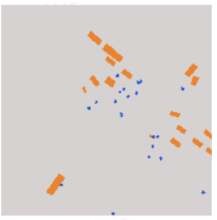
Perception and Decision Making in Intelligent Systems
Homework 4: End-to-end Autonomous Driving Framework
Announcement: 11/19, Deadline: 12/2 23:59

Introduction

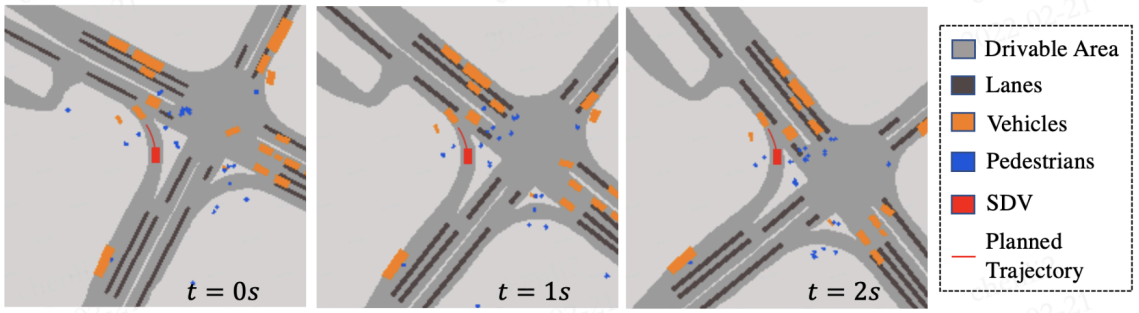
In this assignment, you will work with a pre-trained model to enable ego car control based on the future trajectory of surrounding agents. You will implement functions that process model inputs to transform coordinates to an ego-centered frame and use the model's outputs to make the decisions for ego-motion.

(a) Input Images

(b) Map



(c) Occupancy



(d) Plan

Implementation

We utilize the end-to-end model named ST-P3 [1]. This model adopts the LSS (Lift-Splat-Shoot) approach [2], which extracts perspective features from multi-view cameras, lifts them into 3D using depth estimation, and fuses them into the BEV (Bird's Eye View) space. The model processes the past three frames as input to predict the future motion of traffic participants around the ego vehicle.

For tasks 1 and 2, you are required to write code to complete the functions that we have defined in the project template and make them output the expected results.

Stage 1: Perception

Future Ego-Motion Calculation

As described in ST-P3 [1], to compensate for deviations caused by the ego vehicle's motion, the motion matrix is incorporated into the features by concatenating it into the spatial channel.

Task (TODO1):

Your task is to implement the *get_future_egomotion()* function. This function should calculate the ego vehicle's motion between two consecutive frames and return a pose vector representing the motion between these two poses.

Affine Transformation

Since the model uses the past three frames as input and transforms them into the corresponding BEV features, we must account for the ego car's movement. To ensure the BEV features align with the current ego-centered coordinate system, we employ Affine Transformation, which applies affine matrices to warp the features into the target coordinate space.

Task (TODO2_1):

Complete the calculation process for the affine matrices, which map the BEV representation from one ego-vehicle pose to another. This ensures the features are correctly transformed between different ego poses.

Task (TODO2_2):

Implement the `warp_features()` function. This function should use the affine matrix you calculated to warp the BEV features to the target coordinates, aligning them with the current ego-centered coordinate system.

Stage 2: Prediction

In this stage, the trained model uses the input from the perception stage to generate future-predicted segmentation. This prediction provides the ego vehicle with information about the future motion of traffic participants in its surroundings.

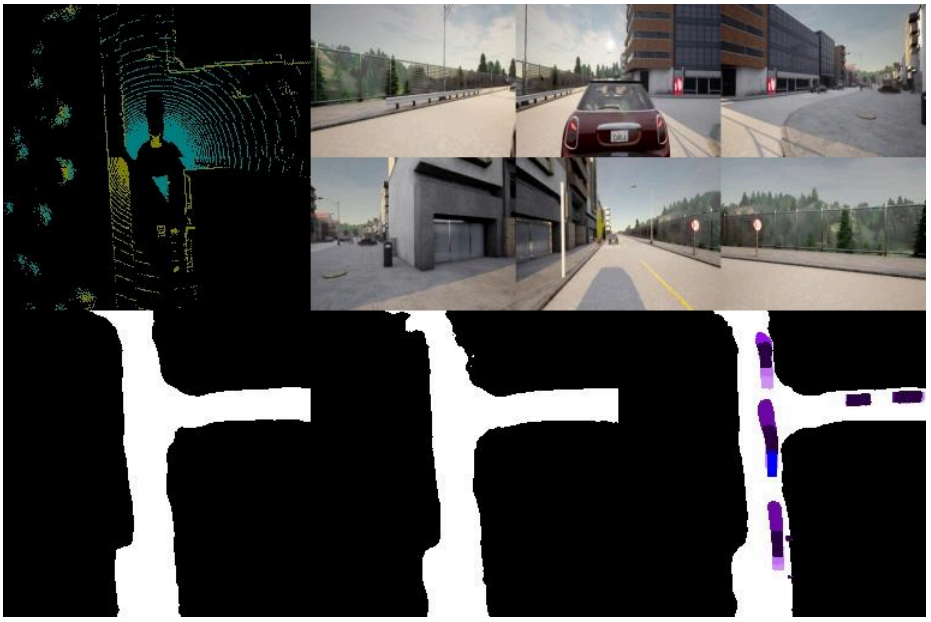
A more precise prediction leads to more accurate control decisions, enabling safer and more efficient navigation.

We have provided the checkpoint you can download on Google Drive, if you do the above tasks correctly, you can get the output like the below images, these images can be checked in the log directory.

Task (TODO_3): Complete the Main Logic for the Perception Stage

In this task, your goal is to complete the main logic for the model's perception stage. There are four main steps in this function, and all necessary subfunctions can be found in stp3.py. Finally, you will apply temporal fusion (refer to [1], Section 3.1) to obtain the input for the prediction module. Below are the steps:

1. **Encode Input Features**
   Call the `encoder_forward()` function to encode the input features.
2. **Project to Bird's-Eye View**
   Use the `projection_to_birds_eye_view()` function to transform the encoded features into the Bird's-Eye View (BEV).
3. **Warp to Ego's Current Coordinate**
   For each frame's BEV feature, call the `warp_features()` function to transform the BEV feature to the ego vehicle's current coordinate system.
4. **Apply Temporal Fusion**
   Use `self.temporal_coef` to apply temporal fusion to the warped features, producing the final input for the prediction module.



Stage 3: Planning

With the output from the prediction stage, the model must determine a feasible trajectory to reach the target position. In the previous homework, we used the RRT algorithm to generate a trajectory. However, in autonomous driving, road conditions are much more complex, and due to physical constraints—such as the fact that vehicle motion follows the kinematic bicycle model—various methods are employed to achieve final control.

Task (TODO_4): Decision-Making and Control Implementation

For this homework, your task is simplified: you only need to determine **throttle** and **brake** controls based on the predicted segmentation(model output) of the environment. The steering control is managed by a rule-based autopilot.

You can implement the `get_our_control()` function using a rule-based approach or training your model. Ensure that the ego vehicle progresses safely along its route while avoiding collisions and adhering to traffic rules.

Your driving model will be evaluated using the **test results of the three Carla routes**. The criteria of the route test are shown below:

Hint:

- The output contains future-predicted segmentation data for agents around the ego vehicle.
- Use the `vis()` function as a reference to extract and visualize segmentation information, such as predicted agents' position.

========== Results of RouteScenario_28035 (repetition 0)
------ SUCCESS =========

Start Time	2024-11-18 14:51:32
End Time	2024-11-18 15:05:07
System Time	814.85s
Game Time	55.3s
Ratio (Game / System)	0.068

Criterion	Result	Value
RouteCompletionTest	SUCCESS	100 %
OutsideRouteLanesTest	SUCCESS	0 %
CollisionTest	SUCCESS	0 times
RunningRedLightTest	SUCCESS	0 times
RunningStopTest	SUCCESS	0 times
MinSpeedTest	SUCCESS	227.01 %
InRouteTest	SUCCESS	
AgentBlockedTest	SUCCESS	
ScenarioTimeoutTest	SUCCESS	0 times
Timeout	SUCCESS	

Please check all the **"TODO_"** comments in

- hw4/team_code/stp3_with_autopilot.py
- hw4/team_code/stp3/models/stp3.py

Report

Please contain the following parts in your report:

1. **Code explanation**
   Please explain in detail how you implement the affine transformation, ego-motion calculation, the high-level concept of your control policy, and the process from the model's output to your final control.

2. **Results and discussion**

   a. Screenshot of the evaluation results on three scenarios.
   b. Discuss your results and observations.

3. **Question Answering**

   a. How do you think self-driving cars turn the surrounding images into BEV feature representations? What may be the advantages and disadvantages?

   b. End-to-end models refer to deep learning systems that map raw sensor data directly to the vehicle's control outputs. While these models simplify the overall system by eliminating the need for manual integration of separate modules, they face significant challenges in terms of **explainability**. How do you think we can enhance the model's explainability? If the model were more explainable, humans might trust autonomous vehicles more.

Grading Policy

1. **Report (45%)**
   a. Code explanation (15%)
   b. Results and discussion (15%)
   c. Question Answering (15%)

2. **Online Demo (55%)**
   - Code explanation & demonstration (25%)
   - Question answering (30%)

The online demo will include questions about the concepts of Lift, Splat, Shoot (LSS) and the pipeline of ST-P3. We recommend reviewing the paper and familiarizing yourself with the overall code structure of ST-P3.

Submission

Due Date: 2024/12/2 23:59

Please directly compress your code files, result file (.json), and report (.pdf) into {STUDENT ID}_hw4.zip and submit it to the New E3 System.

The file structure should look like:

```
📂{student_id}_hw4.zip
  ├ 📜 stp3_with_autopilot.py
  ├ 📜 stp3.py
  ├ 📜 report.pdf
  ├ 📜 pdm_hw4.json
  └ 📜 other files or folders # only needed if you modify some files in other folders
```

Wrong submission format leads to -10 point
Late submission leads to -20 points per day

References

- [1] ST-P3: End-to-end Vision-based Autonomous Driving via Spatial-Temporal Feature Learning
  https://arxiv.org/pdf/2207.07601
- [2] Lift, Splat, Shoot: Encoding Images from Arbitrary Camera Rigs by Implicitly Unprojecting to 3D
  https://research.nvidia.com/labs/toronto-ai/lift-splat-shoot/
- Carla coordinate system
  https://github.com/autonomousvision/carla_garage/blob/main/docs/coordinate_systems.md
- Carla Vehicle control
  https://carla.readthedocs.io/en/latest/python_api/#carlavehiclecontrol