Motion Planning for Autonomous Driving

Philippe Weingertner, Minnie Ho

November 8, 2019

1 Introduction

2 Related Work

There is a rich litterature related to Motion Planning and a very detailed survey of traditional methods is provided in [14]. Among the first 4 successful participants of DARPA Urban Challenge in 2007, the approaches vary. The winner, CMU Boss vehicle used variational techniques for local trajectory generation in a structured environment. This was done in a 2 steps path-velocity decomposition. A first step of path planning, using variational techniques, is performed and for every candidate path, a combination of different velocity profiles (constant, linear, linear ramp, trapezoidal) is applied : the combination of a path and velocity profile defines a trajectory. In unstructured environments (parking lots) or in error recovery situations a lattice graph in 4-dimensional configuration space (position, orientation and velocity) is searched with Anytime D* algorithm to find a collision-free path. More details are provided in [6, 1, 12]. The vehicle from Stanford used a search strategy coined Hybrid A* that constructs a tree of motion primitives by recursively applying a finite set of maneuvers. The search was guided by a carefully designed heuristic. The vehicle arriving 3rd, Victor Tango from Virginia Tech, constructs a graph discretization of possible maneuvers and searches the graph with the A* algorithm. The vehicle arriving 4th, developed by MIT used a variant of RRT algorithm with biased sampling. While all these techniques differ, they fundamentally rely on a graph search where nodes correspond to a configuration state and edges correspond to elementary motion primitives. Although they provide solutions, the runtime and state space can grow exponentially large. In this context, the use of heuristic to guide the search is important. (**NB: uncertainty is not properly considered here in these traditional MP methods. It is like reasoning with the mean state vector provided by sensor fusion output and ignoring the covariance matrix !**)

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 some research papers from Volvo [8] and BMW [7] , an RL agent is trained in simulation to take decision at a higher tactical level: the decisions relate to a maneuver selection, like lane change, rather than a low level acceleration command. DQN is used to train an agent. But the problem with Reinforcement Learning is that the utility is optimized in expectation. So even if the reward is designed to avoid collisions, this will be optimized in expectation: ultimately it is as if safety would be enforced with soft constraints rather than hard constraints. Which is of course not acceptable for a real vehicle. To solve this problem in [7] an additional safety check layer is added after the DQN agent to eventually override the DQN agent decision if it is considered unsafe. Checking a decision wrt to a specific criteria is simpler than designing a decision making system that jointly optimizes efficiency, comfort and safety objectives. With RL applied to AD we have to account for additional safety checks. In [2] Deep RL is applied at the local planner level: the action space is a set of longitudinal accelerations $\left\{-4m/s^2, -2m/s^2, 0m/s^2, 2m/s^2\right\}$ applied along a given path at a T-intersection. Safety is handled in a different way here compared to previous BMW approach: the agent is constrained to choose among a restricted set of safe actions per state. So the safety is enforce before Deep RL. Ultimately car manufacters may want to combine both types of safety checks: constraining the action set per state before enabling an RL agent to make its own decision, and checking again the final sequence of decisions proposed by the RL agent.

Now the interesting topic is how to best combine traditional Motion Planning with RL. What are the limitations of these techniques in isolation and how to use the strengths of both approaches and circumvent their weaknesses. Traditional motion planning relies heavily on tree search and to enable real time solutions good heuristics are required. Designing a good heuristic is hard. What if we could learn it ? By training an agent with model free RL we can potentially end up with an agent that performs pretty well most of the time and from time to times fails miserably in a way that is hard to explain. The main problems with model free RL are sample efficiency (we need a lot of data), enforcing hard constraints and explainability (how can we explain the decision taken by a RL agent which may become a problem for a car manufacturer). While a model based planning method has the advantages of explainability, do not rely on data and can deal in a more systematic way with hard constraints. As demonstrated in [3] in simple situations RL methods have no benefit over rule based methods, pure RL does not enable the agent to act in safer way. But when the situation becomes much more complex with an increasing number of cars and pedestrians, the benefits of Deep RL methods become clear.

In the gaming domain, chess and go, performances superior to human performances have been achieved with AlphaGo Zero [18]: by combining planning with MCTS tree search and learning with 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. While during training, MCTS is used to improve the sample efficiency of RL training. Now there are a few major differences between a game like chess or go and our initial Motion Planning problem. In chess or go the state space is discrete and fully observable while in AD the state space is continuous and partially observable. In terms of action sets in both cases, we can deal with discrete action sets. But another challenge is that self-play can not be used in the context of Motion Planning. These challenges have been recently tackled in different publications. The applicability of AlphaGo Zero to Autonomous Driving has been studied in [9, 4, 15].

TODO:

- analysis of the 3 most relevant papers
- highlight what is specific to our use case

3 Approach

A few notes + baseline / oracle recap + explain our Custom openai-gym env

3.1 MDP model

TODO: use a relative representation of the state (to reduce state space "area", enable easier generalization/learning etc ...)

- **State**: $S_t = \{S_i^t\}_{i=0..11} = \left\{(x, y, v_x, v_y)_{\text{ego}}, (x, y, v_x, v_y)_{\text{obj}_{1..10}}\right\}$
  - with $S_i^t = [x, y, v_x, v_y]^T$ and $i \in [0, 11]$

We use a relative and normalized $\in [-1, 1]$ representation of the state and account for the fact that the ego car drives along the y-axis only (in a further simplified first version we could even assume that the 10 other cars drive only along the x-axis, crossing orthogonally to the ego car path):

- **State**: $S_t = \{S_i^t\}_{i=0..11} = \left\{\left(\frac{y}{y^{max}}, \frac{v_y}{v_y^{max}}\right)_{\text{ego}}, \left(\frac{\Delta x}{\Delta x^{max}}, \frac{\Delta y}{\Delta y^{max}}, \frac{\Delta v_x}{\Delta v_x^{max}}, \frac{\Delta v_y}{\Delta v_y^{max}}\right)_{\text{obj}_{1..10}}\right\}$
- **Actions**: $a \in \left[-2\ ms^{-2}, -1\ ms^{-2}, 0\ ms^{-2}, 1\ ms^{-2}, 2\ ms^{-2}\right]$
  - for ego vehicle we choose an acceleration along y-axis

- **Transitions**: $T\left(s' \mid s,a\right) = P\left(S_i^{t+1} \mid S_i^t, a\right) = \mathcal{N}\left(T_s S_i^t + T_a \begin{bmatrix} a_x \\ a_y \end{bmatrix}\right)$
  - Linear Gaussian dynamics with a Constant Velocity Model
  - $T_s = \begin{bmatrix} 1 & 0 & \mathrm{dt} & 0 \\ 0 & 1 & 0 & \mathrm{dt} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, T_a = \begin{bmatrix} \frac{\mathrm{dt}^2}{2} & 0 \\ 0 & \frac{\mathrm{dt}^2}{2} \\ \mathrm{dt} & 0 \\ 0 & \mathrm{dt} \end{bmatrix}$
  - $\begin{bmatrix} x^{t+1} \\ y^{t+1} \\ v_x^{t+1} \\ v_y^{t+1} \end{bmatrix} = \begin{bmatrix} x^t + v_x^t \mathrm{dt} \\ y^t + v_y^t \mathrm{dt} \\ v_x^t \\ v_y^t \end{bmatrix} + \begin{bmatrix} a_x \frac{\mathrm{dt}^2}{2} \\ a_y \frac{\mathrm{dt}^2}{2} \\ a_x \mathrm{dt} \\ a_y \mathrm{dt} \end{bmatrix} + \begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_{v_x} \\ \sigma_{v_y} \end{bmatrix}$
- **Reward**: efficiency + safety + comfort
  - $R_t = -1 - 1000 \times 1\left[\mathrm{d(ego,obj)} \leq 10\right] - 1\left[|a_t| = 2\right]$

3.2 Algo 1, MCTS tree search

The MDP is solved online via MCTS tree search. Solving it offline with Value Iteration is not an option as we are dealing with a huge state space (it is actually a continuous state space and if we want to discretize it it will be huge anyways). This planning method, RL Model based, has a complexity that does not grow exponentially with the horizon.

```
function SelectAction(s, d)
    loop
        Simulate(s, d, π0)
    end loop
    return arg max_a Q(s, a)
end function
```

```
function Simulate(s, d, π0)
    if d = 0 then
        return 0
    end if
    if s ∉ T then
        for a ∈ A(s) do
            (N(s, a), Q(s, a)) ← (N0(s, a), Q0(s, a))
        end for
        T = T ∪ {s}
        return Rollout(s, d, π0)
    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, π0)
    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, π0)
    if d = 0 then
        return 0
    end if
    a ∼ π0(s)
    (s', r) ∼ G(s, a)
    return r + λ Rollout(s', d − 1, π0)
end function
```

TODO: add progressive widening to deal with a continuous state space.

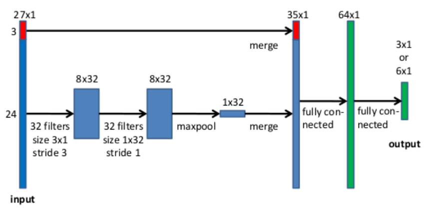
3.3 Algo 2, Approximate Q-learning

- Model Free RL algorithm:
  - $\hat{Q}_{\mathrm{opt}}(s, a; \mathbf{w}) = \mathbf{w} \cdot \phi(s, a)$
  - $\hat{V}_{\mathrm{opt}}(s') = \max\limits_{a' \in \mathrm{Actions}(s')} \hat{Q}_{\mathrm{opt}}(s', a')$
  - Objective$= \left(\hat{Q}_{\mathrm{opt}}(s, a; \mathbf{w})_{\mathrm{pred}} - \left(r + \gamma \hat{V}_{\mathrm{opt}}(s')\right)_{\mathrm{targ}}\right)^2$
  - $\mathbf{w} \leftarrow \mathbf{w} - \eta\left[\hat{Q}_{\mathrm{opt}}(s, a; \mathbf{w})_{\mathrm{pred}} - \left(r + \gamma \hat{V}_{\mathrm{opt}}(s')\right)_{\mathrm{targ}}\right]\phi(s, a)$
- Features Extractor 1: $\phi(s, a) = \begin{bmatrix} s_{42\times1} \\ a_{1\times1} \\ s_{42\times1}^2 \\ a_{1\times1}^2 \end{bmatrix}$

  The state vector has 42 components (10 cars with 4 components each and 2 components for the ego car), action is a single number. We use quadratic components as well: as it is expected that the value of a (s, a) tuple will depend on distances computations. Typically when computing Time To Collision, quadratic terms appear so we want to provide these relevant features and figure out by learning the weights associated to these features.
- Features Extractor 2: $\phi(s, a) = \begin{bmatrix} s_{6\times1} \\ a_{1\times1} \\ s_{6\times1}^2 \\ a_{1\times1}^2 \end{bmatrix}$ we just take into account ego car + closest car to work with a minimum number of features. It may be easier to work with; to speed up learning.

3.4 Algo 3, Deep Q-learning

Model Free RL method. Our Neural Network has 42 neurons as input, corresponding to the state vector, and 5 neurons as output. We will use a Neural Network architecture slightly adapted from [8]. It is based on a CNN network as we want to have translational invariance of the input. It should not matter to provide information about different cars in one order or the other.



3.5 Algo 4, MCTS tree search with a learned heuristic

Combining Planning and Learning, Model Based and Model Free RL. TODO post progress report.

We want 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 may run further MCTS tree search up to some time limit, to find even better solutions: faster or more comfortable. The reward takes into account safety, comfort and efficiency. While there is a big penalty for collisions of -1000, at every time step we penalize the reward by -1 to enforce efficiency and penalize every strong acceleration or deceleration, when $|a| = 2 \quad ms^{-2}$, by -2 to encourage more comfortable trajectories.

4 Experimental Setup and Status

The source code is available here: CS221 Project

- Baseline: simple rule - reflex based. DONE
- Oracle: assumes no uncertainty, UCS/A* tree search. DONE
- Sequential Decision Making with Uncertainty => solve a MDP
  - Planning with MCTS tree search. ONGOING: MDP model implementation + our own MCTS algo implementation
  - Approximate Q-learning: leverage on CS221 hw4 blackjack setup + custom Features Extractor + interface with our MDP model
  - Deep Q-learning: basically the pytorch DQN setup from pytorch.org customized with our proposed CNN network and interfaced with our own gymai environment (gymai-act for Anti Collision Tests)
  - Combining Planning and Learning: we should have MCTS and Q-learning working independently first ..

References

[1] David Bissell et al. "Autonomous automobilities: The social impacts of driverless vehicles". In: *Current Sociology* (Dec. 2018), p. 001139211881674. DOI: 10.1177/0011392118816743.

[2] 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.

[3] Maxime Bouton et al. "Safe Reinforcement Learning with Scene Decomposition for Navigating Complex Urban Environments". In: *CoRR* abs/1904.11483 (2019). arXiv: 1904.11483. URL: http://arxiv.org/abs/1904.11483.

[4] 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.

[5] Michael Everett, Yu Fan Chen, and Jonathan P. How. "Motion Planning Among Dynamic, Decision-Making Agents with Deep Reinforcement Learning". In: *CoRR* abs/1805.01956 (2018). arXiv: 1805.01956. URL: http://arxiv.org/abs/1805.01956.

[6] Dave Ferguson, Thomas M. Howard, and Maxim Likhachev. "Motion Planning in Urban Environments". In: *The DARPA Urban Challenge*. 2009.

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

[8] 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.

[9] 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.

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

[11] Edouard Leurent. "A Survey of State-Action Representations for Autonomous Driving". working paper or preprint. Oct. 2018. URL: https://hal.archives-ouvertes.fr/hal-01908175.

[12] M. McNaughton et al. "Motion planning for autonomous driving with a conformal spatiotemporal lattice". In: *2011 IEEE International Conference on Robotics and Automation*. May 2011, pp. 4889–4895. DOI: 10.1109/ICRA.2011.5980223.

[13] Subramanya Nageshrao, H. Eric Tseng, and Dimitar P. Filev. "Autonomous Highway Driving using Deep Reinforcement Learning". In: *CoRR* abs/1904.00035 (2019). arXiv: 1904.00035. URL: http://arxiv.org/abs/1904.00035.

[14] 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.

[15] 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.

[16] Markus Schratter et al. "Pedestrian Collision Avoidance System for Scenarios with Occlusions". In: *CoRR* abs/1904.11566 (2019). arXiv: 1904.11566. URL: http://arxiv.org/abs/1904.11566.

[17] 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.

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