Lab Guide
LaneNet Lane Estimation

Content Description

The following document describes the implementation of the LaneNet lane estimation in python environment.

Lab Description

In this example, we will capture RGB images from the Intel RealSense camera, use LaneNet to estimate lane locations on the image, overlay the RGB image with the estimation lane, and display result.

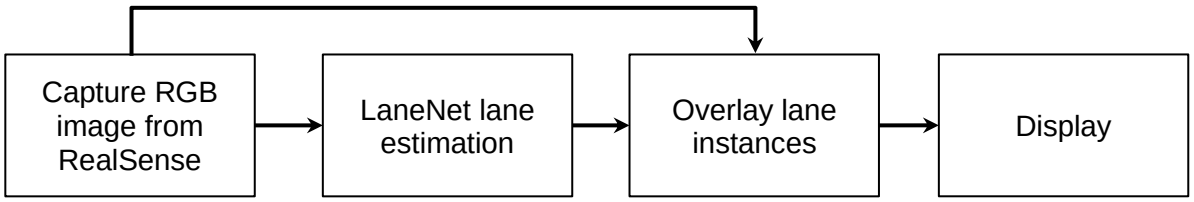

Figure 1. Component diagram

Python

Running the example

1. Check User Manual – Software Python for details on deploying python scripts to the QCar2. The output of the **cv2.imshow()** function should look like Figure 2.

2. When running the script for the first time, make sure the QCar 2 is connected to the internet, as a PyTorch model train by Quanser will be downloaded. In addition, the downloaded model will be converted to a TensorRT engine to improve inference time. The whole process can take up to 20 minutes.

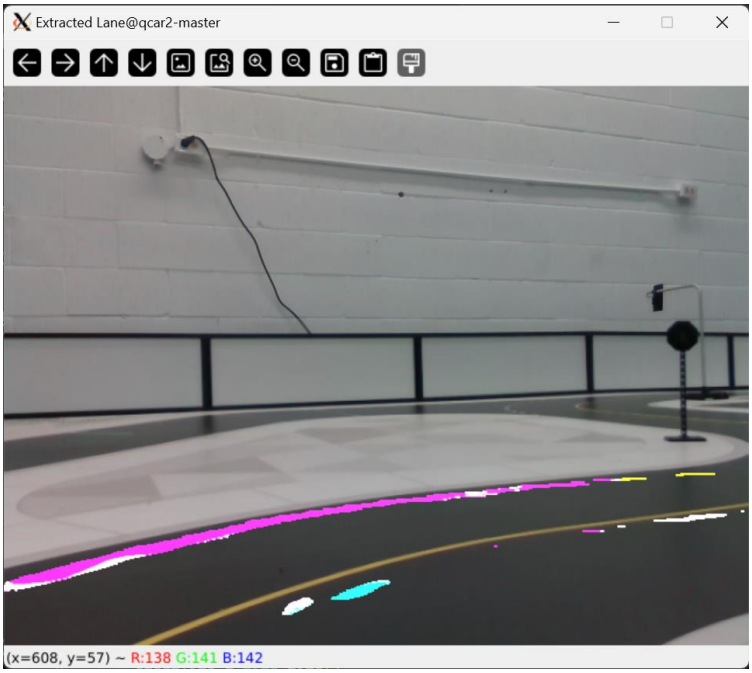

Figure 2. Annotated images of the lanes

Details

1. Initialization

```
myLaneNet = LaneNet(modelPath = 'path/to/model',
                    imageHeight = imageHeight,
                    imageWidth = imageWidth,
                    rowUpperBound = 240
                    )
```

To initialize the LaneNet model, an optional path to a pretrained model can be provided. Otherwise, the default LaneNet model trained by Quanser will be loaded. The height and width of the image stream also need to be provided at initialization, as they will be used to resize the predictions so they can be overlaid with the original image. The **rowUpperBound** defined the row index to crop the input video, which is often the row corresponding to the horizon. A **rowUpperBound** of 0 indicates the entire input image is used as input. After initialization, the LaneNet model is created and stored under the **myLaneNet.net** attribute. This implementation of LaneNet is based on Iroh Cao's implementation.

2. Pre-processing and Prediction

```
rgbProcessed=myLaneNet.pre_process(myCamRGB.imageBufferRGB)
binaryPred , instancePred = myLaneNet.predict(rgbProcessed)
```

Before prediction takes place, the input image needs to be pre-processed into the tensor with a dimension of 1x3x256x512 (BxCxHxW), as well as applying the same batch normalization as the training set.

The processed image should be used as the input for the **predict()** function. The outputs of the **predict()** function are the 512x256 binary image which indicates the location of all estimated lane markings (**binaryPred**), and 512x256x3 embeddings which can be used to determine separate instances of lane markings (**instancePred**).

3. Render

```
annotatedImg = myLaneNet.render(showFPS = True)
```

The **render()** function returns the annotated RGB which has overlaid estimated lane markings. The **cv2.bitwise_and()** function is utilized to combined the binary lane location estimation and embeddings of different lanes. Since the embeddings are configured to have a dimension of 3, it can be easily visualized. Lane instances that are closer in the embedding space will appear to have more similar colors.

4. Performance considerations

The **render()** is a time-consuming process, as their operations are not optimized. For practical self-driving applications, please consider not rendering the predictions for reduced computation time.