Arthur Wang
Jason Ha

ECS518 Robot Learning Report 2: Modular Pipeline

2.1 Lane Detection

A. Homography Transfer

See front2bev() in lane_detection.py

B. Edge Detection

See edge_detection() in lane_detection.py. Base gradient image produces a lot of noisy peaks causing bad lane lines. To remedy this, we first use OpenCV HoughLineP to get line segments which we then RANSAC. We also tested applying HoughLine to the RANSAC results to get stronger peaks for edge assignment but that had issues with curved corners.

C. Obstacle Detection

See DDRNet.py. Unfortunately, the inference FPS and the segmentation image is extremely underwhelming so the obstacle detection is implemented but not utilized.

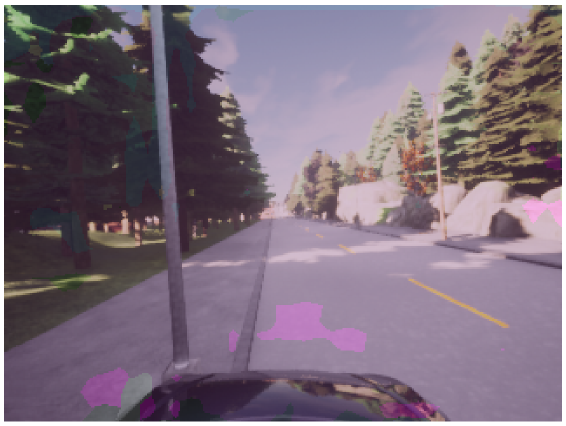

Figure 1. Blended semantic segmentation and input image, not very good.

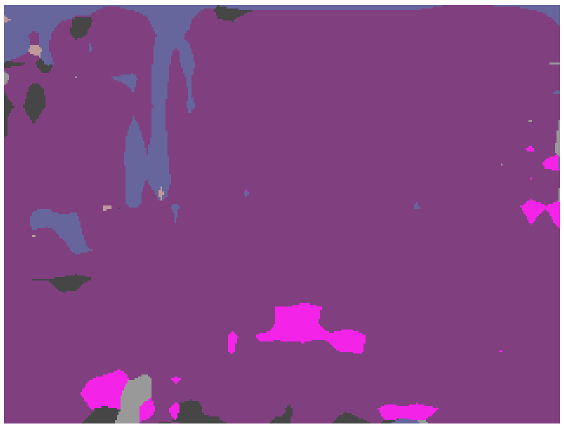

Figure 2. Just segmentation output.

D & E. Assign Edges & Splines

See lane_detection().

F. Testing

We experimented with different gradient threshold values and spline smoothness parameters to optimize lane detection. A lower threshold detected more edges but was prone to noise, while a higher threshold sometimes missed important lane features such as the curb. Adjusting the spline smoothness helped balance precision with the continuity of the lane lines.

Even after tuning, the lane detection struggled with sharp turns and scenarios with inconsistent lighting and shadows, where the lane markings weren't as clear. These issues likely stem from variations in image contrast and rapid changes in curvature that the algorithm couldn't handle well.

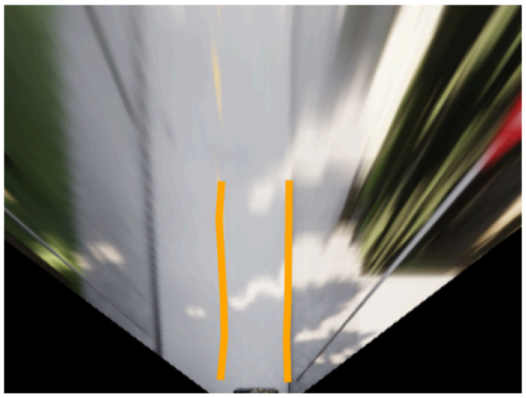

Figure 3. Lane detection on BEV image

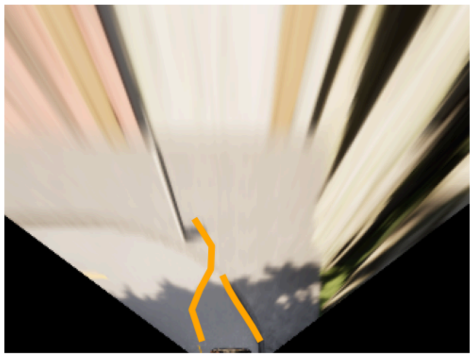

Figure 4. Case of bad lane detection on curves.

2.2 Path Planning

A. Road Center

Using test_waypoint_prediction.py, we validated the waypoints and adjusted the function as needed. We need to include a plot of the resulting waypoints overlaid on the detected lane to show how the car would follow the center path. Waypoint prediction struggles on roads with irregular lane markings or abrupt changes in lane width, where the calculated center can deviate unexpectedly. This is mainly due to the lane boundary splines being inaccurate in such conditions. Another situation is at intersections where dotted lane lines are not present and the curbs might be too near to show up effectively in the BEV image. As such, the waypoints are generally pretty random until the lane detection manages to latch onto something resembling lanes.

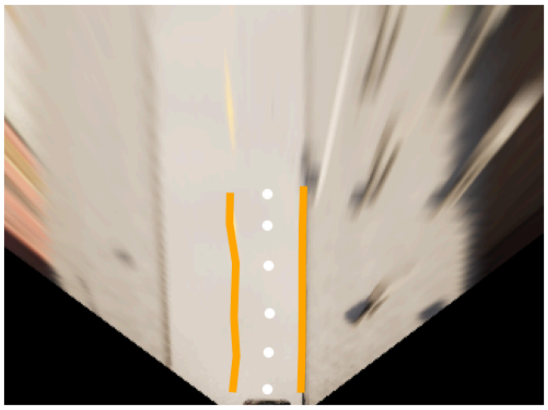

Figure 5. Good way point example on straight road.

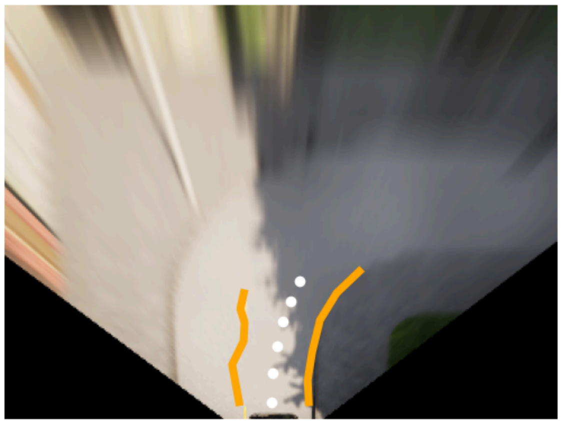

Figure 6. Good waypoint detection on curved corners.

B. Path Smoothing

To improve the car's performance on curvy roads, we smooth the path by minimizing an objective function. The second term of the objective is calculating the angle between two vectors which is the vector between the previous waypoint and the current and the vector between the next waypoint and the current waypoint. This is the per waypoint curvature which is then summed for all waypoints to get the overall path curvature.

C. Target Speed Prediction

See target_speed_prediction() in waypoints.py.

2.3 Lateral Control

See lateral_control.py.

A. Stanley Controller Theory

Orientation error is the arctan2 angle between the car position at [156, 0] and the second waypoint position. We use the second because the first may line up with the car position resulting in the orientation error potentially being noisy.

Cross-track error is simply the x axis difference between the car position and the first waypoint.

B. Stanley Controller

Based on our testing, we chose a k gain of 0.05. At higher velocities, the resulting turn angle should be lower and vice versa. However, we noticed that at lower speeds the car will still swerve quite hard. As such, we attempted to limit the impact of the cross track error by choosing a very low gain. This helped by making the car turn less for any given command signal which helped with not making drastic steering changes which can affect lane detection.

C. Damping

We chose a D value of 0.7 through testing. The damping term helps moderate the steering angle by smoothing out sudden changes to the steering angle. For example, if the previous steering angle is 0, and the current Stanley controller commands a steering angle of 1, the damping term will affect how much of that command signal is actually used. A 70% damping in steering command seemed reasonable and performed okay when testing compared to other values.

2.4 Longitudinal Control

A. PID Controller Implementation

See longitudinal_control.py.

B. Parameter Search

We started by tuning the proportional term (K_p), which had the biggest impact on reducing steady-state error and aligning the car's speed with the target. We preferred that the car did not accelerate too quickly to reach the set speed so we decided 0.05 was a decent proportional gain.

Then we adjusted the derivative (K_d) to smooth out rapid speed changes that could happen when accelerating and decelerating.

Finally, we introduced a small integral term (K_i) to correct for any remaining error over time. We watch the feedback response plot closely to determine how large the integral term should be to force the speed to the set point over time. However, our car crashed a lot and rarely stayed driving in a straight line for long enough to properly evaluate.

After multiple iterations we landed on the values of 0.05, 0.05, 0.15 for Kp, Kd, and Ki respectively.

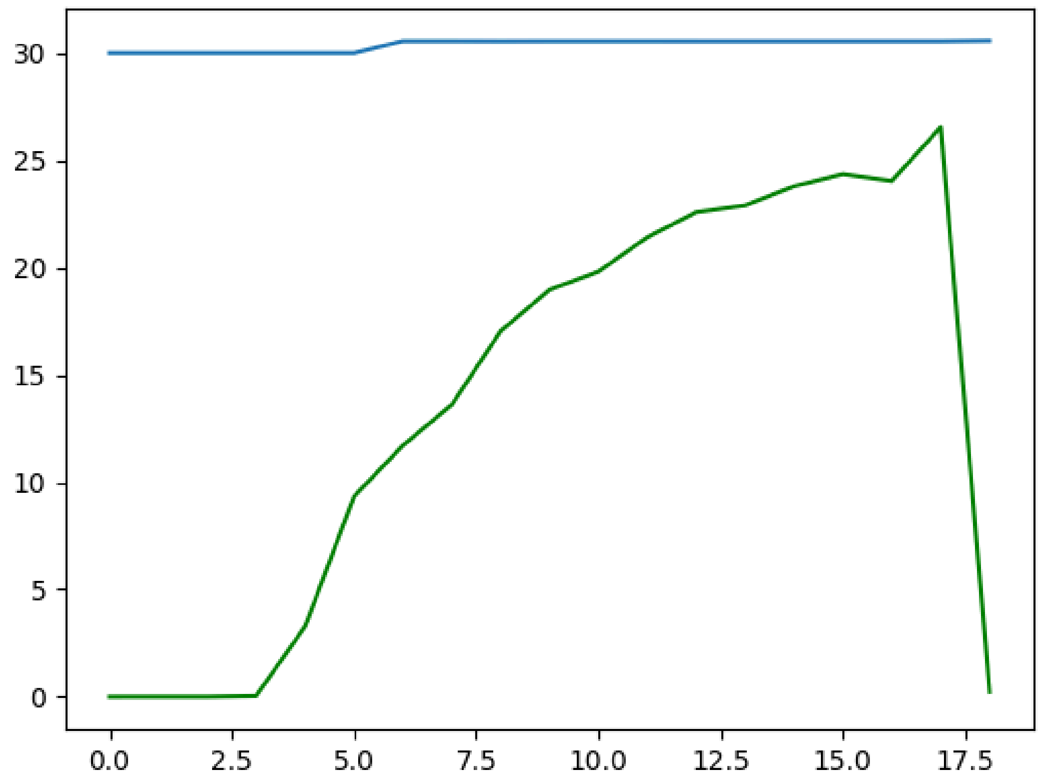

Figure 7. Step vs. Speed graph. Over 18 ticks at clock.tick(30), we accelerate up towards the set speed of 30. Then we crash.

2.5 Competition

See test_agent_lane.py.