**** all videos are stored in:**
https://syncandshare.lrz.de/getlink/fiEvYXKnzgPJZkgR3SwmHRgG/

- **Dataset (Data_collection_depth.py):**
  I collect data from the same starting position, in Town 03, at the starting point at "world.get_map().get_spawn_points()[0]" under the same weather, with cars that are chosen randomly each time. The dataset is collected with the help of auto pilot. When using auto pilot, I found the cars make different choices at random, which means, the car may turn left at an intersection this time, and turn right at the same intersection next time. If the number of turning left and turning right are the same, the training model may tend to output going straight, which minimizes the error. So I only keep the datasets, in which the cars always make the same choice, which means they turn right at the first intersection, turn left at the second one and then turn right at the third one. Each time 110 images are collected, this number is chosen such that the number of images where the car is turning left, turning right and going straight are roughly the same. In total 1320 images are collected. In the training notebook I output the number of images belonging to these 3 classes, and it turns out that in 425 images the cars are turning left, in 460 going straight and in the other 435 turning right.
  To match each image with the steering and put them into the neural network as input together, I put them into a dict, which has a length of 1320 data, each has an "image" and a "steer".

- **Training model (training_depth.ipynb):**
  Since I only have 1320 data for training and validation, it's far from enough to train a totally new neural network, I use the transfer learning. The first part of the network, which means the CNN, I use the first part of the pretrained Alexnet. And the second part is a 2-layer fully-connected neural network, working as a classifier, which is not pretrained. I use the Relu function as activation function. In each training step, only the parameters in the classifier are optimized iteratively, the parameters in the pretrained Alexnet are frozen.
  For training the NN, the whole 1320 data are first shuffled randomly and then 1100 are used for training and the other 220 for validation.
  First I tried to train the NN directly with the original steering data, and the loss was always high with different learning rate. I thought the reason might be that the steering data is too small, most of them were between 0.5 and 0.5, which leads to the vanishing gradient problem. So I multiply all steering data with 100 and tried again, and the loss was much smaller. For the hyperparameters, I tried out different combinations and setting the learning rate to be 0.9e-04, the batch size 50 and number of epochs 25 works best for me.
  To check the output of the network, I test it with several randomly chosen data, and the difference seems to be small, smaller than 0.01.

- **Using model in model_inference.py (model_inference_depth.py; model_inference_depth_visualize.py):**
  Before loading the trained model into model_inference.py, I first tried with the a dummy model, which always output a random number between 1 and 1. The real-time image is captured by camera.listen() function and the image is stored under image.raw_data. I tried to visualize the real-time images (in the model_inference_depth_visualize.py) and they look different from these in my dataset. Then I found out that the images are converted to gray-scale before saving to disk, which I used for training. To reduce the difference, I convert the image.raw_data to gray-scale image according to the official document. After that, the visualization looks good and the steering is randomly chosen, just as I wrote in the dummy model.

Then I tried to run the model_inference.py with the trained NN. The model is saved with "torch.save(steer_model.state_dict(),PATH)", which only saves the state, not the whole model and doesn't require large space. The model is loaded with "load_state_dict(torch.load(PATH))". In each frame I output the steering computed from the model and check if it is the same with the actual steering data output with "vehicle.get_control().steer)".

- **Add 2 extra cameras (data_collection_2extra.py; training_2extra.ipynb; model_inference_2extra.py)**

To add 2 more cameras, I put an extra depth camera on each side, the position and the angle are both fixed. The data are collected in the same way and 1320 data pairs are collected, each with 3 images and a steering data. I combined the 3 images into 1 with 3 channels and then put this new image into the dict, together with the corresponding steering, the same way as with only 1 camera. The network is the same, with pretrained Alexnet and a not pretrianed 2 -layer classifier. I trained these 1320 data with totally the same hyperparamer, with also 25 epochs.

To use all 3 real-time images, two extra camera.listen() are added in the model_inference.py. In each frame, the combined 3-channel image is collected as input for the trained neural network and the ouput steering is used to control the car.

- **The outcome:**

(1) with only one depth camera (/videos/depth):

First I tried with the same start position I used to collect the training data. On the straight road the car runs quite good with the throttle=0.5. At the intersection, it does have the problem discussed in the class, that there is always some delay in comparison to the auto pilot, which means the car needs more time to react before turning left or turning right. So it may sometimes hit the fountain or the building.

roughly 3 of 10 times the car hits the fountain

roughly 7 of 10 times the car runs on the road

Then I tried to run the model under dynamic weather, and it seems to run in the same way.

As I always use the same starting point in the same town for training, I thought my model can not handle other starting positions, but it turns out to be not that bad. The car can even run well under dynamic weather. However, it is not that robust and always end up with

roughly 3 of 10 times the car hits the fountain

roughly 7 of 10 times the car runs on the road

some obstacle. With only 1320 data pairs all at the same starting position, it is quite nice that the model can handle some situation that it has never seen.

(2)

In another town (Town04), without dynamic weather

In another town (Town04), with dynamic weather

cameras(/videos/2_extra):

With 2 extra cameras, it is strange that my model seems to be less robust. Which means I have to set the throttle=0.3 so that the car knows where to go, which means the model need more frames and more time to output the right steering. The improvement is that the cars can turn left/right without so much delay in comparison to only 1 camera.

The model can also handle dynamic weathers but cannot handle different start positions.

roughly 3 of 10 times the car turns right with some delay

roughly 4 of 10 times the car turns right without delay

As the model with extra 2 cameras is not that robust, I tried to train the 3-camera model once again, this time, I put some weight to the images. The images from the left and right cameras are multiplied by 0.5 and from the middle multiplied by 1, which means the middle camera plays a more important role in comparison to the other two. The outcome

Output the same way with dynamic weather

not robust at other starting positions

is however even worse than the model before. Since some may try to turn left/right too early, which hits the obstacles on the side.
(training_2extra_2.ipynb; model_inference_2extra_2.py; /videos/depth):

- 

roughly 3 of 10 runs well

Roughly 7 of 10 hits some obstacles

To get robust outcome, 1320 images are far from enough, but with the help of transfer learning, the outcome seems to be not bad. However, with the 3-camera model, it may be not a good idea to combine the 3 images into a 3-channel image, as the image becomes much more complicated and the outcome is not robust, even setting the start position to be the same as the training dataset. The weather does not have big impact on the model, since the input of the model is gray-scale image, they look quite the same with different weather. But changing the start point or even map is a big challenge for the model and need more dataset to get a better result.