Advanced IOT Applications

Dr. T V Prabhakar
Department of Electronic Systems Engineering
Indian Institute of Science, Bangalore

Lecture – 13
Basic computer vision algorithms Part -2

So after you have completed an edge detection on an image, the next step is to identify lines. We are particularly interested in lines in the context of automotive applications, because we expect that most roads are like straight roads; of course there are bends and particularly when you are moving over a small hill or a mountain you go round taking bends right bends are there, but by and large what is important is that you have to detect the edges at road ends and lane divider or another road edge itself. Our hypothetical example was also about the fact that there is a road and then there as a left edge ,there is a right edge, there is a divider in the middle and then there is a lane marking, right? all of this we had assumed.

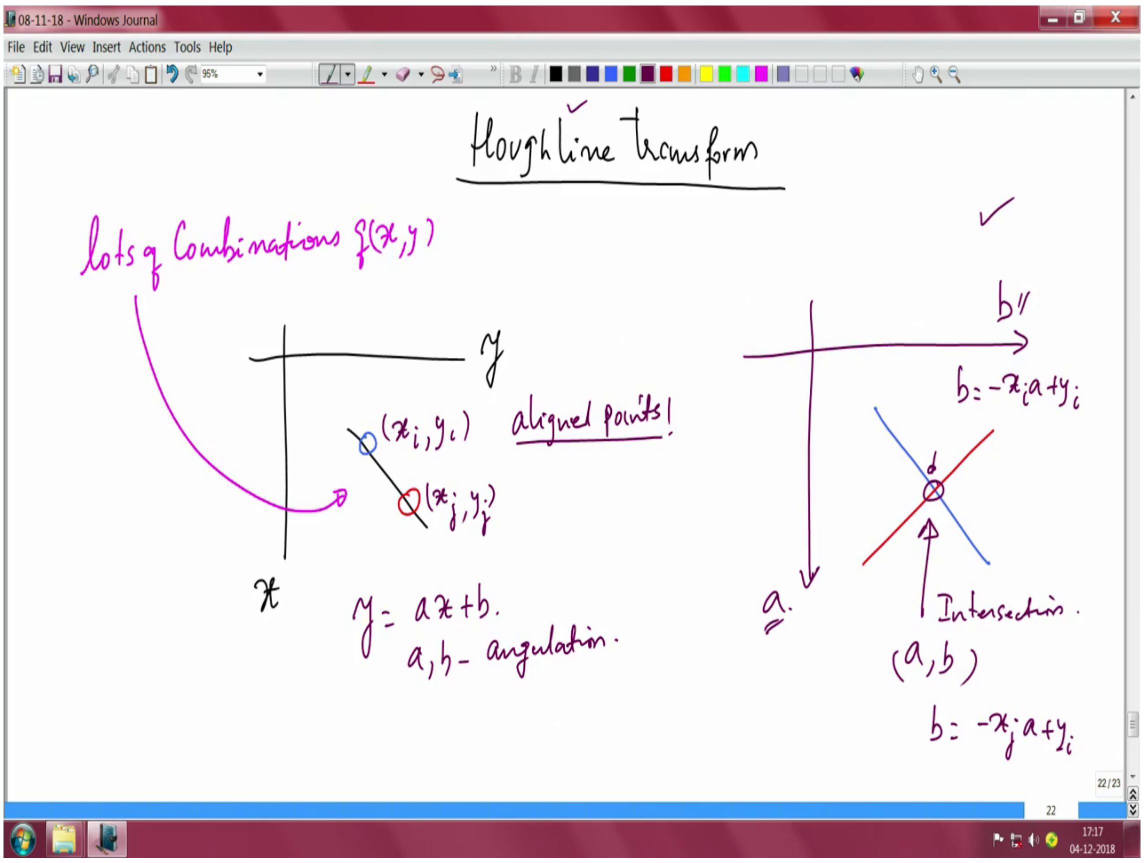
You may argue that edge detection with canny edge detection particularly, would be sufficient if you are just looking at you know straight lines, canny edge is essentially for edge detection. If your edge happens to be a straight edge it will simply detect that as a straight edge, but it will not call it like a line it will not declare anything as a line it continues to call it as an edge, it just that we interpret it as a line.

So, therefore, you cannot assume and perhaps these are some small mistakes that people may end up doing, that they may say let me restrict this example to just try canny edge, because we are only looking at straight lines that is a limiting way because as I said the road can also turn left can turn right and so on and then edge canny edge will only give you that edge, it will not do anything whether this edge is straight or whether its curved or whatever it does not say anything.

Therefore, Hough line transform is the next logical thing to do. See there are so much of literature out on the internet that you could pick any literature that you are comfortable with. So, however in this course I would strongly suggest that you concentrate understand either canny edge or Hough line from OpenCV perspective. So, that it becomes easy for you when you are working on tools like MATLAB and so on right or python or particularly python or see any of them open CV is an important library API, for most image processing application. So, you should look at definitions around that.

So, I have tried to sort of look at a mix of several literature that is available and then I have tried to make a very small module on Houghline, because I do not want to repeat because there are all the good material which is out there on the web.

(Refer Slide Time: 03:59)

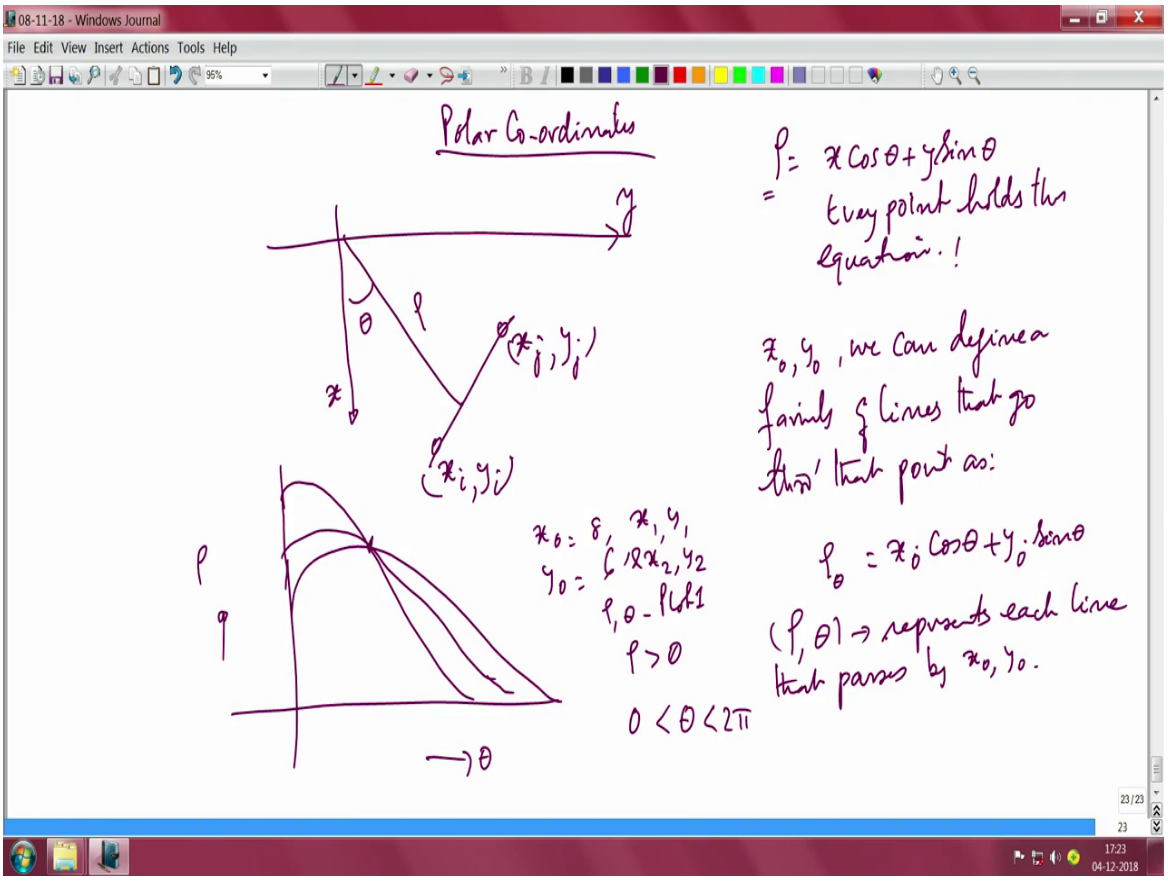


So, how do you detect lines go back to high school, go back to the famous straight line equation you have x here you have y here and you are interested in knowing the set of points starting from Xi to Yi to Xj ,Yj. I have just shown two points, but there are infinite number of points as you go along the line right, as you go along the line shown in the picture there are infinite number of points. If you are able to show that they all are sort of aligned, they all are aligned points then it must be a straight line that is all you have to say. So, the a and b as you know is essentially gives you information about the angulations. So, there lots of combinations for x y in order to arrive at the notion of aligned points.

Now the intuition of a and b as co-ordinates of the plot is outstanding, really you should spend time to understand Houghline from this picture, its a nice picture. See what has actually happened here, you have done a transform as you see the word here Houghline transform what is the transform you are doing? You are transforming from the x y plane to the a b plane and on the a b plane you find that the a b intersection is the most interesting part. If these lines these points Xi Yi Xj Yj and so, many other points all of them are aligned in the same line all of them will come to the same intersection point that is the key here.

So, if you start counting the number of times these lines are intersecting, you actually know the number of points which actually make up the line that is the interesting part. And again I urge you to look up several places on the internet where this material is discussed extremely well so, I do not want to put too much emphasis there.

(Refer Slide Time: 06:31)


Polar Co-ordinates

$\rho = x\cos\theta + y\sin\theta$

Every point holds this equation.!

x_0, y_0, we can define a family of lines that go thro' that point as:

$\rho_\theta = x_0\cos\theta + y_0\sin\theta$

$(\rho, \theta) \rightarrow$ represents each line that passes by x_0, y_0.

$x_0 = 8,\ x_1, y_1$

$y_0 = 6\ \ x_2, y_2$

ρ, θ - Plot

$\rho > 0$

$0 < \theta < 2\pi$

So, that is the most interesting part now how do you actually realize all of that well take the straight line into the polar coordinate, where everything is represented as rho and theta right and you know the famous equation rho is equal to Xcos theta plus Ysin theta every point holds this equation, every point this is the key here, every point holds this equation.

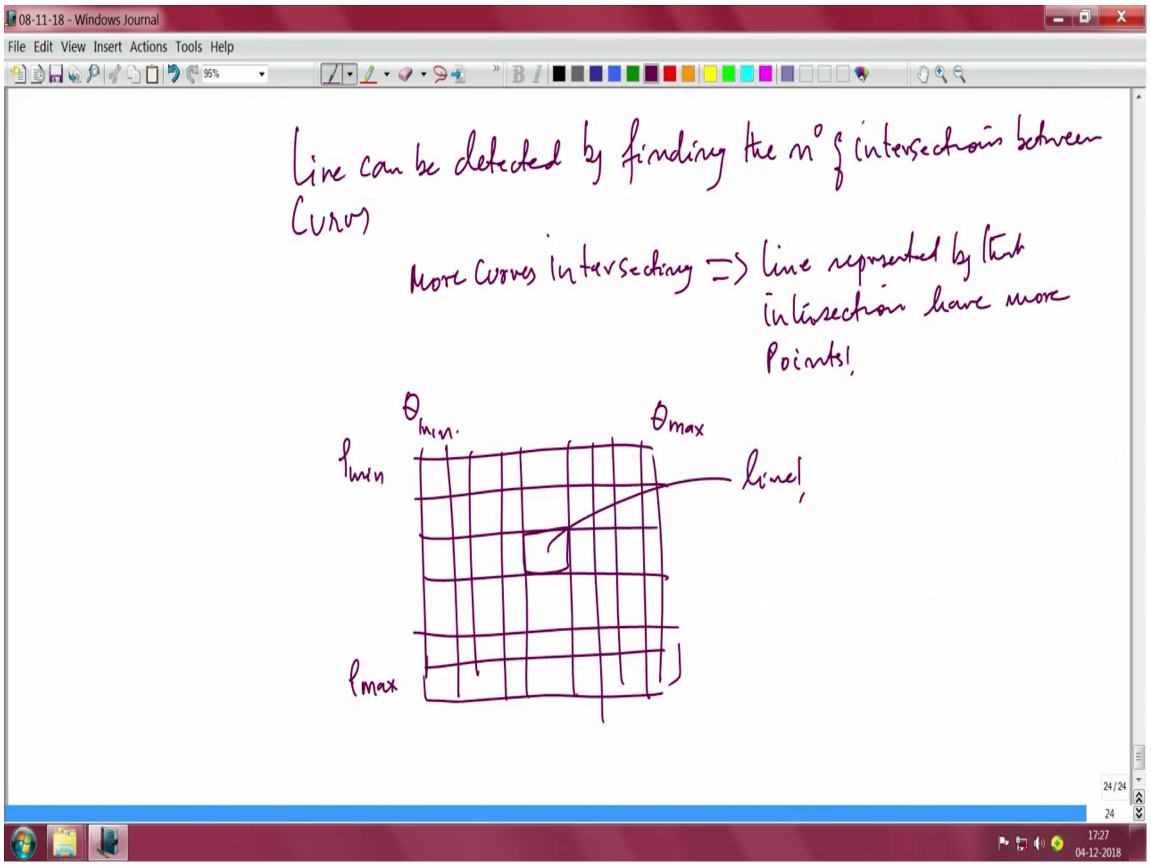
You now what you do is something even more interesting you essentially do the following. In general for each point X_0 Y_0, we can define a family lines, go through that point as what will you do? Rho theta is equal to X_0 cos theta plus Y_0 sin theta, meaning that each pair of RHO theta. So, for this RHO, theta for each pair of this rho theta it represents each line passes through X_0 Y_0, meaning that each pair rho theta represents each line that passes by X_0 Y_0. Now, if we plot the lines for a given X_0 and Y_0 the family of lines that go through it what we get is a nice sinusoidal.

Essentially y axis is theta and X axis is essentially rho you start getting this we get a sinusoids for instance you start putting numbers X_0 you put 8, Y_0 you put 6 you get a plot in the R theta plane plot 1 call it. Then what you do you essentially see you have to note that always rho is greater than 0 that is important and theta is lying between 0 and 2 pi.

Now, what you do you go on giving values for different values of X_0 and Y_0, then you will start getting a family of curves. All of them will be intersecting somewhere or the other, you take one more and intersect it here, goes like this shown below left corner. So, you see you will go on getting these 3 plots which intersect at 1 point, there is 1 intersection point here. These coordinates are the parameters for some rho theta or the line which has these 3 points X_0 Y_0 , X1 Y1 and X2 Y2, all of them are intersecting here which means they are in a line that is all you are saying.

So, that is the point about detecting Houghlines in actual implementation you can actually do this, in general you can say a line can be detected,

(Refer Slide Time: 11:56)



Line can be detected by finding the no of intersections between curves

More curves intersecting ⇒ line represented by that intersection have more points!

by finding the number of what is it now? its a count number of intersections between curves, is a very nice way to you know find out the line and the how it can how you can detect this line. So, essentially you are creating a nice table of all theta and rho values and then you know whichever has the highest count essentially finding the number of intersections means the count intersection every time there is an intersection you can increase the count by 1 and then you know how many points essentially form the line.

So, essentially that is what you are saying the more curves intersecting means the line represented by the intersection have more points simple. So, I will just put it here more curves intersecting implies line represented by that intersection have more points what a nice intuition indeed right and this is what the Hough line is all about.

So to just summarize, you can make a nice table, you can have 0 here for theta this can go to theta max this can go to theta min, you can have rho min here and rho max here then keep filling this table that is all you need to do, this dot line I should remove very simple to see, the count suppose one square has the maximum count, actually shows 1 count; that means, all points which we have chosen X _0 Y_0 X1 Y1 X2 Y2 X3 and so on, all of them form one line this is one line that is all. So, these two put together the canny edge detection in image processing using a camera sensor and the Hough line transform post edge detection form the most important part of our further demonstrations and your own tests and trials that you can do part of this module.

(Refer Slide Time: 16:01)

So, our basic infrastructure for following a lane includes a robotic platform, its a Kobuki robotic platform as shown in the picture. It is from a company called Yujin Robot. On top of that is our basic autonomous vehicle. It has several wheel sensors and then you have to apply thrust on these wheels. Essentially we are going back to our basic Braitenberg algorithm, where the sensor is indeed the camera and the actuator indeed are the two are the motors to which are coupled, two wheels right.

So, its a same model. We are taking a very simple model and trying to show you what are the things, that how complicated even a simple model can actually result in. So, there is a basic camera and this camera is a USB camera to which is connected to a small single board computer which essentially is a raspberry pi platform and then there is a battery bank and all of that. So, this essentially forms our autonomous vehicle and now let us get into the details of the steps involved in showing you a demonstration, before going to the demonstration you must know the platform you should know the steps and then you should actually see the demonstration right. So, we will go to do a code walkthrough and show you what are the major steps involved in the process.

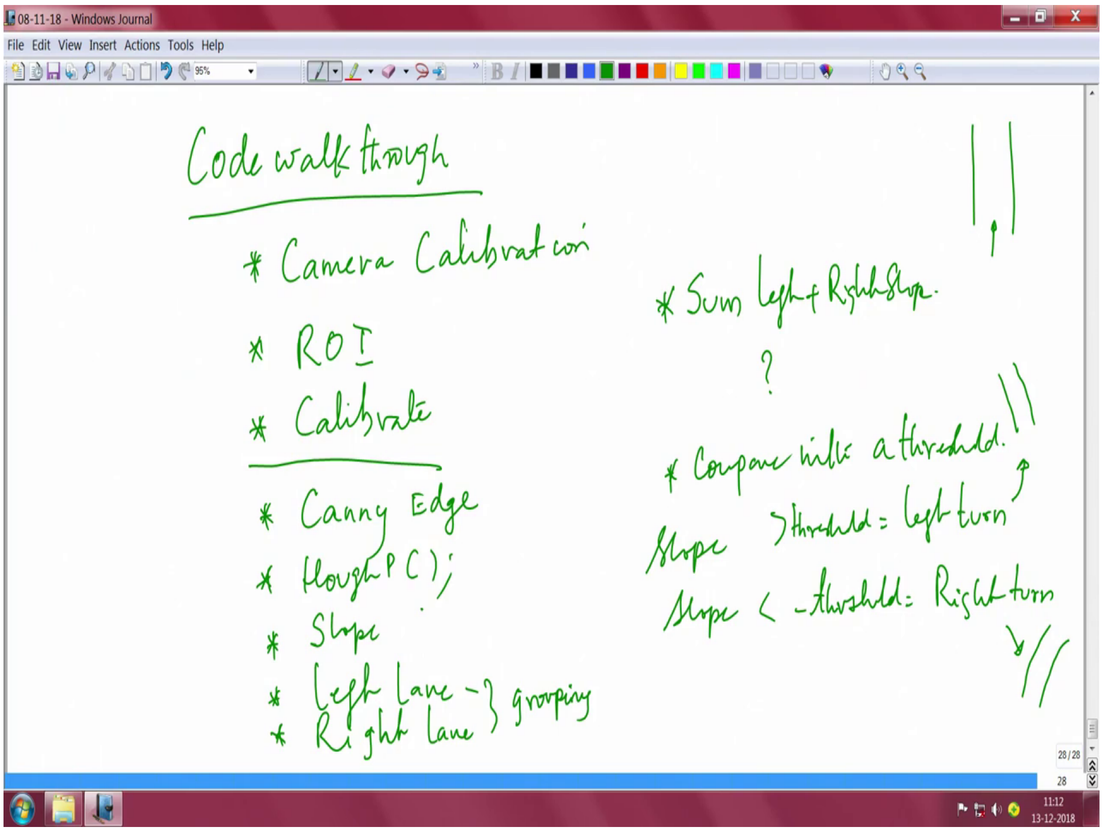
Now, what is important is, if you are using camera as a sensor you must do what is known as camera calibration. Now, camera calibration essentially means that a board like the chess board should be used to take care of several distortions that a camera can essentially inherently have. So, this is very important that is why does the camera have inherent distortions, because you are talking about an optical system there is a lens, there the lens introduces distortions and those distortions have to be corrected and there are steps which you have to follow before you actually start using these sensor.

Now, I can tell you one big mistake that most people do you do not start a calibration like holding the board bending inward or outward. This is already going to introduce distortion for you are expected to mount it rigidly on some sort of a cardboard. I have just taken this to show you to put it on a rigid cardboard. So, that this sheet itself is rigidly held and then you start your calibration process. So, this is the first step, so remember in IOT you are bound to have sensors and if you are going to have the sensor, if it is air quality you are going to have air quality sensor, if it is going to be autonomous driving you are going to have LiDARs radars camera sensors and so on.

Any application you take will have sensors, first step is to calibrate those sensors particularly. If you are using this kind of camera sensors. They say some data sheets talk about this kind of calibration. Very high end cameras perhaps may not needed, because they already do a few things and make it sophisticated for you. But if you are buying your own lab based equipment and lab based simple camera sensors the calibration process is available and so you have to go through those steps.

And let me also assure you that a very nice algorithms that are proposed for camera calibration all of it will be available to you in terms of a nice function call in OpenCV, which you can directly use, but you should know what those functions are and what is the function of the function right. So, you must know what exactly is happening so, that you will be able to appreciate this whole process of calibration right. So, let us get into the details of this whole process and also do a code walkthrough and then we will actually see a demonstration.

(Refer Slide Time: 20:47)


08-11-18 - Windows Journal
File Edit View Insert Actions Tools Help

Code walk through

* Camera Calibration
* ROI
* Calibrate

* Canny Edge
* HoughP();
* Slope
* Left lane } grouping
* Right lane

* Sum left + Right Slope.
?
* Compare with a threshold.

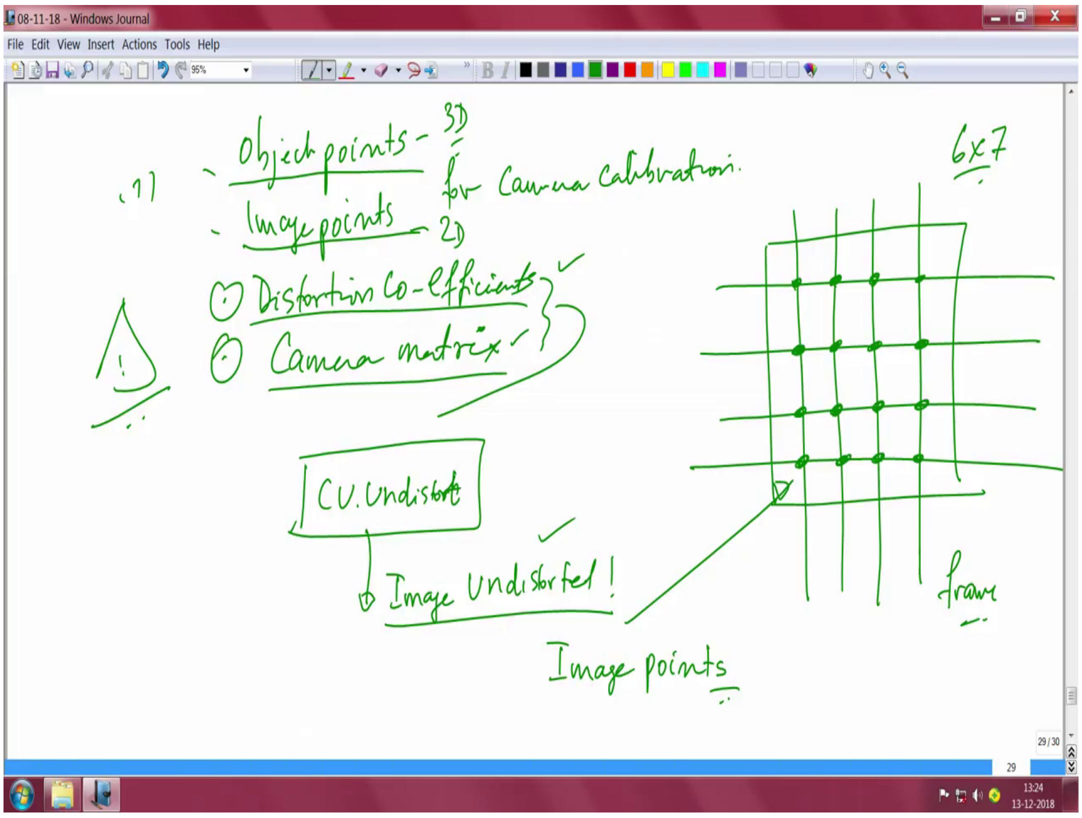
Slope > threshold = left turn

Slope < -threshold: Right turn

The first step towards camera calibration when you want to do code walkthrough through the whole step as I said is trying to establish the camera calibration. Let us see what are the code corresponding to camera calibration. So, let us now quickly do find out all I mean look at the source code of all that we said, first of all what is your objective? your objective is to use some kind of a reference chess board or a checkup board and then try to calibrate the camera.

Most often this camera calibration is something that is done for pinhole type of cameras, larger professional cameras which are used in perhaps in autonomous driving. We will also have something equivalent, but you may not have to do this kind of very elaborate process, but for most lab experiments and all that this is the first step for any optical device that one used because lenses will bring in distortion. So, idea is that these are the points the intersection points that you see, internal intersection points, and this can be easily translated into this picture that I have drawn here.

(Refer Slide Time: 22:10)



So, let us focus on the picture; you can see the internal intersection points. These are essentially the intersection points which should lie exactly sort of red dots which you assign to on top of the basic chessboard that you have. And once this chessboard is twisted, is radially distorted it should be able to continue to track it and know how much distortion has been has been applied on this so, let me give you a demonstration. So, let me go back to the physical chessboard that we have here.

Now, you have the intersection points on each one of these points internal to square box, here all these dots are there and for some reason the image it sees is like distorted; that means, how much is the distortion it should know. So, that information is 2D in a way if you move away or towards there is a change in depth. So, some how you should be able to look at the 2D image and also look at how to convert this into something equivalent in terms of 3D object points all of that is done in the open CV. So, let me again point you back to the steps that I have, the image points are these cross section points or these red dots I mentioned are essentially the 2 D points from that you will be creating this object points which essentially comes, because of tilt in the chess board which will give you the depth information which will become 3 D right?

All of this essentially leads to sort of estimating the distortion coefficients and the camera matrix of interest, that these two once you have estimated through the process of camera calibration, you feed that information to this function called cv.undistort. So, if you do that you will get an image which is completely undistorted.

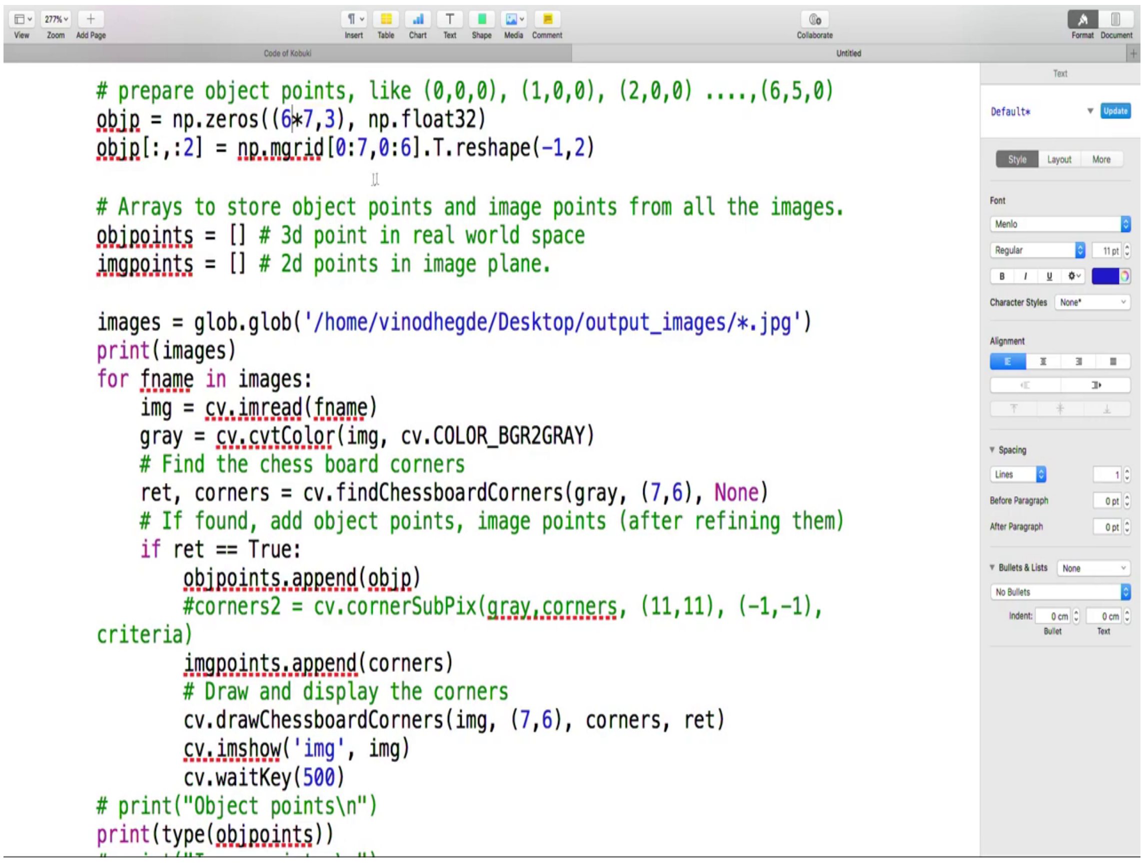
So, essentially, I have sort of glossed over the whole process, because it involves a lot of intricate linear algebra background, but the point really is that you will need to estimate these two, the distortion coefficients and the camera matrix. Essentially, you have to let me show you this picture again chessboard, red dots at these intersection points depth information that is available keep tracking the depth information, correct the image for this depth that comes and essentially keep ensuring that the camera knows how much of distortion how to how much of distortion is there.

Because it has estimated the camera distortion coefficients, has the camera matrix with it applies this correction, using this function I mentioned to you and you ultimately end up within an undistorted image. Let us now look at the source code of that and then move on to the remaining parts of the program, which essentially did the straight line a turn to the right and then a turn to the left.

So, let us do that see take the image take one snapshot of the image of chessboard take one more snapshot, take one more like this; take one more like this, collect as many samples as possible collect about let us say 25 or even 50 images the large number of samples that you take, because you are trying to estimate the distortion coefficient isn't it? and also you are trying to come up with the camera matrix.

So, therefore, as many pictures as possible if you collect it is always good. So, take all those images those little images that you have, in the process of trying to estimate the two things that I mentioned to you; one is the image points essentially those red dots which are exactly at the intersection and from that you are trying to get to the 3 D which essentially is nothing, but the image points because you will be rotating it. So, let me show you the code that does exactly this.

(Refer Slide Time: 26:25)

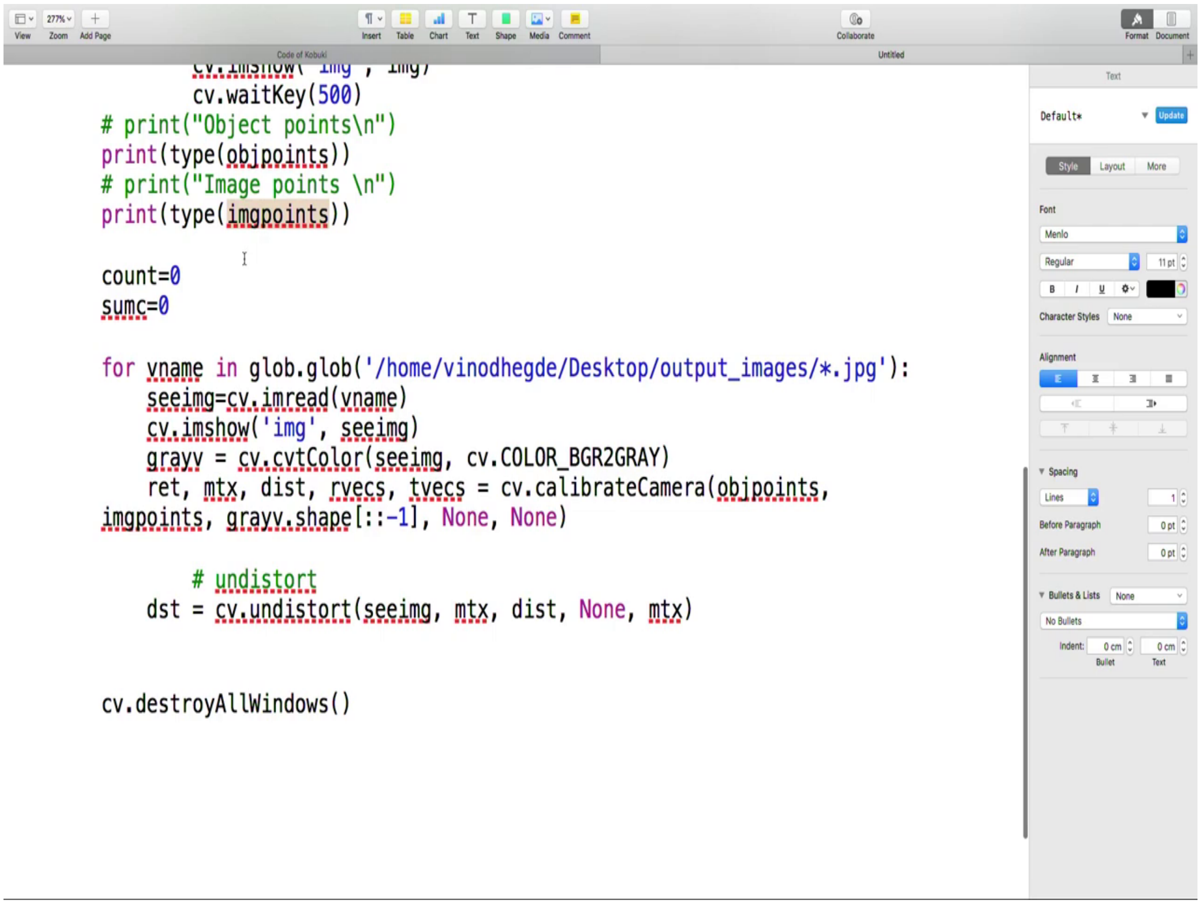


```
# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
objp = np.zeros((6*7,3), np.float32)
objp[:,:2] = np.mgrid[0:7,0:6].T.reshape(-1,2)

# Arrays to store object points and image points from all the images.
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.

images = glob.glob('/home/vinodhegde/Desktop/output_images/*.jpg')
print(images)
for fname in images:
    img = cv.imread(fname)
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    # Find the chess board corners
    ret, corners = cv.findChessboardCorners(gray, (7,6), None)
    # If found, add object points, image points (after refining them)
    if ret == True:
        objpoints.append(objp)
        #corners2 = cv.cornerSubPix(gray,corners, (11,11), (-1,-1), 
criteria)
        imgpoints.append(corners)
        # Draw and display the corners
        cv.drawChessboardCorners(img, (7,6), corners, ret)
        cv.imshow('img', img)
        cv.waitKey(500)
# print("Object points\n")
print(type(objpoints))
```

Essentially np.zeros(6 *7)cross 3 coordnaites points, intersection points those red dots are say 6 on the x axis and 7 on the y axis essentially that is the number of points that you have. So that is one thing , the second thing is the object points that we mentioned to you indeed in the 3d point cloud space, the image points are essentially 2d points, this part of the routine, where the glob.glob is used essentially reading one image after the other right, it is has to read all these images and then take each image at a time.

And then identify the you call this function called cv.findChessboard corners and it is trying to estimate all the corners, each for each of the images. Essentially you will end up with both the object points as well as the image points as and when you run it across all these images.

(Refer Slide Time: 27:27)



```
            cv.waitKey(500)
    # print("Object points\n")
    print(type(objpoints))
    # print("Image points \n")
    print(type(imgpoints))

    count=0
    sumc=0

    for vname in glob.glob('/home/vinodhegde/Desktop/output_images/*.jpg'):
        seeimg=cv.imread(vname)
        cv.imshow('img', seeimg)
        grayy = cv.cvtColor(seeimg, cv.COLOR_BGR2GRAY)
        ret, mtx, dist, rvecs, tvecs = cv.calibrateCamera(objpoints,
    imgpoints, grayy.shape[::-1], None, None)

            # undistort
        dst = cv.undistort(seeimg, mtx, dist, None, mtx)


    cv.destroyAllWindows()
```

At the end you will end up with as you can see we are done in we have done a print of object points and the image points. So, how many if you have 25 images that you have taken how many object points will you have? You will have one set for each image, you see you will get if it is a 6 cross 3. So, it will be a 6 cross 3 6 cross 7 in our case you will get the object points and image points and object points for every frame, this is like one frame right with one image that you have captured.

If you turn it if you turn the picture you will get one more snapshot image for that, snapshot image you will get object points and image points like that you will keep getting for all 25 or 50 or any number of images that you have taken for the purposes of calibration. What you do with all these points when you come to this code here open cv function called cv.calibrate, to which you will pass the object points or the image points and also the size that you are considered which is essentially 6 cross 7 in our case.

And then you are trying to estimate what you known as a camera matrix and the distortion matrix, apart from the radial vector which is rvecs and also the tangent vector which is the other parameter. For the moment we are not looked at the radial vectors and tangent vectors, we are only interested in the camera matrix and the distortion coefficients.

Now, what you have to do? you take this camera matrix, you can see this function cv dot undistort you take the camera matrix take the distortion coefficients, take one of the images that you have of interest and then feed it to this function what you get at the output is a completely undistorted output which is essentially the variable here which is dst with this you have done the camera calibration.

So let us see how to put everything together in terms of a small demonstration my project staff Vinod Hegde has worked extensively on this trying to make a small system working for you. Let us see how exciting this whole set of process, set of algorithms get into a nice process of demonstration. Remember I mentioned to you about camera calibration holding paper is not the way to calibrate, put it on a hard surface and then do the calibration. So, just for holding this in my hand I am just showing you, but this is not the way to do it put it on a rigid surface so, that it is straight and idea is you should be able to detect all the cross sections and then map it back onto the complete camera squares right. Then you are ready to go back as far as the calibrations concerned that is just the first step, then a lot of matrix operations have to be performed before you actually put in the required coefficients and then conclude that the camera is correctly calibrated. Let us think about that as a very first step. After that you will actually start looking at how to mount such a camera and ensure that it is does this line following and even if the line is not straight, but it is turning to the left and right it should be able to go, you know do all the algorithms that we discussed.

For that let us now slowly zoom the camera towards the set up. The set up comprises of a camera, power bank, onboard computer and robotic platform; that is like equivalent to of our autonomous vehicle that we had proposed in this module. Now, look at the straight line, there are two straight line as far as camera is concerned horizontal noisy lines are also a straight line, but orthogonal to the two lines right.

So, canny edge has to be clever enough to take those which are dominant and then start only capturing those important lines of interest. So, any image captured by such a camera first has to be converted to grey scale then you do what is known as a Gaussian blurring and, then you apply the canny edge algorithm number of edges, number of lines are indicated soon after you perform the Hough line.

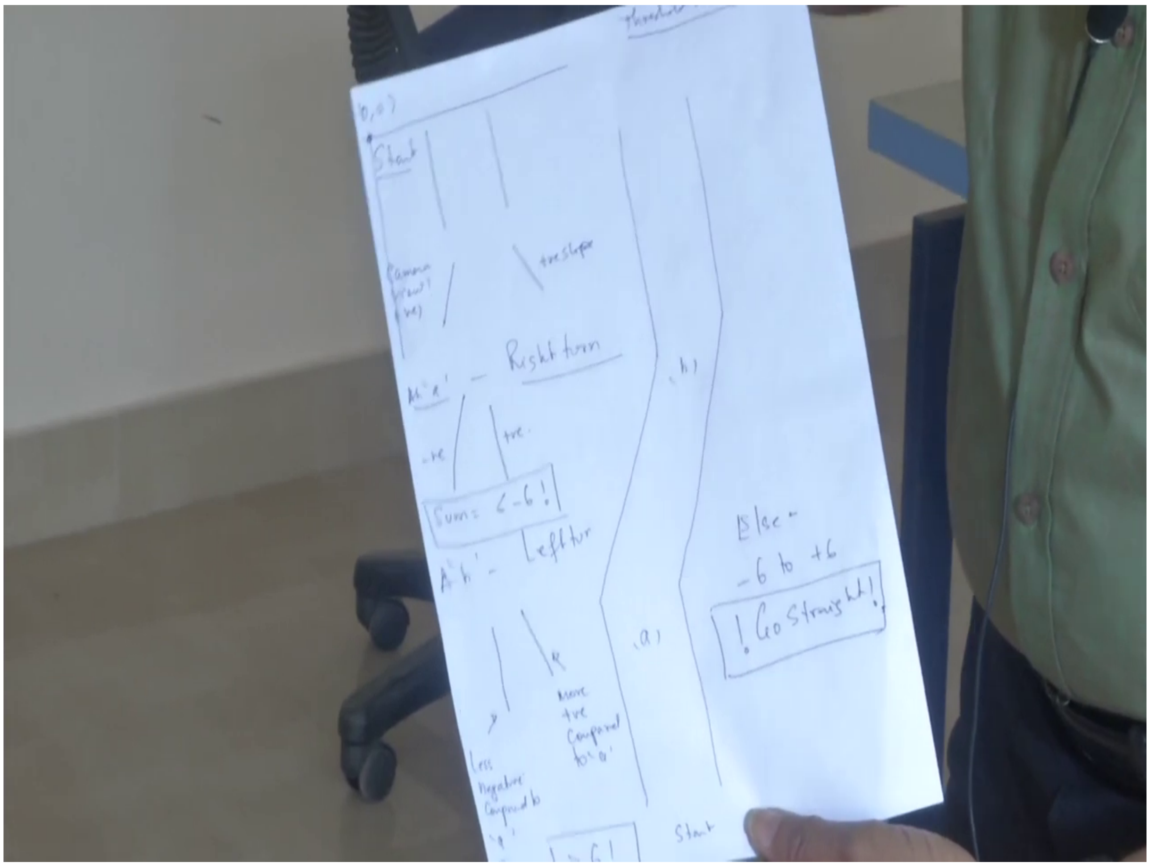
So, the moment Hough line is done number of lines are indicated in this view of the camera. Particularly you can see we have turned the camera such that it just pointing to a very small area, here what you should also do is set your thresholds right just to identify the most two dominant lines and what is also important is the region of interest; the region of interest since the camera is pointing down is like a sort of a triangle and of the calibration and conversion to grey scale in the initial stage it is able to identify these two lines dominant lines. Then start following the two lines so, I may have left things unanswered for you right.

In your region of interest you have to identify the two dominant lines, what does it mean in terms of actually implementing an algorithm? it Simply means that you have to calculate slopes. The effort in making this robotic platform move based on the region of interest is essentially trying to understand the slopes of identified lines, these will have a certain slope and the other have different slopes. You can identify the thickness for instance as an important feature and then look at the thickness of the lines and take two lines and lines separated by the edges at a certain distance and then say these are the two lines of interest.

So, it can be a very interesting algorithm even to identify the two dominant lines, edges in a given region of interest. So, I let you imagine how we would have actually taken care of the millions of lines straight lines, that are coming in this region of interest and just picking these two dominant lines right. So, that is one part, but at the same time you also have to note that one trick to do that is to look at slopes as I mentioned to you.

And then see if you take a straight line of some thickness which has one edge, second edge, third edge and fourth edge and you take the slopes of both of them and add the 2 slopes, you end up seeing that the slopes cancel each other and dominant edges, which cancel out the slopes indeed are the 2 lines of left edge and the right edge of interest. So, that is essentially what would happen, then let me that is essentially what we have done in this demonstration.

(Refer Slide Time: 35:34)



Let us see this paper which is the crux of the demonstration, what is the starting point the you have coordinate 0 0 here, top left corner; assume a is the start, you have two straight lines right the system has to move and that is indeed, this is the start this is the lane of interest that it has to move.

Then the Kobuki the platform comes to this point a, comes to a what would happen at a it has to take a right, you will see that the left edge will have a negative slope the right edge will have a positive slope and together when you sum them up you will get a number which is less than minus 6. If you get a number less than minus 6 the sensor output simply gives thrust to the wheels of the platform to make a right turn, which means this threshold adjustment becomes very critical the threshold is essentially from minus 6 to plus 6.

If you get the sum of the left and the right slopes less than minus 6, you have to take a right turn the same thing will happen when it comes to b, it will again continuously it will be calculating the slopes all across it will be calculating the slopes, at b it realizes that the slope is more positive compared to what it was at a the right edge, I mean the right edge is more positive compared to a and the left edge is less negative compared to what it saw at a,

With this you will find that the sum of the left and the right slope is greater than 6 and moment that happens the sensor signals to the thrust motors to take a left turn and then it makes a left and then proceeds straight. So, if the two slopes cancel each other what you should do? we should simply go straight, this is the simplest possible algorithm that we can look a. So, now, let us turn our attention with this background let us turn our attention to the actual demonstration of all that we have put together.

Refer video – 38:10 ; https://www.youtube.com/watch?v=__RsJ3fRQsk

So, you can see now that the autonomous system is moving forward, the region of interest is captured by the camera the slopes are cancelling out each other, it has started, it has now come to position a, at a what should happen? The sum of slope is less than minus 6, the thrust is given to the motor it takes a nice right turn and tries to move forward and again sees 2 straight edges 4 edges each time the 4 edges the slopes are cancelling out continuously and therefore, decides to move along the straight line.

You can see a slight adjustment it is trying to find the 2 lines and then tries to make a correction perhaps again, let us see what would it would do? you can see that its making its way trying to correct itself away from the 2 lines, it comes to position b at b it has to take a left turn and because the sum of slopes is greater than 6 takes a left turn.

And then it sees the two lines and starts moving forward, it will do something interesting at the end which is I am sure will be quite puzzling for you can see how the algorithm while we have put all these high performance algorithms in place, if the two edges one of the edge actually misses. It starts identifying some two dominant edges in the region of interest that we have defined and starts following those 2 edges thinking that indeed that there is a straight line, you will see now that you will see it has gone till now.

And now it starts drifting there are no lines some two imaginative lines it has found from the floor, because the floor tiles have indicated at some 2 lines, it is doing the same calculation it has found some slope differences, it has taken a right turn and it is moving forward and then perhaps it finds during its continuous calculation finds that there is a difference in slope based on that it would either take a left turn or a right turn and so on.

So, it becomes quite unpredictable now and perhaps these are the dangers of all these algorithms, if you do not design them in a perfect manner. So, you may have to bring in failsafe options you may have to bring in options of you know halting the vehicle in case there is a mismatch and so on and so forth.