Intelligent Driving System HW2 Report

1. Implementation
   a. Setup ground-truth future waypoints (TODO 1)

```
# TODO 1
"""
Setup ground-truth future waypoints.
:R: matrix that converts the future (x,y) into ego's coordinate
:local_command_point: centering the waypoint around the ego
"""
waypoints = []
for i in range(4):
    # First, I have to compute the rotation matrix
    R = np.array([
        [np.cos(np.pi / 2 + ego_theta), -np.sin(np.pi / 2 + ego_theta)],
        [np.sin(np.pi / 2 + ego_theta),  np.cos(np.pi / 2 + ego_theta)]
    ])
    # Before multiple the rotation matrix, I have to calculate the relative distance between future waypoints and ego car position
    local_command_point = np.array([self.future_y[index][i] - ego_y, self.future_x[index][i] - ego_x])
    local_command_point = R.T.dot(local_command_point)
    waypoints.append(local_command_point)
data['waypoints'] = np.array(waypoints)
# End TODO 1
```

   b. How to use high-level command as input (TODO 2)

```
# TODO 2
"""
Create an one hot vector of high-level command.
Ex: command = 3 (straight) -> cmd_one_got = [0,0,1,0,0,0]

VOID = -1
LEFT = 1
RIGHT = 2
STRAIGHT = 3
LANEFOLLOW = 4
CHANGELANELEFT = 5
CHANGELANERIGHT = 6
"""
command = self.command[index]
if command < 0:
    command = 4
# Just created the one hot command as required
command -= 1
assert command in [0, 1, 2, 3, 4, 5]
cmd_one_hot = [0] * 6
cmd_one_hot[command] = 1

data['target_command'] = torch.tensor(cmd_one_hot)
# End TODO 2
```

   c. Forward the inputs to get some latent embedding (TODO 3)
   d. Predict the ego's speed (TODO 3)

```
# TODO 3
"""
Please check the network first!

Forward the inputs to get some latent embedding
:feature_emb, cnn_feature: use self.perception to encode the img.
:measurement_feature: use self.measurements to encode the state. (Please check what "state" refers to in train.py)

Predict the ego's speed
:outputs['pred_speed']: input feature_emb into self.speed_branch to predict the speed.
"""
# state = [speed, target point, command, 1]
# Just did what the comment said
feature_emb, cnn_feature = self.perception(img)
measurement_feature = self.measurements(state)

outputs['pred_speed'] = self.speed_branch(feature_emb)
# End TODO 3
```

e. Predict the waypoints (TODO 4)

```
# TODO 4
"""
Please check the network first!

Autoregressive generation of output waypoints
1. x_in: concat x and target point along second dimension -> shape=[B,4]
2. z: input (input=x_in, hidden=z) into self.decoder_traj to update z
3. dx: input z into self.output_traj to predict waypoint offset dx
4. x: update waypoint x by adding x with offset dx
"""
# First, concate x and target point along second dimension by torch.cat()
# Second, put z into trajectory decoder to update z
# Third, put z intp output trajectory decoder to predict the offset of waypoint
# Last, update the x
for _ in range(self.config.pred_len):
    x_in = torch.cat([x, target_point], dim=1)
    z = self.decoder_traj(x_in, z)
    dx = self.output_traj(z)
    x = dx + x
    output_wp.append(x)

pred_wp = torch.stack(output_wp, dim=1)
outputs['pred_wp'] = pred_wp
# End TODO 4
```

f. Tick input data from CARLA (TODO 5)

```
# TODO 5
"""
Tick data from CARLA
:rgb: input_data['rgb'][1][:, :, :3] (convert it into RGB format)
:bev: input_data['bev'][1][:, :, :3] (convert it into RGB format)
:gps: input_data['gps'][1][:2]
:speed: input_data['speed'][1]['speed']
:compass: input_data['imu'][1][-1]
"""
# Just ticked the data from input data, and converted rgb and bev to RGB format by cv2
rgb = cv2.cvtColor(input_data['rgb'][1][:, :, :3], cv2.COLOR_BGR2RGB)
bev = cv2.cvtColor(input_data['bev'][1][:, :, :3], cv2.COLOR_BGR2RGB)
gps = input_data['gps'][1][:2]
speed = input_data['speed'][1]['speed']
compass = input_data['imu'][1][-1]
# End TODO 5
```

g. Converts waypoints into vehicle control with a PID controller (TODO 6)

```
# TODO 6
"""
Predicts vehicle control with a PID controller. (Use control_pid in self.net)
We strongly suggest that you study "control_pid".
"""
# Waypoint, Velocity, Target
# Predict vehicle control by control_pid function
steer_traj, throttle_traj, brake_traj, metadata_traj = self.net.control_pid(pred['pred_wp'], gt_velocity, target_point)
# End TODO 6
```

h. Apply control to your vehicle (TODO 7)

```
# TODO 7
control = carla.VehicleControl()
"""
Apply control signal: you can clip the value as you want
"""
# Just applied the control to the ego vehicle
control.steer = steer_traj
control.throttle = throttle_traj
control.brake = float(brake_traj)
# End TODO 7
```

2. Discussion

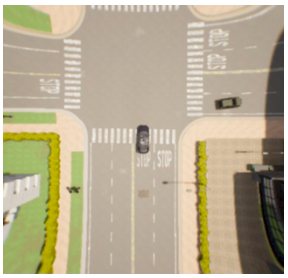
a. Use e2e_driving/statistics.py to parse results in the results_VA.json

```
score_composed = 42.951935314333316
score_penalty = 0.6142555631636888
score_route = 75.2399077349452
collisions_layout = 0.26422668920138688
collisions_pedestrian = 0.0
collisions_vehicle = 0.8988975683351389
red_light = 0.09062639437192661
stop_infraction = 1.3343584272999818
vehicle_blocked = 0.354853083573313347
outside_route_lanes = 0.181252788743853233
```

b. In which aspects does the model perform poorly?

This model perofrm poorly on collision, it is easily collide with others vehicles, leading to low route completion and very low driving score.
And this model also easily to fail the running stop test.


Collided with vehicles, leading to blocked at intersection.

Running stop sign

c. How would you enhance the explainability of the models?

Adding bounding boxes from training the driving model can enhance explainability by providing visual cues about the objects and their spatial relationships in the scene, allowing observers to understand how the model perceives and interacts with its environment, ultimately helping to identify potential collision risks and improve safety.

3. Question Answering

a. What else can we do to improve the performance of the driving model?

Maybe we can add more data for training, for example, depth images, semantic images, bounding boxes, and lidar point cloud. With the help of the diversity training data, I believe that the driving model can improve the performance. Or perphase augmenting the training data with various transformations like rotation, translation, scaling, and adding noise can improve the driving model's robustness and generalization to different driving conditions.

b. Besides imitation learning, how would you train the end-to-end driving models? What are the pros and cons of each paradigm?

Maybe I will use Reinforcement Learning (RL), and it has pros and cons. For example, RL allows the agent to learn from its interactions with the environment, without requiring labeled data. It can adapt to various driving scenarios and optimize for specific objectives, such as reaching a destination quickly while obeying traffic rules. However, RL can be sample-inefficient and prone to instability during training. It often requires careful tuning of hyperparameters and reward functions. Additionally, RL may struggle in safety-critical domains like autonomous driving, where mistakes

can have severe consequences. For instance, ROACH, one of the driving model based on RL, it has a obviously shortcoming that it cannot handel lane changing well.

c. In HW2, we predict the future trajectory and convert it into control through a PID controller. However, there are some works that predict control directly. What do you perceive as the challenges or limitations of each of these methods?

From the above papers, the main limitations of this method stem from the initial hypothesis that demonstration data can always be associated with a constant maximal reward. Firstly, if the demonstration data is not consistently optimal due to factors such as low expert performance, it introduces noise in the reward function. This noise from imperfect autopilot demonstrations, as seen in our dataset on the CARLA simulator, can impact the performance of the model. Although GRIAD showed improvement over vanilla RL, it demonstrates some robustness to noisy demonstrations.

Secondly, a warm-up phase is observed in difficult environments, such as in HalfCheetah-v2 and Humanoid-v2. This phase arises due to a distribution shift where the training expert data primarily represent actions not yet reached by the exploration agents. For instance, in HalfCheetah-v2, the expert agent jumps immediately upon touching the ground, requiring a warm-up phase for the exploration agent to gain momentum. Although the agent eventually learns to jump successfully, this distribution shift affects the learning process. However, increasing the proportion of exploration data helps compensate for this shift.

Lastly, there's inconsistency in the rewards associated with common actions collected by both demonstration and exploration agents. For instance, in HalfCheetah-v2, demonstration actions rewarded with high demonstration rewards may receive poor rewards during exploration. This discrepancy between offline demonstration data and online RL experiences leads to an overestimation of demonstration actions and can impact training. While high rewards for demonstration data may encourage the agent to mimic expert behavior, the practical impact on training remains difficult to assess.

4. Please provide any reference you take
https://arxiv.org/pdf/1904.08980