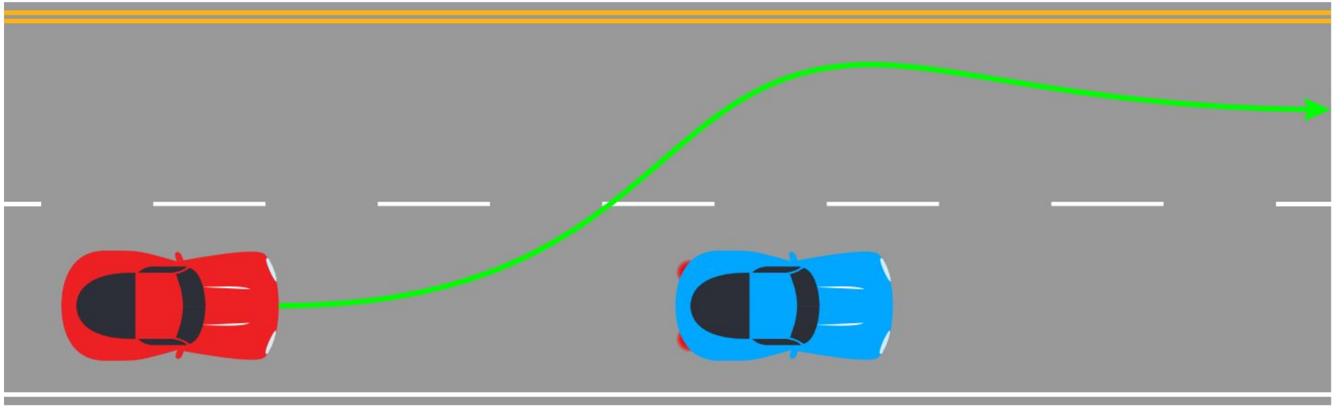
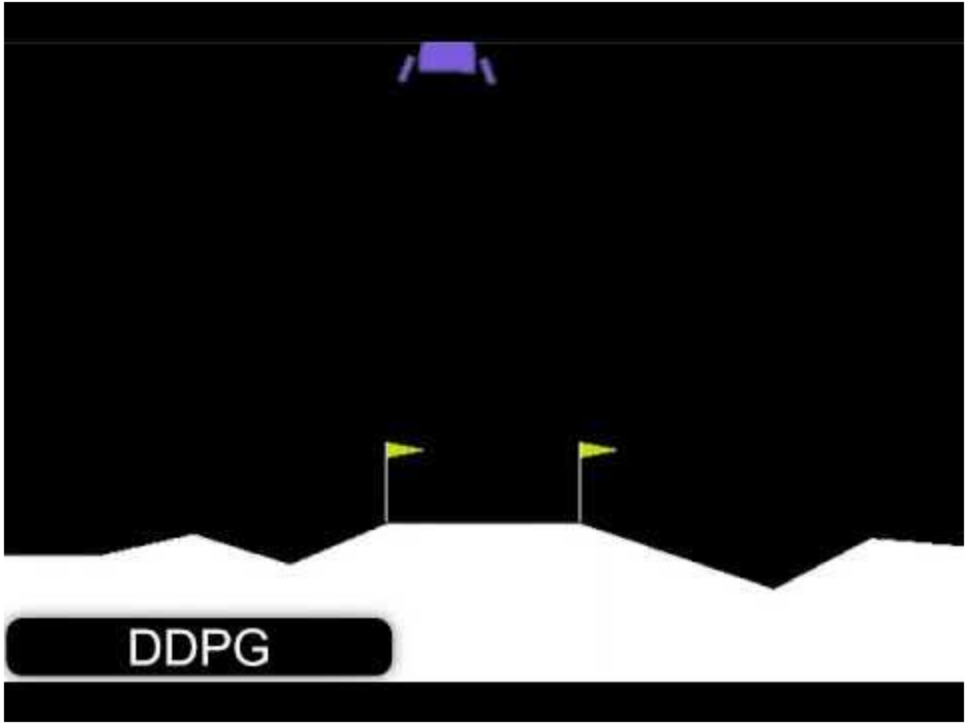
RL in Collision Imminent Environment



Motivation

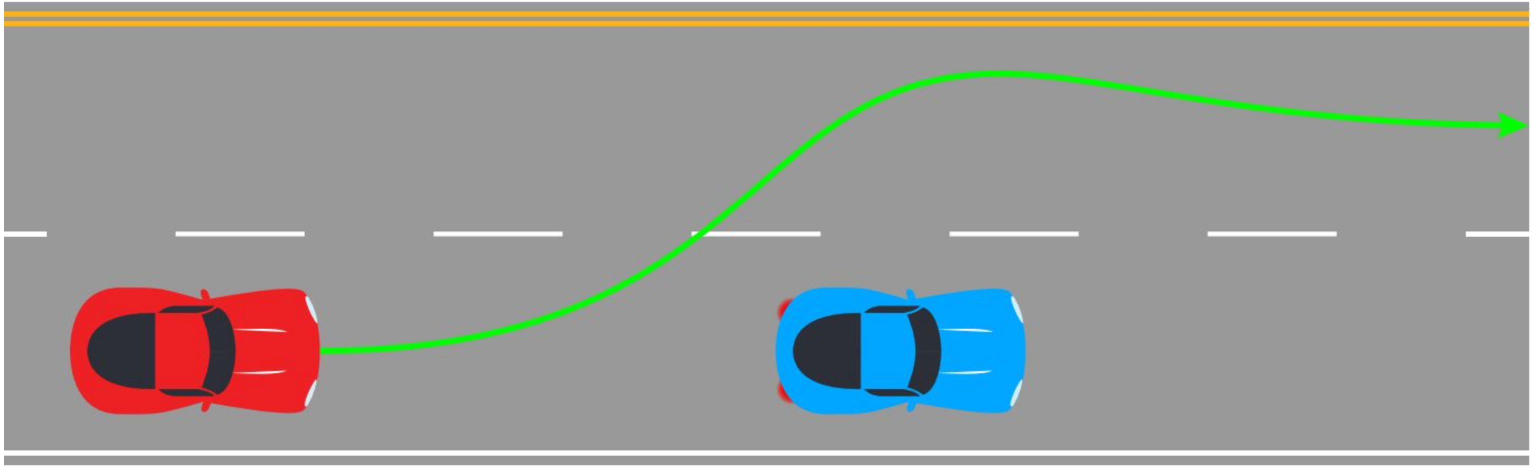
- How can reinforcement learning improve motion planning for autonomous driving?
- What challenges arise for real world applications of RL?



https://github.com/jdlowman2/rl4robotics

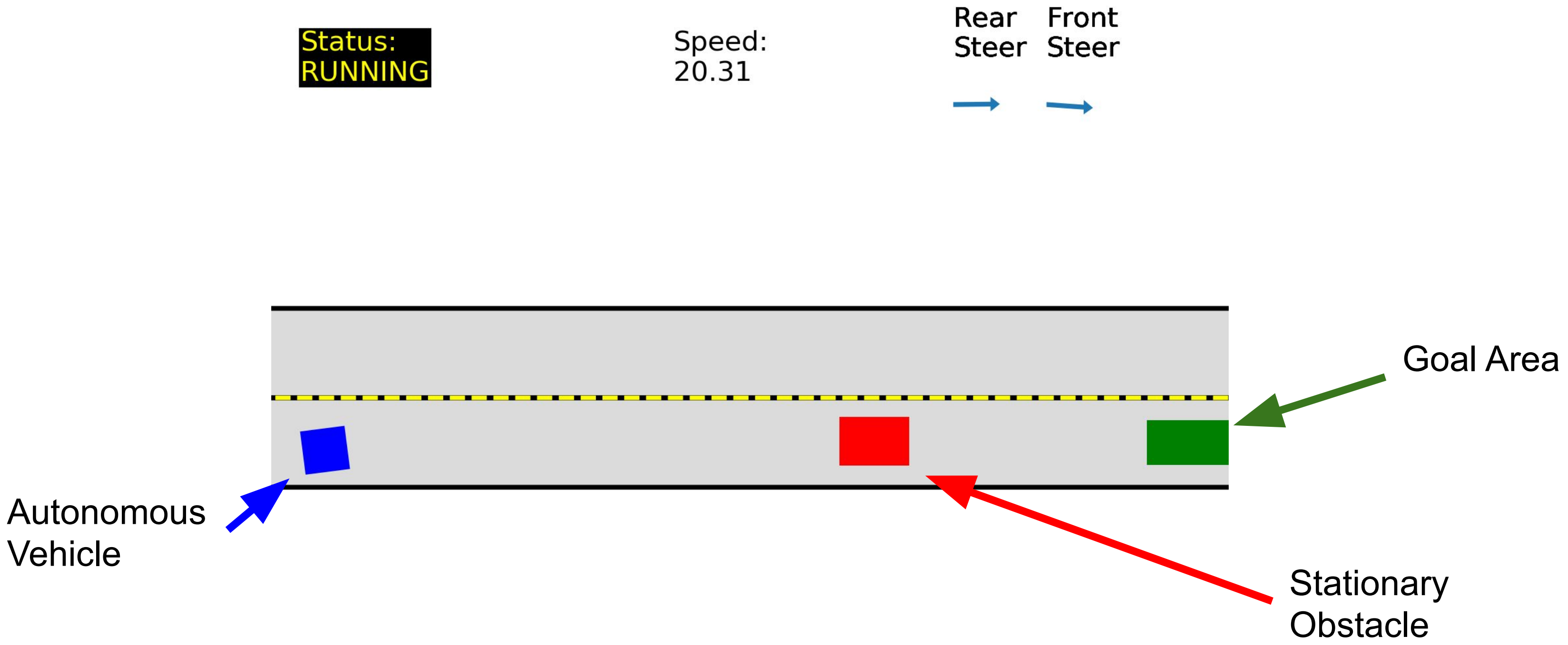
Collision Imminent Environment

- Autonomous vehicle (red) traveling too fast to stop
- Must take evasive steering action to avoid obstacle (blue)



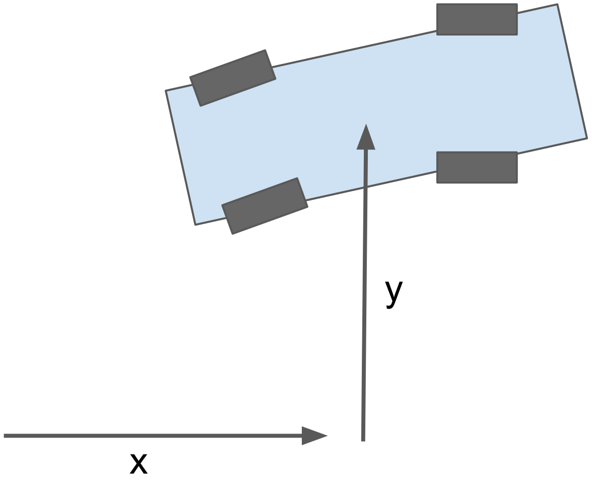
http://www-personal.umich.edu/~tersal/papers/paper73.pdf

Collision Imminent Environment



Environment: Vehicle Model

- All-wheel steering bicycle model



$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \psi \\ u \\ v \\ \omega \\ \delta_f \\ \delta_r \end{bmatrix} \begin{matrix} \text{Longitudinal position} \\ \text{Lateral position} \\ \text{Heading} \\ \text{Longitudinal velocity} \\ \text{Lateral velocity} \\ \text{Yaw rate} \\ \text{Front wheel angle} \\ \text{Rear wheel angle} \end{matrix}$$

$$u = \begin{bmatrix} \dot{\delta}_f \\ \dot{\delta}_r \end{bmatrix} \begin{matrix} \text{Front wheel steering rate} \\ \text{Rear wheel steering rate} \end{matrix}$$

Environment: Vehicle Model

- All-wheel steering bicycle model
- Pacejka magic tire model for tire slip

$$x_{t+1} = x_t + \Delta t \frac{dx}{dt}|_{x=x_t, u=u_t}$$

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \psi \\ u \\ v \\ \omega \\ \delta_f \\ \delta_r \end{bmatrix}$$

$$u = \begin{bmatrix} \dot{\delta}_f \\ \dot{\delta}_r \end{bmatrix}$$

$$\frac{\mathbf{dx}}{dt} = \begin{bmatrix} u\cos(\psi) - v\sin(\psi) \\ u\sin(\psi) + v\cos(\psi) \\ \omega \\ 0 \\ -u\omega + \frac{F_{y,f}\cos(\delta_f) + F_{y,r}\cos(\delta_r)}{m} \\ \frac{F_{y,f}\cos(\delta_f)l_f + F_{y,r}\cos(\delta_r)l_r}{I_{zz}} \\ \dot{\delta}_f \\ \dot{\delta}_r \end{bmatrix}$$

$$F_{y,-} = \mu F_{z,-}\sigma_{y,-}$$

$$\sigma_{y,-} = -\sin(C\arctan(B\frac{V_{y,-}}{V_{x,-}}))$$

$$V_{x,-} = u\cos(\delta_-) + (v + \omega l_-)\sin(\delta_-)$$

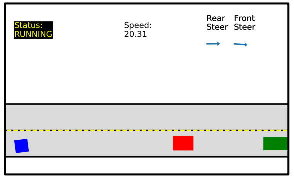
$$V_{y,-} = -u\sin(\delta_-) + (v + \omega l_-)\cos(\delta_-)$$

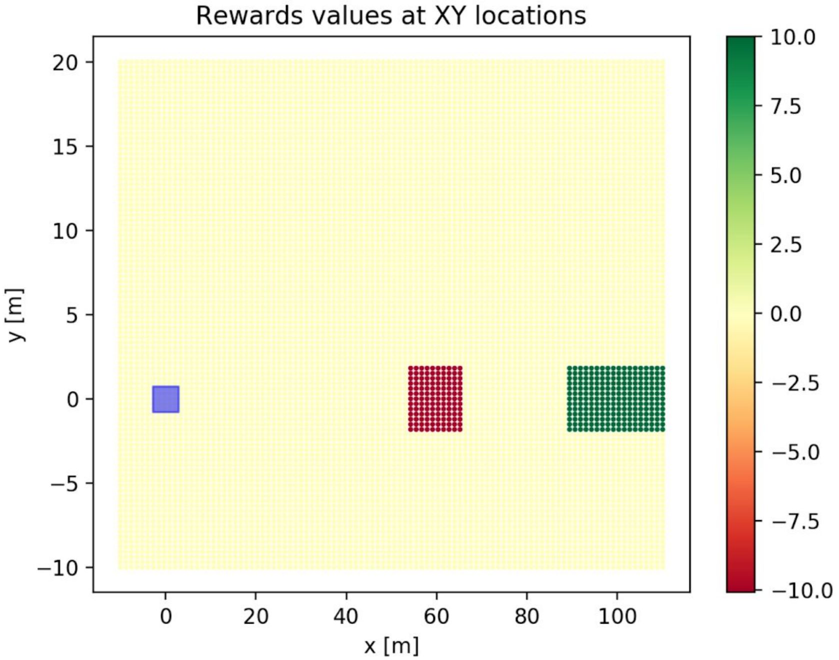
Environment Observation

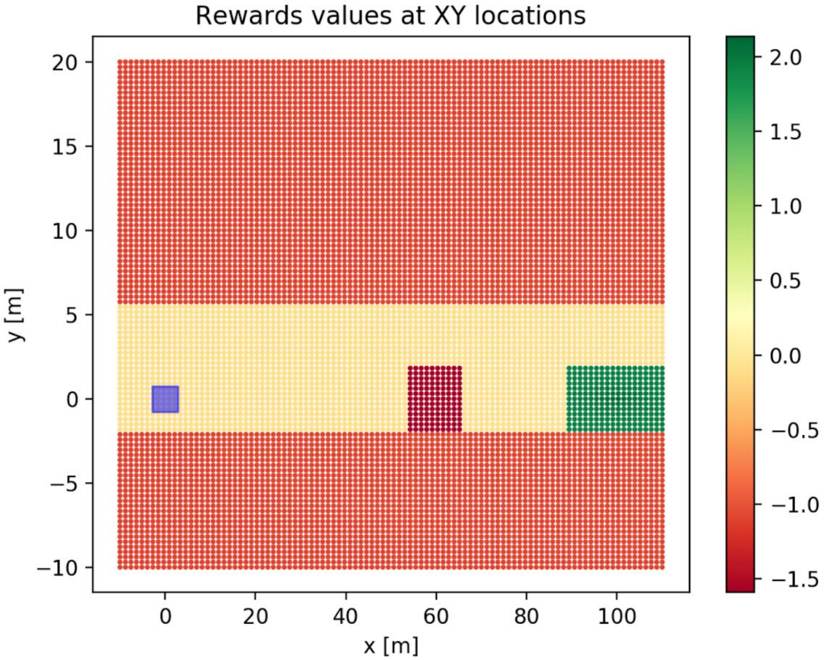
- Observe 8D vehicle state vector and 2D position of the obstacle

$$\mathbf{z} = \begin{bmatrix} x \\ y \\ \psi \\ u \\ v \\ \omega \\ \delta_f \\ \delta_r \\ o_x \\ o_y \end{bmatrix} \begin{matrix} \text{Longitudinal position} \\ \text{Lateral position} \\ \text{Heading} \\ \text{Longitudinal velocity} \\ \text{Lateral velocity} \\ \text{Yaw rate} \\ \text{Front wheel angle} \\ \text{Rear wheel angle} \\ \text{Obstacle longitudinal position} \\ \text{Obstacle lateral position} \end{matrix}$$

Environment: Reward Function

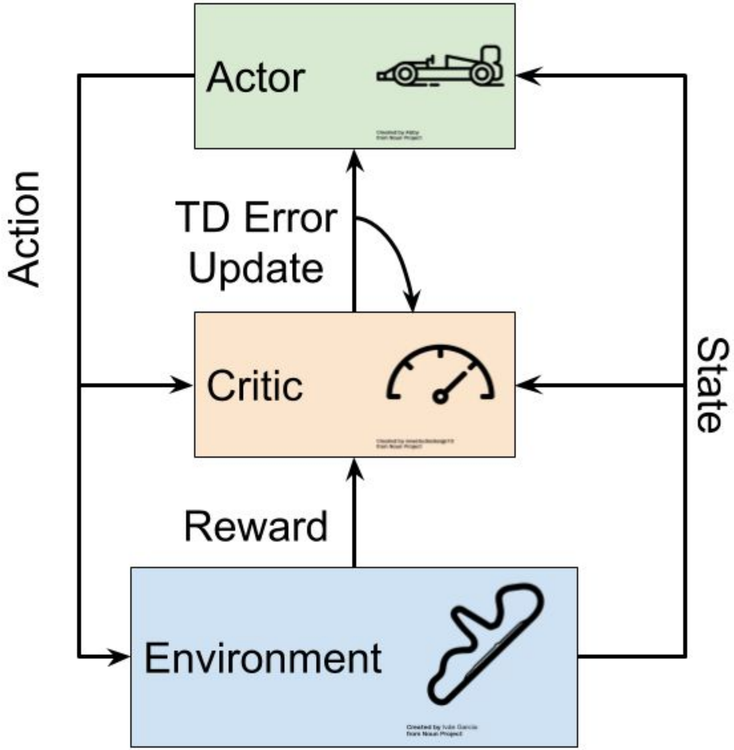






Model-Free Reinforcement Learning

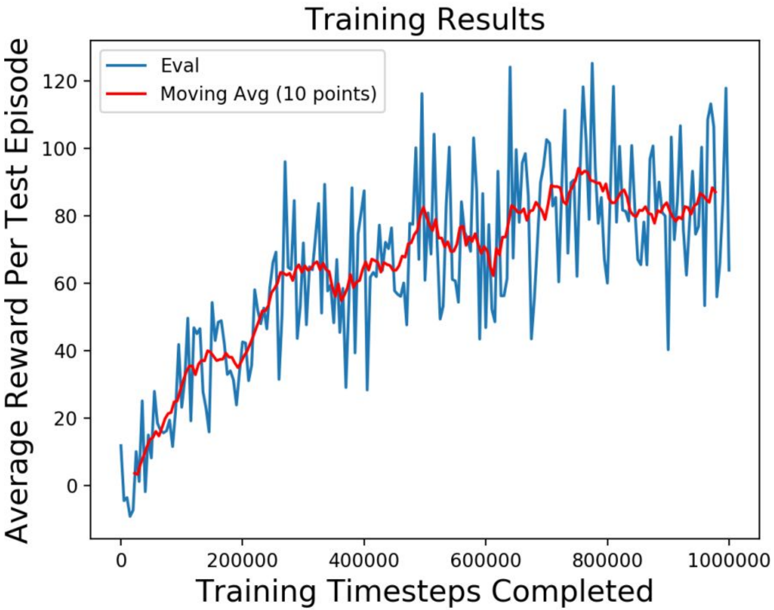
- Actor-Critic Method: Simultaneously train two function approximators:
  - Actor: Approximates optimal policy given a state
  - Critic: Approximates future reward given a state and action



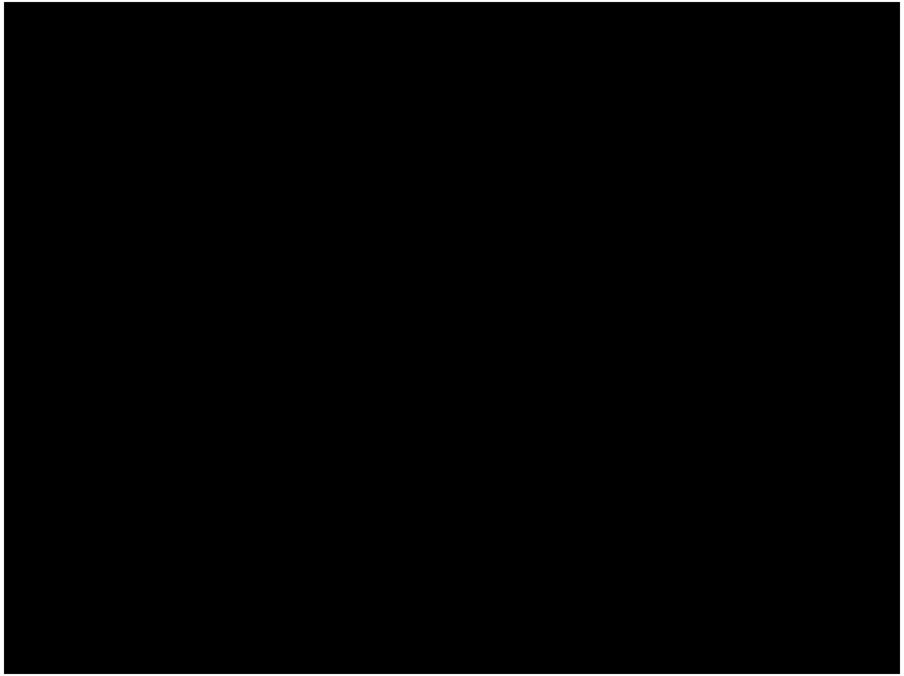
Deep Deterministic Policy Gradients

- Use policy + noise actions at training time to encourage exploration
- Collect sequences of (state, action, next_state, reward)
- Store sequences in “memory” / “replay buffer”
- At regular intervals, sample from replay buffer and perform gradient ascent/descent updates on actor and critic networks respectively

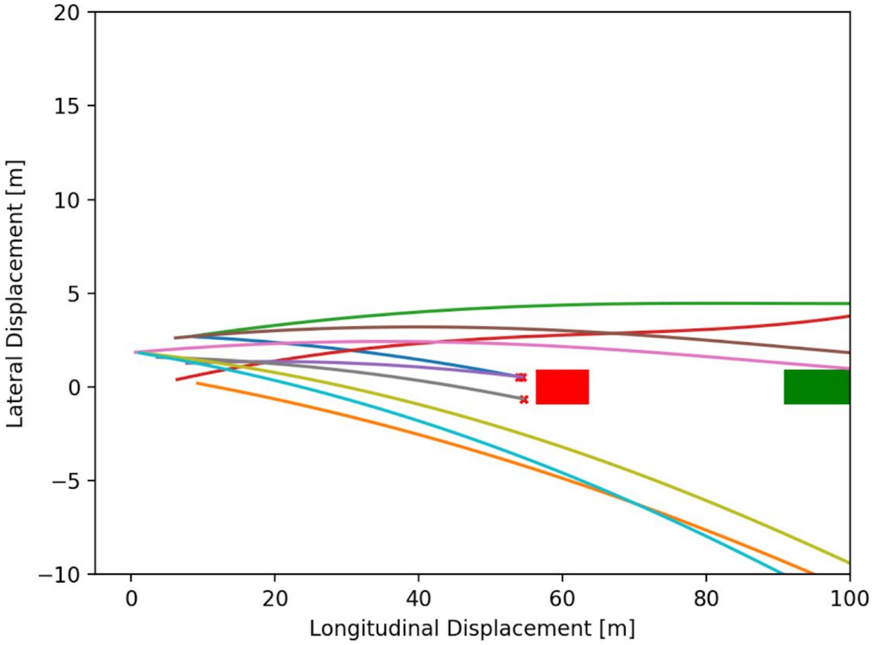
Training Results



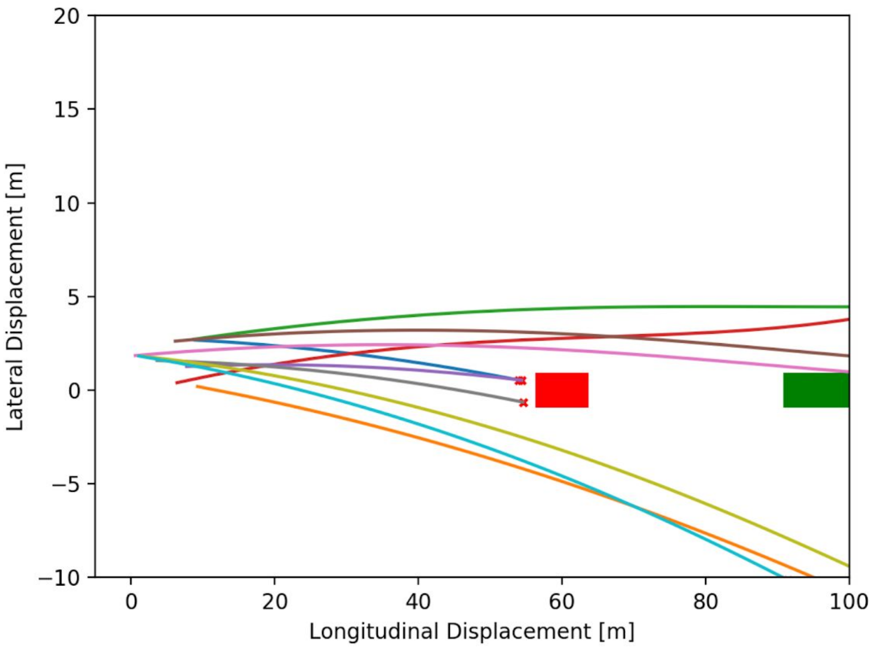
10 test episodes evaluated after every 5,000 training steps



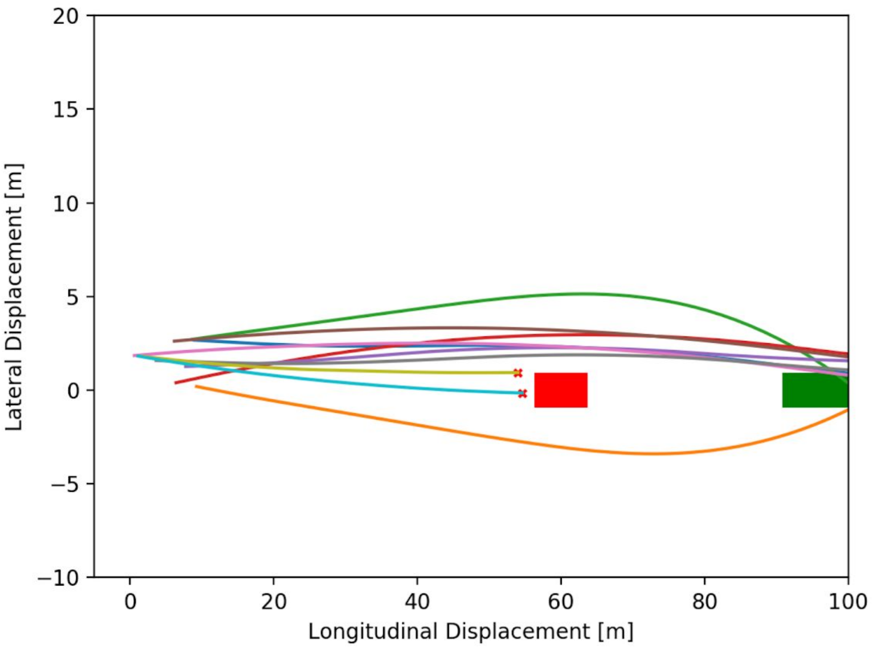
Before Training



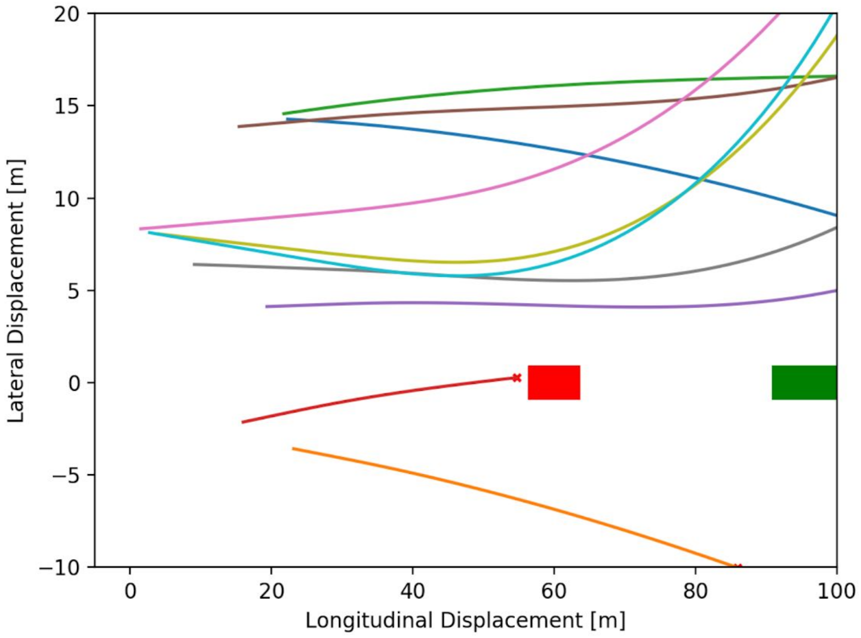
Before Training



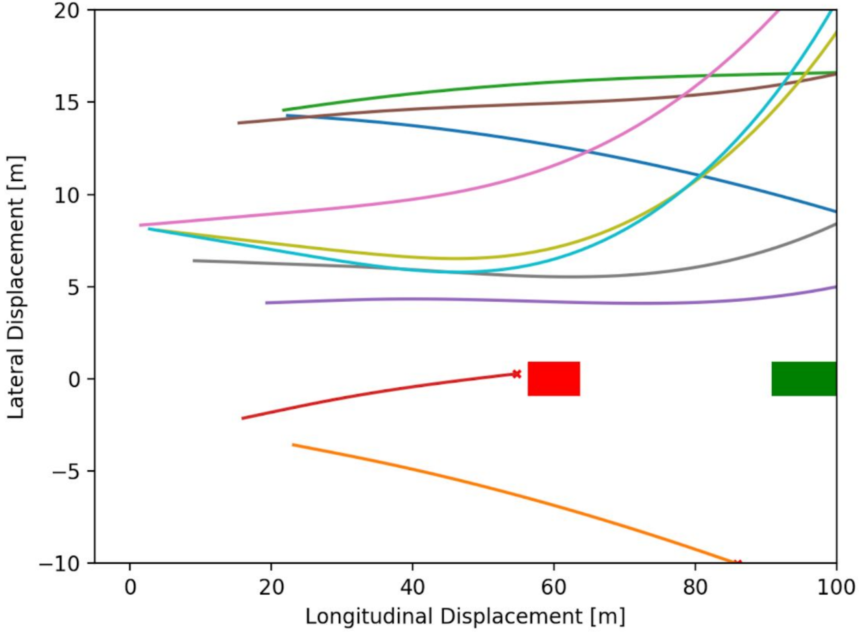
After Training



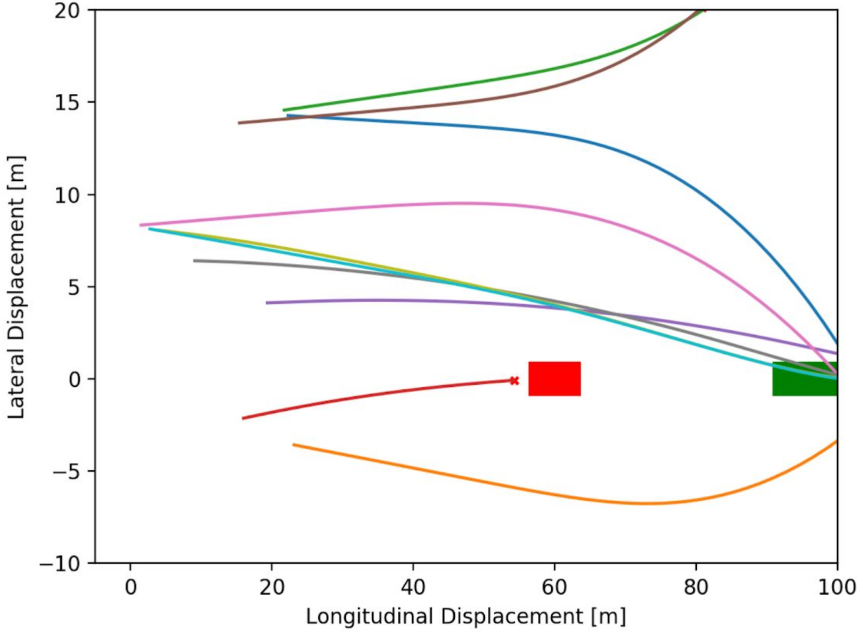
Before Training



Before Training

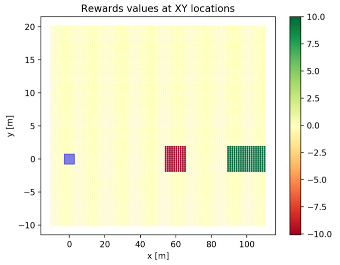


After Training

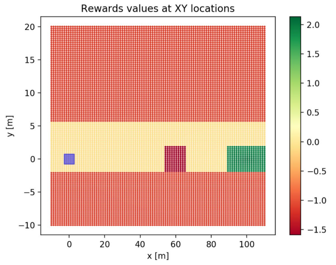


Choice of Reward Function

Sparse



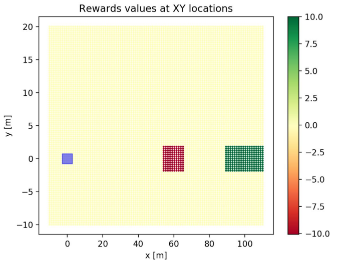
NonSparse



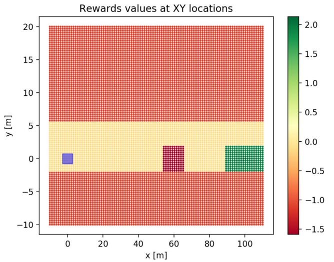
Algorithm	Reward	Steering Rate Limits (front, rear)	Reached goal 500 episodes	Avg Reward (stdev) 500 episodes
Untrained	Sparse	±70, ±30	2.0%	-8.19 (15.21)
Untrained	NonSparse	±70, ±30	11.0%	-33.79 (30.47)

Choice of Reward Function

Sparse



NonSparse



Algorithm	Reward	Steering Rate Limits (front, rear)	Reached goal 500 episodes	Avg Reward (stdev) 500 episodes
Untrained	Sparse	±70, ±30	2.0%	-8.19 (15.21)
Untrained	NonSparse	±70, ±30	11.0%	-33.79 (30.47)
TD3	Sparse	±700, ±300	78.8%	109.21 (71.26)
TD3	NonSparse	±700, ±300	50.4%	-5.81 (18.95)
TD3	Sparse	±70, ±30	85.6%	88.44 (61.02)
TD3	NonSparse	±70, ±30	55.0%	-0.67 (22.81)

Lessons Learned

- Semester project showed encouraging results
- Sensitivity to simulation design choices
    - Reward function
    - Variance of initialization states on robustness of learned policies
- Model-free offers generalized algorithm at the cost of training time and undesirable behaviors

Future Questions

- Performance with randomized position/size/number of obstacles
- Transferability of learned policies?
- Robustness to differences between training and test?
- Reward shaping to eliminate tire slip / encourage good driving behavior?
- Integration with other motion planning algorithms for safe driving?

Github Repositories

Collision Imminent Environment: https://github.com/jdlowman2/collision_imminent_env

TD3 algorithm (forked implementation): https://github.com/jdlowman2/TD3

Writeup: https://github.com/jdlowman2/TD3/blob/master/ROB_590_Project_Final_Report%20(9).pdf

Implementation of DDPG: https://github.com/jdlowman2/rl4robotics

References

[1] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[2] J. Wurts, J. L. Stein, and T. Ersal, "Collision imminent steering using nonlinear model predictive control," in *2018 Annual American Control Conference (ACC)*. IEEE, 2018, pp. 4772–4777. [Online]. Available: http://www-personal.umich.edu/~tersal/papers/paper73.pdf

[3] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," 2018.

DDPG [1]

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

TD3 [3]

Uses the minimum of two Q networks to avoid overestimation

Algorithm 1 TD3

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ

Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$

Initialize replay buffer $\mathcal{B}$

for $t = 1$ **to** T **do**

  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'

  Store transition tuple (s, a, r, s') in $\mathcal{B}$

  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$

  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$

  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$

  Update critics $\theta_i \leftarrow \underline{\text{argmin}}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$

  if t mod d **then**

    Update ϕ by the deterministic policy gradient:

    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$

    Update target networks:

    $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$

    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$

  end if

end for

Vehicle Parameters

Parameter	**Value**
Width	1.8 m
Length	4.8 m
Mass	2041 kg
I_{zz}	4964
L_f	1.56
L_r	1.64
$F_{z,f}$	51.40%
$F_{z,r}$	48.60%
μ	0.8
B	13
C	1.285
Δt	0.05

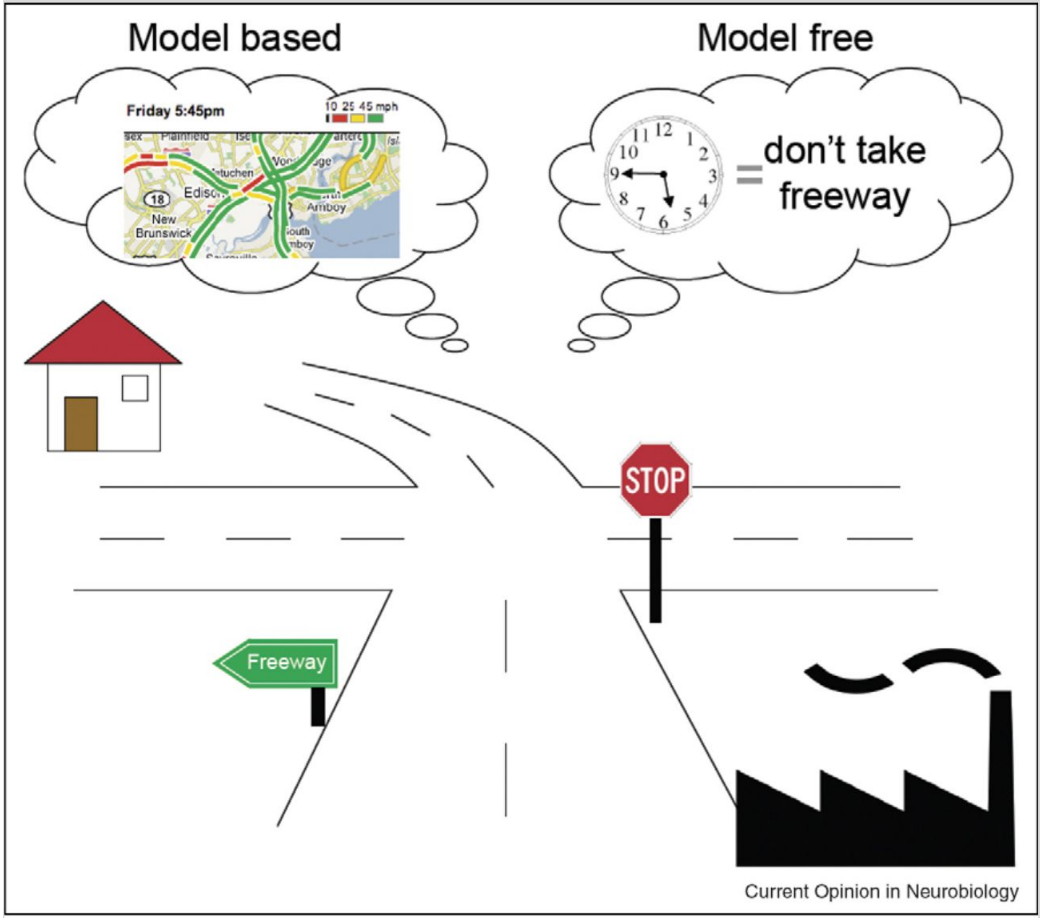


Figure 1: Two ways to choose which route to take when traveling home from work on friday evening.

https://www.sciencedirect.com/science/article/pii/S0959438808000767?casa_token=j1N5BJ7sx2kAAAAA:6TiQ4NIlkIOsYvQY1fGLnF6A44HFeRJOvMG9I8-fq5jSuiCtrRpuLfGoy0RbXr0PviB0-ZCz8A