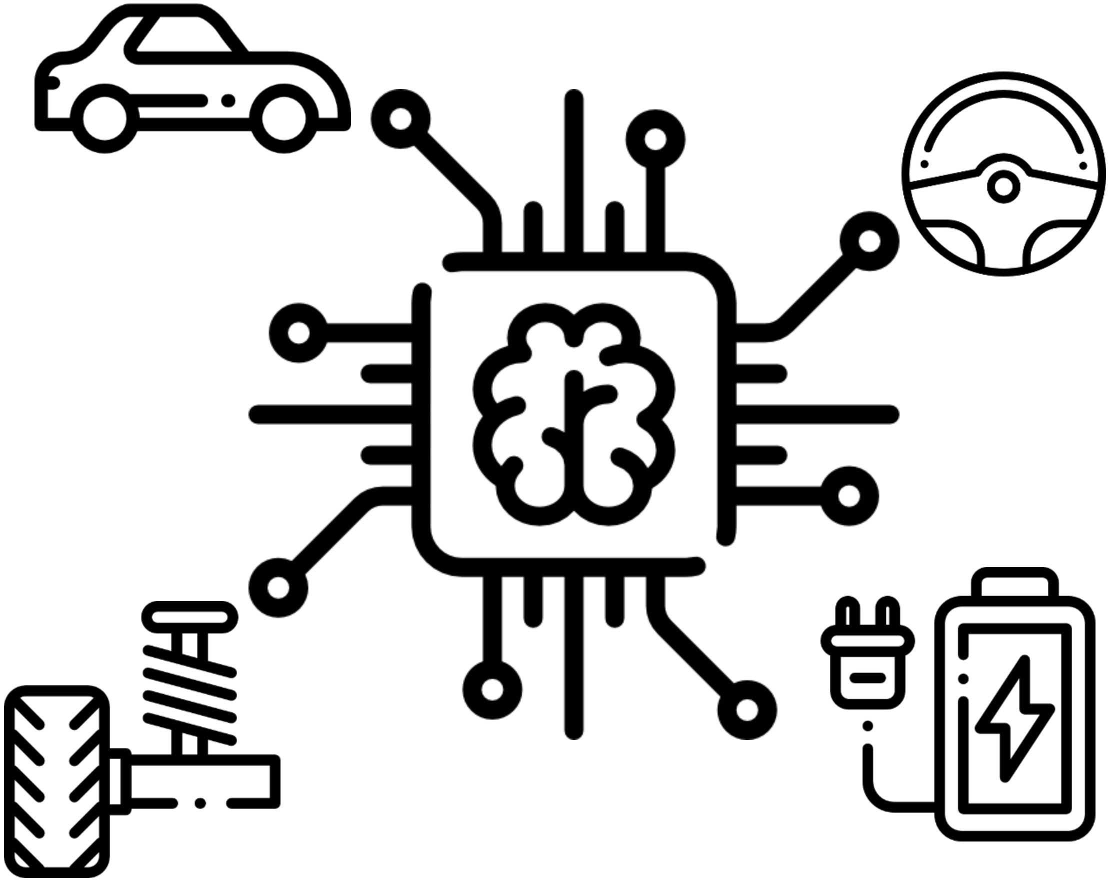
Institute of Automotive Technology
Faculty for Mechanical Engineering
Technical University of Munich

Artificial Intelligence in Automotive Technology

Johannes Betz / Prof. Dr.-Ing. Markus Lienkamp / Prof. Dr.-Ing. Boris Lohmann



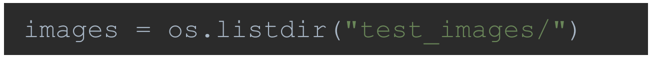
Practice Session 2

- In this practice session we will define a special computer vision pipeline
- The goal of the pipeline is to detect **lane lines** in images and video
- We are using different images to check if we worked correctly
- We are applying our CV Knowhow from the lecture

0. Define the pipeline

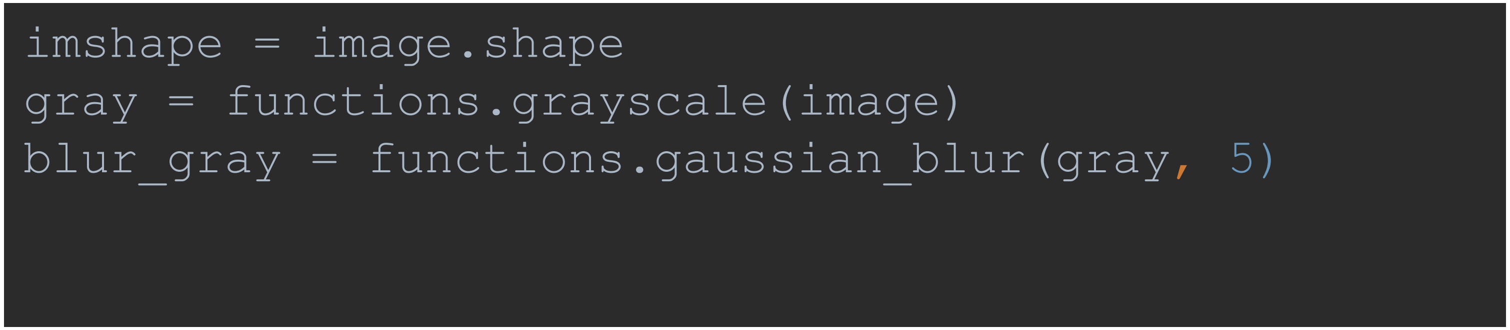
- First of all we have to define the pipeline for processing the image
- We are splitting the code in two parts
  - A function which is called “process_image”
  - A main software part which is loading the image
- In addition, for a better overview we are moving all functions for processing the image to a library called “functions.py”

1. Load an image

- In the second step we are loading the images
- We define one folder called „test_images“
- In this folder we can integrate as much pictures as we want
- With  `images = os.listdir("test_images/")` we can list all images in this folder
- After that we are calling each image in an for loop and we are processing each image in an foor-loop

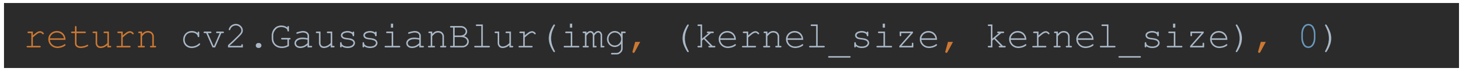
2. Apply Gaussian Blur

- In the first steps of image processing, we have to
  - Get the scale of the image
  - Transfer image to greyscale
  - Apply Gaussin Blur to the image



```
imshape = image.shape
gray = functions.grayscale(image)
blur_gray = functions.gaussian_blur(gray, 5)
```

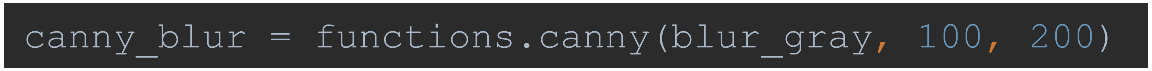
  - The Guassian Blur function is applied with openCV



```
return cv2.GaussianBlur(img, (kernel_size, kernel_size), 0)
```

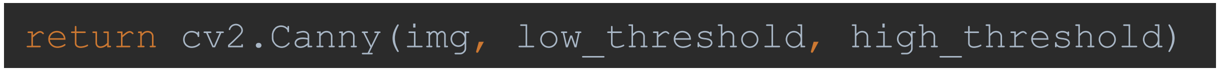
3. Perform Canny Edge Detection

- In the first steps of image processing, we have to apply
  - The canny edge detection algorithm



```
canny_blur = functions.canny(blur_gray, 100, 200)
```

  - The Canny Edge Algorithm is applied with openCV



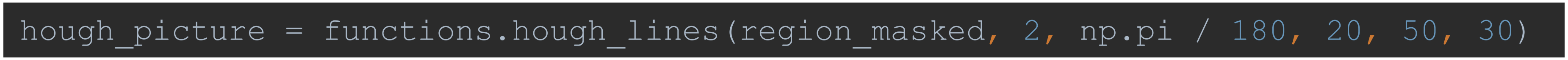
```
return cv2.Canny(img, low_threshold, high_threshold)
```

4. Define a region of interest

- In the third step we have to find a region of interest to lower the computational load
- What could be a good region of interest in this picture?

5. Retrieve Hough lines

- In the last processing step we are applying the hough line detection to the image to find the points which are on one line



```
hough_picture = functions.hough_lines(region_masked, 2, np.pi / 180, 20, 50, 30)
```

- The Hough line algorithm needs a lot of input parameters

6. Apply lane lines to the image

- In the last step we are applying the lines to the image
- We are combining the original image with the hough line image

7. Apply lane lines to vidoes

- Now we can do the same not only with pictures

8. Conclusion

9. Improvements