Reinforcement Learning based Suspension Dampening Control System for Automotive Applications

Mubeen Khan
Stanford University
mkhan3@stanford.edu
December 7, 2018

1 Introduction

This study explores the application of Reinforcement Learning (RL) methods to a suspension dampening control system used to regulate the articulation of the control arm connecting the wheel assembly to the chassis of an ordinary road going automobile. Different learning methods based on model-based and model-free approaches have been contrived and their application to a simulated model of the intended device-under-test has been studied. The closed-loop policies precipitated from each implemented method have been compared and contrasted based on their measured performance in the simulated environment.

While the application area is not entirely new, there is tremendous commercial interest in devising new more novel solutions that are increasingly economical without compromising reliability or performance at the same time. There is a burgeoning trend in the automotive industry towards introducing software based solutions for complex control systems for reasons ranging from brand marketing all the way to allowing the consumer to engage with the hardware more interactively. The gamut of suspension dampening applications is particularly broad, ranging from body roll prevention in the automotive sport segment to improving comfort and road handling in the luxury segment at the other end of the spectrum. Needless to say software configurable systems offer considerably higher Reliability, Accessibility and Serviceability (RAS) for automotive assemblies that are deployed on a commercial scale.

The problem is framed as Markov Decision Process where the state of the system is fully observable using a set of sensors measuring position, angle, and velocity (linear and angular). A model of the agent or the system-under-test has been constructed based on physical characteristics typical to this apparatus. The changes in state of the device are directly observed while it interacts with the control system in the simulated environment. The optimal policy for devising *action* based on *state* derived from the RL algorithms is applied and the effects of these policies are measured and compared. The objective is to use RL algorithms to make the agent learn to control suspension articulation in an unknown environment by interaction in a simulation framework.

2 Devising the Model

The development and testing of the RL algorithms for this application pivots on the availability of a platform to emulate the interaction of the device with the environment. Considerable effort was dedicated to developing a model of reasonable complexity to emulate the system under test with the objective of running episodic simulations against the model for testing the performance of different algorithms in various situations. All simulations were conducted using Octave 4.4.0.

For this exercise a Quarter Car model has been considered for emulation simplicity as opposed to a half or full car model. As evident by the name, only ¼ of the vehicle is modelled to develop a suspension low level controller. The quarter car representation basically consists of a single wheel which is represented in the form of a spring. However in some cases the wheel can be considered as equivalent to a parallel combination of spring and a damper. The actual shock absorber is assumed to support only one fourth of the weight of the total car body mass.

This model is two dimensional because movement along the z direction (translational motion of the vehicle along the vertical-axis is called "HEAVE" or "BOUNCE") only is taken into consideration. It basically consists of a single wheel which is represented in the form of a spring and an active damper. Based on the damper used, generally speaking vehicle suspension systems can be classified into three types: (1) Passive (2) Semi Active, and (3) Active suspension systems. A graphical rendering of the device from actual simulation is shown in Figure 1.

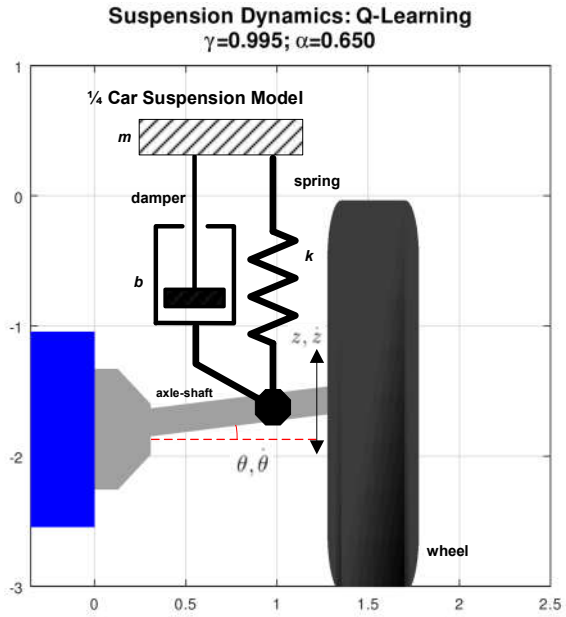

Figure 1 1/4 Car Suspension Model

This study focuses on devising a RL technique for an Active Suspension control system (the third type) using the algorithm to adjust the damper coefficient *b*, as shown in Figure 1, to control the articulation of the axle-shaft. Fully active suspension systems use electronic monitoring of vehicle conditions, coupled with the means to impact vehicle suspension and behavior in real time to directly control the motion of the car. The electronic monitoring is enabled using sensors to communicate the complete attitude of the mechanical assembly at any given point in time to the control logic.

2.1 Active Suspension System

In active suspension systems, sensors are used to measure the position/acceleration of sprung mass, unsprung mass and various other components, and the analog signals from these sensors are sent to a controller. The controller is designed to take necessary actions to improve the performance abilities already set. The algorithm running on the controller will generate actions that may then be fed to the actuator controlling the damper to generate the required forces to form a closed loop system. Obvious tradeoff is the requirement of external power and the increase in complexity, cost and weight all of which are important considerations in the automotive industry. The majority of the studies in the area of semi-active suspensions use a two-degree-of freedom (2DOF) model for representing single suspensions.

2.2 Mathematical formulation

This particular study captures the behavior of the model using sensor indications for bounce, $\{z, \dot{z}\}$ and deviation of the axle-shaft $\{\theta,\dot{\theta}\}$ from the default 0-degree equilibrium position. The compensation applied by the control system is based on the behavior learned from these four measurements by the RL algorithm at each simulation time step. The current model has been devised on a formulation where the state of the system is described by four real parameters:

(1) Vertical position of the suspension component, z
(2) Rate of change of position, $\dot{z}$
(3) Angle of connection of the suspension component, θ
(4) Rate of change of angle, $\dot{\theta}$

The transfer function for the quarter car suspension takes as input values of each one of the four parameters plus the *action* in each simulation time step and returns updated real numbered values for the new state parameters. There are two actions possible in any given state, since action in this case pertains to movement of the damper either up or down (without considering the rate of movement).

The new state parameters are then (a) individually discretized into ranges (b) a differing offset is added to each range so as to avoid overlap (c) offset'd ranges for the four parameters are then collapsed to get a single combined integer value derived from the individual discretized values. This single integer representation of the state variable is what is then consumed by the simulation. The granularity of the discretization directly affects the quality of the learned policy at the expense of increased computation. The discretization granularity (number of states) was initially kept low (288) for faster development and then increased later (576) to improve the learned policy once the infrastructure had somewhat matured.

3 Finite State MDP

The problem is formalized as a Markov Decision Process, i.e. using set of states and set of actions governed by state transition probabilities using a reward function whose expectation the learning algorithm will attempt to maximize over an infinite horizon. This framework is then used to conduct discrete time simulations (simulation proceeds in discrete steps) starting with an initial state. Uncertainties related to the variability of the driving surface modelled using random variables that may be sampled at each step of the simulation. At every step the controller must devise the force on the control arm to counteract the movement of the wheel. The cumulative effect of these events will take the system to the next state.

For the reward formulation, a penalty is exacted on the system if the suspension goes out of bounds, i.e. extends or contracts maximally. The bounds for extension and contraction are determined by the angle of deviation or the current position of the control arm exceeding a certain limit. Reward will be kept at zero in the normal operating range.

For the model based techniques, the transition function is learned during exploration where transition statistics are monitored and stored as the agent takes an action in its present state to transition to the next.

The agent's performance is evaluated as it goes about acquiring understanding of its environment by interacting with it. The modus operandi related to this process of discovery offers challenges typical to such problems namely, (a) balancing exploration with exploitation (b) decision-reward latency i.e. attributing credit for later rewards to earlier decisions, and (c) generalizing from limited experience to taking suitable action in previously un-encountered states.

3.1 Simulation

The simulation framework is setup to test each implemented control system for a number of iterations or episodes. The selected algorithm endeavors to learn the policy over successive episodes. Each episode entails simulating the suspension dynamic for a discrete number of time steps, until the system experiences failure, i.e. suspension is detected to have gone out of bounds. The out-of-bounds condition is detected from state parameter values $\{z, \theta\}$ in which case the discretization function returns a reserved exit-state value. On observing the next state value set to the terminal state the main control loop declares end of that particular simulation episode. The number of simulation time steps that define the length of each episode from start to failure is used as a measure of how well the policy has performed in that particular time frame. The following is a list of operations that are typically performed in each iteration irrespective of what algorithm is run:

(a) Bellman/Q-Learning residual $|U_k - U_{k-1}| < \delta$ is used as the criteria for convergence
(b) Number of time steps to failure is recorded for each episode
(c) State to action policy mapping is updated depending on the experience gained over the iteration as dictated by each algorithm
(d) Main simulation loop continues until convergence is detected for a consecutive number of episodes, or some maximum terminal iteration count is reached

The hyper-parameters governing the algorithms, mainly the Discount Rate γ, Learning Rate α and exponential or trace decay λ had to be tuned in each case to ensure that the best possible policy could be precipitated from one complete simulation run. This involves evaluating policies from several simulations, run with the same seed, with parameters values varying over a range from one run to the next, concluding with selecting parameter values that yielded the best policy comparatively.

The simulation is equipped to render graphically the interaction of the algorithm running on the control system with the suspension model for each experiment that was run. The runtime performance of the algorithm, and the improvement in control system functioning with learning progression can actually be visualized for each time step of the simulation along with the variation in state parameter values.

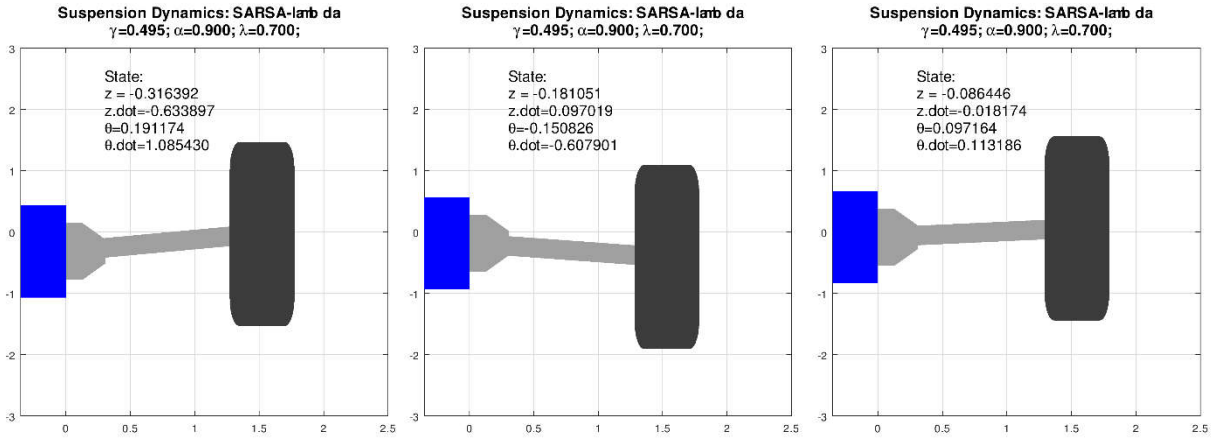

Figure 2 Active Suspension Dynamics – Simulation states

4 Methods

4.1 Gauss-Seidel Value Iteration

This is an application of approximate dynamic programming (DP) using asynchronous update to the state value function to find an optimal policy for a system with continuous state and action space. The state space in this case has been quantized for finite-state DP method consumption. Since the agent under test is a physical mechanical device so the local approximation assumption that states close to each other may have similar values performs reasonably well. This gives an approximation of the value function where the optimal solution quality depends on how finely the state space has been discretized. Simulation results show that local approximation converges invariably and VI yields an adequate closed loop plan based on greedy action selection for reacting to uncertainty in this environment. This is a model based technique and the transition function $T(s' \mid s,a)$ is estimated directly from experience ($N(s,a,s')$ transition counts) over the course of simulation using Maximum Likelihood methods. This algorithm implements the well known Bellman update equation and greedy policy selection shown in (1).

$$U_{t+1}(s) \leftarrow \max_a \left[R(s,a) + \gamma \sum_{s'} T(s'|s,a) U_k(s') \right]$$
$$\pi(s) \leftarrow arg \max_a \left[R(s,a) + \gamma \sum_{s'} T(s'|s,a) U^*(s') \right] \quad (1)$$

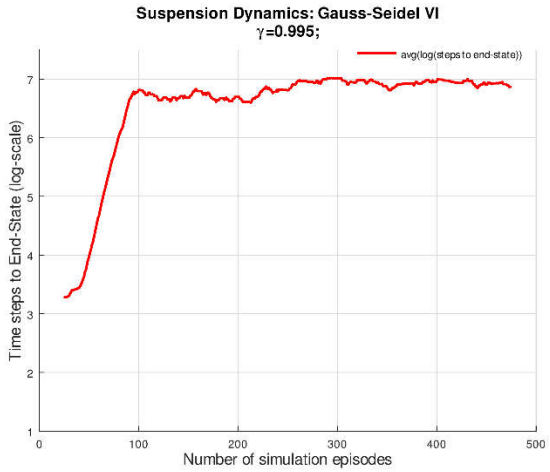

Figure 3 Gauss-Seidel VI closed-loop policy performance

Figure 3 illustrates the performance of the VI learned policy from one such simulation. Learning was observed to complete in 100-200 episodes after which the value function would converge in one iteration over the state-action space. Asynchronous DP involves operations over the entire state space of the MDP at the end of each episode which tends to become computationally prohibitive when using finer quantization of the {state, action} space for the suspension dynamics.

4.2 Q-Learning with Forward Search

Q-Learning applies incremental estimation to the utility function update in the Bellman equation while using the Q(s,a) value function as shown in (2).

$$Q(s_t, a_t) \leftarrow (1-\alpha).\underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} . \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount rate}} . \underbrace{\max_a Q(s_{t+1}, a)}_{\text{est. of optimal future value}} \right)}^{\text{learned value}} \quad (2)$$

This is a model free learning method that can handle stochastic transitions and rewards without requiring adaptations. An epsilon-greedy non-stationary annealing based exploration strategy was instrumented. The action selection was biased towards exploration in the initial training phase rapidly transition to full greedy using the highest Q value as criteria for choosing the action. For states that the Q-learning agent has not seen before it has no clue as to which action to take since it does not have the ability to estimate value for unseen states. The learning rate parameter α determines to what extent newly acquired information overrides the old. When compared to results from Gauss-Seidel VI (Figure 3) it can be seen that the Q-learning agent unsurprisingly converges to the optimal policy much more slowly.

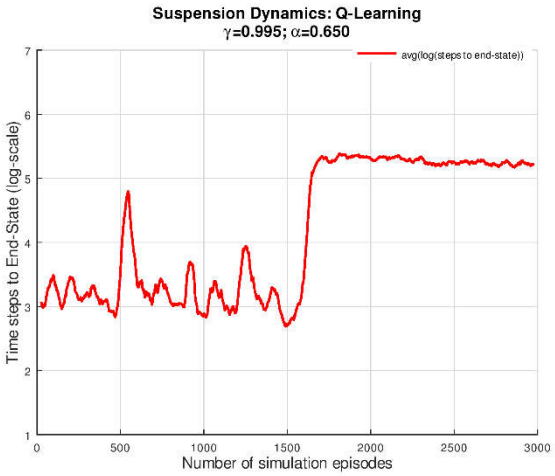

Figure 4 Q-Learning policy performance

The best performing policy from several simulation seeds is then chosen and was tested in a purely exploitative environment ($\epsilon = 0$) as a measure of fitness. For state quantization that limits the number of states to less than 500 with 2 possible actions (lateral z movement, up or down) in each state, a predominantly exploitative scheme was found to perform well.

Forward Search method was incorporated for determining action in higher dimensional environments when the agent encounters a state that it has not visited before with look ahead horizon set to 3 and restricting the next states to a subset denoting the neighborhood of the current state.

The trained agent was evaluated by running the simulation 10 times for ~500 episodes in each run. The results from one such run are shown in Figure 5.

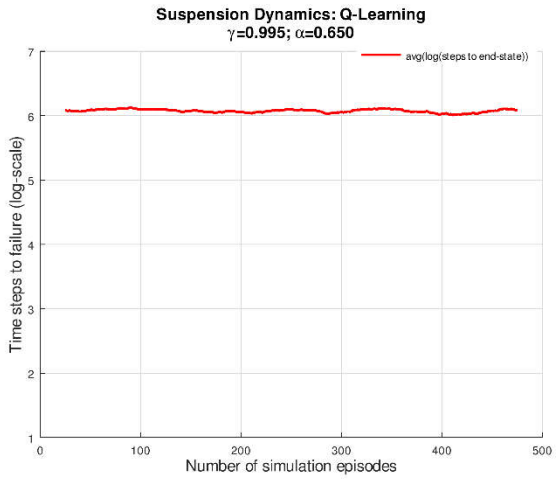

Figure 5 Q-Learning: Exploitative policy with Forward Search

4.3 Sarsa

Sarsa is a derivative from Q-learning that uses the actual action taken to update Q instead of maximizing over all possible actions. With a suitable exploration strategy a_{t+1} will converge to $\arg_a\max Q(s_{t+1},a)$ that is used in the Q-Learning update and should converge to optimality. Sarsa is an online policy algorithm that learns the Q value based on the action performed by the current policy rather than the greedy policy. This also effects an improvement in computation efficiency of the order O(|A|) since it does not need to calculate $\max_a Q(s,a)$ as seen in (2).

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma\, Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \tag{3}$$

One limitation that poses as an impediment to the learning process for the suspension dynamics model, for both Q and SARSA is the sparsity of rewards over the state space. This would be one area where the model would benefit from additional discernment. The mainstream reward model used in these simulations returns zero for all states except the terminal state where it returns -1. One supplement to the reward model to incentivize the algorithm further was to introduce positive reward value for states corresponding to suspension dynamics closer to equilibrium, i.e. $\{z = 0 \pm \zeta_z\ , \theta = 0 \pm \zeta_\theta\}$ where ζ_z, ζ_θ denote some miniscule offset for z and θ from equilibrium. The results from this approach however did not mature enough to make them suitable for presentation here.

The policy derived from the converged Sarsa algorithm was found to perform somewhat sub optimally when compared to Q. Greedy policy using the max Q value for selecting the next action was employed.

4.4 Sarsa with Eligibility Traces

The Sarsa online-policy was supplemented with temporal difference learning using the trace decay parameter λ to catalyze the learning process in the sparse reward environment such as the one which characterizes our suspension dynamics model. The algorithm is modified to assign credit to achieving the goal to past states and actions using eligibility traces. Rewards associated with reaching the objective are propagated backwards and the credit is decayed exponentially.

Tuning the hyper-parameters required a fair amount of experimentation. The standard Q value update as shown in (4) was implemented in each iteration of the loop with state-action pair N(s,a) being reset for each iteration.

$$
\begin{aligned}
&N(s,a) \leftarrow N(s,a) + 1 \\
&\delta \leftarrow r_t + \gamma\, Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \\
&for\ s \in S\ ,\ a \in A \\
&\quad Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\delta N(s,a) \\
&\quad N(s,a) \leftarrow \gamma\lambda N(s,a)
\end{aligned}
\tag{4}
$$

The results from simulation with the trained agent using the learned Sarsa- λ policy are shown in figure 5.

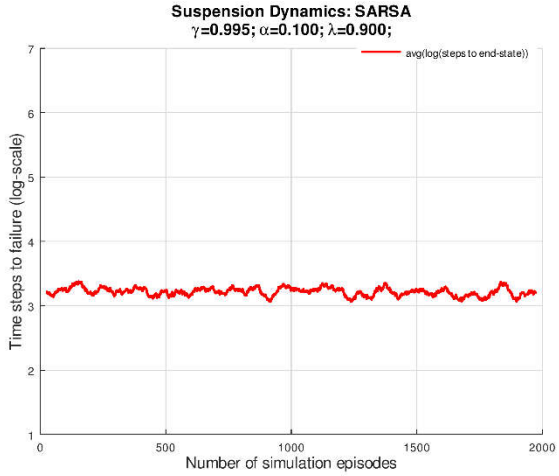

Figure 6 Sarsa- λ TD Learning Exploitative Policy from Simulation

5 Conclusions

As evidenced by the results presented in the sections above the Q-Learning algorithm illustrated considerable success especially when tested in a purely exploitative environment where the best learned policy was put to the test. The behavior of the policy was tested for ~20 simulation seeds. The parameter values of λ=0.995 and α=0.65 were found to yield the best performing policy. Results from Gauss-Seidel VI are presented more from an academic perspective rather than practical, computational intractability in high dimension problems would prohibit actual implementation.

In hindsight usage of a slightly more sophisticated and discerning reward model would have improved the learning performance of the algorithm and this is one area where the current model would benefit from improvement.

The state uncertainty is currently ignored under the pretext of sensor recordings being deemed 100% accurate. A POMDP model to incorporate uncertainty from sensor readings was initially planned but the development of the simulation model proved somewhat of a challenge and ended up consuming bulk of the budgeted time to preclude the possibility of any POMDP development.

6 References

[1] Effects of Passive and Semi-Active Suspensions on Body and Wheel Hop Control, Mehdi Ahmadian and Robert H. Marjoram, *SAE Transactions* Vol. 98, Section 2: JOURNAL OF COMMERCIAL VEHICLES (1989), pp. 596-604
[2] http://www-anw.cs.umass.edu/rlr/domains.html
[3] Using Eligibility Traces to Find the Best Memoryless Policy in Partially Observable Markov Decision Processes, John Loch and Satinder Singh
[4] Learning from delayed feedback: neural responses in temporal credit assignment, Matthew M. Walsh and John R. Anderson, *Cogn Affect Behav Neurosci.* 2011 June ; 11(2): 131–143. doi: 10.3758/s13415-011-0027-0.
[5] M. J. Kochenderfer. Decision making under uncertainty: theory and application. MIT press, 2015.
[6] Learning from Delayed Rewards. Cambridge Univ., Cambridge, England. Watkins, C. J. C. H. (1989).
[7] Learning to predict by the methods of temporal differences. Machine Learning, 3,9–44. Sutton, R. S. (1988)
[8] Reinforcement Learning: An Introduction. Cambridge, MA: MIT Press. Sutton, R. S. & Barto, A. G. (1998).