# Motion Planning for Autonomous Driving

*Philippe Weingertner, Minnie Ho*

*pweinger@stanford.edu, minnieho@stanford.edu*

Stanford
CS 221 Final Project

## Objective

- Given a path and ten other moving vehicles, determine a velocity profile for an ego vehicle.
- Velocity profile: collision-free, comfortable (no huge brake/floor), efficient (move to goal), and real-time (<250 ms).

## Motivation

- The challenge is solving a **continuous state-space** problem in **real-time**: high state space complexity requiring fast decision
- Combine planning methods (Tree Search based) with learning based methods in the most efficient real-time/accuracy tradeoff

## MDP Model

To make it generic and scalable to uncertainties handling (sensors and drivers) we model the problem as a MDP model

- **State space**: is continuous (scene with 11 cars)
- **Action set**: is a discrete set of 5 acceleration commands

- **States:** $S^t = \left\{(x, y, v_x, v_y)_{\text{ego}}, (x, y, v_x, v_y)_{\text{obj}_{1..10}}\right\}$, meters $m$ for positions $(x, y)$ and $\frac{m}{s}$ for velocities $(v_x, v_y)$.
- **StartState:** $S^0 = \left[100, 0, 0, 0, (x, y, v_x, v_y)_{\text{obj}_{1..10}}\right]$, where the object vehicle positions and velocities are chosen from uniform distributions.
- **EndState:** $S^E = [100, 200, ?, ?, ..., ?]$, where the end goal occurs when the ego vehicle reaches the goal.
- **Actions:** $A = [-2, -1, 0, 1, 2]$, with $\frac{m}{s^2}$ for acceleration.
- **Transitions:**

$$T\left(S_i^{t+1} \mid S_i^t, a_i^t\right) = \begin{cases} T_s S_i^t + T_a a_i^t & \text{for ego} \\ T_s S_i^t & \text{for obj} \end{cases}$$

$$T_s = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, S_i^{t+1} = \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix}, T_a = \begin{bmatrix} \frac{dt^2}{2} & 0 \\ 0 & \frac{dt^2}{2} \\ dt & 0 \\ 0 & dt \end{bmatrix}, a_i^t = \begin{bmatrix} a_x \\ a_y \end{bmatrix}.$$

- **Reward**:

$$R_1(s,a) = -1 - 1000 \times \mathbb{1}\left[\min_{\text{obj}\in 1..10} \|(x,y)_{ego} - (x,y)_{obj}\| \leq 10\right] - \mathbb{1}\left[\|a\| \geq 2\right]$$

$$R_2(s,a) = \mathbb{1}\left[(v_y)_{ego} \geq 0\right]\left(1 - e^{-\text{TTC}/100}\right) + \mathbb{1}\left[\text{TTC} >> 100\right]\frac{(v_y)_{ego} - 20}{20}$$

$R_2$ is used by algorithms that require a normalized reward between (0,1) for convergence. TTC is Time-to-Collision.

- **Discount**: $\gamma = 1$.

## Selected References for Poster

[1] Carl-Johan Hoel et al.: Combining Planning and Deep Reinforcement Learning in Tactical Decision Making for Autonomous Driving. https://arxiv.org/abs/1905.02680

[2] David Silver et al.: Mastering the game of go without human knowledge. alphago-zero-starting-scratch

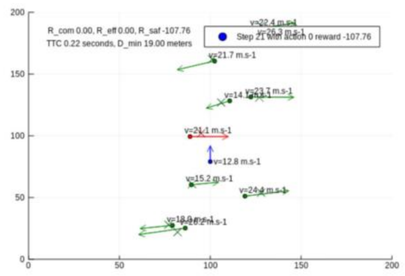
[3] Panpan Cal et al.: LeTS-Drive: Driving in a Crowd by Learning from Tree Search. https://arxiv.org/abs/1905.12197

## Acknowledgments

We thank Chris Waites for his project guidance.

## Test Setup

- 5 vehicles (objects to avoid) from right.
  Initial pos~(U(0,50), U(25,190)) and vel~(U(10,25), U(0,5)) m/s
- 5 vehicles (objects to avoid) from the left.
  Initial pos~(U(150,200), U(25,190)) and vel~(-U(10,25), -U(0,5)) m/s
- Ego vehicle travels in y-direction only, with constant acceleration.
- Object vehicles have zero acceleration, and constant velocity.
- Collision when distance(ego, car) < 10 meters
- Naive policies, random actions, constant speed lead to collisions




Test Setup, based on OpenAI Gym

## Algorithms

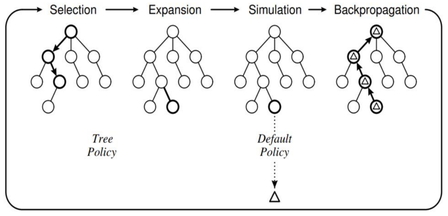
- **Baseline**: Reflex-based (if min TTC < 10, decelerate, else accelerate)
- **Oracle**: Uniform Cost Search and Dynamic Programming
- **Value Iteration**: used over a subset of the state space (limited depth)
- **Q-learning**: with feature extraction
- **DQN**: Deep Q-learning with a neural network (DNN and CNN have been implemented)
- **MCTS**: Monte Carlo Tree Search
- **MCTS enhanced with a pre-trained DQN neural network, acting as a heuristic (a la AlphaGo Zero)**
  - Monte Carlo Rollouts are replaced with nnet.GetV(s)
  - Default Q(s,a) initialisation is replaced with nnet.GetQ(s,a)

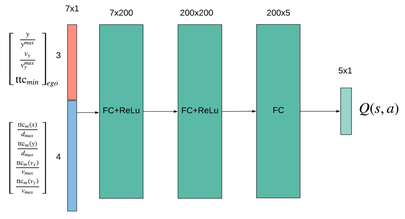
Feature Extraction $o \in 1, 2, 3, 4$ :

$$\phi_1(s,a) = \text{bias} = 1, \phi_2(s,a) = \text{TTC}_x^o, \phi_3(s,a) = \text{TTC}_{v_x}^o, \phi_4(s,a) = \text{TTC}_{v_y}^o$$

$$\phi_5(s,a) = \text{y}^o, \phi_6(s,a) = \text{v}_y^o, \phi_7(s,a) = \text{TTC}^o, \phi_8(s,a) = \text{action}^o$$

## **Proposed MCTS-NNET:** Monte-Carlo Tree Search enhanced with a DQN heuristic




1: **function** SELECTACTION($s, d$)
2: **loop**
3: SIMULATE($s, d, \pi_0$)
4: **end loop**
5: **return** arg $\max_a$ $Q(s,a)$
6: **end function**

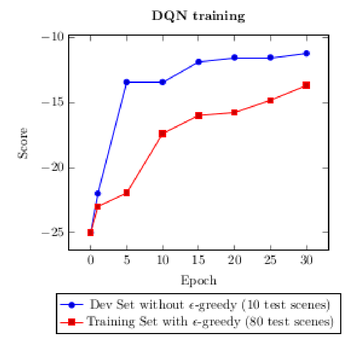
1: **function** ROLLOUT($s, d, \pi_0$)
2: **if** $d = 0$ **then**
3: **return** 0
4: **end if**
5: **if** mcts-nnet **then**
6: **return** nnet.getV(s)
7: **else**
8: $a \sim \pi_0(s)$
9: $(s', r) \sim G(s, a)$
10: **return** $r + \lambda$ ROLLOUT($s', d-1, \pi_0$)
11: **end if**
12: **end function**

1: **function** SIMULATE($s, d, \pi_0$)
2: **if** $d = 0$ **then**
3: **return** 0
4: **end if**
5: **if** $s \notin T$ **then**
6: **for** $a \in A(s)$ **do**
7: **if** mcts-nnet **then**
8: $Q(s,a), N(s,a) \leftarrow$nnet.getQ(s,a), 1
9: **else**
10: $Q(s,a), N(s,a) \leftarrow 0, 0$
11: **end if**
12: **end for**
13: $T = T \cup \{s\}$
14: **return** ROLLOUT($s, d, \pi_0$)
15: **end if**
16: $a \leftarrow argmax_a Q(s,a) + c\sqrt{\frac{logN(s)}{N(s,a)}}$
17: $(s', r) \sim G(s, a)$
18: $q \leftarrow r + \lambda$ SIMULATE($s, d-1, \pi_0$)
19: $N(s,a) \leftarrow N(s,a) + 1$
20: $Q(s,a) \leftarrow Q(s,a) + \frac{q - Q(s,a)}{N(s,a)}$
21: **return** $q$
22: **end function**

**DQN:** Q function is approximated by a Neural Network $\hat{Q}(s, a; w)$

$$\text{Loss}_w = \left(\hat{Q}(s,a;w)_{\text{pred}} - \left(r + \gamma\, max_{a'}\hat{Q}(s', a'; w_-)\right)_{\text{target}}\right)^2 \text{ with}$$

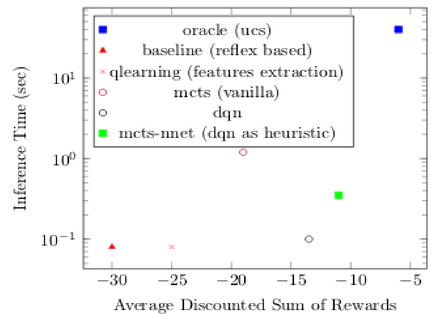
- $w \leftarrow w - \eta \nabla_w \text{Loss}_w(s,a)$ via Batch Gradient Descent (batch size 64)
- Use Experience Replay buffer to avoid training with correlated samples
- Use Fixed Q-target network $\hat{Q}(s', a'; w_-)$ to stabilize training
- Every 1000 iterations update Q-target network weights $w_- \leftarrow w$




## Analysis

- Tree Search without sampling is not applicable as the state space is so huge (Value Iteration is not applicable either)
- Learning based methods provide very good inference times
- Q-learning with linear function approximation and features extraction do not provide good results, even when using expressive models with high order polynomial approximations of features (universal function approximation with Taylor Series)
- DQN training is difficult in our context. We had to use normalized features, experience replay, fixed target network to be able to overfit over a reduced training set in less than 1 hour
- To enable better generalization we have reduced the state space representation to a pre-processed, higher valued information feature set (providing direct information of object vehicle with smallest Time-To-Collision)
- MCTS enables hard constraints per state, restricting the action set depending on the state, and is model based: more explainable
- **Combining MCTS planning and DQN learning is promising as illustrated by our initial results**

### Comparison of Techniques




## Conclusions

- Implementations: CS221 algorithms and custom code for mcts/dqn
- By default tree search based methods are too slow (even MCTS)
- Q-learning and DQN have very good inference time
- DQN outperforms Q-learning
- MCTS enhanced with a DQN neural network is most promising
  - Much faster than vanilla MCTS
  - Better performance (higher score, less collisions)
  - DQN net is used as a powerful heuristic to guide tree search

## Future Work

- Consolidate analysis and results for project report
- So far tests are done on a limited training, dev and test sets to keep training time around 1 hour
- Larger scale experiments are required with much longer training
- Additional MCTS and DQN hyperparameters tuning must be done
- Increase uncertainty in the transition dynamics (using mean so far)