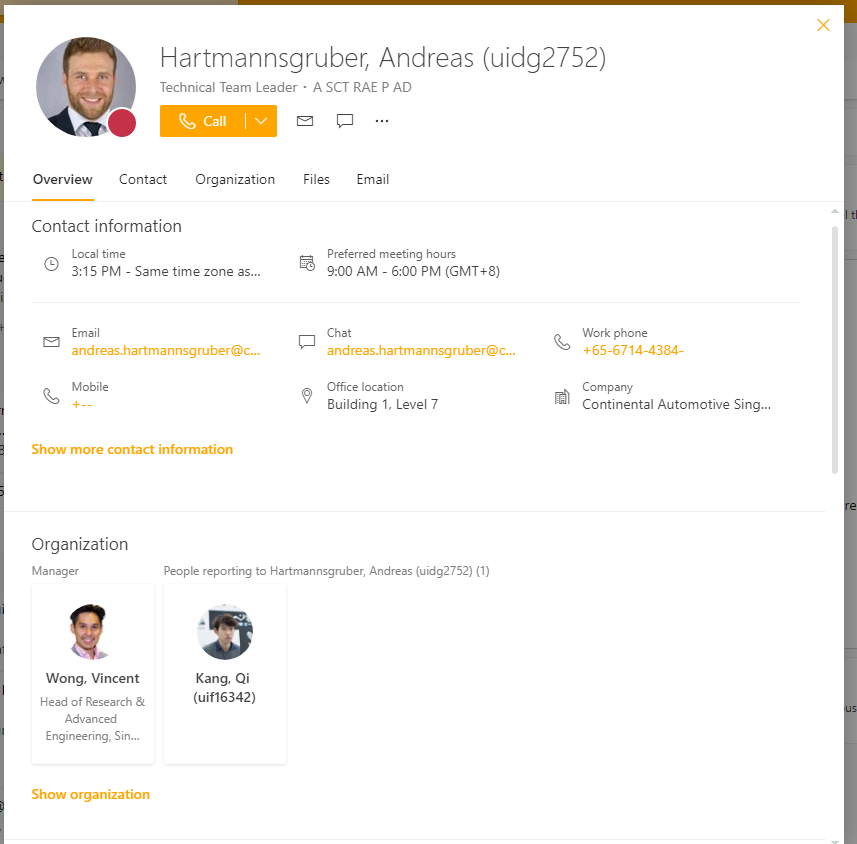
Reinforcement Learning for Truck Trailer

Problem statement

This project aims to devise a mid-end controller for truck trailer system based on hybrid deep reinforcement learning algorithm, which enables the autonomous tractor to drive safely in reaction to its current situation and plan short trajectories for the future.

The vehicle models we used for simulation is based on both a bus-like model and a truck&trailer system, and our training process will be conducted in 3D environment with respect to pybullet.



Existing works

According to our literature review, though there are quite a lot works on either deep reinforcement learning approach on autonomous vehicle and control approach on Truck& Trailer system, the practice on applying DRL algorithms for motion control problem is still limited. However, we still can somehow get some clues form these existing works:

Works on Autonomous driving based on DRL:

Works on Truck Trailer control:

Widyotriatmo and his team [2] adopt the classic kinematic model for truck& trailer, which help us understand the dynamics and establish the primary model for the vehicle

Zhou and his team [3] proposed a unique truck& trailer control model and achieve pretty good results, which can help us to think about more control strategies instead of being constraint by the classic tricycle kinematic model when we create the agent training environment.

Works on DRL based Truck& Trailer Control:

Bejar and his team [4,5] focusing on the backing up and path following issue of truck& trailer system. They applied the DDPG algorithm to devise the controller on the basis of classic system control method, i. e. finding the setpoint of the state and state space manipulation instead of planning the motion of the vehicle body. Also, the steering angle is discrete, which is not appropriate for real-world autonomous driving.

Carl-Johan and his team also working the DRL based truck& trailer manipulation and their focus is lane changing and the algorithm they applied is Double DQN.

potential outcome and contribution

the overall controller

{add flow chart here}

Agenda

environment set up

Recent progress

Reference

Investigation on simulators:

After one month struggling with the commonroad package, we found that its reinforcement learning function is not that functional even though they can provide us with rich library of road information and scenarios. Besides, they didn’t add the truck-trailer model into their garage, which means we need to wait for their next release if we want to implement that. In that case, we have to change to another simulator, the one we decide to use is pybullet. It is easy to start with and coordinate well with reinforcement algorithm. The vehicle model and scenarios are loaded in URDF format, which is simple to get from 3D modeling software such as Solidworks. The initial environment we built with pybullet is shown below and has passed all environment checkers.

Environment setup

For a reinforcement learning environment, the most vital parts will be the definitions of state space, action space and reward function. In the part we will talk about them in detail

State space:

For state space, we are going to make sure that the agent can perceive enough information for it to do the prediction and control itself. In that case, we devise a quite comprehensive state space representation in the form of 10-length-vector.

The information our state space representation contains can be classified in to 5 categories: coordinates, orientation, velocity, distance to goal and deviation.

For coordinates, we store the location of the agent mass point under world coordination. Only x and y coordinates are taken into consideration since we don’t have the vertically motions.

For Orientation, we get the yaw angle of the agent under the world coordination. Instead of directly using the angle, we use the cos and sin value of it.

For velocity, we put the velocity in x and y directions and angular velocity of the agent under world coordination.

Finally, we define a concept of deviation in our customized environment. In our environment, it is unnecessary to physically create a track model in simulator, we use a reference line, which is the middle lane of a track to represent the track. In that case, since we want the agent to keep to the reference line, we treat the distance between rear wheels of the vehicle and the reference line as the deviation. According to the definition, we have two elements: right rear wheel to the reference line and left rear wheel to the reference line.

The full state representation is shown as follows:

Action space

For action space we want to mimic the real control scheme of vehicle, so we only include the throttle and steering control of the agent. However, according to our discussion on the training results, the agent cannot slow down based on current action space. In that case we will replace the throttle with acceleration control in the future.

Reward function:

Since design of the reward function is one of the priorities that will affect the results of our training results, we will have several versions of that. The detailed reward function and the training results of it will be shown in section 4 training results.

Summary for experiments on Environment v1

Reward function v1

To begin with, our target is to train the agent following the reference line and moving forward to reach the target. In that case, we include 3 parts in the reward function: reward for proceeding, reward for sticking to the reference line and reward for reaching the goal. It is hard unpractical for the agent to stick to the reference line strictly, we loosen the constraint by adding a range region. The proceeding reward and rang reward will be given for each timesteps, which is in dense structure.

Then we got the training curve shown above. From the graph we can see that, after 20 million timesteps, it starts to going up. However, when we save the model and replay the policy in the animation, we found that it just stands still without moving forward, which is not an expected result. Then we abandon this reward structure. Also, in case of making the agent staying in the simulation forever, we set a timestep limit to be 1000 timesteps according to the maximum speed and the distance between the origin and the target.

In this experiment, we finish the tests on pybullet simulator, which turns out to be suitable for our project. Furthermore, we construct the vehicle model and the initial gym framework, which lays a good foundation for future research.

Summary for experiments on reward function v2

In this experiment, we try out the structure of the combination of dense and sparse reward function, from the learning curve we can see that, the reward keeps going up and converge at 25 million timesteps. However, the reward for the convergent one is till very low and act badly in animation. After analysis on the results, we think that the reward function might be too complex for the agent to learn and values for each part of the reward is not reasonable, which leads to the misunderstanding of the task for the agent. In that case, we decide to simplify the reward structure and only keep the dense part, which we think can help the agent to learn continuously.

Summary for experiments on reward function v3

In this section, we can see from the learning curve, the reward converges fast at around 800 thousand timesteps. Also, the curve going up smoothly without sudden peak or fall. After checking the animation, we found that the agent can follow the reference line for some time and move forward, but still end up deviating from it. It makes us to think whether it is a good way to craft the issue like that. After discussion about that, we decide to change the environment, using a series of discrete way point to represent the reference path which is easily to shift its shape and the length of it can be infinite. For that reason, we build our environment v2, which is a goal reaching task. From our perspective, once it can learn to reach one target, it will successively learn how to reach a series of waypoints.

Reward: the reward function we use for this environment is from paper[10], which is a linear function with respect to the movement of the vehicle

Summary for experiments on world coordination

From the three learning curves above we can conclude that, even they somewhat converge to the same value at the end of the training, when applying for a larger K, the highest reward the agent could get during the training process would be larger. When look deep into the structure and the value from the reference paper, the product of the K value and the distance between the origin point and the target should be big enough to offset the penalty of time out. In our case, it should be 13 or higher. According to the animation, the policy our agent learnt can approach to the target but hardly reach. One contributor of this issue may be due to the large exploration space when we apply the world coordination. In that case, we are going to adopt the relative representation of goal and eliminate the coordination of vehicle itself in observation space.

Summary for experiments on relative coordination

In this section, we adopt a relative coordination in state space. For the first experiment, we make the K variant to the starting distance between the agent and the goal by the relationship of K=1200/distance. In that case, the overall reward of the episode should be robust enough with different starting distances. However, it can be seen clearly from the training curve that the reward cannot converge, and the agent seldom reach the target according to the animation. In that case, variant K is not the key factor that can solve the problem.

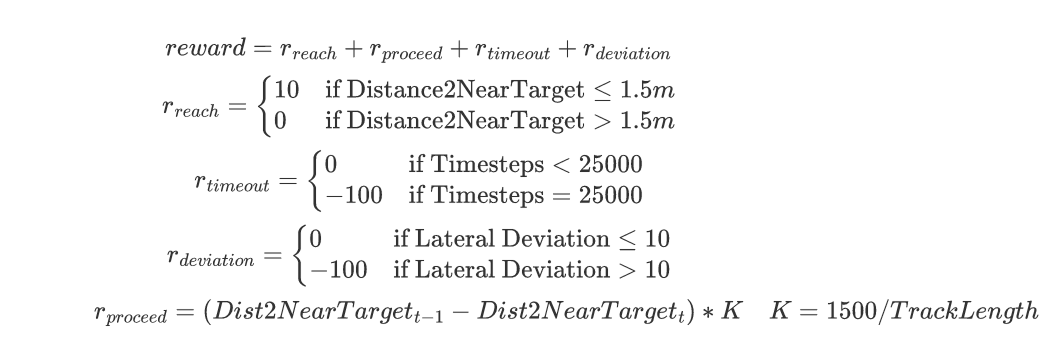
Then we try with lowering learning rate to be 5E-5, which is used to be 3E-3. Then we get the learning curve above. We can clearly observe that the reward converges fast within 1000 episodes. Also, the animation shows a successful rate of 100%, which is beyond our expectation and enabling us to move forward.

To sum up, in this goal reaching environment, we suffer a lot from wrong observation space, unreasonable k setting and high learning rate. They all contribute to our previous failure. Luckily, we manage to solve them all step by step, and have a practical experience with tuning the parameter of reinforcement leaning environment. Goal reaching environment is only an intermediate stage for our project. Since we have succeeded in solving it, we can move forward to the track environment.

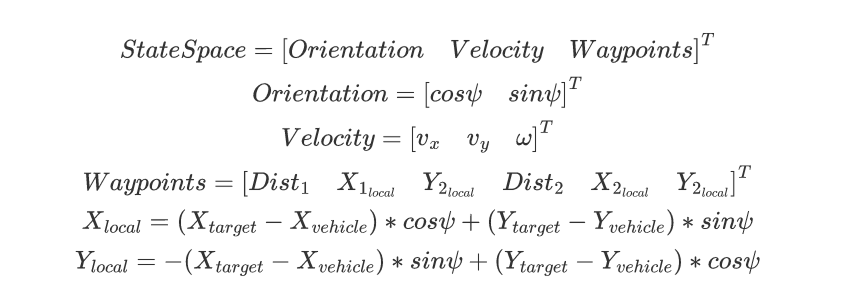
In the track environment we need the agent to have a broad view of the road so that we think more information need to be observed by the agent. The observation space for this setting will includes the local coordination of two waypoints under the vehicle ego coordination system. The agent will keep updating the target points when it surpasses the waypoints on the reference path. The criteria to whether to update the target point in view is that the longitudinal coordination of the current waypoint is negative, or the distance of the vehicle and the current target point is smaller than 1 meter. The full observation space is shown below.

The reward function we adopted for this environment was inherited from goal reaching task, which contain 4 parts reward for proceeding, penalty for time out, penalty for large deviation and reward for reaching waypoints. The full structure of the reward function is shown below. The total

Bus model state space:



In order to prove the advantage and necessity to observe more than one waypoints.



Truck trailer state space

