# Reinforcement Learning for Autonomous Steering

SRIDHAR THIAGARAJAN, SANJEEV SHARMA

**Abstract**

Reinforcement Learning is a field of Machine Learning which deals with the learning of policies via interaction with an environment. RL agents learn from their own experience by taking actions, using the reward function they obtain as feedback. They may be able to handle non-differentiable cost functions which can create challenges for supervised learning problems, and take actions keeping in mind future state possibilities. In this work, a real autonomous vehicle is equipped with a single camera. The control of the steering of the vehicle is formulated as a reinforcement learning problem, and since the aim of the work is to address the control aspect of the vehicle rather than perception, the goal of the car was to follow a single trajectory on the road. Integrating the control framework with more complex perception modules (eg. road detection), could allow for a wider range of use cases. [1]

CONTENTS



## 1 INTRODUCTION

The field of autonomous driving is one which has seen a lot of growth in the past few years, with the contribution of both novel algorithms as well as computational power helping its cause. Autonomous driving requires a level of intelligence that surpasses that achieved so far by AI agents. Recently, end-to-end training of self driving vehicles has been tried [6] with reasonable success, albeit they train the vehicle in a supervised learning fashion. We propose a reinforcement learning framework for this task, due to the capability of an RL system to learn control policies which may surpass that of its driver, as its sole motive is to maximize its expected cumulative reward. This preliminary work focuses on getting the vehicle to follow a predefined trajectory. We discuss formulation of the state space, action space, and the reward function. A suitable algorithm is tested and tried based on this formulation.

[1] Our code for testing on OpenAI gym environments is available in this github repo

### 1.1 *Reinforcement Learning*

The Reinforcement Learning (RL) framework has been used in control tasks for a long time. Recently, the onset of deep networks has allowed RL to scale to tasks larger than ever before. Humanlevel control was demonstrated on Atari games using the Deep Q Networks (DQN) algorithm [2], which is the combination of the earlier proposed Q-learning algorithm [] with deep networks. The advantage of reinforcement learning over supervised learning for autonomous driving, is that RL has the capability to find behaviors which might be better than the human itself, provided this behavior provides its an even greater reward. On the other hand, the supervised learning framework treats the behavior of the drivers as optimal.

### 1.2 *Markov Decision Processes*

A Markov Decision Process (MDP) framework, is a tuple of $(S, A, P_{sa}, \gamma, R)$ pairs, where: $S$ is the set of environment states, $A$ is the Set of Actions, $\gamma$ is the discount Factor, $R$ the reward, while $P_{sa}$ are the State Transition Probabilities The Reinforcement learning agents goal is to maximize the expectation of discounted cumulative sum of rewards from a given state, known as the value function.

$$V_\pi(s) = E[R(S_0) + \gamma * R(S_1) + \gamma^2 * R(S_) + ...|s_0 = s, \pi(s)]$$

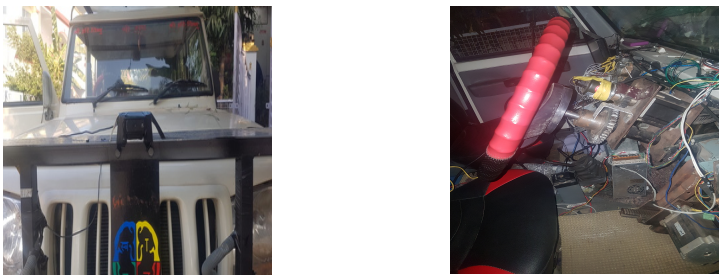
$$\pi^*(s) = argmax_\pi V_\pi(s)$$

Hence, the goal is to find a policy $\pi^*(s)$. Several algorithms exist to solve this problem for finite state-spaces like Value iteration and Policy iteration

## 2 DATA COLLECTION

Due to the absence of a simulator which can accurately simulate the dynamics of the vehicle, we had to make do with off-policy reinforcement learning i.e train the RL agent with offline data collected, An off-policy agent learns the value of a policy which need not necessarily be followed by the agent. A single USB camera was affixed to the front of the vehicle, and its video stream was collected at 30 FPS. The Steering of the vehicle had an encoder attached to it, to read the absolute steering angle at any particular point of time. A Wheel Encoder was also affixed to the rear wheel to give a rough estimate of the velocity of the vehicle, seeing as its dynamics would depend on it. This lead to data collect of the form

$$[Image, SteeringEncoderValue, WheelEncoder]$$



(a) (b)

Figure 1: 1a) USB camera affixed to front of vehicle 1b) Steering+Encoder Mechanism

## 3 STATE REPRESENTATION

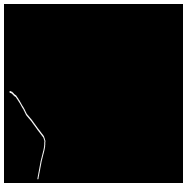
In an MDP, the choice of state representation is often a crucial decision. The following choices can be made 1) Raw image 2) Low level State representation - In this work, we choose to go with the latter, as the RL algorithm will invariably end up needing lesser data to obtain better cumulative reward, due to the more informative and lower dimensional representation. The following pipeline is followed do achieve the state vector

- Identify the trajectory from the image : A Convolutional neural network [5] with cross-entropy loss was trained to reliably identify the trajectory in the image. Sample outputs shown in Fig 2.
- Once the trajectory is extracted, a N dimensional state vector is created, where N is the number of the rows of the image (640). Each entry of the vector corresponds to the median location of the trajectory along that row, normalized by the width of the image.
- Concatenate $(S_{t-6}, S_{t-4}, S_{t-2}, S_t)$ : State should ideally be markov, i.e the future states should be conditionally independent of the previous states, given the current state and action. Another possibility is to use LSTM networks [11], as it can keep the relevant part of the previous states to enable good control policy.

(a) (b) (c)

Figure 2: Output of 2 layer CNN with 32 filters each on samples images after training. Morphological Operations were applied on this to get a clearly segmented trajectory

(a)



(b)

$$I_t = \begin{bmatrix} -1 \\ -1 \\ \vdots \\ 0.2622 \\ \vdots \\ 0.2247 \end{bmatrix}$$

(c)

$$S_t = \begin{bmatrix} I_{t-6} \\ I_{t-4} \\ I_{t-2} \\ I_t \\ V \end{bmatrix}$$

(d)

Figure 3: 1a) Camera input at time t. 1b ) Output of Trajectory detection 1c) Representation of image at time t. Rows with no trajectory perceived set to -1. 1d) The concatenate state vector of $S_t$, where $V$ is the velocity calculated with wheel encoder values

## 4 ACTION SPACE

The action that the RL algorithm takes was decided to be how much more to move the steering at a particular timestep, rather than the absolute value of the steering at any point of time. This seemed to be closer to how humans drive, choose how much more to turn the steering, rather than to decide on its absolute position at any timestep. In order to apply RL algorithms which work work with discrete actions like DQN, the steering encoder values need to be discretized. This needs to be done taking into account

- Each discrete action is sampled sufficient number of times in order for the algorithm to learn its Q-value to reasonable degree of approximation.
- There is not too much distortion error, i.e most continuous actions are discretized to an action reasonably close in value.

Actions 0,-20,-50,20,50 was chosen, and each continuous action assigned to its corresponding cluster centre. The histogram of actions in the data collected was inspected to ensure each action was sufficiently sampled.

## 5 REWARD FUNCTION

The reward function is the only form of feedback in a reinforcement learning system. It needs to be chosen to be as informative as possible, making sure the intent of the task is conveyed. RL systems only optimize the expected cumulative reward, so there might be cases where the expected reward in the long run is maximized, but the task that we wanted to perform is not being done satisfactorily. This means the reward function was wrongly chosen.

In the case of steering, the reward needs to be chosen taking into account the following:

- Higher reward for more closely following the trajectory
- At most timesteps, the steering is kept constant i.e avoid unnecessary non zero-actions.

For the first part, instantaneous tangent to the segmented trajectory was computed at the lowest point in the image; this then, was extended to intersect the lower part of the image. Seeing as the camera is at the centre of the vehicle, the negative of the distance of the intersection point with the centre axis of the image is taken as the reward. When the trajectory is perfectly followed, for example on a straight line, this distance is zero, and the reward attains a maximum value of zero. In all other cases, the reward is negative. For the latter objective, a small negative reward was assigned for taking a non zero action - if steering is moved at a particular timestep, the agent receives an additional negative reward. This is done to encourage the RL agent to learn smooth policies, rather than jerking the steering too much. The final reward function is

$$R = -(\kappa * |i - w/2| + \beta * \mathbb{1}(a \neq 0))$$

, where $\kappa$ and $\beta$ are small positive constants, $i$ is the intercept, and $w$ is the width of the image

## 6 TIME STEP

In discrete time MDP's, the choice of timestep is also one which must chosen carefully. If the timestep is chosen to be large or too small, the effect of the action on the state of the MDP cannot be observed correctly. Too small a time-step, and there will be negligible change in the state. If the time-step

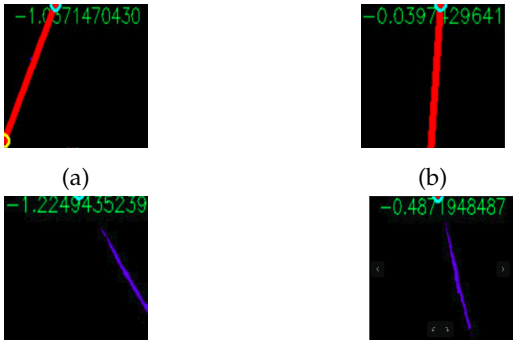



Figure 4: Sample Reward Calculations. As the intercept is farther off from the centre of the window, more negative the reward. Action's influence on reward is ignored for the purposes of this illustration

chosen is too large, the effect of the action on the MDP cannot be assesed correctly. Taking these into consideration, we choose a timestep of

$$\Delta T = 250ms$$

Hence, the camera, wheel encoder and steering values are sampled at $4Hz$

## 7 DEEP-Q NETWORKS

Q-learning [12] is the most fundamental algorithms to solve MDPs. The actions a $\in$ A are obtained for every state s $\in$ S based on an action-value function called

$$Q : S \times A \mapsto \mathbb{R}$$

The Q-learning algorithm is based on the Bellman equation, which in essence, is a dynamic programming update.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma * max'_a(Q(s'_{t+1}, a') - Q(s_t, a_t)]$$

When the states are discrete, the Q-function is a lookup table. For continuous states, for generalization, the Q-function is formulated as a parameterized function of the states, actions; $Q(s, a, w)$. The solution then lies in finding the best setting of the parameter w . Using this formulation, it is possible to approximate the Q-function using a Deep Neural Network (DNN). The objective of this DNN shall be to minimize the Mean Square Error (MSE) of the Q-values as follows:

$$l(w) = E_{s,a\sim\rho(.)}[(r + \gamma * max'_a * Q(s', a', w) - Q(s, a, w))2]$$

The the optimization problem can be easily solved using Gradient-based methods (Stochastic Gradient Descent (SGD), Conjugate Gradients (CG),etc). The algorithm is called Deep Q-Networks (DQN) [2]

## 8 REFERENCES

1. RS Sutton, AG Barto: Reinforcement Learning : An Introduction (MIT Press, 1998)
2. V Mnih et al, Human Level Control Through Deep Reinforcement Learning, Nature, 518 (7540):529533, 2015
3. Hado V : Deep Reinforcement Learning through Double-Q-Learning arxiv preprint: 1509.06461

4. Tesauro, G. Temporal difference learning and TD-Gammon. Commun. ACM 38, 5868 (1995)

5. Krizhevsky, A., Sutskever. I, Hinton, G. ImageNet classification with deep convolutional neural networks.Adv.Neural Inf.Process.Syst.25, 11061114 (2012)

6. M Bojarski et al ,End-to-End Learning for Self-Driving Cars, arxiv preprint:1604.07316

7. Moore, A. , Atkeson, C.Prioritized sweeping: Reinforcement Learning with less data and less real time. Mach. Learn. 13, 103130 (1993).

8. L. Baird. Residual algorithms: Reinforcement learning with function approximation. In Machine Learning: Proceedings of the Twelfth International Conference, pages 3037, 1995.

9. L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. Journal of Artificial Intelligence Research, 4:237285, 1996.

10. J. N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. IEEE Transactions on Automatic Control, 42(5):674690, 1997.

11. Hochreiter, S., Schmidhuber, J. (1997). Long short-term memory. Neural computation , 9 (8), 1735-1780.

12. Watkins, C. J., Dayan, P. (1992). Q-learning. Machine learning , 8 (3-4), 279-292.