Computer vision and lane detection

Nacho Sañudo
University of Modena and Reggio Emilia
Ignacio.sanudoolmedo@unimore.it

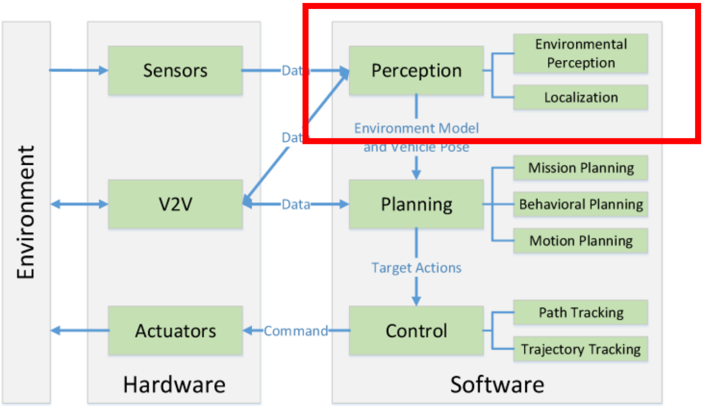
HiPeRT Lab

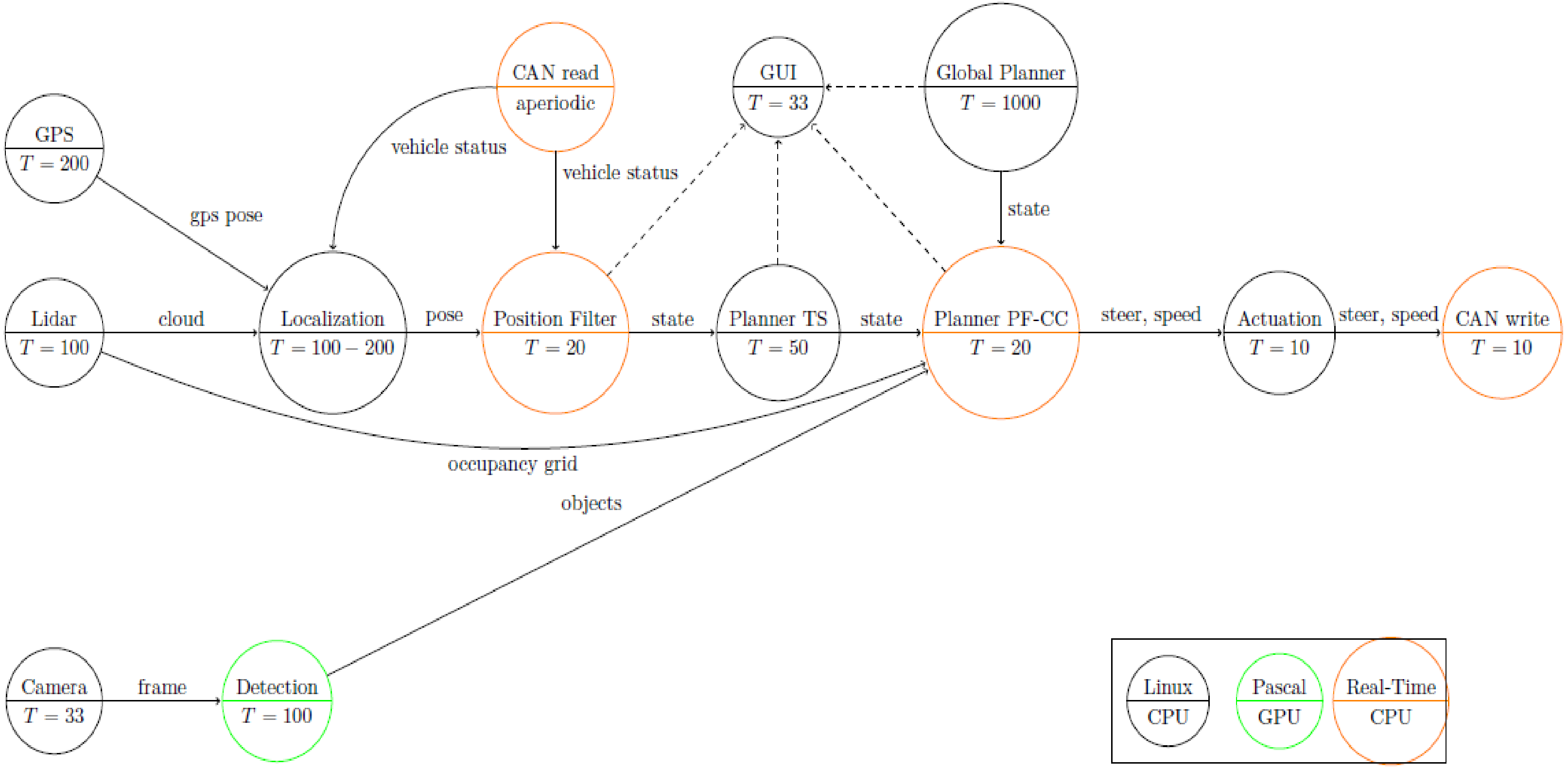
UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Contents

- Perception
- Cameras
  - Images, RGB model
- Computer vision
  - Machine learning
  - Geometry approach
- Use cases
  - Calibration
  - Distance estimation
- Optional assignment
  - Smoothing, Canny, Hough transform...

AD stack





sudo apt install python3-opencv

Perception

- We use our eyes to drive
  - Perceiving the surrounding environment and extracting information is important for autonomous driving systems
  - Monocular and stereo cameras and other sensors allow us to achieve similar functions
    - Adaptive Cruise Control
    - Vehicle, Pedestrian and Cyclist Detection
    - Automatic Emergency Braking
    - Front and Rear Vehicle Classification
    - Worldwide Traffic Sign Detection
    - Road Edge Detection
    - Lane Departure Warning
    - Lane Keep Assist
    - General Object Detection
    - Road Surface Preview
    - Free Space Detection
    - Parking Assist
    - Small Object Detection
    - Traffic Light Detection
    - Vehicle Detection at Any Angle

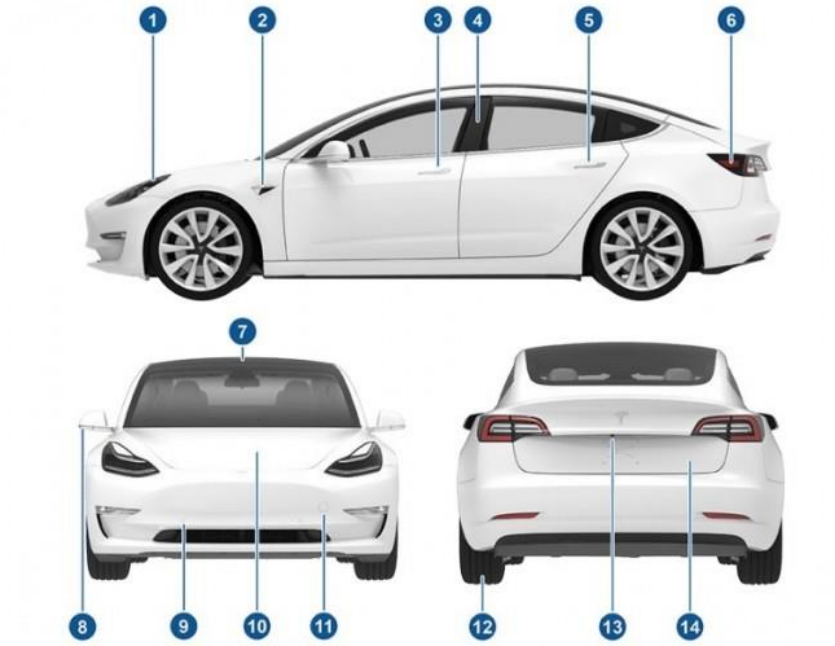
Source: veoneer

Perception

- Perceiving the surrounding environment and extracting information is important for autonomous driving systems
- Cameras are used to identify the presence and semantic attributes of the driving environment:
  1. Localization
  2. **<u>Environmental perception</u>**
     - Road user's detection, classification and tracking
     - Traffic light detection
     - Road sign detection
     - **Lane detection**
     - **We can use computer vision techniques or neural networks**
- **<u>Objective (optional assignment)</u>: identify and track the position of the lines** in different images

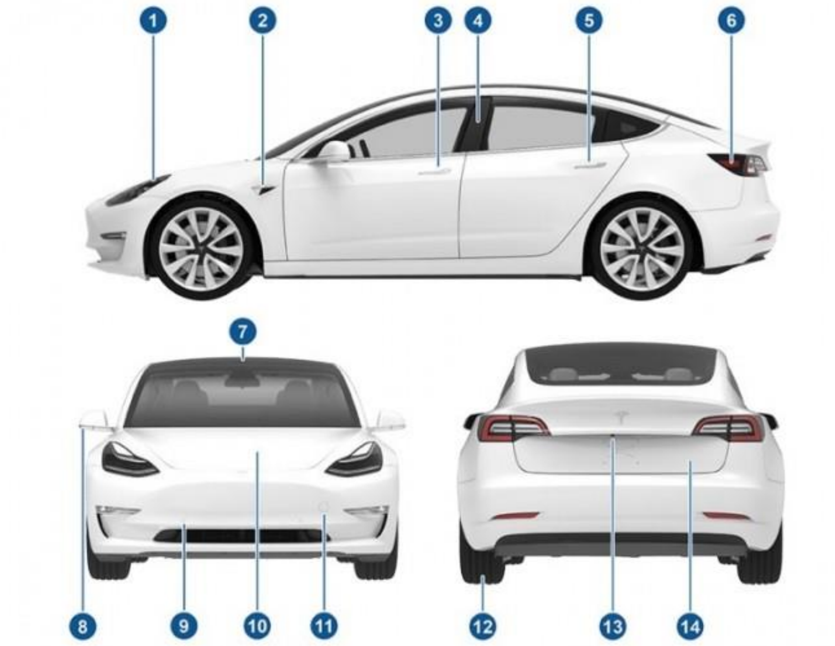
Monocular camera

- A monocular camera is a camera device with one lenses
- It is designed to capture images for enhancing the safety of the driver
- They are generally used to detect and track objects or detect lane boundaries
- Some companies use omni directional cameras to localize the vehicle



Monocular camera

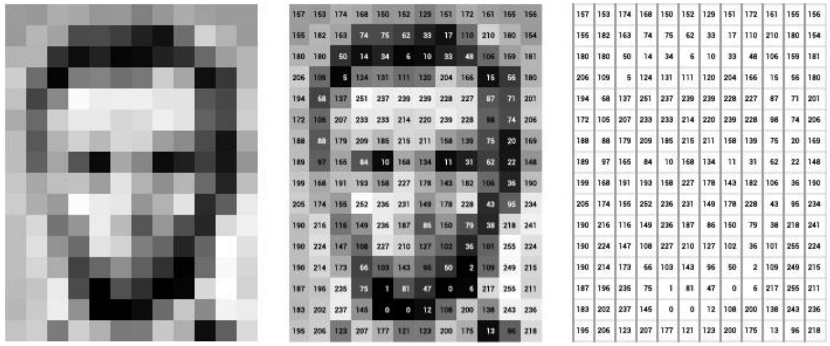
- Metrics:
  - Resolution (number of pixels that create an image)
  - Field of view (horizontal and vertical angle extent that is visible to the camera)
  - Dynamic range

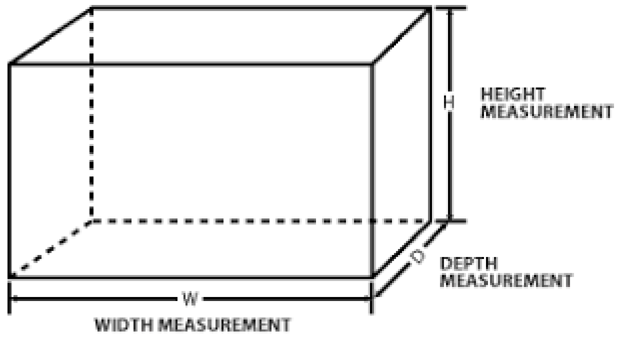


- **What is the difference between the information provided by the cameras and the rest of the sensors?**

Image

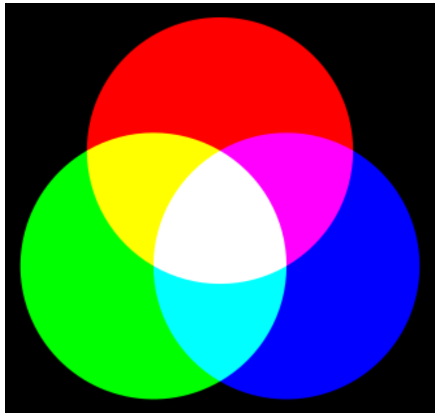
- An image is a collection of **pixels** which can be represented as a **matrix**
  - contains a numerical value representing the **color/intensity**
- **Most color and shape techniques operate over this matrix**
- Each image is characterized by a height, width and depth
  - Depth is the number of color channel
    - **Red, Green, Blue (RGB)**





RGB color model

- The RGB color model is an additive color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors
- A color in the RGB color model is described by indicating how much of each of the red, green, and blue is included. The color is expressed as an RGB triplet (*r*,*g*,*b*)



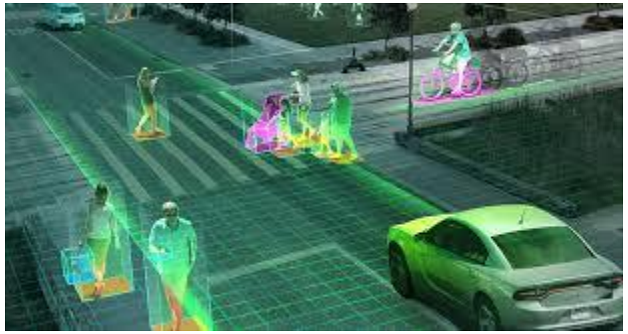
RGB color table

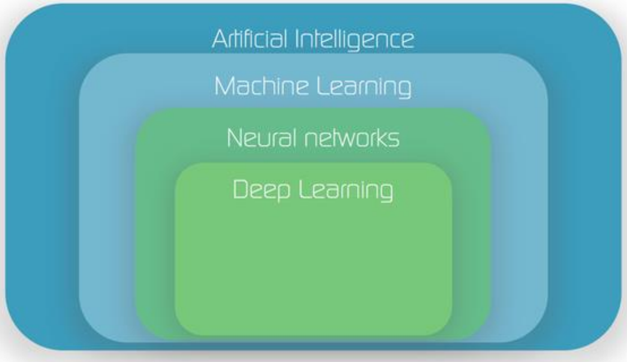
HTML / CSS Name	Hex Code #RRGGBB	Decimal Code (R,G,B)
White	#FFFFFF	(255,255,255)
Red	#FF0000	(255,0,0)
Lime	#00FF00	(0,255,0)
Blue	#0000FF	(0,0,255)

Source: https://en.wikipedia.org/wiki/RGB_color_model

Computer vision

- Computer vision is an interdisciplinary scientific field that deals with how computers can gain high-level understanding from digital images or videos
  - **Machine learning / neural networks**
    - **Most object detectors are implemented using neural networks/machine learning**
  - Geometry/algorithmic-based approach
    - Classic approach





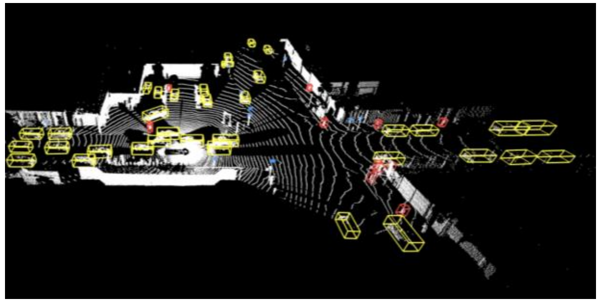
Source: Wikipedia

Machine learning

- Traditionally we should program by hand the instructions needed to recognize the shape of the car, the color → prone to error
- Machine learning (ML) is the study of computer algorithms that improve automatically through **experience**
  - Detection, tracking, semantics, behavior prediction…
- Machine learning algorithms uses a mathematical model based on sample **data** (**training data**)
- To make predictions or decisions another mathematical model is applied based on the training model (**inference phase**)

Source: FF AACHEN. Alexander Ferrein & Wikipedia

Datasets



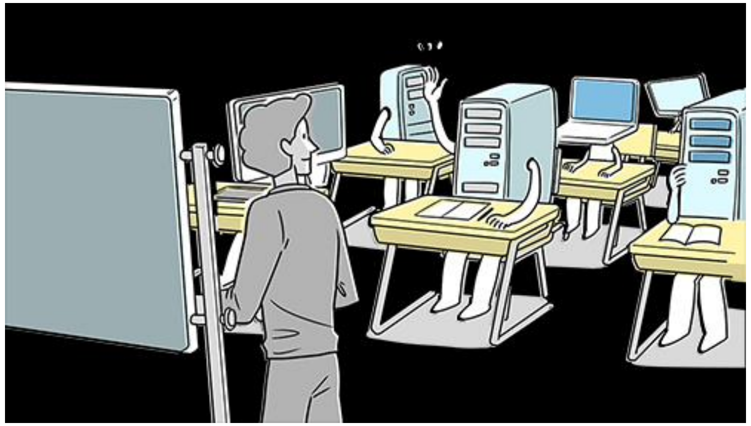
- Used to feed the machine learning models
- Videos with ground truth data from different sensors
  - Generated using real-world data
  - Many options with different configurations
    - Waymo
    - Kitti
    - Argoverse
    - NuScenes
    - ....
- They are used in many cases to validate the accuracy of research

	KITTI	NuScenes	Argo	Ours
Scenes	22	1000	113	1150
Ann. Lidar Fr.	15K	40K	22K	230K
Hours	1.5	5.5	1	6.4
3D Boxes	80K	1.4M	993k	12M
2D Boxes	80K	–	–	9.9M
Lidars	1	1	2	5
Cameras	4	6	9	5
Avg Points/Frame	120K	34K	107K	177K
LiDAR Features	1	1	1	2
Maps	No	Yes	Yes	No
Visited Area (km^2)	–	5	1.6	76

Source: Scalability in Perception for Autonomous Driving: Waymo Open Dataset. Pei Sun

Supervised learning

- The objective is to imitate the concepts taught by the **teacher**
  - For instance let's assume that we have to teach the system to detect a car
- We train the agent by defining where a car is in a given image
  - During the training phase, the agent learns where a car is by testing during certain intervals how good his estimation of a car is
  - A function to determine how good the estimation of the agent, is defined
- In the inference phase the system predicts where the car is



https://www.youtube.com/watch?v=VUrqddjkxok

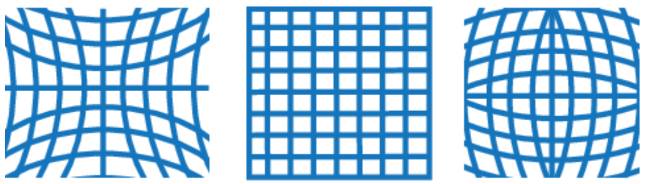
Machine learning

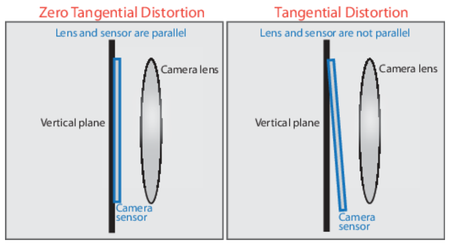
- It is very difficult to teach a system to perform as we want
  - The agent only learns what you teach it
    - It is very easy to “cheat” an agent
- Still a very immature technology for safety related domains

Source:
databasecultures.irmielin.org/how-to-hack-artificial-intelligence/

Camera calibration

- **Most computer vision geometric/algorithmic approaches rely in the quality of the image**
  - **Some cameras introduce distortion to images → this is important because of the shape of the images**
    - **Radial** distortion and **tangential** distortion (they change the shape of the image!!!)
- **To solve the distortion we have to calibrate the camera**
- The process of estimating the parameters of a camera is called **camera calibration**
  - **Intrinsic** are optical, geometric, and digital characteristics of the camera
    - **Focal length**: **distance** between the **lens** and the image sensor
    - **Lens distortion**: radial and tangential
  - **Extrinsic** are parameters that are external to the camera and define the **position** and **heading/orientation** of the camera with respect to the world
- **Idea: We can solve the distortion by using the intrinsic parameters of the camera. How do we obtain these?**





Source: https://de.mathworks.com/help/vision/ug/camera-calibration.html
http://ftp.cs.toronto.edu/pub/psala/VM/camera-parameters.pdf

$$\boldsymbol{x} = \mathbf{P}\mathbf{X}$$

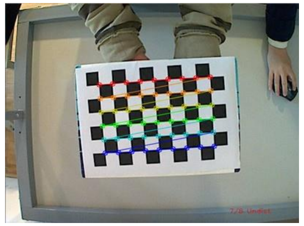
2D image point — camera matrix — 3D world point

$$\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Intrinsic properties
(Optical Centre, scaling)

Camera calibration

- **A camera is a mapping between the 3D world and a 2D image**
- **The process of estimating the parameters of a camera is called <u>camera calibration</u>**
  - The goal of the calibration is to find the matrix that describes the camera parameters
- To solve this problem we provide some sample images with a well-defined pattern, i.e., **a known size and structure like a chess**
  - We find some specific points that **we already know the relative positions** (e.g. square corners in the chess board)
  - **<u>As we know the coordinates of these points in real world space and we know the coordinates in the image, so we can solve for the distortion coefficients</u>**



https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html

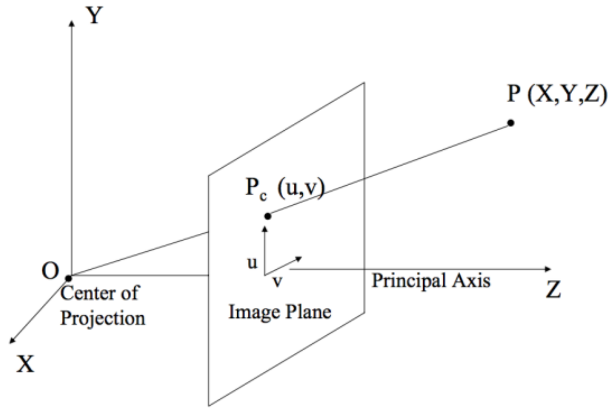
Camera calibration

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

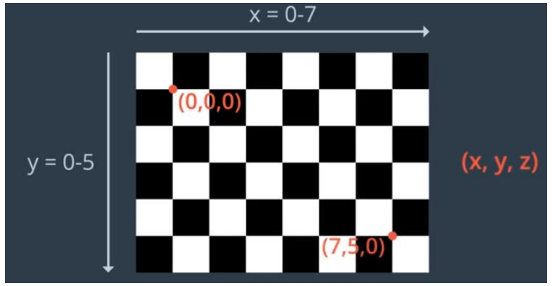
$$x_{corrected} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$
$$y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2 xy]$$

$$Distortion\ coefficients = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3)$$

- We need some method to transform from 3D real world to 2D image coordinates
  - *A lot of math...*
  - We can extract the intrinsic parameters using the camera calibration method
- **As we know the coordinates of the corners in the chessboard image (2D space) and where these points are in the real-world space (3D space), so we can get the values for the distortion coefficients**
  - **<u>In a nutshell: 2D image = 3D image * intrinsic parameters (you now how the 2D image is and how the 3D image is)</u>**
  - We will use the opencv function



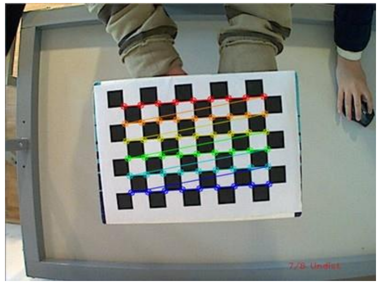
Camera calibration



› The process of calibrating involves two steps
  - We need a chessboard pattern in a flat surface to identify the corners
  - Try to take several pictures of the chessboard on a white surface

› **The distortion is detected considering the difference between the shape of the images and the real size**

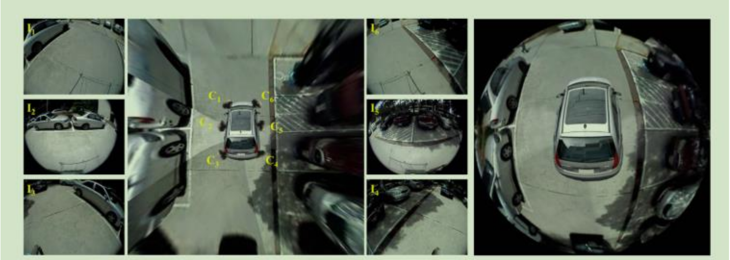
› To measure how good the calibration is we can do either observe the error when projecting the points or by observe the resulting distortion



```
ret, corners = cv2.findChessboardCorners(gray, (6,8), None)
```

Bird’s view

- Elon Musk recently announced a new functionality called “*Vector-space bird’s eye view*”
  - vision system that assists drivers by providing the panoramic image of vehicle surroundings in a bird’s-eye view
  - implemented stitching together 5/6 cameras surrounding the vehicle
  - the **perspective** of the cameras are changed to the ground plane



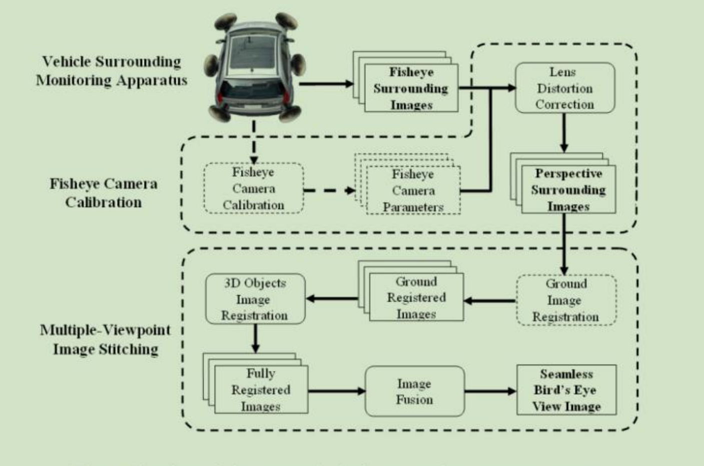
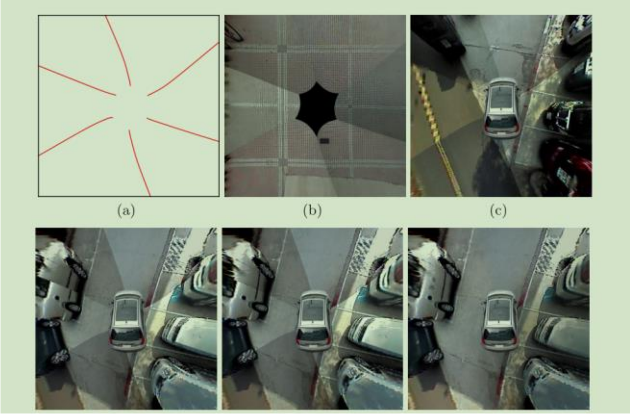


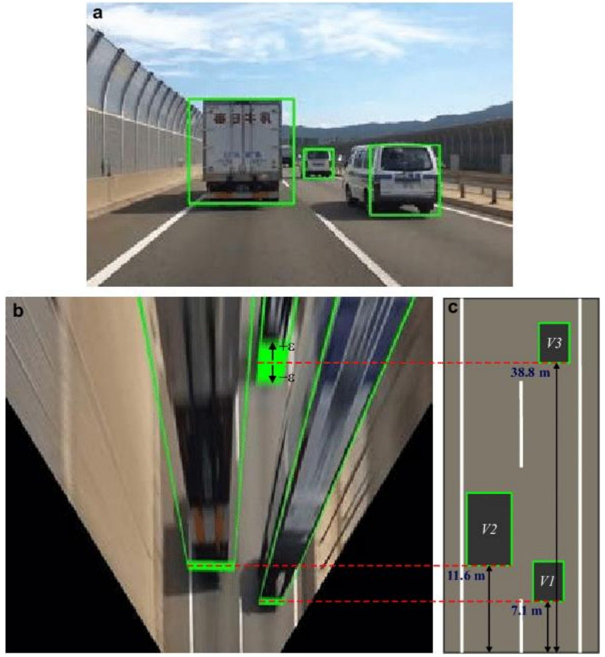
Fig. 2. Flowchart of the proposed vehicle surrounding monitoring system



Source: Bird’s-Eye View Vision System for Vehicle Surrounding Monitoring
Liu YC., Lin KY., Chen YS. (2008)

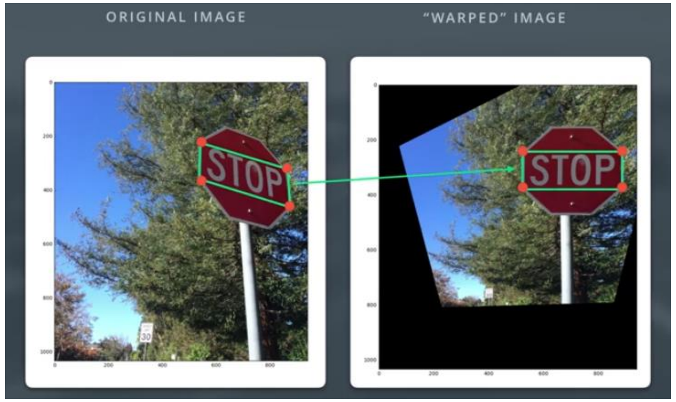
Perspective transform

- **Perspective transform is a method to see the same scene from different viewpoint and angles**
  - **It is used in many applications**
    - **Distance estimation**
    - **Object recognition**
    - Parking assistance (bird view)
    - Advanced lane detection
- The farther away the object is, the smaller the objects appear
  - the ***Z*** information is used to transform the images
- A warp transformation method is used



Perspective transform

1. We first choose 4 points that define the area that we would like to warp
2. We then select 4 points that define the desired rectangle plane for the warp image
3. The function **getPerspectiveTransform** returns a matrix “M” used to change the perspective
4. Next we apply the transform matrix “M” to the original image to get the warped image using the function **warpPerspective**



To Calculate the location of values of each pixel on Destination . Source Pixel values are multiplied with Transformation Matrix as given below .

Step 1 -

Transformation Martix is multiplied with Pixel x,y cordinate values .

$$\begin{bmatrix} 2 & 0.5 & -100 \\ 0 & 2 & 0 \\ 0 & 0.005 & 1 \end{bmatrix} . \begin{bmatrix} 100 \\ 100 \\ 1 \end{bmatrix} = \begin{bmatrix} 150 \\ 200 \\ 1.5 \end{bmatrix}$$

Step 2 -

Values obtiaind of first two rows are divides by third row to obtain (x,y) as given below.

$$[150/1.5, 200/1.5] = [100, 133.33]$$

There for source to Destination Pixel location would be as Below

$$Source\ [100, 100] = Destination\ [100, 133.33]$$

Source: https://medium.com/analytics-vidhya/opencv-perspective-transformation-9edffefb2143

Optional assignment

› **Detect the lines of the road**

– What kind of features can be helpful to identify the lines?

- Color
- Shape
- Orientation
- Position

Algorithm pipeline

1.Grayscaling

2.Gaussian smoothing

3.Canny edge detection

4.Mask region of interest

5.Hough transform

6.Draw lines

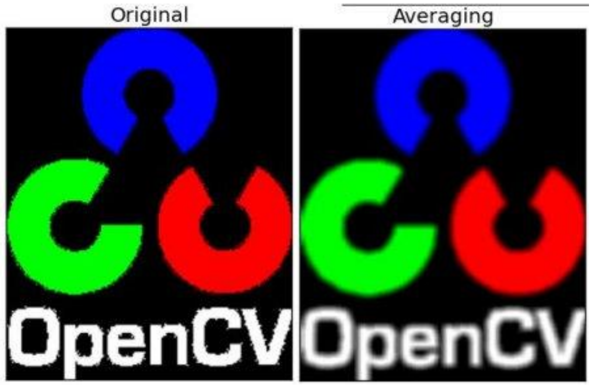
Grayscaling

- The first step is to transform the image into **grayscale**
- **Grayscale** are images with only two colors: *black and white*
  - For many applications, grayscale images are enough to perform computer vision tasks
    - Color information doesn't help us identify important edges or other features
    - Simplify the code
    - Easy to compute


```
import cv2

image = cv2.imread(pathtoimage)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

cv2.imshow('Original image',image)
cv2.imshow('Gray image', gray)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Gaussian smoothing

- Gaussian smoothing/blurring is the result of blurring an image with a Gaussian function
- **Used to reduce image noise or/and reduce detail**
- In terms of image processing, any sharp edges in images are smoothed while minimizing too much blurring




```
import cv2
import numpy
src = cv2.imread('/home/img/python.png', cv2.IMREAD_UNCHANGED)
dst = cv2.GaussianBlur(src,(5,5),cv2.BORDER_DEFAULT)
cv2.imshow("Gaussian Smoothing",numpy.hstack((src, dst)))
cv2.waitKey(0) # waits until a key is pressed
cv2.destroyAllWindows() # destroys the window showing image
```

Canny Edge Detection

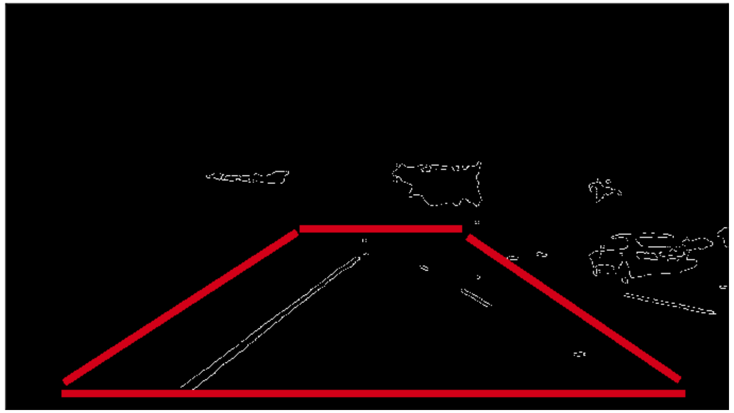
- **Edges correspond to a change of pixels intensity**
- The Canny edge detector is an **edge detection operator** that uses a multi-stage algorithm to detect a wide range of edges in images
  - It is mainly used to **identify the boundaries/edges** of an object in an image
  - For example, if the threshold is [0.1 0.15] then the edge pixels above the upper limit(0.15) are considered and edge pixels below the threshold(0.1) are discarded
- The algorithm follows the next steps
  1. Noise reduction
  2. Finding Intensity Gradient of the Image
  3. Non-maximum Suppression
  4. Hysteresis Thresholding

```
edges = cv.Canny(img,low_th,high_th)
```

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('messi5.png',0)

# Converting color image to grayscale image
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
edges = cv.Canny(gray,100,200)
plt.subplot(121),plt.imshow(gray,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
plt.show()
```

Mask region of interest

- In this phase we select the area or region that may content the lanes
  - We can assume that the camera is fixed



```
# This time we are defining a four sided polygon to mask
imshape = image.shape
vertices = np.array([[(0,imshape[0]),(460, 310), (460, 310), (imshape[1],imshape[0])]], dtype=np.int32)
masked_edges=region_of_interest(edges, vertices)
```

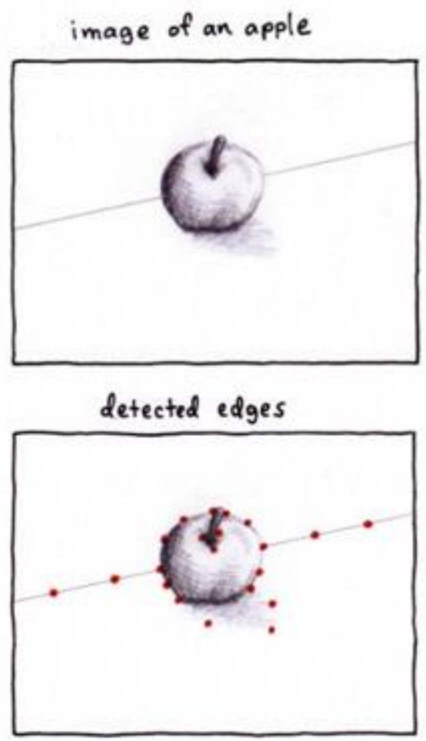
Mask region of interest

- We have to exclude everything that is out of the predefined area
  - “Vertices” are used to represent the region that we will retain, everything else is masked out
  - We remove everything that is out of the **region of interest**

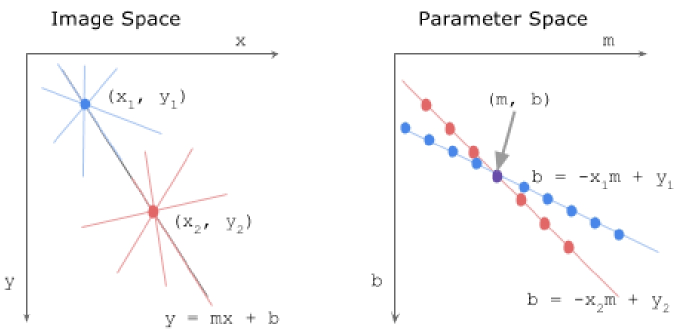
```
def region_of_interest(img, vertices):
    """
    Applies an image mask.

    Only keeps the region of the image defined by the polygon
    formed from `vertices`. The rest of the image is set to black.
    """
    #defining a blank mask to start with
    mask = np.zeros_like(img)

    #defining a 3 channel or 1 channel color to fill the mask with depending on the input image
    if len(img.shape) > 2:
        channel_count = img.shape[2]  # i.e. 3 or 4 depending on your image
        ignore_mask_color = (255,) * channel_count
    else:
        ignore_mask_color = 255

    #filling pixels inside the polygon defined by "vertices" with the fill color
    cv2.fillPoly(mask, vertices, ignore_mask_color)

    #returning the image only where mask pixels are nonzero
    masked_image = cv2.bitwise_and(img, mask)
    return masked_image
```

Hough transform (line detection)

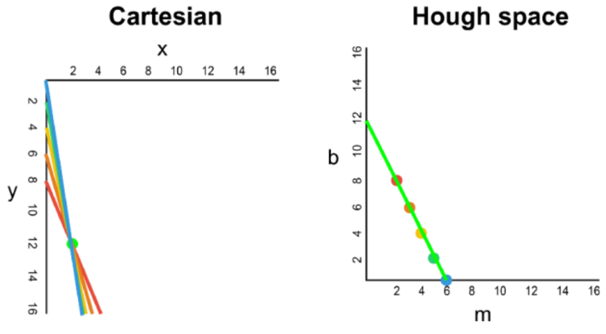
- **How do we know that several pixels in our canny image form a line?**
- The purpose of the Hough transform technique is to find imperfect instances of objects within a certain class of shapes by a **voting procedure**
- **<u>Objective</u>: find as many lines that connect these points in image space**
  - **<u>Idea</u>: Collect the points that form a line and pick the best line model**



Hough transform



- A lines is a collection of points
  - EQ cartesian line: y=mx+c | (**m is the slope, c is the intercept of the y axis)**
- It is possible to use a simpler model that still characterizes a line with one point in the so-called "**m,c space**"
  - a line with the equation y = 2x + 1 may be represented as (2, 1) in Hough space
- A point in the x,y space can be represented as a line in the m,c space
  - For example, a point at (2, 12) can be passed by y = 2x + 8, y = 3x + 6, y = 4x + 4, y = 5x + 2, y = 6x, and so on. These possible lines can be plotted in Hough space as (2, 8), (3, 6), (4, 4), (5, 2), (6, 0)



Source: https://towardsdatascience.com/tutorial-build-a-lane-detector-679fd8953132

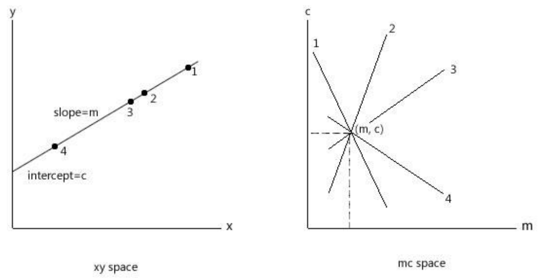
Hough transform

› **Algorithm**: 1) iterates over each pixel, draw the line in the m,c space. 2) Count how many lines intersect. 3) The one with more votes is the best approximation of a line

› For each white pixel, we can draw lines in the m,c space

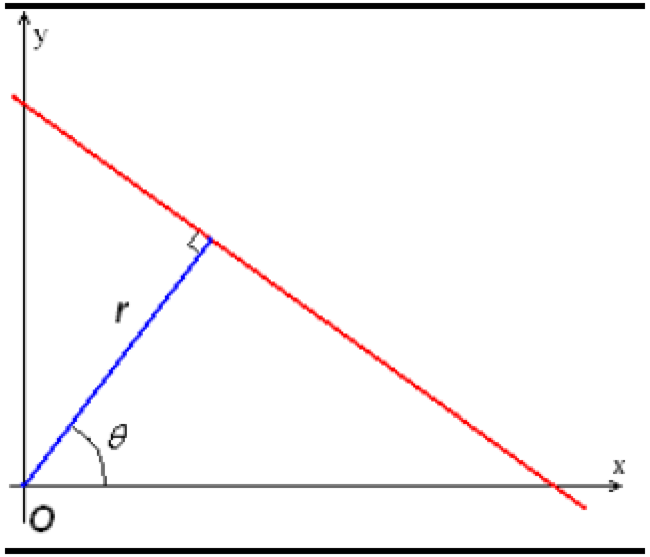
– **The value of *m* tends to infinite for <u>vertical lines</u> (and we don't have infinite space to store infinite lines)**

› We need a different way to represent a line **(Hough space)**



Hough transform

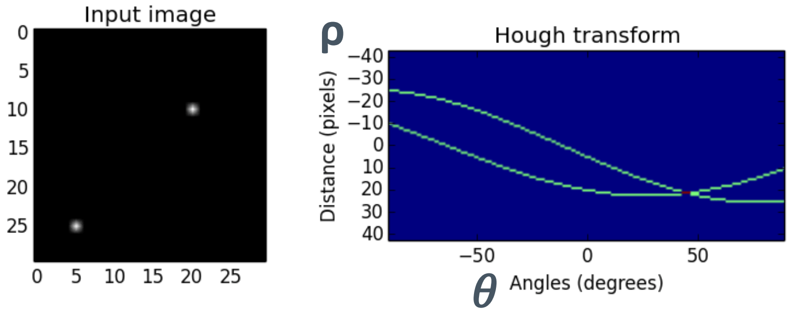
- (Hough space) uses the **polar coordinate** system
  - p (rho) is the perpendicular distance from the origin to the line
  - θ (theta) is the angle formed by this perpendicular line and horizontal axis
  - a line in Polar coordinate system: p=$x\cos\theta + y\sin\theta$
- A line in the x,y space is still equivalent to a point in the p,θ space. **But a point in the xy space is now equivalent to a sinusoidal curve in the p,θ space**



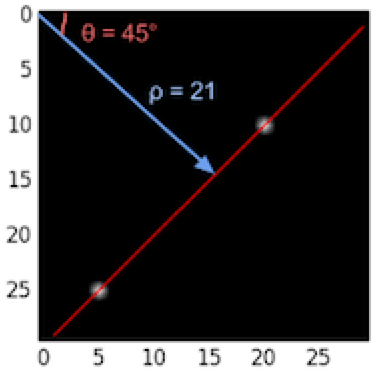
https://aishack.in/tutorials/hough-transform-normal/

Hough transform

- Image of size 30 x 30 pixels with points at (5, 25) and (20, 10)
- The image is transformed to the Hough space by calculating ρ with a point at each angle from −90º to 90º
  - The points in Hough space make a sinusoidal curve
  - **We see that the curves in Hough space intersect at θ =45º and ρ=21**
  - **We can form a line in the cartesian space using as a parameters θ =45º and ρ=21**
- **The more curves intersect at a point, the more "votes" a line in image space will receive**



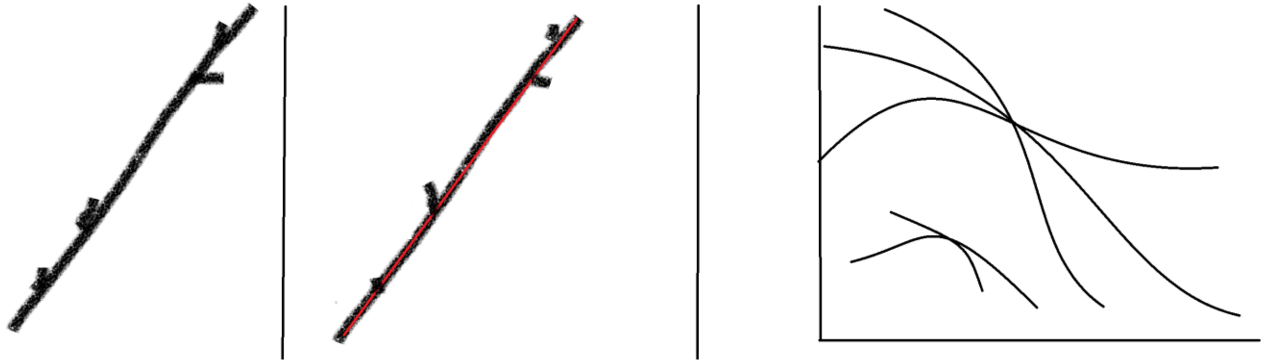
point	$-90°$	$-45°$	$0°$	$45°$	$90°$
(5, 25)	-25	-14	5	21	25
(20, 10)	-10	7	20	21	10



Source: https://alyssaq.github.io/2014/understanding-hough-transform/

Hough transform

- You loop through every pixel of the edge image
  - If a pixel is zero, you ignore it. It's not an edge, so it can't be a line. So move on to the next pixel
  - If a pixel is nonzero, you generate its sinusoidal curve (in the p,θ space).
    - To generate the sinusoidal curve: you take θ = -90 and calculate the corresponding p value. Then, you increase the value of this cell by 1. Then you take the next θ value and calculate the next p value. And so on till θ = +90. And for every such calculation, making sure they "vote“
  - The more curves intersect at a point, the more “votes” a line in image space will receive



Source: https://aishack.in/tutorials/hough-transform-normal/

Hough transform

- **void cv::HoughLinesP**: Finds line segments in a binary image using the probabilistic Hough transform.

```
void cv::HoughLinesP ( InputArray   image,
                       OutputArray  lines,
                       double       rho,
                       double       theta,
                       int          threshold,
                       double       minLineLength = 0 ,
                       double       maxLineGap = 0
                     )
```

Parameters

image	8-bit, single-channel binary source image. The image may be modified by the function.
lines	Output vector of lines. Each line is represented by a 4-element vector (x_1, y_1, x_2, y_2) , where (x_1, y_1) and (x_2, y_2) are the ending points of each detected line segment.
rho	Distance resolution of the accumulator in pixels.
theta	Angle resolution of the accumulator in radians.
threshold	Accumulator threshold parameter. Only those lines are returned that get enough votes ($>$ **threshold**).
minLineLength	Minimum line length. Line segments shorter than that are rejected.
maxLineGap	Maximum allowed gap between points on the same line to link them.

```python
def hough_lines(img, rho, theta, threshold, min_line_len, max_line_gap):
    """
    `img` should be the output of a Canny transform.

    Returns an image with hough lines drawn.
    """
    lines = cv2.HoughLinesP(img, rho, theta, threshold, np.array([]), minLineLength=min_line_len,
                            maxLineGap=max_line_gap)
    return lines
```

```python
# Defining the Hough transform parameters
# Distance resolution in pixels of the grid
rho = 1
# Angular resolution in radians of the grid
theta = np.pi/180
# Intersections in Hough grid cell
threshold = 
# Minimum number of pixels making up a line
min_line_length = 
# Maximum gap in pixels between connectable line segments
max_line_gap = 

lines = hough_lines(masked_edges, rho, theta, threshold, min_line_length, max_line_gap)
```

Drawing lines

- We need to trace a full line that connects the multiple lane markings of the image
  - **We differentiate between left and right lines**
    - Capture negative and positive slopes (/ \)
    - **X increases, y decreases → slope is negative → left line**
    - **X increases, y increases → slope is positive → right line**
- Then we interpolate all the points to form a line using the polyfit function