AUTONOMOUS VISION GROUP
PROF. DR.-ING. ANDREAS GEIGER

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

SELF-DRIVING CARS
EX. 5 – MODULAR PIPELINE

Deadline: 31th January 2022; 8pm

For this exercise you need to submit a **.zip** folder containing your **.py** code files (namely the files *lane_detection.py, waypoint_prediction.py, lateral_control.py, longitudinal_control.py*) and the filled out submission form (*submission.txt*). Please use the provided code templates for these exercises. The given `modular_pipeline.py` file will be used for evaluating your code. Add your best choice of parameters to the `modular_pipeline.py` file. You are not allowed to use learning based approaches for any subtask e.g., deep learning based approaches for lane detection or imitation learning/reinforcement learning for planning or control. Since we are not using learning methods, the use of PyTorch (or any other deep learning/GPU acceleration libraries) is not allowed. In detail, you are not allowed to use packages other than Numpy and Scipy. Additionally, changes to the gym environment will not be considered.
As you do not need a GPU, you should be able to run the code on your local machine. You are however allowed to use the TCML cluster if required (do not request a GPU, please set `--gres=gpu:0` in your .sbatch file).
Comment your code clearly, use docstrings and self-explanatory variable names and structure your code well. For the challenge we only evaluate your submitted pipeline. But we also provide some questions and tasks to deepen your understanding of the topic.

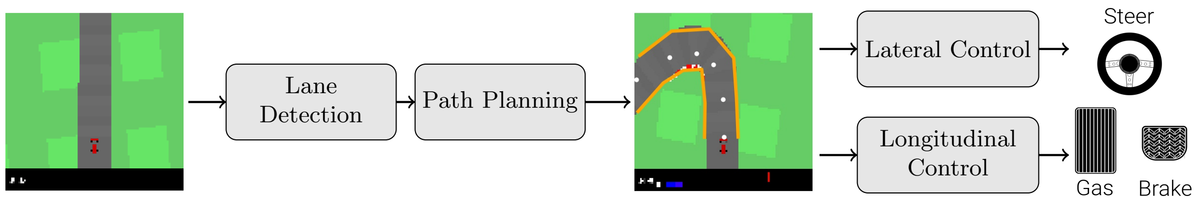


Figure 1: Modular pipeline consisting of a lane detection module, path planning and a control unit.

5.1 Setup

We first ensure that our environment is up-to-date and all required packages are installed.

a) Install a new version of the gym environment on your local conda environment, which can be downloaded here: https://drive.google.com/file/d/1J5TVuATqaCxZJiR5ohbo7NEVZvhYWp53/view?usp=sharing (*Please follow the instructions from Exercise 0*). We have changed some function signatures in this new gym environment and thus the gym environments from the previous exercises are no longer compatible.

b) Install `scipy` on your local conda environment with the command `pip install scipy`.

c) To run your code on the TCML cluster use following singularity image available here: https://drive.google.com/file/d/1naBCcx-bm4PU9npmY09kyVa48huIsFpV/view?usp=sharing (*The singularity image is updated to include all dependencies, you should be able to run your code with the following line* `singularity exec sdc_new_gym21.simg python modular_pipeline.py --score` *in your .sbatch file. Note that you should always use .sbatch files to submit jobs in the TCML cluster and not run jobs on the login nodes*).

5.2 Lane Detection

The first module of the pipeline aims to detect the lane boundaries by finding edges in the state image, assigning the edges to a lane boundary and fitting splines to the point set of the lane boundaries. Please find the module template class in `lane_detection.py` and use `test_lane_detection.py` for testing.

a) Edge Detection: Let's start with the edge detection. First, implement the function `LaneDetection.cut_gray()`. It should translate the state image to a gray scale image and crop out the part above the car. Second, derive the gradients of the gray scale image in `LaneDetection.edge_detection()`. In order to ignore small gradients, set all gradients below a threshold to zero. Third, write a function that derives the maxima of the thresholded gradients per pixel row. *Hint: To find the maxima in each row, you can use for example* `scipy.signal.find_peaks()`.

b) Assign Edges to Lane Boundaries: Given a set of edges (maxima) and assuming that these edges are on the lane boundaries, we would like to assign each maximum to one of the two lane boundaries. We suggest the following approach: find the maxima in the image row closest to the car. Now, by searching for the nearest neighbor edges along each boundary, we can assign the edges to the lane boundaries. The function `LaneDetection.find_maxima_gradient_rowwise()` detects the initial edge. In `LaneDetection.lane_detection()`, fill in the missing code. *Note: you are free to improve upon our suggested approach.*

c) Spline Fitting: For fitting the lane boundaries it is common to use parametric splines. In `LaneDetection.lane_detection()`, fit the spline as documented in [1] to each lane boundary. We are using a parametric spline, which allows us to sample points given a single parameter that represents the length of the curve. As we shall see in the following, such a representation would help us plan the path of the vehicle given the lane boundaries.

d) Testing: Test your implementation by running `test_lane_detection.py`. You can drive using the arrow keys and check the determined lane boundaries in the additional window. Try to look for failure cases. Find a good crop for the part above the car, a good approach to assign edges to lane boundaries and a good choice of parameters for the gradient threshold and the spline smoothness.

5.3 Path Planning

This section is about planning the path as well as setting the speed of the car.

a) Road Center: A simple path for the car would be to follow the road center. Implement function `waypoint_prediction()` to output N waypoints at the center of the road in file `waypoint_prediction.py`.

Towards this goal, use the lane boundary splines and derive lane boundary points for 6 equidistant spline parameter values. *Hint: For best results the first waypoint should be as close to the vehicle as possible i.e. spline parameter values should include 0.* Next, determine the center between lane boundary points corresponding to the same spline parameter, respectively. Use the script `test_waypoint_prediction.py` to verify your implementation. Again, try to find situations where waypoint prediction fails.

b) Path Smoothing: Since we are creating a fancy racing car, we need to tune the waypoints to the road's course, e.g. cutting the corners. To this end, we smooth the path by minimizing the following objective:

$$\underset{x_1,\ldots,x_N}{\operatorname{argmin}} \sum_i |\mathbf{y}_i - \mathbf{x}_i|^2 - \beta \sum_n \frac{(\mathbf{x}_{n+1} - \mathbf{x}_n) \cdot (\mathbf{x}_n - \mathbf{x}_{n-1})}{|\mathbf{x}_{n+1} - \mathbf{x}_n| \, |\mathbf{x}_n - \mathbf{x}_{n-1}|}. \tag{1}$$

Here, $\mathbf{x}_i \in \mathbb{R}^2$ are the waypoints that are varied in order to minimize the objective, $\mathbf{y}_i \in \mathbb{R}^2$ are the center waypoints estimated in task 3.2 a). The first term ensures that the path stays close to the center path. What is the purpose of the second term? Implement the second term of the objective in the `curvature()` function. *Hint: For more details on path smoothing, read section 8.1 in* [2].

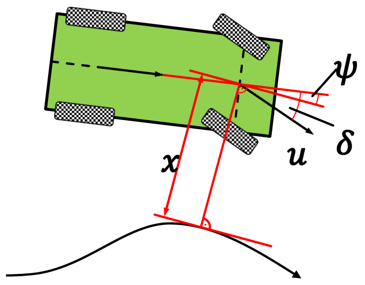
c) Target Speed Prediction: In addition to the spatial path, we need to know how fast the car should drive on the path. Heuristically, the car should accelerate to a maximum velocity if the path is smooth and decelerate before corners, so it makes sense to choose the target speed based on the path curvature. A measure of the curvature is provided by the second term in equation 1.

Implement a function `target_speed_prediction()` that outputs the current target speed for the predicted path in the state image, using

$$v_{\text{target}}(\mathbf{x}_1, ..., \mathbf{x}_N) = (v_{\max} - v_{\min}) \exp\left[-K_v \cdot \left| N - 2 - \sum_n \frac{(\mathbf{x}_{n+1} - \mathbf{x}_n) \cdot (\mathbf{x}_n - \mathbf{x}_{n-1})}{|\mathbf{x}_{n+1} - \mathbf{x}_n| |\mathbf{x}_n - \mathbf{x}_{n-1}|} \right| \right] + v_{\min} \tag{2}$$

As initial parameters use: $v_{\max} = 60$, $v_{\min} = 30$ and $K_v = 4.5$. You can tune these parameters for improving your driving score.

5.4 Lateral Control



In this section, you build a controller for steering the vehicle on the predicted path. Use the template in `lateral_control.py`.

a) Stanley Controller Theory: Read section 9.2 of [2], understand the heuristic control law for the steering angle which consists of two terms,

$$\delta_{SC}(t) = \psi(t) + \arctan\left(\frac{k \cdot d(t)}{v(t)}\right) \tag{3}$$

where $\psi(t)$ is the orientation error, $v(t)$ is the vehicle speed, $d(t)$ is the cross track error and k the gain parameter.

b) Stanley Controller: Implement the control law in equation 3 in `lateral_control.py` and determine a reasonable gain parameter empirically. *Note: To obtain the orientation error and the cross track error you can exploit the fact that the orientation of the car in the state image is fixed in the y-direction. The function* `env.step()` *already outputs the current vehicle speed.*

c) Damping: We now improve the steering angle control by damping the difference between the steering command and the steering wheel angle of the previous step. Implement a damping term as follows:

$$\delta(t) = \delta_{SC}(t) - D \cdot (\delta_{SC}(t) - \delta(t-1)) \,. \tag{4}$$

Find a good choice for the damping parameter D. What impact does damping have on the behavior of the car?

5.5 Longitudinal Control

After implementing the lateral control law for steering, we need to set up a control law for gas and braking. We implement a PID controller that follows the predicted target speed which we have implemented in 3.2 c). Use `longitudinal_control.py`.

a) PID Controller: For implementing the PID controller, we must use a discretized version of the control law:

$$e(t) = v_{\text{target}} - v(t) \tag{5}$$

$$u(t) = K_p \cdot e(t) + K_d \cdot [e(t) - e(t-1)] + K_i \cdot \left[\sum_{t_i=0}^{t} e(t_i)\right] \tag{6}$$

If the control signal $u(t)$ is larger than 0, we choose the gas value equal to the control signal. If u is smaller than 0, we choose the brake value equal to the negative control signal:

$$a_{\text{gas}}(t) = \begin{cases} 0 & u(t) < 0 \\ u(t) & u(t) \geq 0 \end{cases} \qquad a_{\text{brake}}(t) = \begin{cases} 0 & u(t) \geq 0 \\ -u(t) & u(t) < 0 \end{cases} \tag{7}$$

Implement the control step as described. *Hint: The integral term often leads to the so-called integral windup which means it can accumulate a significant error and yield strong overshooting. Implement an upper bound for the error sum.*

b) Parameter Search: Optimizing the parameters of the PID controller (K_p, K_d, K_i) is a lot of engineering and hand tuning. To find good parameters of the PID controller (K_p, K_d, K_i) and the target speed $(v_{\max}, v_{\min}, K_v)$ (from 5.2 c) use the plots of the target speed and the actual speed generated by `test_longitudinal_control.py`. Start with only the proportional term (K_p) and modify only a single term at a time. The car should follow the target speed quite accurately, but pay attention to the acceleration after tight corners, if it is to strong the car may loose grip. *Hint: If you need more information about parameter tuning, see* [3].

After tuning the parameters, set the parameters of your modular pipeline in the function `calculate_score_for_leaderboard()` in `modular_pipeline.py`.

5.6 Competition

With each coding exercise sheet you are welcome to participate in our competition! The winners for each of the 3 exercise sessions will be given the opportunity to present their approach in the very last lecture. The evaluation works as follows: the models are tested on a set of validation-tracks. For each track, the reward after 600 frames is used as performance measure and the mean reward from all validation tracks then forms the overall score. For the final ranking, we will run that evaluation on a set of secret tracks for every submission. The reward is -0.1 every frame and +1000/N for every track tile visited, where N is the total number of tiles in track. Good luck and Merry Christmas!

5.7 References

[1] https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.splprep.html#id3
[2] http://isl.ecst.csuchico.edu/DOCS/darpa2005/DARPA%202005%20Stanley.pdf
[3] https://homepages.inf.ed.ac.uk/mherrman/IVRINVERTED/pdfs/PID_control.pdf