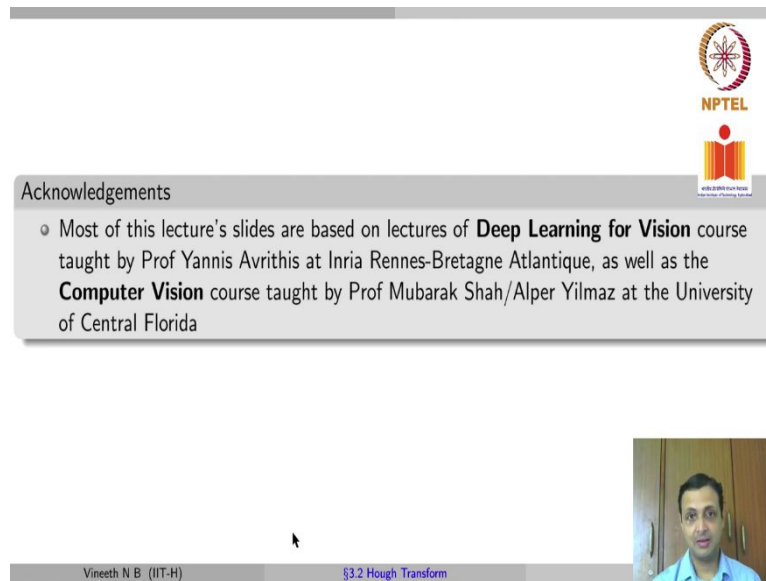


Deep Learning for Computer Vision
Professor Vineeth N Balasubramanian
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Lecture 16
Hough Transform

The next topic that we are going to talk about in trying to match shapes or any other templates on an image is the method known as Hough Transform.

(Refer Slide Time: 00:31)



The screenshot shows a slide with the following content:

- Acknowledgements**
- Most of this lecture's slides are based on lectures of **Deep Learning for Vision** course taught by Prof Yannis Avrithis at Inria Rennes-Bretagne Atlantique, as well as the **Computer Vision** course taught by Prof Mubarak Shah/Alper Yilmaz at the University of Central Florida

Logos for NPTEL and IIT Hyderabad are visible in the top right corner. A small video inset of the professor is in the bottom right corner. The footer contains the text: Vineeth N B (IIT-H) §3.2 Hough Transform

Once again, these lecture slides are also based on the excellent lectures of Professor Yannis, at Inria Ren, as well as the lectures of professor Mubarak Shah at the university of Central Florida.

(Refer Slide Time: 00:48)

Line Fitting:

already seen a couple of line fitting algorithms: Least squares fit and RANSAC
 how do they perform when multiple lines are present?

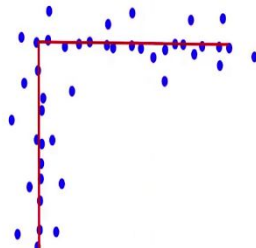


Figure 1: Example line configuration in an image.

Source: Alper Yilmaz



© Hough, Method and means for recognizing complex patterns, U.S. Patent No. 3,069,654, Dec 1962

Line equation in Cartesian co-ordinates is:
 $y = mx + c$ m is slope, c is y-intercept

Rearranging it slightly, we get:
 $c = (-x)m + y$ which for a specific point (x_i, y_i) becomes $c = (-x_i)m + y_i$

This can be thought of as the equation of line in parameter space; i.e. in the (m, c) coordinate system with slope $-x_i$ and c-intercept y_i

Each point in parameter space is a model

We have already seen a couple of line fitting methods. We saw least squares fit and then we saw RANSAC. The question that you are going to ask now is, what do you do if there are multiple lines placed in a particular orientation or placed in a particular configuration in an image? It could be multiple lines, it could be a polygon, it could be a circle, then how do you deal with fitting a shape onto the set of data points that you have? And that is what we are going to talk about now, when we Hough Transform.

(Refer Slide Time: 1:30)

Line Fitting: Hough Transform

- Hough, Method and means for recognizing complex patterns, U.S. Patent No. 3,069,654, Dec 1962
- Line equation in Cartesian co-ordinates is:
 $y = mx + c$ m is slope, c is y-intercept
- Rearranging it slightly, we get:
 $c = (-x)m + y$ which for a specific point (x_i, y_i) becomes $c = (-x_i)m + y_i$
- This can be thought of as the equation of line in parameter space; i.e. in the (m, c) coordinate system with slope $-x_i$ and c-intercept y_i
- Each point in parameter space is a model

Source: Alper Yilmaz, Mubarak Shah, Fall 2011 UCF



© Hough, Method and means for recognizing complex patterns, U.S. Patent No. 3,069,654, Dec 1962

Line equation in Cartesian co-ordinates is:
 $y = mx + c$ m is slope, c is y-intercept

Rearranging it slightly, we get:
 $c = (-x)m + y$ which for a specific point (x_i, y_i) becomes $c = (-x_i)m + y_i$

This can be thought of as the equation of line in parameter space; i.e. in the (m, c) coordinate system with slope $-x_i$ and c-intercept y_i

Each point in parameter space is a model

This method dates back to the early 60s, by Hough, who filed the U.S. patent for this, but since then, there have been many efforts that have translated this into what we see today. In

the early 70s, the Hough Transform was used for detecting lines and curves. Then in the early 80s, there was a method that came that was called a generalized transformation and we will see all of these over the next few slides.

Let us start with the simple equation of a line and Cartesian coordinates, which we all know is $y = mx + c$, where we say that m is the slope and c is the y intercept. You could just play with the terms of the equation a little bit, and then write this as $c = -mx + y$.



Rather we now live in an m, c space where x is c intercept, y define the line in that space. So, were originally in the x, y space, where m define the slope and c define the y intercept. We could just rearrange these terms slightly to now go to an m, c space where $-x$ becomes the slope and y becomes the c intercept. In this kind of a context, every point in m, c space becomes the equation of a line in the x, y space, remember for every choice of m and c , you will get the equation of a line as we see here. And every point in the x, y space similarly would become the equation of a line in the m, c space. There is a dual correspondence between these two spaces. So what do we do with this?

(Refer Slide Time: 3:35)


Line Fitting: Hough Transform

- N samples needed to fit a model (2 points to fit a line)
- But even one sample brings some information
- In the space of all possible models, vote for ones that satisfy a given sample
- Collect votes for all samples, and seek for consensus

Source: Yannis Avrithis, Deep Learning for Vision, Spring 2019 SIF

Vineeth N B (IIT-H)
§3.2 Hough Transform



We know that to fit any model we need a certain number of samples. For example, to fit a line, we need two samples. If it was any other model, if you were fitting a square or a circle, we need a different number of points. To fit a line, you may need two points. Let us start with that example. But even if you had just one point instead of two points, you know that it could

belong to a certain family of lines. It gives you some partial information about the line equation itself.

So, what we can ask every point in your x, y space to do is to vote for certain configurations of m and c , in case you are talking about a line, if aligned. Every point can vote for what configuration of m and c would have resulted in this point being at that particular location. And then you collect words for all such points and try to seek consensus and wherever the majority vote, you are going to say that that is the equation of the line that I am looking for. Let us elaborate on this and go through slowly.

(Refer Slide Time: 4:50)

Hough Transform: Polar Parametrization

- The Cartesian formulation is problematic for vertical lines. Why?
 - The slope is unbounded for vertical lines.
- Consider a polar parametrization of the line:
 - $\rho = x \cos \theta + y \sin \theta$
 - ρ is distance of line from origin and θ is angle made by normal to x-axis
 - For given line, $\rho \geq 0$ and $0 \leq \theta \leq 360^\circ$ are bounded

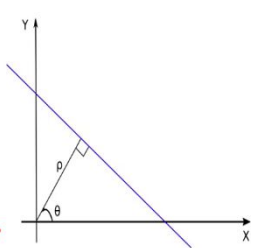




Figure 2: The ρ, θ parametrization.



Source: Alper Yilmaz, Mubarak Shah, Fall 2011 UCF
Vineeth N B (IT-H) §3.2 Hough Transform 5



Before we go forward, we spoke about the Cartesian coordinates on the previous slide. Unfortunately, the Cartesian formulation can be problematic for vertical lines. You may say why? When you have a vertical line, your slope is unbounded, it becomes difficult to represent such a line in your m, c space because m has to be infinity.

So, in such a scenario, it may be wiser to use a slightly different parameterization, the polar parameterization, which is been written in terms of ρ and θ , where ρ is the distance of the line from origin and θ is the angle made by the normal to the X axis. You can also write $\rho = x \cos \theta + y \sin \theta$. This is a standard polar parameterization that is used going from Cartesian coordinates to polar coordinates.


We know now that in this space, ρ is at least lower bounded by 0 and θ lies between 0 and 360. We know that these quantities are bounded, which makes it slightly easier to work with.

(Refer Slide Time: 6:17)

Hough Transform: Polar Parametrization

- A point (x_1, y_1) 'votes' for many points in parameter space \rightarrow **Hough Voting**
- Each line through a point (x_1, y_1) is a vote for a point in parameter space which satisfies $\rho = x_1 \cos \theta + y_1 \sin \theta$

voting in parameter space



Source: Yannis Avrithis, *Deep Learning for Vision, Spring 2019 SIF*
Vineeth N B (IIT-H) 6

So, every point in your original space, in your Cartesian space (x, y) votes for many points in your parameter space. So, if you had a certain line, certain point x_1 in your original Cartesian space. For the moment, we are going to talk about this method generally, in terms of a Cartesian space and fitting a line or fitting a circle and so on and so forth.

A little later in this lecture, we will talk about how you use this with images. If you already have a sense of what is coming, we are trying to find out how do you use Hough Transform to match lines or circles or any other shape in an image. That is the goal of this lecture or the method discussed in this lecture.

But before we go to an image, we are just going to talk generally about Cartesian coordinates. So, if you have a point x_1 in the (x, y) space, you know that there are a family of lines that can pass through x_1 , and each of those lines has a correspondent m, c it is going to have a slope and the y intercept, and all of them will result in corresponding points in the m, c space or the ρ, θ space. If you go to the log space, you will have a slightly different variation. They will correspond to a point here in the r theta space.

So, for every point x, y can vote for many points in your parameter space, which that particular x_1 point could have been lying on. So, each line to through this point, (x_1, y_1) is a



vote for a point in a parameter space that satisfies $\rho = x \cos \theta + y \sin \theta$. Let us see a few examples.

(Refer Slide Time: 8:12)


Hough Transform: Polar Parametrization

- A point (x_i, y_i) 'votes' for many points in parameter space \rightarrow **Hough Voting**
- Each line through a point (x_1, y_1) is a vote for a point in parameter space which satisfies $\rho = x_1 \cos \theta + y_1 \sin \theta$

voting in parameter space

Source: Yannis Avrithis, Deep Learning for Vision, Spring 2019 SIF



Vineeth N B (IIT-H)
§3.2 Hough Transform
6 /



Here is another line that passes through x_1 and that would work for a different point in the parameter space, which is here.

(Refer Slide Time: 8:23)


Hough Transform: Polar Parametrization

- A point (x_i, y_i) 'votes' for many points in parameter space \rightarrow **Hough Voting**
- Each line through a point (x_1, y_1) is a vote for a point in parameter space which satisfies $\rho = x_1 \cos \theta + y_1 \sin \theta$

voting in parameter space

Source: Yannis Avrithis, Deep Learning for Vision, Spring 2019 SIF



Vineeth N B (IIT-H)
§3.2 Hough Transform
6 /

Here is another line that passes through x_1 , that would vote for a different point in the parameter space. Remember the parameter space now is defined in a polar parameterization.

(Refer Slide Time: 8:36)

Hough Transform: Polar Parametrization

- Each line through a point (x_2, y_2) is a vote for a point in parameter space which satisfies $\rho = x_2 \cos \theta + y_2 \sin \theta$

voting in parameter space

Source: Yannis Avrithis, *Deep Learning for Vision, Spring 2019 SIF*
Vineeth N B (IT-H) 3.2 Hough Transform 7

And you keep repeating this for several lines that could have passed through x_1 and all of them get votes in the parameter space.

(Refer Slide Time: 8:46)

Hough Transform: Polar Parametrization

- Each line through a point (x_2, y_2) is a vote for a point in parameter space which satisfies $\rho = x_2 \cos \theta + y_2 \sin \theta$

voting in parameter space

Source: Yannis Avrithis, *Deep Learning for Vision, Spring 2019 SIF*
Vineeth N B (IT-H) 3.2 Hough Transform 7



Similarly, you could have another point x_2 in your original space through which you would have another set of lines that can pass in your original Cartesian space and each of those lines may vote for them for a different parameterization in your parameter space. You can see that again, here, these are little examples for several lines that pass through x_2 , you get several votes in the parameter space.

(Refer Slide Time: 9:17)

Hough Voting

```
1: procedure HOUGH VOTING( $X, \Theta$ )
2:    $X$ : data    $\Theta$ : quantized parameter  $\theta_{min}, \dots, \theta_{max}$ 
3:    $A$ : accumulator array, initially zero
4:   for  $(x, y) \in X$  do
5:     for  $\theta \in \Theta$  do
6:        $\rho = x \cos \theta + y \sin \theta$ 
7:       ▷ for each set of model parameters consistent with a sample, increment  $A$ 
8:        $A[\theta, \rho] = A[\theta, \rho] + 1$ 
9:     end for
10:  end for
11:  Non-maximum Suppression: detect local maxima in  $A$ 
12: end procedure
```

Source: Yannis Avrithis, *Deep Learning for Vision, Spring 2019 SIF*
Vineeth N B (IIT-H) 3.2 Hough Transform



Let us try to recount this through an algorithm. So, you have your data X , you have a set of quantized parameters, theta min to theta max. For example, if you take polar parameterization, you may not want to vote for every angle between 0 and 360, you may want to divide 0 to 360 into 10-degree bins and have only 36 possible theta values. That is what we mean by a quantized parameter here.

You also initialize an accumulator array, which is simply a frequency count. It is simply a frequency count of vote. You can look at it that way. So, for every point (x, y) in your Cartesian space, we try to see for theta belonging to theta, you have 10 possibility thetas. If you write out $\rho = x \cos \theta + y \sin \theta$. We try to see for each set of model parameters consistent with the sample, so which rho and which theta would align with this particular X, Y that we have taken. You increment A for that theta and rho.

So, you imagine the accumulator being a 2D matrix defined by theta values on one axis and rho values on the other axis, which could also be quantized, you could quantize distance into, distance of say a certain number of units. And whichever rho and theta we know corresponds to this X and Y that we have, you go to that rho and theta and in each of these cases, you could check for consistency and increment that cell in the accumulator matrix by 1. And you keep doing this for every X, Y point that is given to you. Those are your set of points that are given to you and you keep accumulating and at the end, you do a non-maximum suppression in A .

So, if you have many different bins that have a high vote, you do a non-maximum local, local non-maximum suppression to see which bin has the really highest votes and you define that theta and rho to be corresponding to the model that corresponds to your, that directly has a bearing on the data in your X matrix. This gives you a way of estimating the theta and rho or in Cartesian coordinates, finding the equation of the line, corresponding to this set of points that are given to you.

As you can see this is very similar to a least square fit, but doing it in a slightly different way. We are still trying to fit a line to a set of points that is given to us, but we are doing it using the Hough voting approach.

(Refer Slide Time: 11:54)

Line Detection

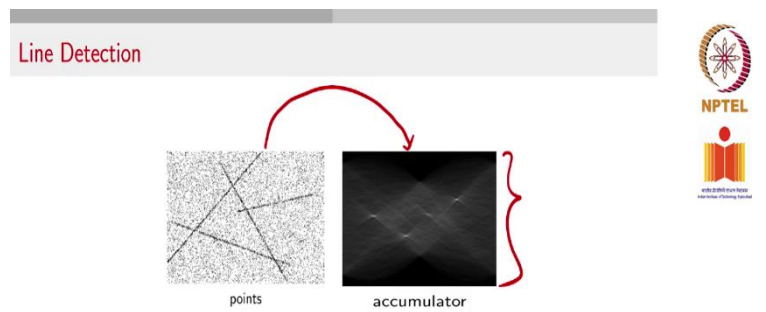
points

Source: Yannis Avrithis, *Deep Learning for Vision*, Spring 2019 SIF

Vineeth N B (IT-H) 3.2 Hough Transform 9

Now, as I promised, here is an example from an image perspective. Let us say we have an image and the image has several lines and we want to find the equations of those lines. That could be several applications where this may be required. Let us say for instance, that you are trying to take images of a computer chip and trying to find out how those lines are aligned on a computer chip and so on and so forth. You want to find the equations of those lines.

(Refer Slide Time: 12:24)

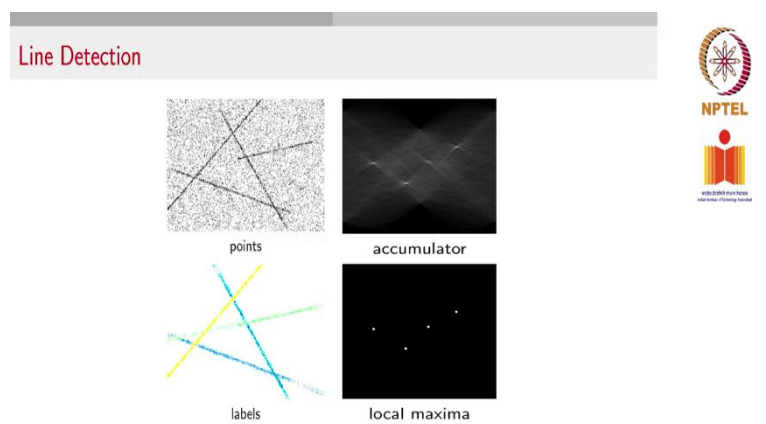


Source: Yannis Avrithis, *Deep Learning for Vision, Spring 2019 SIF*

Vineeth N B (IIT-H) §3.2 Hough Transform 9

So, using the same approach that we just talked about on the earlier slide, you assume a certain parameterization, you ask for each point to vote to which parameterization may have generated in that particular point, and you end up getting votes over your accumulator 2D matrix, let us just say a ρ and θ . So, you can see here that you have a bunch of votes that is spread across many different accumulator bins, we said that accumulated as a matrix that is rho and theta.

(Refer Slide Time: 12:54)



Source: Yannis Avrithis, *Deep Learning for Vision, Spring 2019 SIF*

Vineeth N B (IIT-H) §3.2 Hough Transform 9

Now, you are doing non-maximum suppression to maintain only the local minima in your accumulator. And once you do this local maxima, you find that there are four maxima here

which corresponds to four values or four parameterizations of lines, which are the four lines that you actually have in the original image. And those are the points that were voting for each of those accumulator bins. This way you can find out the equations of those lines, as well as which points in the original image correspond to those equations, to be able to extract that shape in that image. So, you can actually find equations of lines in any image given to you using this approach.

(Refer Slide Time: 13:44)

Hough Transform: Finding Circles

- Circle fitting similar to line fitting
 - $(x - x_0)^2 + (y - y_0)^2 - r^2 = 0$
- What are the dimensions of accumulator A for circle fitting?



NPTEL
National Programme
on Technology Enhanced Learning

Vineeth N B (IIT-H) §3.2 Hough Transform 10




Let us make this a bit harder now. Let us say we want to now find circles in an image. We did talk about block detection earlier, but now we want to exactly find the circle with its radius and center and be able to parameterize it perfectly. So, circle fitting is very similar to line fitting, that is going to be the fitting that you are looking for $(x - x_0)^2 + (y - y_0)^2 - r^2 = 0$. What should be the dimensions of the accumulator? For line, the accumulator was a 2D matrix. What would be the dimension of the accumulator for circle fitting?

(Refer Slide Time: 14:29)


Hough Transform: Finding Circles

- Circle fitting similar to line fitting
 - $(x - x_0)^2 + (y - y_0)^2 - r^2 = 0$
- What are the dimensions of accumulator A for circle fitting?
 - 3D accumulator with dimensions x_0, y_0, r
- Fix one of the parameters (generally radius is fixed) and loop for the rest
- Increment accumulator A
- Find local maxima in A



Source: Ioannis Gkioulekas, 16-385 Computer Vision, Spring 2020, CMU

Vineth N B. (IT-H) §3.2 Hough Transform



It would be a 3D three-dimensional accumulator, where you have x_0, y_0, r , which corresponds to the center of the circle and r which corresponds to the radius of the circle. Those are the three votes that each point is going to be voting for. Since we have three possibilities here, one option is to fix one of the parameters and generally you fix the radius and then look for the rest, ask every point to vote what is the center and then you can keep going around and going in a round Robin manner to ensure that every point works for all three parameters at the end.

So, you keep incrementing the accumulator for every vote that a point makes to each of these values and at the end, you do a local maxima in a, and this way you can actually estimate the parametrization of the circle that you are looking for. Here is a visual illustration, where given a set of coins in an image, you can actually now find out where those coins lie and what is the radius of that coin by first extracting edges.

So, in this, you have to first extract edges, and then be able to use those edge information to be able to find the exact parameterizations on the surface. So when I say parameterizations, I mean define the centers and radii of each of those circles that you have,

(Refer Slide Time: 15:56)

Generalized Hough Transform

- Used for shapes with no analytical expression
- Involves a training phase where R-table is computed
- Given object of interest, compute R-table as follows:
 - Compute centroid (x_c, y_c) .
 - For each edge point (x_i, y_i) , compute distance to centroid r_i and find edge orientation ϕ_i .
 - Construct a table of angles and r -values

ϕ_1	$r_1, r_2, r_3 \dots$
ϕ_2	$r_{14}, r_{21}, r_{23} \dots$
ϕ_3	$r_{41}, r_{42}, r_{33} \dots$
ϕ_4	$r_{10}, r_{12}, r_{13} \dots$

Source: Alper Yilmaz, Mubarak Shah, Fall 2011 UCF

Vineth N B. (IT-H) §3.2 Hough Transform 1

You need not stop with just lines or circles, you can obviously do this for any other shape. As long as you can parameterize any shape in an analytical manner, you can give a set of equations to define a shape. All you have to do is take a point in your original image and ask you to vote for what would be the parameters of a polygon that that point belongs to or lies on. And then you just count which parameterization got the highest votes amongst all the possibilities in your parameter space, and simply define that to be the polygon or any shape that you are looking for.

But what, if you are looking for the shape that has no analytical description. For example, on this slide, you see this irregular shape, which has no analytical distribution, which has no analytical definition. How do you estimate, how do you find such shapes in an image? We do assume that the shape has given to us.

So, you do know that this is the shape that you are looking for in a given image and it could be of different sizes, could be rotated in different ways, it could be scaled, all of those transformations are possible. But we want to find where in the image it lies. The only thing we know is that the shape in this particular case may not have an analytical definition. You cannot write it as an equation. What do you do in this particular case?

You define a reference point in the shape. Like you could define, for example, an x naught y naught as a reference point in that shape and now every edge point of this shape can be defined with respect to that reference point. So every point here (x, y) has a certain length

from your reference point (x_0, y_0) and a certain angle with respect to the reference point (x_0, y_0) . As an example, (x_0, y_0) could be the centroid of that object that you are looking for. But it need not be the centroid, it could be any reference point.

So, once you have a centroid or any other reference point, for each edge point that you have, so remember again, that given a new image, you have to first do an edge detection to get the edge points. Then for each edge point you compute what is the distance to the centroid. You would first do this for the template object that you have before you run it on any new image. This is when you know that this is the shape you are looking for, you define a reference point for each edge point in this particular shape, you compute a distance to the centroid, which is r_i and an each orientation ϕ_i .

And then you create something called an r table. And what the r table says is that, for a given edge orientation ϕ_1 , what are all the possibilities? What are all the points that could have a certain radii with the same orientation, with respect to the reference point? Similarly, for another point with another orientation ϕ_2 , let us say ϕ_2 is this particular edge point. That has a certain radius with respect to the reference point. Similarly, there could be another point with the same angle, which would have a certain radius r_{21} , there could be a third point with the same angle, same image orientation and that could have a radius r_{23} and you create an r table, which connects these orientations and these distances to that centroid of the reference point. So once you have built this r table, what do we do?



(Refer Slide Time: 19:31)


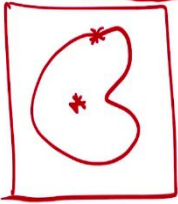
Generalized Hough Transform


```

1: procedure GENERALIZED HOUGH TRANSFORM( $X, R, A[x_c, y_c]$ )
2:    $X$ : data   $R$ : R-table   $A[x_c, y_c]$ : Accumulator array with quantization  $x_{cmin}, \dots, x_{cmax}$ 
   and  $y_{cmin}, \dots, y_{cmax}$ 
3:    $A$ : accumulator array, initially zero
4:   for  $(x, y) \in X$  do
5:     for each  $(r_i, \phi_i) \in R$  do
6:        $x_c = x + r_i \cos \phi_i$ 
7:        $y_c = y + r_i \sin \phi_i$ 
8:        $A[x_c, y_c] = A[x_c, y_c] + 1$ 
9:     end for
10:  end for
11:  Non-maximum Suppression: detect local maxima in  $A$ 
12: end procedure

```



Vineth N B (IIT-H)
§3.2 Hough Transform

Here is the algorithm of what is known as Generalized Hough Transform. So, you have your data. You are given your r table, which you have already built from the shape that was given to you. Remember once again, that the shape is defined for you. Not defined is given to you, it is not well defined mathematically is given to you. Using that shape you construct something called the r table and you also construct an accumulator array for how many of the dimensions you have to estimate.

So, if that reference point has two coordinates, you have only two coordinates to vote for. Otherwise, you may have other parameters too. If you allow your shape to change in multiple ways, you could have other dimensions in your accumulator for which also the points can vote.

In this particular case, to keep it simple let us assume that the shape is going to be the same, just that the shape could be placed anywhere in the image. No scaling, no rotation. Let us assume that that is the scenario here. So, the only thing that we are trying to find is where would the centroid of that object?

So, you have an accumulated array, which is of two dimensions going from $X_{c_{min}}$ to $X_{c_{max}}$, $Y_{c_{min}}$ to $Y_{c_{max}}$ are the maximum values that the centroid can assume in the X axis. Similarly, $Y_{c_{min}}$ and $Y_{c_{max}}$. And you initialize the accumulator, every cell of the accumulator to 0. Then for every point for every r_i, ϕ in your r table, you try to write if your centroid can be written as this $x + r_i \cos \phi_i, y + r_i \sin \phi_i$.

If you can write your centroid with respect to this, given x, y using this particular formula, you would increase the accumulator for that particular centroid. And you keep doing this for various different centroid possibilities, where your centroid possibilities will come from $X_{c_{min}}$ to $X_{c_{max}}$ and $Y_{c_{min}}$ to $Y_{c_{max}}$, where you discretize your possibilities of the centroid and you create an accumulative array.

That is the way you would do it. And at the end, you would get a set of values, a set of frequency counts as to what each x, y voted for and you do a non-maximum suppression to find the local maxima and that is going to give you the final x_c, y_c which correspond to this new set of points that are given to you.

Speaking in terms of images. Once again, let us assume that this is the shape that you are looking for. As I said, we assume in this example here, that we are only looking for the shape

as is, no scale change no rotation change. So, which means the only thing that can change is the center of the object could be all different parts. So, that is why we are voting only for where the center is. And so given a new image, you run an edge detector and for each edge point, let it work for where the centroid would be and wherever you get the maximum votes for the centroid is where your centroid potentially could be.

You could use this kind of an approach to say, find a car in an image. If you knew a car has a particular shape, you defined that shape, you now ask every, you run an edge detector on the new image, and then ask each point on the edge to vote for where the center of the car is and based on that, you can probably trace where the car is located in the given image.

So, you could use such an approach even for object detection. Again, within sudden changes in your shapes in the possibilities of the shape. If a car's pose completely changes, obviously this kind of an approach will not work in that scenario. So, it works only with certain tolerance.



(Refer Slide Time: 23:23)

Hough Transform

- Can be used for detection of:
 - shapes
 - objects, including multiple instances
- Advantages
 - Deals with occlusion well
 - Robust to noise
- Disadvantages
 - Can be computationally expensive
 - Setting parameters is not easy

Source: Ioannis Gkioulekas, 16-385 Computer Vision, Spring 2020, CMU

Vineeth N B (IT-H) §3.2 Hough Transform 13



To summarize the Hough Transform, it is an effective approach for detection of shapes, objects, including say multiple, even if there were multiple cars, you can probably find all of them. You could have multiple maxima in where, edges vote for the centroids of the cars and each of them may be different instances of a car in a given image. So, you can also use this for detection of multiple instances of object in an image.


The advantages of the Hough Transform is that it deals with occlusion fairly well because it is a voting procedure. As long as the number of votes is high, it does not matter if certain votes went wrong. So, even if a certain part of the object was occluded it can work. Obviously, if a significant part of the object is occluded it will not work. It is also robust to noise for the same reason, because you only need to look at certain set of pixels, and as long as they vote for a majority, as a majority, you are going to get your shape or object in the image.

The problem here is it can be computationally expensive because depending on how you quantize your parameter space. Remember that every point has to cast a vote for different possibilities in your parameter space, which can be time consuming. And setting your parameters and quantizing them may not be easy for different kinds of shapes. For line, circle and a few really defined shapes, it may be straightforward, for certain other shapes this may not be easy or quantizing them may not give very accurate answers. Those are some limitations to work with here.


(Refer Slide Time: 25:02)

Generalized Hough Transform: Example

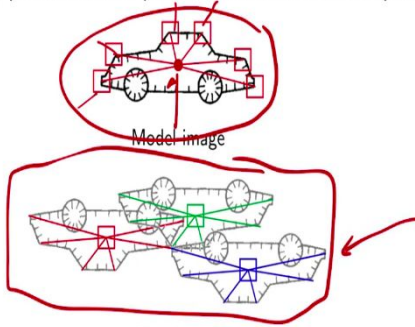
- **Build model:** Record coordinates relative to reference point
- **Test phase:** Each point votes for all possible coordinates of reference point



NPTEL



NPTEL




Model image

Test image

Vineeth N B (IIT-H)

3.2 Hough Transform

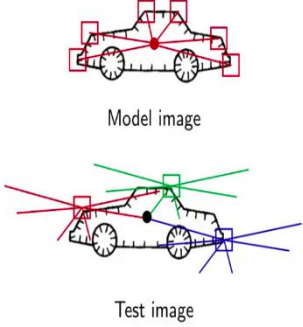


Here is the visualization of the car example. So, you have a model image and there are some key points with respect to a reference point at the center of the image. So, you record the coordinates relative to the reference point in the model image and for every test image, you look for the same configuration of the key points with respect to the centroid of the object.

(Refer Slide Time: 25:37)

Generalized Hough Transform: Example



- **Build model:** Record coordinates relative to reference point
- **Test phase:** Each point votes for all possible coordinates of reference point



Model image

Test image


Vineeth N B (IIT-H) 3.2 Hough Transform



So, in these particular cases, the car is inverted, but when the test image is of the same configuration as the original object, you would find a match in this particular scenario.

(Refer Slide Time: 25:47)



Generalized Hough Transform: Example



Model image

Test image

Vineeth N B (IIT-H) 3.2 Hough Transform



Here is one more example. So if you consider your model image to be the Eiffel Tower and here is a test image which is also the Eiffel Tower taken in a different scale on a different day with different lighting conditions.

(Refer Slide Time: 26:02)

Generalized Hough Transform: Example

Model image points

Test image points

Accumulator

Local Maxima

Vineeth N B (IIT-H) §3.2 Hough Transform

You then have your accumulator after you find your model image points in your original image and your test image, you have your accumulator that votes for the centroid of the object with respect to various key points in that object. You consider a local maxima, you do not see very clearly here, but the local maxima is somewhere in the middle.

(Refer Slide Time: 26:29)

Generalized Hough Transform: Example

Model image points

Test image with location

Accumulator

Local Maxima

Vineeth N B (IIT-H) §3.2 Hough Transform

And based on that you vote for where the Eiffel Tower is located in the test image.

(Refer Slide Time: 26:36)

Homework Readings

Homework




Readings

- Chapter 4.3, Szeliski, *Computer Vision: Algorithms and Applications*

Questions

- How would you use Hough transform to detect ellipses, squares and rectangles?
- Your friend working in a diagnostics startup asks you how to use Hough transform to automatically count Red Blood Cells in a blood sample. What would you advise your friend?

Vineeth N B (IIT-H) 3.2 Hough Transform 14



Your home work for this lecture is Chapter 4.3 of Szeliski's book and a couple of questions for you to think about. How would you Hough Transform to detect ellipsis, squares and rectangles? Try to work out what the parameterizations or the analytical forms of each of these shapes is, and try to find out how the accumulator array would look in each of these cases. That should help you address these problems.



And a real-world use case. Let us assume your friend working in a diagnostic startup asks you how to count the number of red blood cells in a blood sample automatically? So you have a blood sample, you want an automated way to count the number of red blood cells using Hough Transform. What would you advise him or her? Something for you to think about.

(Refer Slide Time: 27:29)

References

- Richard O. Duda and Peter E. Hart. "Use of the Hough Transformation to Detect Lines and Curves in Pictures". In: *Commun. ACM* 15.1 (Jan. 1972), 11–15.
- John Illingworth and Josef Kittler. "A survey of the Hough transform". In: *Computer vision, graphics, and image processing* 44.1 (1988), pp. 87–116.
- Richard Szeliski. *Computer Vision: Algorithms and Applications*. Texts in Computer Science. London: Springer-Verlag, 2011.

Vineeth N B (IIT-H) 3.2 Hough Transform 15



And here are the references.