Self-Driving Cars

Exercise 5 - Modular Pipeline

Apratim Bhattacharyya

Autonomous Vision Group

University of Tübingen

Exercise Setup

Files

- `ex_05_modular_pipeline_exercise.pdf`
- `modular_pipeline.py`, `lane_detection.py`, `waypoint_prediction.py`, `lateral_control.py`, `longitudinal_control.py`
- `test_lane_detection.py`, `test_waypoint_prediction.py`, `test_lateral_control.py`, `test_longitudinal_control.py`
- `submission.txt`

Submit

- submission information: please fill out `submission.txt`
- your code: `modular_pipeline.py`, `lane_detection.py`, `waypoint_prediction.py`, `lateral_control.py`, `longitudinal_control.py` as a `.zip` file

Deadline: **Mon, 31th January 2022; 8pm**

Exercise Setup

- run it on your local machine, no learning
- no GPU required
- TCML cluster (without requesting for a GPU)

Exercise Setup

Do’s

- comment your code
- use docstrings
- use self-explanatory variable names
- structure your code well

Exercise Setup

Do’s

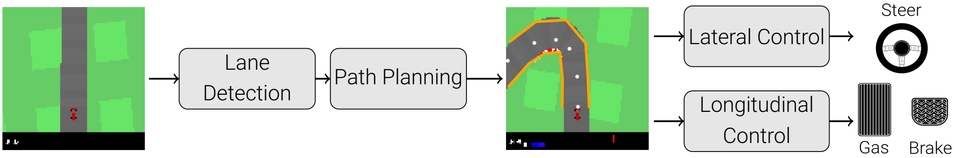
- comment your code
- use docstrings
- use self-explanatory variable names
- structure your code well

Do not’s

- learning based approaches for any subtask
- PyTorch (or any other deep learning/GPU acceleration libraries)
- install more packages (only use Numpy and Scipy)
- change the gym environment

Modular Pipeline

Modular Pipeline



3.1
Lane Detection

Lane Detection

Template

- `lane_detection.py`
- `test_lane_detection.py` for testing

a) **Edge Detection:**

- Translate the state image to a grey scale image and crop out the part above the car
  → `LaneDetection.cut_gray()`
- Derive the absolute value of the gradients of the grey scale image and apply thresholding to ignore unimportant gradients.
  → `LaneDetection.edge_detection()`
- Determine arguments of local maxima of absolute gradient per pixel row
  → `LaneDetection.find_maxima_gradient_rowwise()`
  Hint: use for example `scipy.signal.find_peaks()`

Lane Detection

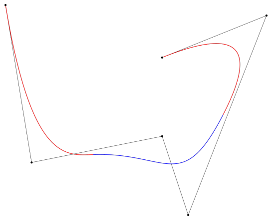
b) **Assign Edges to Lane Boundaries:**

- Find arguments of local maxima in the image row closest to the car
  $\rightarrow$ `LaneDetection.find_first_lane_point()` (already implemented)
- Assign the edges to the lane boundaries by successively searching for the nearest neighboring edge/maximum along each boundary
  $\rightarrow$ `LaneDetection.lane_detection()`
- *Note: you are free to improve upon our suggested approach*

Lane Detection

c) **Spline Fitting:**

- Fit a parametric spline to each lane boundary
  $\rightarrow$ `LaneDetection.lane_detection()`
- Use `scipy.interpolate.splprep` for fitting and `scipy.interpolate.splev` for evaluation



Given a list of points, which represents a curve in 2-dimensional space parametrized by s, find a smooth approximating spline curve $g(s)$.

Lane Detection

d) **Testing:**

- 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.
- Try to find failure cases

3.2
Path Planning

Path Planning

Template

- `waypoint_prediction.py`
- `test_waypoint_prediction.py` for testing

a) **Road Center:**

- Use the lane boundary splines and derive lane boundary points for 6 equidistant spline parameter values
  $\rightarrow$ `waypoint_prediction()`
- Determine the center between lane boundary points with the same spline parameter
  $\rightarrow$ `waypoint_prediction()`

Path Planning

b) **Path Smoothing:**

- Improve the path by minimizing the following objective regarding the waypoints x given the center waypoints y

$$\underset{\mathbf{x}_1,\ldots,\mathbf{x}_N}{\operatorname{argmin}} \sum_i |\mathbf{y}_i - \mathbf{x}_i|^2 - \beta \sum_n \frac{(\mathbf{x}_{n+1} - \mathbf{x}_n) \cdot (\mathbf{x}_n - \mathbf{x}_{n-1})}{|\mathbf{x}_{n+1} - \mathbf{x}_n| \, |\mathbf{x}_n - \mathbf{x}_{n-1}|}.$$

- Explain the effect of the second term
- Implement second term

  $\rightarrow$ `curvature()`

c) **Target Speed Prediction:**

- Implement a function that outputs the target speed for the predicted path in the state image , using

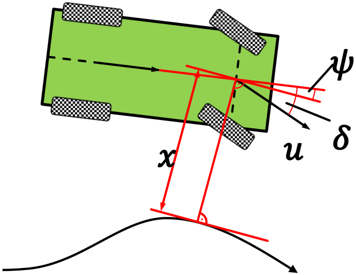
$$v_{\text{target}}(\mathbf{x}_1, ..., \mathbf{x}_N) = (v_{\text{max}} - v_{\text{min}}) \exp\left[-K_v \cdot \left| N - 2 - \sum_n \frac{(\mathbf{x}_{n+1} - \mathbf{x}_n) \cdot (\mathbf{x}_n - \mathbf{x}_{n-1})}{|\mathbf{x}_{n+1} - \mathbf{x}_n| \, |\mathbf{x}_n - \mathbf{x}_{n-1}|} \right|\right] +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

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

$\rightarrow$ `target_speed_prediction()`

3.3
Lateral Control

Lateral Control



$$\delta_{SC}(t) = \psi(t) + \arctan\left(\frac{k \cdot d(t)}{v(t)}\right) \tag{1}$$

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.

Lateral Control

Template

- `lateral_control.py`
- `test_lateral_control.py` for testing

a) **Stanley Controller:**
   - Read section 9.2 in the Stanley paper

     `http://isl.ecst.csuchico.edu/DOCS/darpa2005/DARPA%202005%20Stanley.pdf`
   - Understand the parts of the heuristic control law

Lateral Control

b) **Stanley Controller:**

- Implement controller function given waypoints and speed
  $\rightarrow$ `LateralController.stanley()`
- Orientation error $\psi(t)$ is the angle between the first path segment to the car orientation
- Cross track error $d(t)$ is distance between desired waypoint at a spline parameter of zero to the position of the car
- Prevent division by zero by adding as small epsilon
- Check the behavior of your car

Lateral Control

c) **Damping:**

- Damping the difference between the steering command and the steering wheel angle of the previous step

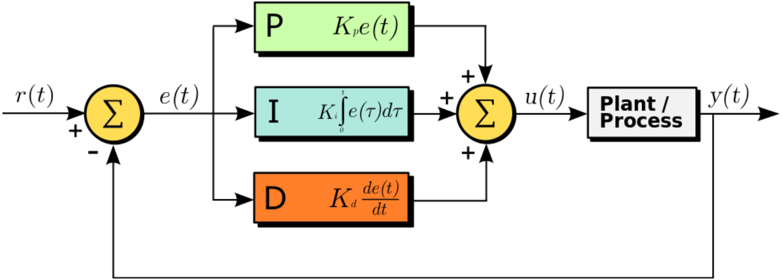
$$\delta(t) = \delta_{SC}(t) - D \cdot (\delta_{SC}(t) - \delta(t-1)) \,. \tag{2}$$

- Describe the behavior of your car

3.4
Longitudinal Control

Longitudinal Control

Proportional - Integral - Derivative Controller for gas and braking



Longitudinal Control

Template

- `longitudinal_control.py`
- `test_longitudinal_control.py` for testing

a) **PID Controller:**

- Implement a PID control step for gas and braking
- Use a discretized version:

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

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

where $u(t)$ is the control signal and $e(t)$ error signal

Longitudinal Control

a) **PID Controller:**

- Due to integral windup, implement an upper bound for integral term.
- From control signal to gas and brake action values

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

Longitudinal Control

b) **Parameter Search:**

- Run `test_lateral_control.py` and have a look at plots of the target speed and the actual speed
- tune parameters (K_p, K_i, K_d) and $(v_{\text{max}}, v_{\text{min}}, K_v)$
- Start with $(K_p = 0.01, K_i = 0, K_d = 0)$ and $(v_{\text{max}} = 60, v_{\text{min}} = 30, K_v = 4.5)$
- Only modify a single term at a time !!!!

Competition

Competition

- For the competition you are free to modify and improve your basic modular pipeline and the parameters
- Run `modular_pipeline.py --score` using your parameters
- Submit your code

Questions?