slide 25. Histogram-based template matching tends to be more resilient than the appearance-based template matching to changes in object pose, scale, and even occlusion.

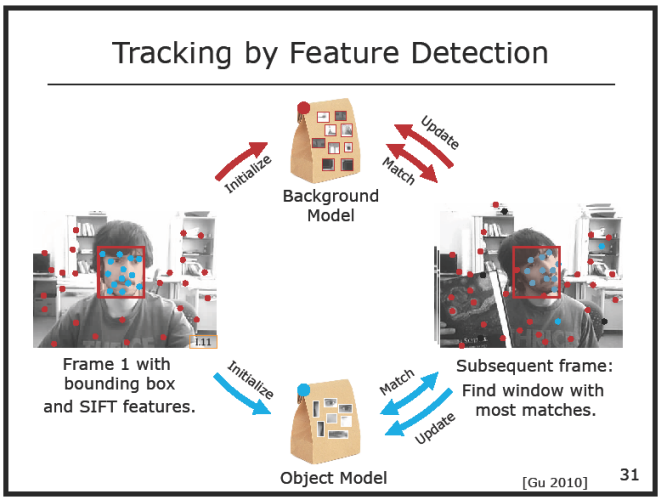
Shown here is an example of histogram-based template matching. Our region of interest is one of the players. As seen in the online lecture video, the method works well in this case, except when the player is crossed by another player with the same uniform. When this happens, sometimes, the window gets attached to the crossing player.



We now present a robust tracking algorithm that is based on feature detection. This algorithm is developed by Gu.




Shown here on the left is the initial image, within which a region of interest (red window around the face) has been specified. This window could be the result of a face detection algorithm or manual selection. The goal is to track the face over an entire video. First, the SIFT detector is applied to the image. The detected features in the target window are shown in blue and the ones outside the window in red. Remember that each SIFT feature has a location and a descriptor. The collection of SIFT features inside the window represent the object model, and all the features outside the window represent a background model. That concludes the initialization process, and now we are ready to do tracking.

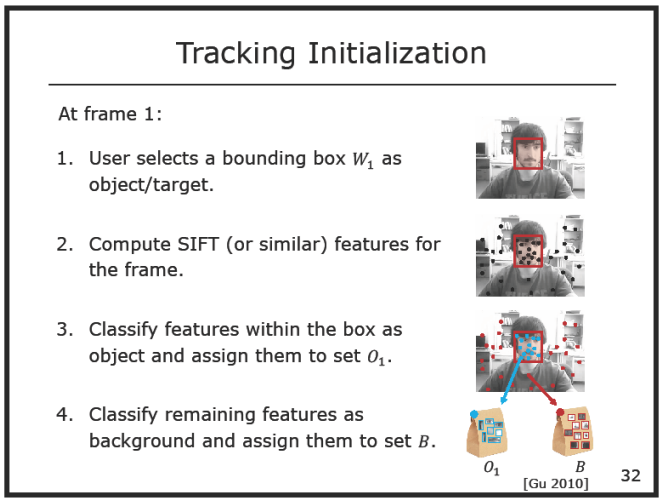


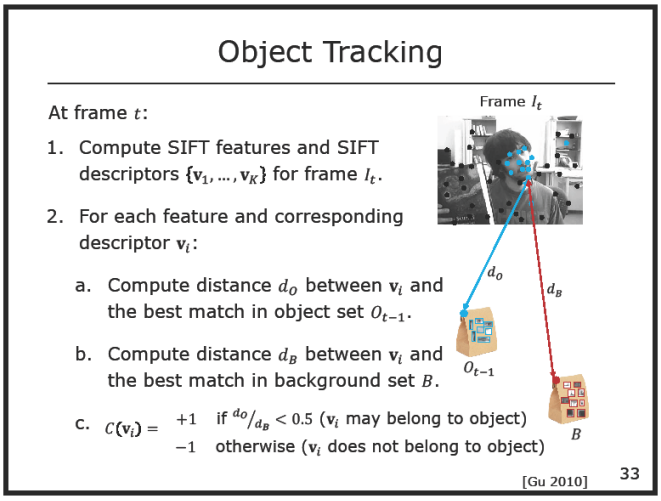

We want to find our object of interest, the face, in the new image shown on the right. First, we apply feature detection to this image to find all its SIFT features. For each of these features, we are going to find the closest match in the object model and in the background model. All the blue points match well with the object model, and the red points with the background model. There are also a few black dots, which are not assigned to either model as they did not match well with either.

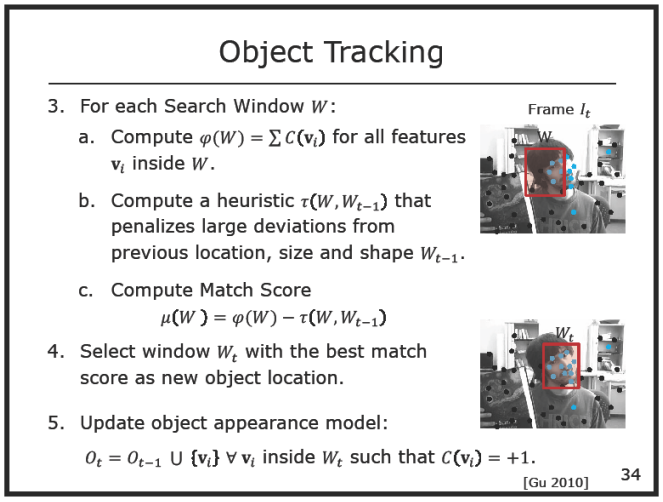
We now take the window in the left image and place it on the right image. Our goal is to apply shifts and distortions to this window to arrive at a window that maximizes the number of object model features (blue dots). To ensure that new window does not deviate significantly in terms of location, size, and shape with respect to the old one, we use a cost function related to the shift and distortion of the new

window. Once the new window has been found, since the object and the background may have changed in appearance, we use the features within the window to update the object model and the features outside it to update the background model. This process is repeated for each new frame of the video.

We now describe the algorithm in detail. Let us start with the initialization process, which involves building the object and background models. In the first frame, we define the object of interest as a window $W$ (red), and then we apply SIFT to the image. Then, we use all the features inside the window to create the object model, and the features outside the window to create our background model. This concludes the initialization process and we are ready to perform tracking.




Here is how tracking works for frame $t$. First, we find SIFT features in the image. For each feature $v_i$, we find the distance $d_o$ between it and the best matching feature in the object model, and the distance $d_b$ between it and the closest matching feature in the background model. If $d_o$ is much smaller than $d_b$ ($d_o$ divided by $d_b$ is less than 0.5), then we give that feature a confidence, $C(v_i) = +1$, meaning it belongs to the object. If not, we give it a confidence of -1, as we believe it does not belong to the object. Every feature now has a $C$ value associated with it. In slide 33, each blue dot has a confidence of 1, and each red one a confidence of -1.

Now, we need to figure out where the window $W$ has moved to in the image. In other words, we search for the new location and shape of the window $W$. For each window in the search, we are going to add up all the $C$ values inside the window to get a total confidence $\varphi$. We also apply a penalty, $\tau$, for the change in the window shape and position with respect to the window in the previous frame. The match score $\mu$ for the window is then simply $\varphi$ minus $\tau$. The window $W_t$ that maximizes the match score $\mu$ is the new position of the object.

The last step is to update the object and background models. Since the appearance of the object and background may change between frames, the features in the new image can differ slightly from those in the previous one. We use the matched features inside the window $W_t$ to update the object model, and the ones outside to update the background model.

The effectiveness of the above algorithm is best seen in the online video lecture. In the two examples shown here, the method demonstrates high resilience to changes in scale and orientation. On the left, the lighting of the scene changes quite a bit. On the right, the person spins around and yet is successfully tracked.

Tracking Results: Scale and Orientation

Resilient to changes in scale and orientation

35

The algorithm is also quite robust to occlusion. In the two examples shown here, the faces are partially occluded by the hat and the magazine, respectively. The tracking remains robust because the features on these new objects have not been seen before and therefore are not added to the object model. These examples demonstrate the advantage of using a feature-based approach instead of using a template-based approach for object tracking.

Tracking Results: Occlusion


Resilient to occlusion

36

Let us look at a few real-world applications of object tracking. This system developed by Benfold can track people in the wild. Most of the people shown here are detected and tracked reliably. A rectangle is displayed around each person, and a smaller rectangle within it indicates the location of the head of the person. The smaller rectangle can be used to perform face recognition.

## Tracking Applications

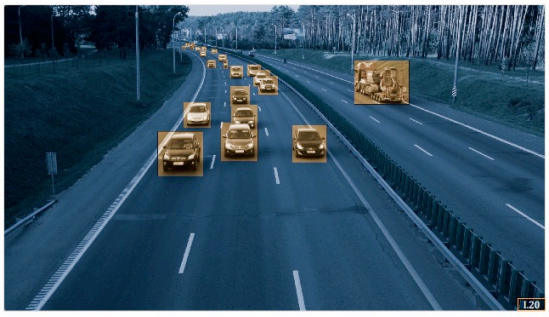
Tracking People in the Wild

[Benfold 2011]

37

A popular application of tracking is in traffic monitoring. Since we know the location of the highway with respect to the camera, we can use the estimated speed of each vehicle in the image to determine its speed on the highway. This information can be used to detect moving violations as well as gauge traffic conditions.

## Tracking Applications



Traffic Monitoring

38

There are more recent applications of tracking and motion analysis. Shown here is the use of motion to determine customer behavior in a store. The red regions correspond to ones where customers have spent the most time. This information is useful as it informs the shop owner about which products are attracting more attention. Such data can be used for effective product placement and inventory management.

## Tracking Applications

Customer Behavior for In-Store Analytics

39

## References and Credits

Shree K. Nayar

Columbia University

Topic: Object Tracking, Module: Perception

First Principles of Computer Vision

40

## References: Papers

[Benfold 2011] B. Benfold and I. Reid. "Stable Multi-Target Tracking in Real-Time Surveillance Video." CVPR 2011.

[Comaniciu 2000] D. Comaniciu, V. Ramesh and P. Meer. "Real-time tracking of non-rigid objects using mean shift ". CVPR 2000.

[Gu 2010] S. Gu, Y. Zheng and C. Tomasi. "Efficient Visual Object Tracking with Online Nearest Neighbor Classifier" . ACCV 2010.

[Stauffer 1998] C. Stauffer and W. E. L. Grimson. "Adaptive background mixture models for real-time tracking" . CVPR 1998.

41

## Image Credits

I.1 www.changedetection.net N. Goyette et al., changedetection.net: A new change detection benchmark dataset, CVPR 2012.

I.2 www.changedetection.net N. Goyette et al., changedetection.net: A new change detection benchmark dataset, CVPR 2012.

I.3 www.changedetection.net N. Goyette et al., changedetection.net: A new change detection benchmark dataset, CVPR 2012.

I.4 www.changedetection.net N. Goyette et al., changedetection.net: A new change detection benchmark dataset, CVPR 2012.

I.5 www.changedetection.net N. Goyette et al., changedetection.net: A new change detection benchmark dataset, CVPR 2012.

I.8 www.changedetection.net N. Goyette et al., changedetection.net: A new change detection benchmark dataset, CVPR 2012.

I.10 https://www.epfl.ch/labs/cvlab/research/research-surv/research-balltracking/ Pascal Fua.

I.11 Carlo Tomasi. Used with permission.

I.12 Carlo Tomasi. Used with permission.

I.13 Carlo Tomasi. Used with permission.

42

## Image Credits

I.14 Carlo Tomasi. Used with permission.

I.15 Carlo Tomasi. Used with permission.

I.17 https://abcnews.go.com/Business/stores-snooping-holiday-shoppers/story?id=21153815 . ABC News. Prism Skylabs.

I.19 Ian Reid. Used with permission.

I.20 Purchased from iStock by Getty Images.

43

**Acknowledgements**: Thanks to Roshan Kenia, Tracy Cui and Nikhil Nanda for their help with transcription, editing and proofreading.

**References**

[Benfold 2011] B. Benfold and I. Reid, “Stable Multi-Target Tracking in Real-Time Surveillance Video,” CVPR 2011.

[Comaniciu 2000] D. Comaniciu, V. Ramesh and P. Meer, "Real-time tracking of non-rigid objects using mean shift," CVPR 2000.

[Gu 2010] S. Gu, Y. Zheng and C. Tomasi, "Efficient Visual Object Tracking with Online Nearest Neighbor Classifier," ACCV 2010.

[Stauffer 1998] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," CVPR 1998.

[Nayar 2022F] Image Processing II, Nayar, S. K., Monograph FPCV-1-5, First Principles of Computer Vision, Columbia University, New York, March 2022.

[Nayar 2022G] Edge Detection, Nayar, S. K., Monograph FPCV-2-1, First Principles of Computer Vision, Columbia University, New York, May 2022.

[Nayar 2022H] Boundary Detection, Nayar, S. K., Monograph FPCV-2-2, First Principles of Computer Vision, Columbia University, New York, June 2022.

[Nayar 2022I] SIFT Detector, Nayar, S. K., Monograph FPCV-2-3, First Principles of Computer Vision, Columbia University, New York, August 2022.

[Nayar 2025B] Face Detection, Nayar, S. K., Monograph FPCV-2-5, First Principles of Computer Vision, Columbia University, New York, February 2025.

[Nayar 2025J] Optical Flow, Nayar, S. K., Monograph FPCV-4-3, First Principles of Computer Vision, Columbia University, New York, April 2025.