Robot Learning and Vision for Navigation

Exercise 2 – Modular Pipeline

Release date: Thursday, 21 Feb. 2023 - **Deadline for Homework: Monday, 13 Mar. 2023 - 23:59**

For this exercise you need to submit a **.zip** folder containing your report as a **.pdf** file (up to 5 pages) and your **.py** code files.

As in the previous exercise, please use the provided code templates. Add your best choice of parameters to the `modular_pipeline.py` file.

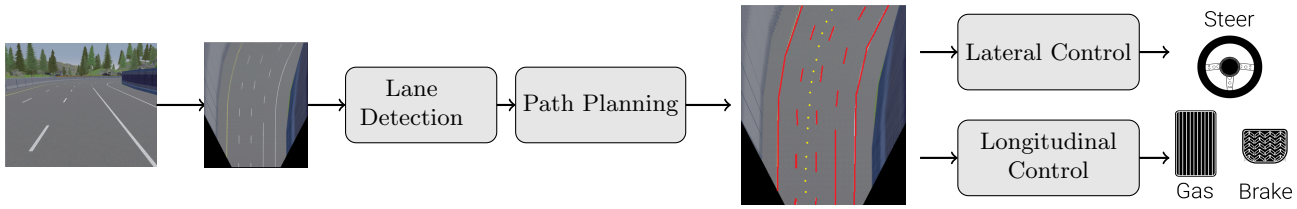


Figure 1: Modular pipeline consisting of a lane detection module, path planning and a control unit.

2.1 Lane Detection (2+2+2+1+1+1 Points)

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) Homography Transfer: In a typical autonomous driving stack, Behavior Prediction and Planning are generally done in this a top-down view (or bird's-eye-view, BEV), as hight information is less important and most of the information an autonomous vehicle would needs can be conveniently represented with BEV. In this question, you will need to implement the function `front2bev` to transform a front-view image into a bird-eye view image [4, 5].

b) Edge Detection: Now we have the BEV image, and let's start with the edge detection. First, implement the function `LaneDetection.cut_gray()`. It should translate the state image to a grey scale image and crop out the part above the car. Second, derive the gradients of the grey 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`.

c) Obstacle Detection: Obstacle detection is a crucial component of autonomous driving systems as it helps the vehicle detect and avoid obstacles in its path. Now we are going to apply a pre-trained Faster R-CNN [6] to detect the potential obstacles in the front-view image, and do the tomography transfer and combine with the edge detection. *Hint: a pre-trained Faster R-CNN can be found https://github.com/facebookresearch/detectron2/blob/main/MODEL_ZOO.md, and a tutorial of how to use it can be found https://colab.research.google.com/drive/16jcaJoc6bCFAQ96jDe2HwtXj7BMD_-m5* .

d) 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. Our approach consists of finding the maxima in the image row closest to the car. 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 missing code.

e) **Spline Fitting:** For fitting the lane boundaries it is common to use parametric splines. In `LaneDetection.lane_detection()`, fit the spline documented in [1] to each lane boundary. We are using a parametric spline, since we can sample points given a single parameter, that represents the length of the curve.

f) **Testing:** Test your implementation by running `test_lane_detection.py`. You can drive using arrow keys and check determined lane boundaries in the additional window. Find a good choice of parameters for the gradient threshold and the spline smoothness. If there are still some failure cases, describe them in the report and try to find reasons. Add some state images including the lane boundaries to the report.

2.2 Path Planning (1+2+1 Points)

This section is about how to plan the car's path as well as the speed.

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. These values should include 0. Next determine the center between lane boundary points with the same spline parameter. Use the `test_waypoint_prediction.py` to verify your implementation. Add a plot to the report. In which situations does the waypoint prediction fail? Why?

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:

$$\underset{x_1,\dots,x_N}{\operatorname{argmin}} \sum_i |y_i - x_i|^2 - \beta \sum_n \frac{(x_{n+1} - x_n) \cdot (x_n - x_{n-1})}{|x_{n+1} - x_n|\,|x_n - x_{n-1}|}. \tag{1}$$

There, x_i are the waypoints that are varied in order to minimize the objective, y_i are the center waypoints estimated in task 3.2 a). The first term ensures that the path stays close to the center path.

Describe the purpose of the second term in the report. Then, 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

$$v_{\text{target}}(x_1, \dots, x_N) = (v_{\max} - v_{\min}) \exp\left[-K_v \cdot \left| N - 2 - \sum_n \frac{(x_{n+1} - x_n) \cdot (x_n - x_{n-1})}{|x_{n+1} - x_n|\,|x_n - x_{n-1}|} \right|\right] + v_{\min}, \tag{2}$$

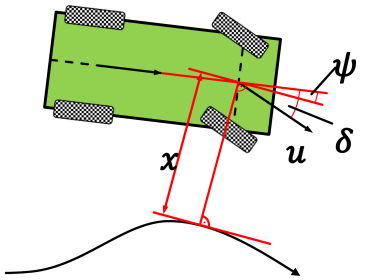
As initial parameters use: $v_{\max} = 60$, $v_{\min} = 30$ and $K_v = 4.5$.

2.3 Lateral Control (1+2+2 Points)

In this section 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 and explain the two parts of the control law

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



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. Describe the behavior of your car. *Note: We fixed the orientation of the car in the state image in y-direction. The function* `env.step()` *also outputs the current 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 and describe the impact of damping on the behavior of the car.

2.4 Longitudinal Control (3+3 Points)

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 use the plots of the target sped and the actual speed generated by `test_longitudinal_control.py`. Start with only the proportional term 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 the parameter tuning, see* [3].

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

2.5 Competition (0 Points)

Congratulations! You have just built a basic modular pipeline for a navigation. For the competition you are free to modify and improve your basic modular pipeline and the parameters.

For a final ranking, we will run the `modular_pipeline.py score` on a secret set of tracks for every submission. For each track, the cumulative reward after 600 frames is used as the performance measure. Evaluation is stopped early if the agent is `done` (e.g. has left the track). The overall score is computed by taking the mean cumulative reward from all validation tracks. The reward function is defined as -0.1 for every frame and $+1000/N$ for every track tile visited, where N is the total number of tiles in track.
Good luck!

2.6 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] http://saba.kntu.ac.ir/eecd/pcl/download/PIDtutorial.pdf
[4] https://csyhhu.github.io/2015/07/09/IPM/
[5] https://towardsdatascience.com/a-hands-on-application-of-homography-ipm-18d9e47c152f
[6] Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." Advances in neural information processing systems 28 (2015).