LabMLISP: Lab Course Machine Learning in Signal Processing

Exercise II: Road simulator

Prof. Veniamin Morgenshtern

Author: Sebastian Lotter, Ahmad Aloradi, and Veniamin Morgenshtern

The goal in this exercise is to build a road simulator. The simulator is intended to create artificial images of a cartoon-like road together with a binary image that indicates pixels belong to a lane.

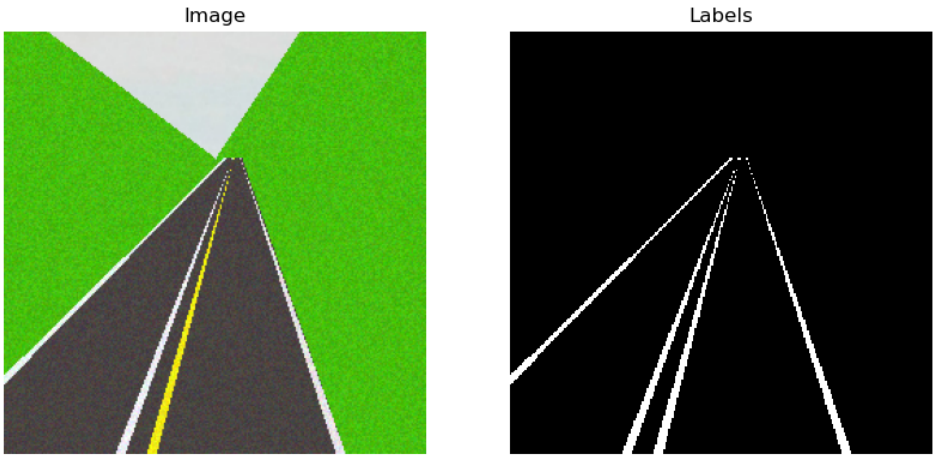


Figure 1: Sample output from the simulator

This data will then be used to perform supervised learning with a neural network for lane recognition in a later exercise. The image will serve as input to the network and the binary image will provide the labels for training the network.

Exercise Materials

You are given a skeleton code of the simulator and you are required to complete the missing code. To access the lab materials for this task, execute:

```
$ cd ~/labmlisp

$ cp -r ~/SHARED/DATA/mlisp-lab/ps2-simulator/* .

$ cp -r ~/SHARED/DATA/mlisp-lab/data .
```

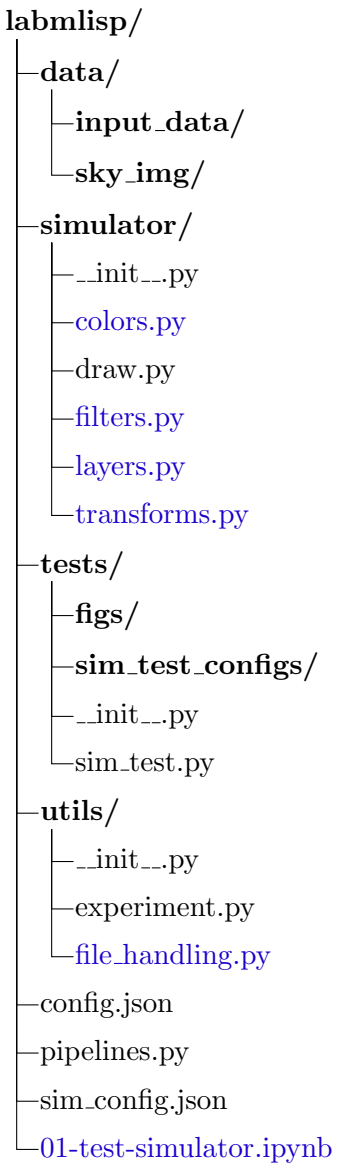


Figure 2: The file structure of the simulator. The folders are indicated in bold-face font and the files that needs to be completed as part of this task are indicated in blue.

The file structure of the simulator is given in figure 2. Make sure that you have the files listed in the figure in your working directory i.e., **labmlisp**. In case any file is missing, please inform the lab tutors. You also may find some extra files that are not listed in the figure. You can safely ignore those extra files as you will not be directly working on those files in this task. But please don't delete those files!

In the working directory, you will find the **01-test-simulator.ipynb** notebook. This is for testing your simulator. After each step, you can test your implementation and visually compare your results with the expected results.

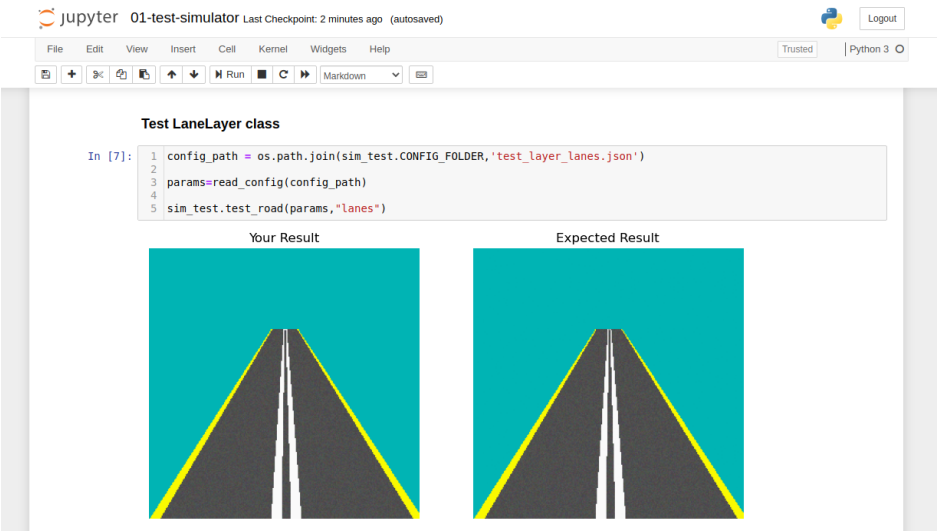


Figure 3: Visual inspection can be done by comparing results from your implementation with the expected result using the **01-test-simulator.ipynb** notebook.

Structural Overview

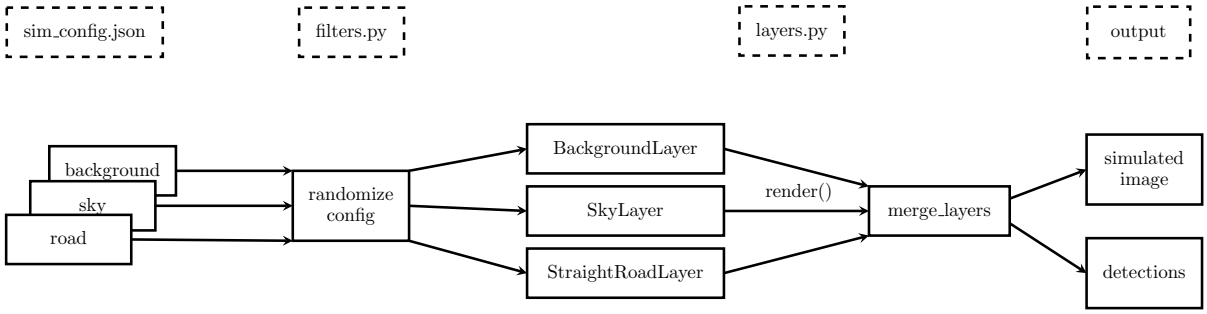


Figure 4: Structural Overview of the simulator

The figure 4 shows a high-level overview of the simulator. Inside the **simulator** folder, you will find the following python files: **colors.py**, **draw.py**, **filters.py**, **layers.py** and **transform.py**. These files are used to generate the simulated road images and their corresponding lane detections. These files contain incomplete pieces of code. The task is to complete the missing parts of the code in the files such that the simulator generates the required road images.

The file **sim_config.json** contains an example of the simulator configuration. This file provides a dictionary specifying those configurations. Configuration of the various layers and their composition is completely up to the user. The file **sim_config.json** is complete and need not to be modified. There are 'filters' (see **filters.py**) that randomize the layers configuration to provide a wide variety of images for training the neural network. The user is, however, free to manually access the dictionary entries to change some certain configurations of the layers. In the directory **sim_test_config**, we provide various simpler configuration files which would be used for testing in the **01-test-simulator.ipynb**. You are encouraged to read through the relevant test config file

from this directory before implementing each component of the simulator.

The simulated images consist of several overlayed layers. The module **layers.py** provides implementations of some basis layer types that together will form the simulated images. It consists of one base class and five derived classes as shown in figure 5. Each of the derived classes implements a different layer in the image. For every layer, there is an associated binary image (fmask), with ones corresponding to the pixels of the layer itself. For example, the background layer has a bitmask that is defined as an all ones array, since all of the background should cover the whole image.

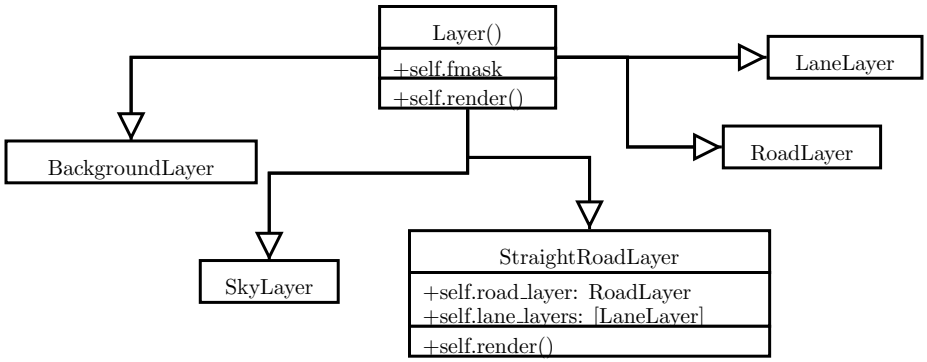


Figure 5: Layer Class Hierarchy

The simulator has two outputs: cartoon-like road images and lanes detections. The simulator renders the image in the following way: obtaining the configuration of each layer, applying the filters, and then merging the different layers. At the end of this file, a suggested roadmap is supplied to help you throughout the process of developing the simulator. It is advisable to keep testing your results in the notebook file **01-test-simulator.ipynb**

Recommended Road Map

Hint: To test the functionality below it might be useful to use the Python / Jupyter debugger.

1. Complete the missing code in `BackgroundLayer` class in **layers.py**
2. Complete `color_w_constant_color` in **colors.py**.

   ——— test ———

3. complete `color_w_noisy_color` in **colors.py**.

   ——— test ———

4. complete `init_transform` and `project` functions in **transforms.py**.

   ——— test ———

5. Complete `RoadLayer` in **layers.py**.

   ——— test ———

6. Complete `LaneLayer` in **layers.py**.

   ——— test ———

7. Complete `get_image_list` in **file_handling.py**.

   ——— test ———

8. Complete `SkyLayer` in **layers.py**.
9. Complete `color_w_image` in **colors.py**.

   ——— test ———

10. Complete `TiltRoadFilter` in **filters.py**.

    ——— test ———

11. Complete `ShiftRoadFilter` in **filters.py**.

    ——— test ———

12. Complete `LaneWidthFilter` in **filters.py**.

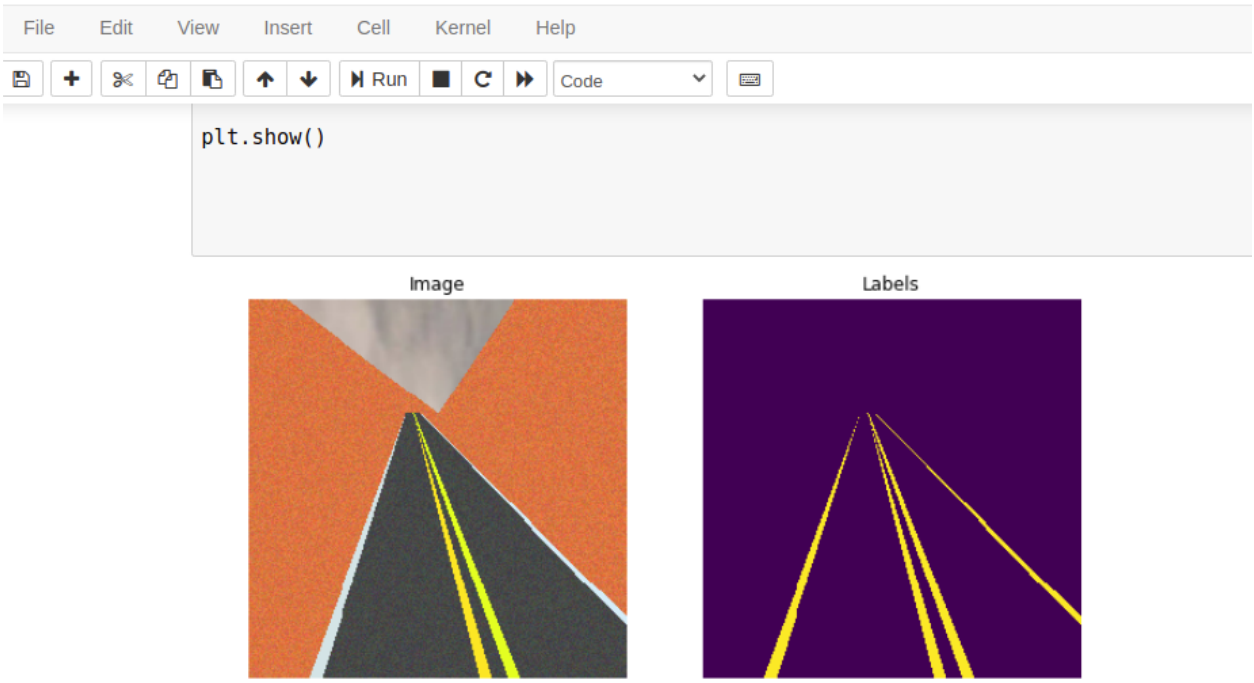
    ——— test ———

13. Complete last two cells in the notebook **01-test-simulator.ipynb**.

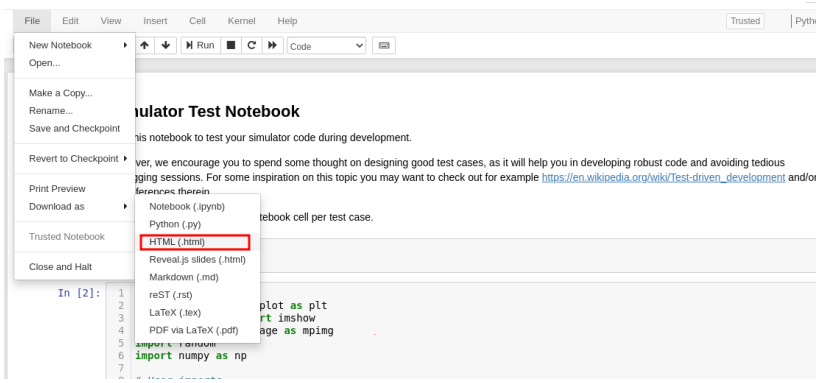
    ——— test ———

Submitting your work

- After you complete all the steps mentioned in the previous section, show your completed Simulator Test notebook to the tutor. The last cell of the notebook should look something like:



- Convert the notebook into a HTML file as shown below



- Upload this HTML file to StudOn.

Hints

1. To bind the function to a parameter, you need to use partial, please see how the `partial` function is used in **simulator.py**. Here is a basic example of using this function

```
from functools import partial

def add(x,y):
     print('current value of x is:', x)

     print('current value of y is:', y)
     return x+y

binded_add = partial(add,x=1)
binded_add(y=3)
```