LabMLISP: Lab Course Machine Learning in Signal Processing SS 2019

Part II: Road simulator

Prof. Veniamin Morgenshtern

Author: Sebastian Lotter, Ahmad Aloradi, and Veniamin Morgenshtern

Your next goal is to create a road simulator that will be used to create images for training the network.

Study object oriented programming

The system you will build will use the object oriented programming framework. If you are not familiar with object oriented programming in Python, begin by going through a short tutorial notebook that is available at the shared folder `/SHARED/DATA/mlisp-lab/OOP_Tutorial/`. Copy the notebook by running the following command in your working directory.

```
$ cp ~/SHARED/DATA/mlisp-lab/ps2_oop_tutorial/*.ipynb .
```

If needed, here is a link to a more comprehensive object oriented programming tutorial:

`https://python.swaroopch.com/oop.html`

Study debugging in Python / Jupyter

In this laboratory you will be asked to complete a lot of code. It's best to make your work modular and after implementing each function or class test that the function you have implemented works as expected.

Even if your code does not work as a whole, you can still test individual functions. For this a built-in Jupyter debugger is very useful. In the end of the `OOP_Tutorial.ipynb` notebook you will find a simple example on how to use the debugger.

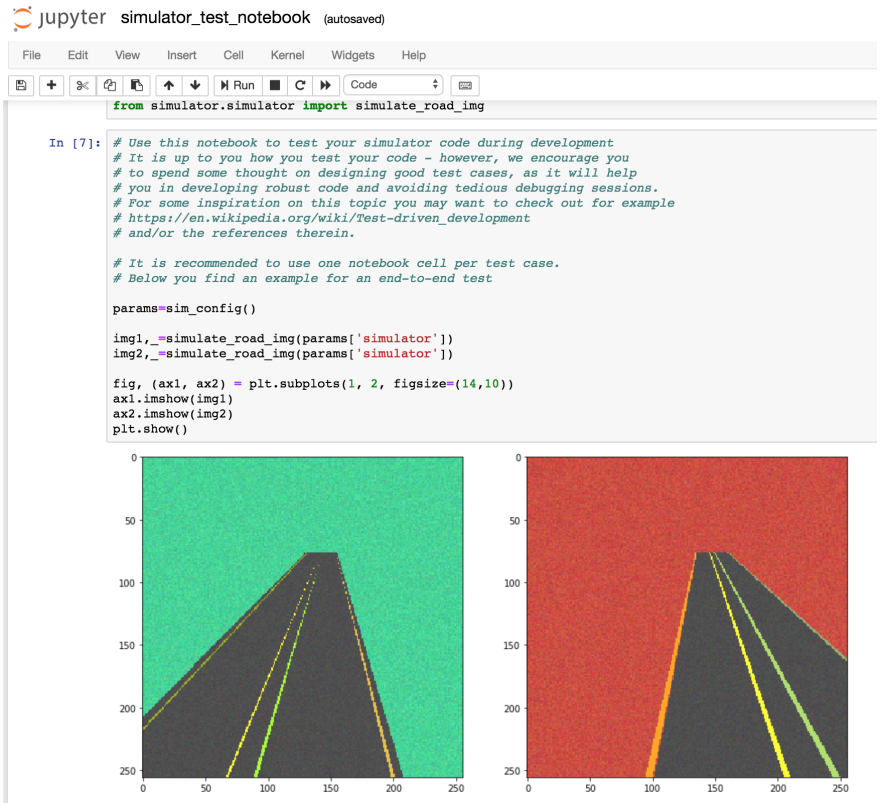
Create road simulator

You are given a prototype of the simulator and you need to reconstruct the missing details. To access the code, execute:

```
$ cd ~/labmlisp
```

```
$ cp -r ~/SHARED/DATA/mlisp-lab/ps2_simulator/* .
```

In the working directory your will now find the `simulator_test_notebook.ipynb` file. This is for testing your simulator. After you are done, the output of the fourth cell in `simulator_test_notebook.ipynb` notebook should look like this:

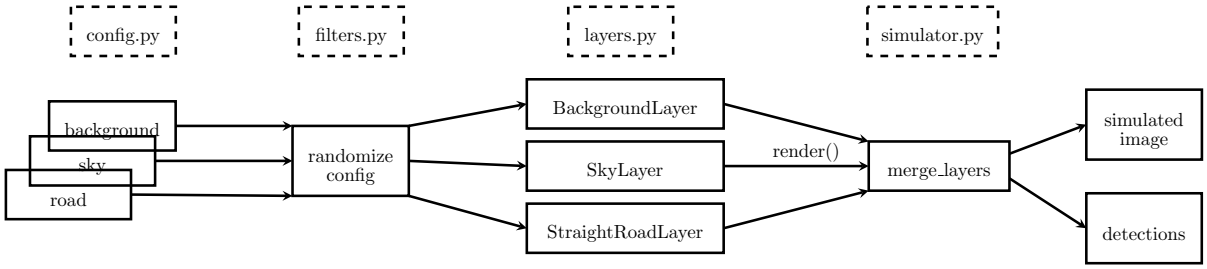


In the working directory your will now find the `simulator` subfolder with `colors_ps.py`, `filters_ps.py`, `layers_ps.py`, and `simulator_ps.py` files. Make copies of these files and call them `colors.py`, `filters.py`, `layers.py`, and `simulator.py`, respectively. Follow the instructions found in the files and fill in the missing code. Test your code as you develop it by making new cells in `simulator_test_notebook.ipynb` and calling the corresponding functions.

In your work you can use the following sequence:

- Start by implementing one coloring function (`color_w_constant_color` would be a good choice) from `colors.py`.
- Reduce the `config.py` to a single layer: `BackgroundLayer`.
- Use `color_w_constant_color` together with this single layer. This makes a complete working simulator, creating deterministic background images.
- Test that this basic simulator works.
- Now you can extend your simulator by implementing functions in `layers.py` and `colors.py` in any arbitrary order. Every time you have implemented some new layer or color, you can test it by adopting the config and running the simulator.
- `filters.py` should be the last file you work on, this one is for adding randomness and can be implemented in the end.

Structural Overview

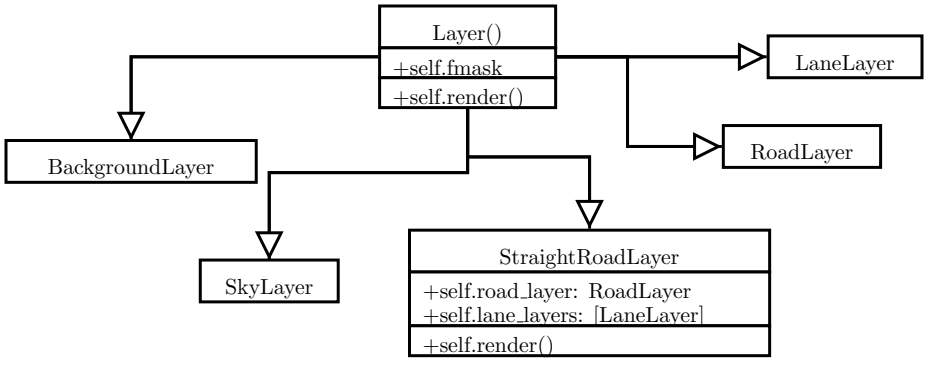


The figure above shows a high-level overview of the simulator. Inside the simulator directory, you will find the following python files: `config.py`, `layers.py`, `colors.py`, `filters.py`, `simulator.py`, `utils.py` and `__init__.py` . These files together should generate the simulated road images and 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 `config.py` contains an example configuration of the simulator. The file has only one function that returns a dictionary specifying those configurations. Configuration of the various layers and their composition is completely up to the user. The file `config.py` is complete and need not to be filled. 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.

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 the below figure. 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.

Layer Class Hierarchy



The simulator (`simulator.py`) 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 pdf 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 `simulator_test_notebook.ipynb`

Hints

1. If you get an indentation error in `filters.py`, its because the constructor(`__init__`) is not initialized. This is part of your work. You can either choose to write `pass` in the those constructors or write the code for those constructors right away.

2. To bind the function to a parameter, you need to use partial, please see how partial 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)
```

More information can be found online.

Recommended Road Map

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

1. type the statement `pass` after each empty constructor in the file `filters.py`, specifically in lines 27, 41, 55, 70.
2. complete `color_w_constant_color` in `colors.py`.
3. complete `color_w_random_color` in `colors.py`.
4. complete `BackgroundLayer` in `layers.py`
5. reduce `config.py` to a single layer: `BackgroundLayer` by commenting lines 29-79.

——— test ———

6. complete `color_w_constant_color_random_mean` in `colors.py`.
7. uncomment lines 29-43 in `config.py` and change the value of `'prob'`, in line 32, from 0 to 1.

——— test ———

8. complete `StraightRoadLayer` and `LaneLayer` in `layers.py`.
9. uncomment lines 44-48 and line 79 in `config.py`.

——— test ———

10. complete `TiltRoadFilter` in `filters.py`.
11. uncomment lines 49-78 in `config.py`.

——— test ———

12. complete `ShiftRoadFilter` in `filters.py`.

——— test ———

13. complete `ShiftLanesFilter` in `filters.py`.

——— test ———

14. complete `LaneWidthFilter` in `filters.py`.

——— test ———