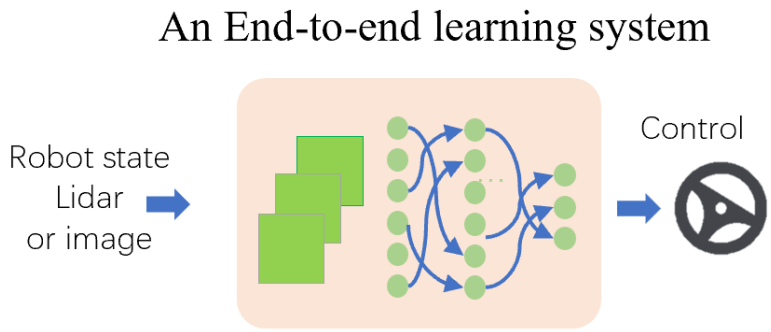
Learning for End-to-End Robot Policies: From Perception to Control

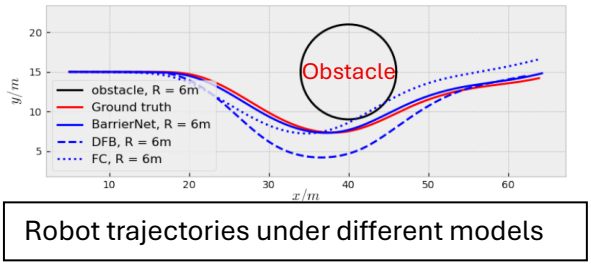
Table of Content

1. Introduction

"End-to-end control" refers to control mechanisms or processes that manage a task from its absolute beginning to its absolute conclusion, encompassing all the necessary steps and components without relying on external assistance for the core operation.

In this course project, we aim to build a (tiny) deep learning model that can learn control policies from state, Lidar or image observation of the robot while ensuring collision avoidance to the environment obstacles. A learning model framework is shown in the figure below:





Robot trajectories under different models

2. Software Setup

Install the following packges (and any others that are required) by running:

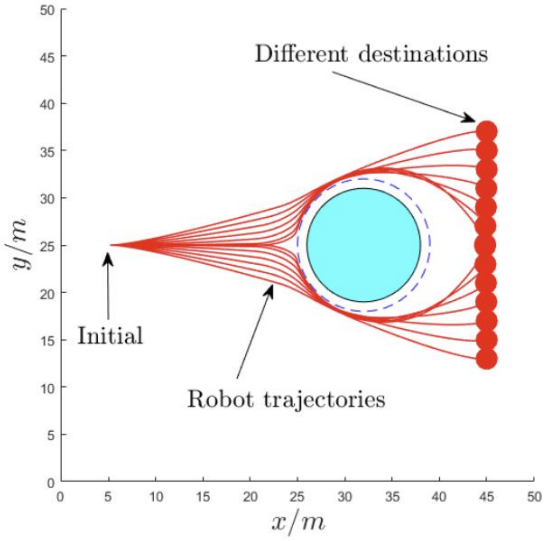
pip install torch==1.10.1+cu113 torchvision==0.11.2+cu113 -f https://download.pytorch.org/whl/cu113/torch_stable.html

pip install opencv-python==4.5.2.54 matplotlib==3.5.1 ffio==0.1.0 descartes==1.1.0 pyrender==0.1.45 pandas==1.3.5 shapely==1.7.1 scikit-video==1.1.11 scipy==1.6.3 h5py==3.1.0

pip install qpth cvxpy cvxopt

3. Learning from Robot State to Control (Option 1)

In this experiment, we have the robot state and destination of the robot as input for the model, and the output is the steering rate and acceleration control of the robot.



Dataset:

There are around 100 trajectories starting from the same initial location to different destinations (as shown in the figure above). There are three files:

dataM_train.mat – Dataset for training

dataM_valid.mat – Dataset for validation during training

dataM_test.mat – Dataset (a single trajectory) for open-loop or closed-loop testing

Data format: data[:,0:5] – x, y, theta, v, destination_y (destination_x is a constant/fixed)

data[:,5:7] – u1, u2 (steering and acceleration control of the model)

where (x,y,theta, v) denotes (location_x, location_y, orientation, linear speed)

The file reading and data loader are implemented in lines 32-66 of the file train.py

Task:

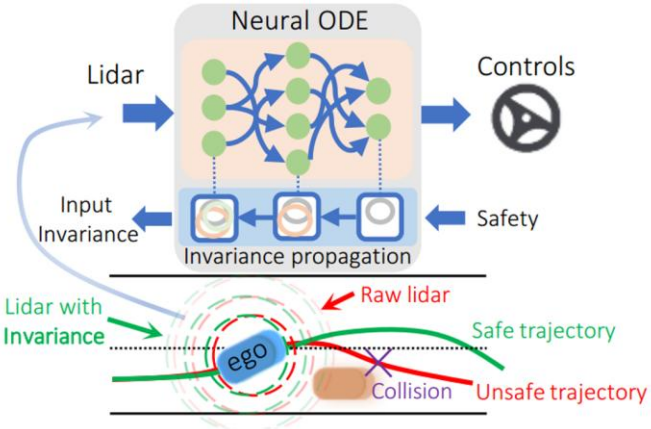
1. Build your own deep learning model based on the provided template (specifically, complete #todo parts in train.py, test.py, models.py)
2. Train the model to get reasonable performance (by tuning the model, training methods, etc.) – python train.py
3. Plot both open-loop and closed-loop control profiles (as instructed in train.py and test.py), the closed-loop robot trajectories (compared with ground truth trajectory), as well as plot the closed-loop safety profile – python test.py
4. Run the testing for at least 100 times with different destinations (within the range of training dataset), and get the statistics (create a table) of safety violation (e.g., how many runs violate the safety constraint in which case the minimum safety metrics (safety) becomes negative) and testing error
5. Add a safety filter layer (e.g., using the control barrier function method [1]) in the model to ensure the safety of robot, and then run the testing for at least 100 times with different destinations (within the range of training dataset) to get the statistics of safety violation and testing error, and add it to the table of task 4 above. (this is optional, count for an extra 50% score)

References:

[1] Wei Xiao, and Calin Belta. "High-order control barrier functions." *IEEE Transactions on Automatic Control* 67, no. 7 (2021): 3655-3662.

4. Learning from Lidar to Control (Option 2)

In this experiment, we have the Lidar sensor information of the robot as input for the model, and the output is the steering rate and acceleration control of the robot (an overtaking behavior in highway driving).



Dataset:

There are around 200 trajectories starting from the random initial location to random destinations when overtaking another moving vehicle (as shown in the figure above). There is one file: data4.pkl – Dataset for training, validation and testing

Data format: data['lidar']– 100 dimensions Lidar points from the ego vehicle

data['ctrl']– training labels (ground truth controls for steering, acceleration)

data['ego'] - (x,y,theta, v) of the ego vehicle

data['other'] - (x,y,theta, v) of other moving vehicle (vehicle being overtaken)

where (x,y,theta, v) denotes (location_x, location_y, orientation, linear speed)

The file reading is implemented in lines 51-76 of the file train.py

Task:

1. Build your own deep learning model based on the provided template (specifically, complete #todo parts in train.py, eval.py). The template uses a neural ODE model [1], you are welcome to use any other models (e.g., state space models, LSTM, etc).
2. Train the model to get reasonable performance (by tuning the model, training methods, etc.) – python train.py
3. Plot both open-loop and closed-loop control profiles (as instructed in train.py and test.py), the closed-loop robot trajectories (compared with ground truth trajectory), as well as plot the closed-loop safety profile – python eval.py
4. Run the testing for at least 100 times with Lidar noise (python eval.py --with_noise), and get the statistics (create a table) of safety violation (e.g., how many runs violate the safety constraint in which case the minimum safety metrics (safety) becomes negative) and testing error
5. Add a safety filter layer (e.g., using the control barrier function method [2]) in the model to ensure the safety of robots, and then run the testing for at least 100 times with noise to get the statistics of safety violation and testing error, and add it to the table of task 4 above. (this is optional, count for an extra 50% score)

References:

[1] Chen, Ricky TQ, Yulia Rubanova, Jesse Bettencourt, and David K. Duvenaud. "Neural ordinary differential equations." *Advances in neural information processing systems* 31 (2018).

[2] Wei Xiao, and Calin Belta. "High-order control barrier functions." *IEEE Transactions on Automatic Control* 67, no. 7 (2021): 3655-3662.

Some important parameters in train.py of the template:

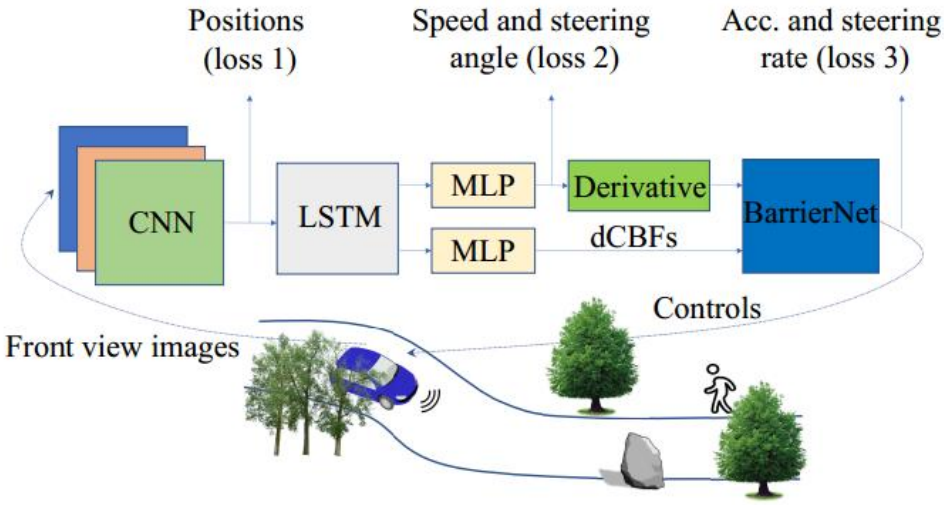
--batch_time: the sequence length of each sampled trajectory during training

--niters: number of training iterations looping through the whole dataset

--test_freq: frequence of validation during training

5. Learning from image to Control (Option 3, Advanced)

In this experiment, we have the front-view RGB image of the ego vehicle as input for the model, and the output is the steering rate and acceleration control of the vehicle in autonomous driving.



Tutorial and dataset: please refer to: https://github.com/Weixy21/BarrierNet

Codebase: https://github.com/Weixy21/BarrierNet/tree/main/Driving

Task:

1. The codebase is complete, re-implement on your computer [1]
2. Ablation studies of models, inputs, etc. (optional, count for an extra 50% score)

References:

[1] Xiao, Wei, Tsun-Hsuan Wang, Ramin Hasani, Makram Chahine, Alexander Amini, Xiao Li, and Daniela Rus. "Barriernet: Differentiable control barrier functions for

learning of safe robot control." *IEEE Transactions on Robotics* 39, no. 3 (2023): 2289-2307.

6. Grading rubrics

1. Complete all tasks (except optional safety filters): 30

2. Project reports with all necessary contents from all tasks: 30

3. Project presentations: 40

4. Add safety filters: 50 (bonus)

Total: full credits (100/100), credits with bonus (150/100)

Note: You may complete both task options 1 and 2, and there will also be a bonus based on the performance (determined by the lecture and TA).

7.Report guidelines

Report must be in pdf generated from the Latex template (provided in the project file).

Submission includes the following:

1. Complete code of the project
2. Report includes:
    - Open-loop control profiles compared with ground truth controls
    - Closed-loop control profiles compared with ground truth controls
    - Robot trajectories
    - Safety profiles (safety metrics)
    - A statistical table showing comparison of safety violation rates and testing errors
    - Safety filter formulation (optional)
3. Presentation slides (to be used for final course presentation)