Motion Planning for Autonomous Driving

Philippe Weingertner, Minnie Ho

November 12, 2019

1 Introduction

2 Related Work

There is a rich litterature related to Motion Planning and a very detailed survey is provided in [9]. Among the first 4 successful participants of DARPA Urban Challenge in 2007, the approaches vary but they fundamentally rely on a graph search where nodes correspond to a configuration state and edges correspond to elementary motion primitives. The runtime and state space can grow exponentially large. In this context, the use of efficient heuristic is important.

More recently, Reinforcement Learning and Deep RL have been investigated in the context of Autonomous Driving for Decision Making either at the Behavioural Planning or Motion Planning level. In papers from Volvo [5] and BMW [4], a DQN RL agent is trained to take decision at a tactical level: selecting maneuvers. But the problem with Reinforcement Learning is that a utility is optimized in expectation. So even if this utility is designed to avoid collisions, this will be optimized in expectation only. In [4] an additional safety check layer is added after the DQN agent to eventually override the DQN agent decision. In [1] RL is applied in the local planner: the action space is a set of longitudinal accelerations applied along a given path at a T-intersection. The agent is constrained to choose among a restricted set of safe actions per state. So the safety is enforce before Deep RL. Ultimately we may want combine both types of safety checks before and after an RL agent.

In gaming, AlphaGo Zero [12] has defeated human world champions: thanks to a combination of MCTS tree search and learning with Deep RL. A neural network biases the sampling towards the most relevant parts of the search tree: a learnt policy-value function is used as a heuristic during inference. There are a few differences for Motion Planning. The state space is continuous, not discrete, and only partially observable. Also self-play can not be used. These challenges have been recently tackled in different publications. The applicability of AlphaGo Zero ideas to Autonomous Driving has been studied in [6, 2, 10]. In [10] the Motion Planning problem is addressed in a 2 steps path-velocity decomposition. The path planner employs hybrid A* to propose paths that are driveable and collision free wrt static obstacles. In a second step a velocity profile is generated by issuing acceleration commands. The problem is formulated as a POMDP model and solved with an online DESPOT solver. DESPOT is a sampling based tree search algorithm like MCTS which uses additional lower bounds and upper bounds values. To guide the tree search of DESPOT, a NavA3C neural network is used. The NavA3C network is trained in simulation and is expected to provide tighter bounds. In [6] the problem is considered at the behavioral planning level. A set of lane changes decisions are taken to navigate a highway and to reach an exit. The problem is formulated as a MDP problem. MCTS tree search is used as an online MDP solver and a learned policy-value network is used to efficiently guide the search.

We consider the problem of the local planner and velocity profile generation, similar to the first paper, but with an approach mainly aligned with the later one.

3 Test Setup

The problem statement is as follows. Given an ego vehicle (E) with a given path of (x, y) coordinates, find a set of acceleration decisions (a_x, a_y) at discrete time steps to enable E to avoid a set of intersecting vehicles $\{V\}$.

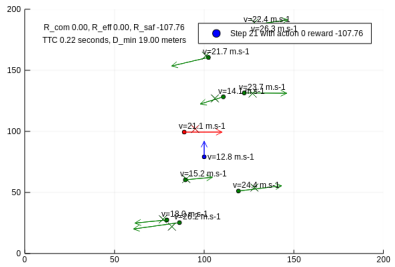

Figure 1: Custom openai gym Act-v1 env

The ego car in blue has to avoid 10 intersecting vehicles to reach a goal point. The position and speed of intersecting vehicles is not known precisely: the ground truth is represented via dots while the position reported by the sensors is represented by crosses. A Time To Collision based on ground truth information is displayed and if there exists an intersecting car with a computed TTC below 10 seconds it is displayed in red. We developped our own test framework but made it compatible with standard openai gym interfaces. Our simulation and test environment can be downloaded and installed from gym-act.

4 Approach

We are dealing with a local planning task where a path to follow has been derived by some other algorithms (a variational technique or some A* or D* tree search algorithm). We have to derive a set of acceleration commands, such that we avoid dynamical obstacles that may cross our path. We handle a sequential decision making problem under uncertainty: the sensors provide noisy estimates. We first use a model based approach (an online MCTS tree search algorithm). We benchmark it against a model free approach (Q-learning or DQN). We then combine the 2 approaches, to guide the MCTS tree search with a learned heuristic. We target a fast, safe and explainable solution as required for Autonomous Driving. We expect the safety and explainability to be obtained thanks to the model based approach and to gain some speed-up by using a model free learned heuristic.

4.1 MDP model

A representation of the states, position and speed information, in absolute coordinates would be $S_t = \left\{(x, y, v_x, v_y)_{\text{ego}}, (x, y, v_x, v_y)_{\text{obj}_{1..10}}\right\}$. But we use a relative and normalized representation for easier generalization and learning. In our setting the ego car drives along the y-axis only.

- **States:** $S = \left\{\left(\frac{y}{y^{max}}, \frac{v_y}{v_y^{max}}\right)_{\text{ego}}, \left(\frac{\Delta x}{\Delta x^{max}}, \frac{\Delta y}{\Delta y^{max}}, \frac{\Delta v_x}{\Delta v_x^{max}}, \frac{\Delta v_y}{\Delta v_y^{max}}\right)_{\text{obj}_{1..10}}\right\}$ which is a vector $\in \mathbb{R}^{42}$

While the state space is continuous we use a discrete action space.

- **Actions:** $A = \left[-2\ ms^{-2}, -1\ ms^{-2}, 0\ ms^{-2}, 1\ ms^{-2}, 2\ ms^{-2}\right]$ corresponding to the longitudinal acceleration.

The Transition model corresponds to standard linear Gaussian dynamics with:

- **Transitions:** $T(s' \mid s, a)$ for each car_i $P\left(S_i^{t+1} \mid S_i^t, a_i^t\right) = \mathcal{N}\left(\mu = T_s S_i^t + T_a a_i^t, \Sigma\right)$

Using a Constant Velocity Model with $T_s = \begin{bmatrix} 1 & 0 & \text{dt} & 0 \\ 0 & 1 & 0 & \text{dt} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, S_i^{t+1} = \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix}, T_a = \begin{bmatrix} \frac{\text{dt}^2}{2} & 0 \\ 0 & \frac{\text{dt}^2}{2} \\ \text{dt} & 0 \\ 0 & \text{dt} \end{bmatrix}, a_i^t = \begin{bmatrix} a_x \\ a_y \end{bmatrix}$. The reward model accounts for efficiency (we penalize every step), safety (we heavily pernalize collisions) and comfort (we penalize strong accelerations and decelerations:

- **Reward**: $R(s, a) = -1 - 1000 \times 1\left[\text{d(ego,obj)}_s \leq 10\right] - 1\left[|a| = 2\right]$

4.2 Algo 1, MCTS tree search

The MDP is solved online with MCTS tree search. Solving it offline with Value Iteration is not an option as we are dealing with a huge state space. MCTS [7] is one of the most successfull sampling-based online approaches used in recent years. It is the core part of AlphaGo Zero [11]. This algorithm involves running many simulations from the current state while updating an estimate of the state-action value function $Q(s,a)$ along its path of exploration. Online algorithms enable to reduce the search space to reachable states from a current state. MCTS balances exploration and exploitation via a method called Upper Confidence Bound: during the search we execute the action that maximizes $Q(s,a)+c\sqrt{\frac{\log N(s)}{N(s,a)}}$ where $N(s), N(s,a)$ track the number of times a state and state-action pair are visited. c is a parameter controlling the amount of exploration in the search: it will encourage exploring less visited (s,a) pairs and rely on the learned policy via $Q(s,a)$ estimates for pairs that are well explored. Once we reach a state that is not part of the explored set, we iterate over all possible actions from that state and expand the tree. After the expansion stage, a rollout is performed: it consists in running many random simulations till we reach some depth. It is a Monte Carlo estimate so the rollout policy is typically stochastic and does not have to be close to optimal. The rollout policy is different than the policy used for exploration. Simulations, running from the root of the tree down to a leaf node expansion, followed by a rollout evaluation phase, are run until some stopping criterion is met: a time limit or a maximum number of iterations. We then execute the action that maximizes $Q(s,a)$ at the root of the tree. The pseudo code of the algorithm is provided below:

```
function SelectAction(s, d)
    loop
        Simulate(s, d, π_0)
    end loop
    return arg max_a Q(s, a)
end function
```

One remaining problem is that the above algorithm does not cope with continuous state space: the same state may never be sampled more than once from the generative model which will result in a shallow tree with just one layer. The Progressive Widening variant of MCTS [13, 3] solves this problem by controlling the sampling of new states and the sampling among already existing states to enable exploration in depth and not just in breadth.

```
function Simulate(s, d, π_0)
    if d = 0 then
        return 0
    end if
    if s ∉ T then
        for a ∈ A(s) do
            (N(s, a), Q(s, a)) ← (N_0(s, a), Q_0(s, a))
        end for
        T = T ∪ {s}
        return Rollout(s, d, π_0)
    end if
    a ← arg max_a Q(s, a) + c sqrt(logN(s) / N(s,a))
    (s', r) ~ G(s, a)
    q ← r + λ Simulate(s, d − 1, π_0)
    N(s, a) ← N(s, a) + 1
    Q(s, a) ← Q(s, a) + (q − Q(s, a)) / N(s, a)
    return q
end function
```

```
function Rollout(s, d, π_0)
    if d = 0 then
        return 0
    end if
    a ~ π_0(s)
    (s', r) ~ G(s, a)
    return r + λ Rollout(s', d − 1, π_0)
end function
```

4.3 Algo 2, Approximate Q-learning

While similarly to [12, 2, 6, 11, 10] we intend to use Deep Learning to learn an evaluation function to be used as a heuristic to guide the MCTS tree search, we will first investigate Approximate Q-learning with a

reduced set of features. Weuse the following features extractor. $\phi(s,a) = \begin{bmatrix} s_{6\times1} & a_{1\times1} & s^2_{6\times1} & a^2_{1\times1} \end{bmatrix}^T$ where we take into account the state vector of the ego car (2 components) and the state vector of the car with the smallest TTC (4 components). We also take into account the acceleration command of the ego car. We use quadratic components as well: as it is expected that the value of a (s,a) tuple will depend on distances computations. We focus on a reduced set of relevant features to speed up training. The Q-function is parametrized by a vector w with $\hat{Q}_{\text{opt}}(s,a;\mathbf{w}) = \mathbf{w}\cdot\phi(s,a)$. We have $\hat{V}_{\text{opt}}(s') = \max_{a'\in\text{Actions}(s')} \hat{Q}_{\text{opt}}(s',a')$ and use the objective $\left(\hat{Q}(s,a;\mathbf{w})_{\text{pred}} - \left(r+\gamma\hat{V}_{\text{opt}}(s')\right)_{\text{targ}}\right)^2$ which leads to the following update rule while performing Sochastic Gradient Descent: $\mathbf{w} \leftarrow \mathbf{w} - \eta\left[\hat{Q}_{\text{opt}}(s,a;\mathbf{w})_{\text{pred}} - \left(r+\gamma\hat{V}_{\text{opt}}(s')\right)_{\text{targ}}\right]\phi(s,a)$. One of the problem we may encounter is that the data in simulation is not iid, but highly correlated from one simulation step to the other and the targets will vary a lot. This problem is typically handled by using an experience replay buffer, which is possible with an off policy algorithm, and using a different fixed Q-network for targets evaluation, which is updated less frequently than the Q-function used for predictions, as described in DeepMind papers [14, 8].

4.4 Algo 3, Deep Q-learning

We use a DQN algorithm with a replay memory buffer to ensure we are dealing with iid samples and a target network, updated less frequently than the optimisation network, to stabilize the training procedure as described in DeepMind papers [14, 8]. We use a Huber Loss which acts as MSE when error is small and as a mean absolute error when error is large: to make it more robust to outliers when the estimates of Q are very noisy. Exploration is done with an ϵ -greedy policy. We use a batch size of 128, $\gamma = 0.999$ and Adam optimizer for the Batch Gradient Descent updates. Programming is done with pytorch and we leverage on this code pytorch.org as a starting point for the DQN setup.

Our Neural Network has 42 neurons as input, corresponding to all cars position and speed information (relative and normalized coordinates), and 5 neurons as output, corresponding to the 5 possible control commands. We use a CNN architecture similarly to [5], leveraging on CNN translational invariance properties: the order of information about different cars should not matter.

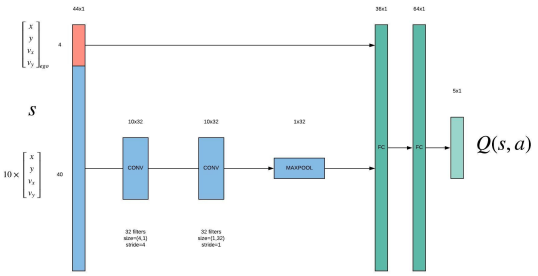

Figure 2: Q-function as a CNN network

4.5 Algo 4, MCTS tree search with a learned heuristic

We plan to use our learned Q-network $\hat{Q}(s,a;\mathbf{w})$ via approximate Q-learning or Deep Q-learning as a heuristic for MCTS tree search: to expand the tree in the most promising areas and hence come up faster with a good solution. A solution is considered good as soon as it is estimated collision free. We are dealing with uncertainty so it is an estimate. We may run further MCTS tree searches up to some time limit, to find even better solutions: faster or more comfortable.

5 Experiments

The source code is available here: CS221 Project. The baseline (simple rule - reflex based) and oracle (assuming no uncertainty using UCS/A* tree search) have been implemented at the proposal stage. For the progress report, we have a first version of Q-learning with some results summarized in the appendix. We are working on the MCTS implementation.

References

[1] Maxime Bouton et al. "Reinforcement Learning with Probabilistic Guarantees for Autonomous Driving". In: *CoRR* abs/1904.07189 (2019). arXiv: 1904.07189. URL: http://arxiv.org/abs/1904.07189.

[2] Panpan Cai et al. "LeTS-Drive: Driving in a Crowd by Learning from Tree Search". In: *CoRR* abs/1905.12197 (2019). arXiv: 1905.12197. URL: http://arxiv.org/abs/1905.12197.

[3] Adrien Couëtoux et al. "Continuous Upper Confidence Trees". In: *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*. LION'05. Rome, Italy: Springer-Verlag, 2011, pp. 433–445. ISBN: 978-3-642-25565-6. DOI: 10.1007/978-3-642-25566-3_32. URL: http://dx.doi.org/10.1007/978-3-642-25566-3_32.

[4] Andreas Folkers, Matthias Rick, and Christof BÃŒskens. "Controlling an Autonomous Vehicle with Deep Reinforcement Learning". In: June 2019. DOI: 10.1109/IVS.2019.8814124.

[5] Carl-Johan Hoel, Krister Wolff, and Leo Laine. "Automated Speed and Lane Change Decision Making using Deep Reinforcement Learning". In: *CoRR* abs/1803.10056 (2018). arXiv: 1803.10056. URL: http://arxiv.org/abs/1803.10056.

[6] Carl-Johan Hoel et al. "Combining Planning and Deep Reinforcement Learning in Tactical Decision Making for Autonomous Driving". In: *CoRR* abs/1905.02680 (2019). arXiv: 1905.02680. URL: http://arxiv.org/abs/1905.02680.

[7] Mykel J. Kochenderfer. *Decision Making Under Uncertainty: Theory and Application*. MIT Press, 2015.

[8] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

[9] B. Paden et al. "A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles". In: *IEEE Transactions on Intelligent Vehicles* 1.1 (Mar. 2016), pp. 33–55. DOI: 10.1109/TIV.2016.2578706.

[10] F. Pusse and M. Klusch. "Hybrid Online POMDP Planning and Deep Reinforcement Learning for Safer Self-Driving Cars". In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. June 2019, pp. 1013–1020. DOI: 10.1109/IVS.2019.8814125.

[11] David Silver et al. "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm". In: *CoRR* abs/1712.01815 (2017). arXiv: 1712.01815. URL: http://arxiv.org/abs/1712.01815.

[12] David Silver et al. "Mastering the game of Go without human knowledge". In: *Nature* 550 (2017), pp. 354–359.

[13] Zachary Sunberg and Mykel J. Kochenderfer. "POMCPOW: An online algorithm for POMDPs with continuous state, action, and observation spaces". In: *CoRR* abs/1709.06196 (2017). arXiv: 1709.06196. URL: http://arxiv.org/abs/1709.06196.

[14] Chris Urmson et al. "Autonomous Driving in Traffic: Boss and the Urban Challenge". In: *AI Magazine* 30 (June 2009), pp. 17–28. DOI: 10.1609/aimag.v30i2.2238.