REINFORCEMENT LEARNING: MDP APPLIED TO AUTONOMOUS NAVIGATION

Mark A. Mueller

Georgia Institute of Technology, Computer Science, Atlanta, GA USA

ABSTRACT

The problem of autonomous vehicle navigation between lanes, around obstacles and towards a short term goal can be solved using Reinforcement Learning. The multi-lane road ahead of a vehicle may be represented by a Markov Decision Process (MDP) grid-world containing positive and negative rewards, allowing for practical computation of an optimal path using either value iteration (VI) or policy iteration (PI).

KEYWORDS

Reinforcement learning, MDP, value iteration, policy iteration, autonomous navigation, self-driving car

1. INTRODUCTION

Markov Decision Process (MDP) "worlds" are usually small rectangular environments composed of squares or cells which contain states which are connected by actions. The ideas presented herein extend the MDP "small world" to one which may be large or even infinite. In particular, MDP's and Reinforcement Learning (RL) can model a self-driving car in an environment comprised of a road with obstacles (negative rewards) and desired goals (positive rewards). We can thus allow a self-driving car to make decisions by modeling the road ahead of a car as a grid world. Different roads and road conditions in front of a moving car may be represented by different reward matrices.

2. MODELING THE ROAD AHEAD: TO FINITE AND BEYOND

If a driver were to embark on a coast to coast drive from San Francisco to New York City, the planning and the driving would require different levels of decision-making. For example, mapping a route (shortest or best path) requires a "big picture" view of roads and traffic, which we routinely do using smartphones' navigation apps. Another level of planning may involve where to stay overnight, where to purchase gas and eat, etc. Finally, at another level, a driver looks out at the road ahead. This view provides information required for subsecond decisions such as obstacle avoidance and staying in a lane versus changing lanes. This paper focuses on this aspect of autonomous driving and planning.

The road ahead can be represented by a grid-world. It's not necessary to model the road beyond the horizon for the purposes of short-term navigation. It may be possible to represent the road ahead as a 15x15 grid, where each square forward represents the next second of travel, and the squares to the left and right represent lane positions and the edges of the road. Three highway lanes in one direction can thus be represented as the row E|333|222|111|E, where "|" lines separate the three lanes denoted by numbers 1 to 3, each lane has 3 positions within that lane (left, center, right), and "E" represents the edge of the road. A segment of highway may be represented by multiple rows as described. If a car is moving 100kph (36 meters per second), and each square forward represents one second of travel, 15 squares represents about 0.5km in the direction of travel.

DOI:10.5121/mlaij.2017.4401

Furthermore, as the car advances to the next square, it is possible to move the "world" forward so that the end square is never reached! This contrasts with standard MDP in which cumulative rewards are optimized for a path through a gridworld from a starting state to a termination state. MDP can be employed to respond to a reward, which can be appropriately set up to move down the road and avoid obstacles. The reward matrix may be stated most simply as a "blank slate" of zeros with a reward at the forward end of the gridworld, so that the policy is to move forward. More complex rewards may be used to model lanes, so an autonomous car is rewarded for staying off lane dividers and away from the edges of roads. Different obstacles may be modeled with negative rewards: A bump in the road may be represented by a small negative number, while a pedestrian or a car should be represented by a very large negative reward.

3. SMALL AND SIMPLE: 3X3 WORLD

This work utilizes Python 2.7.10, equipped with Anaconda packages, and mdptoolbox (downloaded from pymdptoolbox.com). A simple 3x3 gridworld illustrates the size of R (reward matrix) and P (transition matrix) required for mdptoolbox and MDP type calculations in general. This illustration serves as the model for larger gridworlds discussed below.

The "world" is thus defined by R, P, and gamma, the discounting factor for future rewards. The 3x3 gridworld contains 10 states, where the tenth state is a terminal state which provides zero reward, and for which actions keep it in this terminal state. Figure 1 shows two different worlds' R (represented in code as vectors but displayed below as grids), and the resulting values and policies computed for gamma=0.9. 3 actions which can be described as motions: up (^), up-right (/), and up-left (\). Illustrations shown make use of Excel to add color to results which have been cut and pasted from python output.

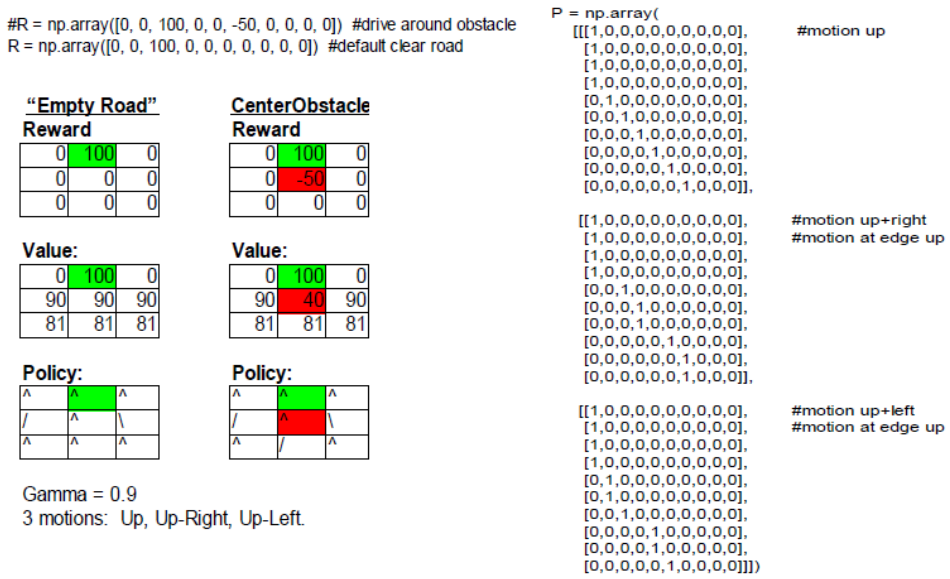

Figure 1: 3x3 World.

Figure 1 Upper Left shows 2 different rewards for open road and center obstacle (shown commented from code). Right: P requires 300 elements and shows impracticality of hand-typing P for worlds larger than 3x3. Figure 1 Lower Left: Results from value iteration, policy iteration and Q-learning (same results). The "Goal" square is denoted with green, obstacles with red.

P (policy) is also shown. For this 3x3 case, the author has hand-typed P containing 300 elements. Most are zeros; the 1's represent transitions to the next state for each motion (up is the 0^{th} or top matrix, up+right is the next square matrix, and up+left is the last square matrix). Border locations do not permit a motion "off the grid", and the world does not wrap around (as in some video games where the top wraps around to the bottom, and left wraps around to the right). Specifying a left-up move from the left boundary simply transitions upards (and the right boundary is symmetrically similar). The top "world" row simply points to a zero-reward terminal state that points to itself for all actions. For larger worlds such as 5x5, P is not shown but is similar in structure. Stochastic P's are not described here, but their matrices are similar in structure containing nonzero probabilities centered around the 1's as shown above in Figure 1.

Values and policies are computed three different ways: Value iteration, policy iteration and Q-Learning. Other methods of computing are also available with the mdptoolbox package (not used here) include: finite horizon, relative value iteration, value iteration GS (Gauss-Seidel) and policy iteration modified. The values and policies shown in Figure 1 were computed these three ways which agree. Q-learning is considerably slower, even for a low number of iterations, so MDP computing for the remainder of this paper focuses on value iteration (VI) and policy iteration (PI).

Computing Times for 3x3:	
Algorithm	Time (sec)
Value Iteration	0.002
Policy Iteration	0.006
Qlearning, 10k iters	1.2
Qlearning, 30k iters	3.6
Qlearning, 100k iters	12
Qlearning, 300k iters	36
Qlearning, 1e6 iters	119

Figure 2: Computing times comparing VI, PI and Qlearning. Value iteration is 3X faster than Policy Iteration; Q-Learning is magnitudes slower than either of these.

4. 5X5 WORLD

We can similarly create examples using a 5x5 world. At this point it is not practical to show the P matrix, which is similar to that shown in Figure 1 but already large enough to fill an entire page. However, one P matrix can be used with a number of R matrices to demonstrate obstacle avoidance and "road edge" examples. Four different reward vectors (shown in matrix form for illustration) and the resulting P and V are shown in Figure 3. These include "Empty Road", "Obstacle" and "Obstacle and Edge". Code for 5x5: A4_mdp_5x5.py. "Empty Road" reward is simply 50, 100, 50 rewarding reaching the center of the top of the gridworld. "Obstacle" is adds two negative reward squares of -100, denoted in red in Fig. 3. Finally, "Obstacle and Edge" provides a -10 reward (punishment) for driving on the edge of the gridworld or road, denoted in yellow in Fig. 3. Unless noted, gamma=0.9.

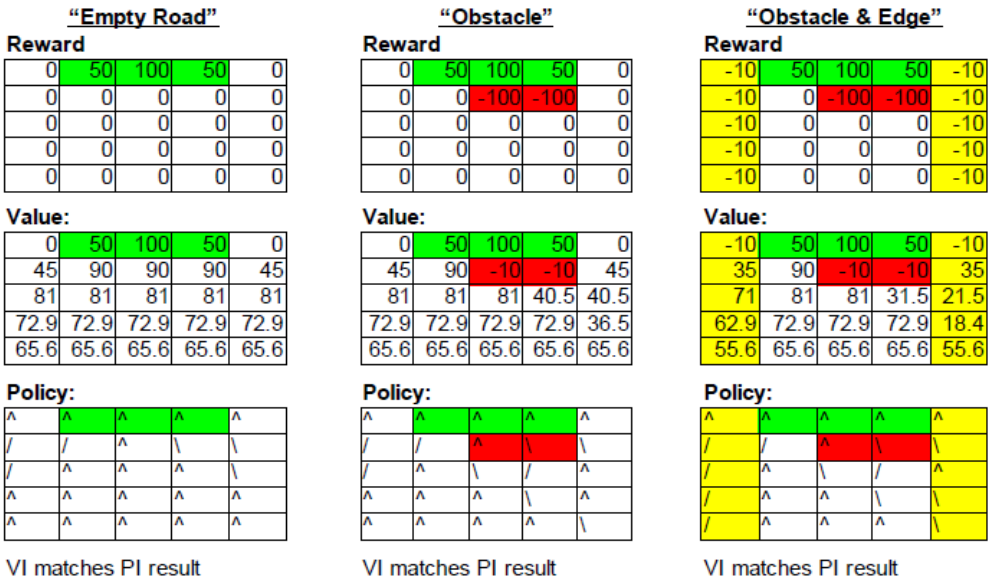

Figure 3: Three different 5x5 rewards result in different policies.

In Figure 3, the policies for center and right cases show how the obstacle is avoided. Note, however, that a vehicle at the bottom row does not act proactively. Some lateral motions are "last second" maneuvers, which can be changed with different R.

We can observe in Fig. 3 that the choice of rewards "forces" the vehicle to avoid the obstacle (red). This model incentivizes ending in the center of the lane (or road/world), but the rightmost model also punishes with -10 for driving on the edge of the road. In Figure 4, driving decisions are illustrated with a focus on improving the model for "Obstacle & Edge". In particular, the optimal center-starting path on the left, shown in blue, could be placed away from the obstacle by moving left sooner. This would provide more margin for avoiding the obstacle. Similarly, the right path, which starts from the right 2nd square from the bottom shown in blue, swerves in front of the obstacle, then turns back to the edge of the road.

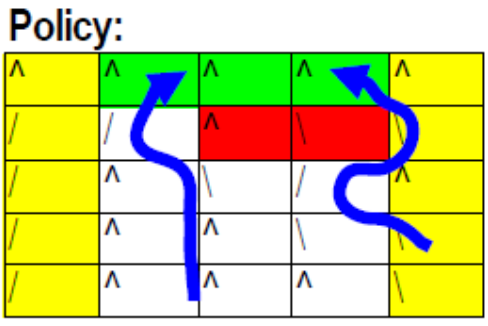

Figure 4: Obstacle and Edge moves critiqued.

In Figure 4, blue drawn lines indicate paths which move over late or unnecessarily swerve in front of the obstacle.

By modifying the reward matrices, "driving behavior" or policy can be modified to one which suits the modeler. In this case, we can punish occupation of a square in front of an obstacle in order to create a driving behavior with greater safety margin: we can place -20 rewards in the squares below the obstacle squares, and expect different paths. Results are shown in Figure 5 (right, blue paths). Figure 5 shows earlier turning in front of the obstacle compared to the paths described in Figure 4.

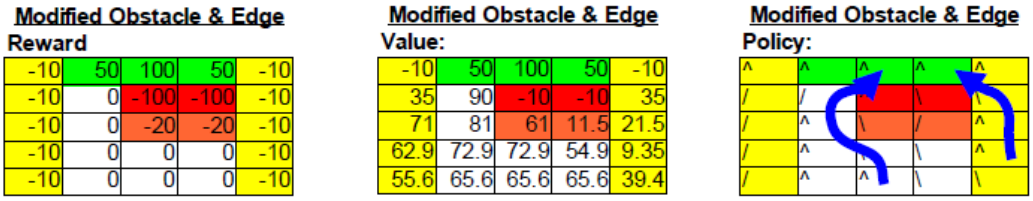

Figure 5: Changing the obstacle model slightly shows improved driving behavior.

Different discount values (or gamma) are considered for the another 5x5 case; results are shown in Figure 6. Only gamma values between 0.0 and 1.0 may be chosen, as required by mdptoolbox. We can see from the table that policy results are robust. Policies are compared to those computed for gamma=0.9 (shown in Figures 5 above). Thus, the model is fairly robust to gamma changes.

	Policies compared to gamma = 0.9	
gamma	VI policy	PI policy
0.1	different	different
0.2	different	different
0.4	same	same
0.6	same	same
0.8	same	same
0.99	same	same
>=1.0	N/A	N/A

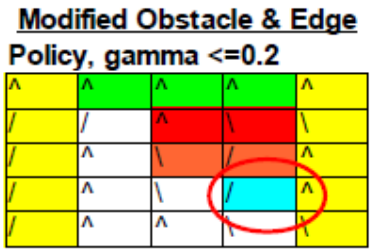

Figure 6: Policies computed for VI and PI (value iteration and policy iteration).

In Figure 6, results for different gammas are mostly the same as for gamma=0.9, except for gamma <=0.2 where we see one state's policy (indicated in cyan, circled in red) changes from left-up to right-up.

Computing times for these 5x5 models: 0.003 sec for VI, 0.008sec for PI. VI required 9 iterations, while PI required 4 iterations to reach zero change.

5. 5X5: TO FINITE AND BEYOND

The 5x5 grid world model can be used in a much larger world by continuously updating, or moving forward, the world with each move. As the vehicle moves forward, the world moves forward and the car is placed in the "moved-to" column but kept in the last row, while the features of the road move towards the car in the 5x5 world and roll off the back of the world.

In this manner, the 5x5 world can be used to drive in a larger or even infinite world. This is similar to video games which show a finite-screen world, updated as the screen boundaries cross into "new" space. Such perception of finite space within a much larger world also mimics

human driving and other human tasks. A human will process what he/she can observe within a horizon of observability or relevance, with a higher weighting for objects/situations which are closer. The models presented here similarly allow for an artificially intelligent (AI) agent to "think" in terms of "closer is more relevant" versus "far away planning" by tuning gamma, and and AI agent can more efficiently compute by limiting the horizon to the nearby world. A sequence for a vehicle moving step by step is illustrated in Figure 7.

Fig. 7, left shows a series of grids in order of time steps 1 to 12. At each step, a reward vector is processed to form a Value vector (not shown, but similar to those shown previously) and a Policy vector. The vector can be presented as a matrix in the shape of the 5x5 world for showing the path of a vehicle. In each succeeding time, the "world" is moved forward so the vehicle stays in the bottom row.

The vehicle position at the bottom of the 5x5 grids is shown in dark blue, the "next position" is shown in cyan (light blue). As before, edge of the road is yellow, obstacles are red, and pre-obstacle squares are orange. Goal squares with positive reward are colored green. Separate files (with R for each grid) are generated as a function of time. In a real-world application of self-driving vehicle, R would be generated by a vision system or by other sensors.

Figure 7, right shows the resulting "map" of the vehicle driving through obstacles. Value and Policy iteration both produce the same Value and Policy results, in 0.002 and 0.006 seconds, respectively. The algorithms are not too computationally intensive for real-world application.

In summary, this example illustrates how an autonomous vehicle can effectively utilize an MDP representation and use either Value Iteration or Policy Iteration to update state and choose actions.

It is worth pointing out that the MDP models described in this paper are more appropriately applied to a motorcycle (narrow width) rather than a car, since the grid positions shown occupy one square even with increasing resolution. However, models could be extended to vehicles with greater width simply by summing the cumulative rewards for several squares across as the car with corresponding width moves forward. In this manner, larger vehicles may be modeled. A progressively complex model could even model the tire-touching portions of road separately from the other parts of a car, in order that a car could avoid tire-puncturing objects such as glass which could otherwise be driven over with a tire path which avoids such objects.

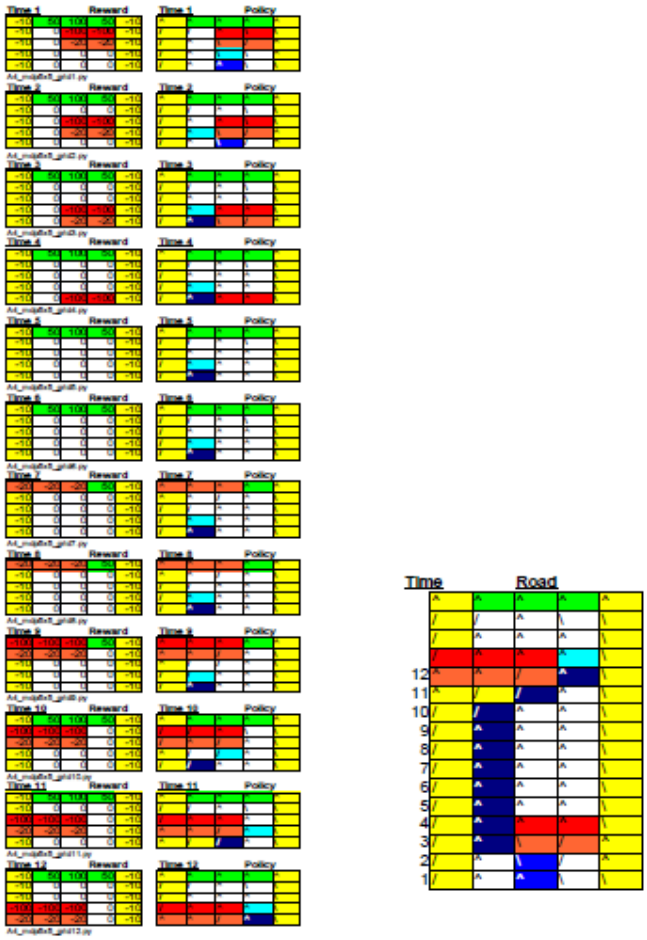

Figure 7: Car (blue) maneuvering around obstacles (red). Self Driving Vehicle time sequence of rewards and resulting policies (left) and resulting map showing the path traveled (right).

6. LARGE: 15X15 WORLD REPRESENTING A 3-LANE HIGHWAY

With a 15x15 grid, we can represent a 3 lane (in one direction) highway. The center median occupies two squares, the edge of the road occupies 2 squares, each of three lanes occupy three squares as before, and each pair of lanes is separated by one-square lane dividers. The "end of the world" goals are rewards shown in green; road edges and lane dividers are -10 rewards shown in yellow. Figure 8 shows the R, Value and Policy for the open road (gamma=0.9 discount factor used for calculation). Figure 9 also demonstrates that anything larger than 15x15 is difficult to represent in this format.

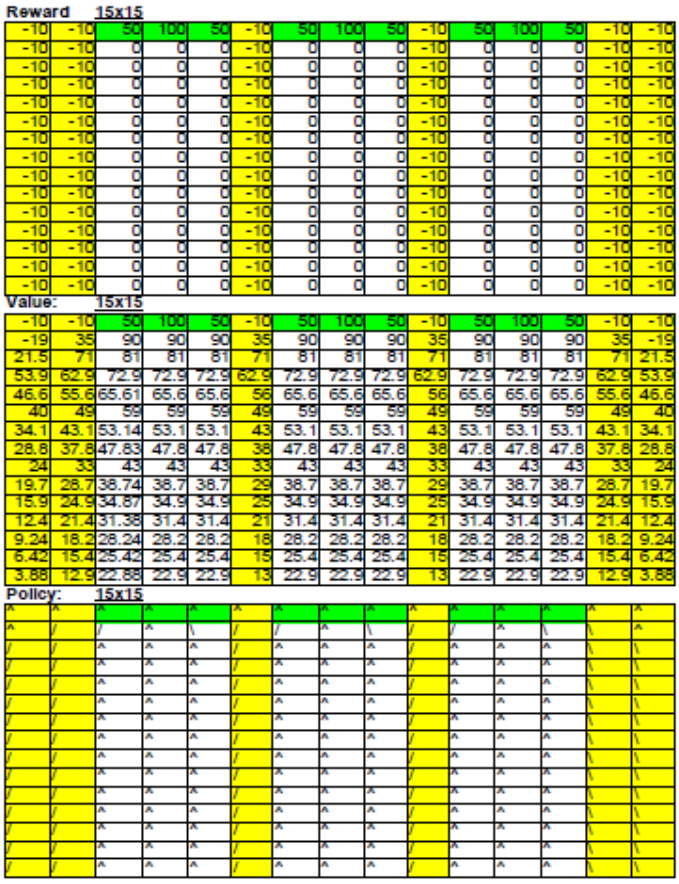

Figure 8: 15x15 grid representation of 3 lane highway "open road".

Top: reward, Middle: Value, and Bottom: Policy.
Computing times for VI and PI are 0.029 sec and 0.031 sec, respectively. VI ran through 16 iterations to reach zero changes; PI ran through 3 iterations to zero change. Q-Learning is not utilized for this model.

Figure 9 shows another model of the 15x15 highway, in which 2 lanes are blocked and one lane is open. This is similar to Figure 8, but with obstacles added. These obstacles are shown in red/orange. A blue line, drawn badly, shows how the policy points towards the largest reward (100) at the top center; the others are either blocked or replaced with (red) negative rewards. The policy is consistent with the rewards and values shown.

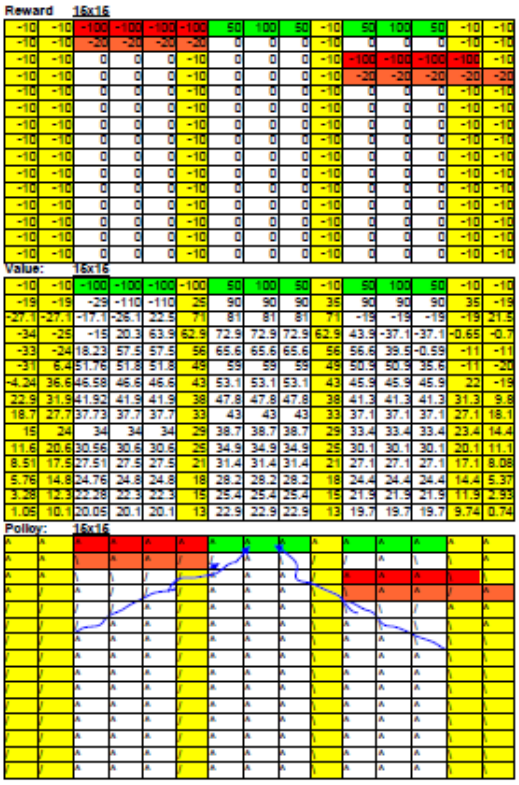

Figure 9: Three lane highway with obstacles represented by 15x15 grid. Reward is given (top); Value (middle) and Policy (bottom) are computed. Obstacles are red/orange. Lane boundaries and road edges are yellow.

7. COMPUTING SPEEDS FOR LARGER WORLDS

Computing speeds were measured for worlds between 20x20 and 50x50. Adding random values to R (to eliminate zeros) did not alter computing speeds. Summary: VI is always faster than PI, and the difference in computing speed grows with world size. Results for times and number of iterations are shown in Figure 10. Summary: VI is always faster than PI for the gridworld models considered here.

grid size	Value Iter time (sec)	Value Iter iterations	Policy Iter time (sec)	Policy Iter iterations
20x20	0.015	21	0.047	8
25x25	0.047	26	0.203	14
30x30	0.079	31	0.724	22
40x40	0.283	41	3.956	29
50x50	0.703	51	9.366	37

Figure 10: Computing times and iterations vs world size, comparing VI and PI. VI is always faster.

For value iteration, a comparison of 25x25 and 50x50 computing times shows a factor of nearly 20 difference. The PI difference is even more pronounced. As the gridworld model size becomes larger, VI is progressively faster.

The computing speed results show that is reasonable, especially for autonomous vehicles with distributed processing, to compute local grids and communicate goals to neighboring grids/vehicles, and use a smaller moving world model rather than computing for a too-large world.

8. CONCLUSIONS

The python package for mdptoolbox is useful for performing value iteration (VI) and policy iteration (PI) calculations. We introduced this package with 3x3 grid-worlds with 3 possible forward motions. These were further applied to 5x5 worlds representing a road. For all of the computations in this paper, PI and VI consistently produce the same results in terms of Value and Policy, even for a wide range of gamma (discount factor) considered.

Modeling of the road ahead was was accomplished by moving the world ahead along with the movement of a vehicle. MDP grids were utilized to model a real road containing obstacles, edges and lanes. Both VI and PI can be used for relatively fast computation for moving a vehicle to avoid an obstacle and reach the designated goal, which moves forward with the vehicle. However, VI is consistently faster than PI, and progressively faster for larger gridworlds.

Finally, models of a 3-lane 15x15 gridworld highway were generated. This size of world is sufficient for representing lane boundaries, road edges, obstacles and goals. Larger grids are possible; these require different visual representations than those presented here. VI Computing times for larger grids were shown to be practical for autonomous cars.

REFERENCES

[1] pymdptoolbox http://pymdptoolbox.readthedocs.org/en/latest/api/mdp.html

[2] R package mdptoolbox https://cran.r-project.org/web/packages/MDPtoolbox/MDPtoolbox.pdf

[3] matlab http://www7.inra.fr/mia/T/MDPtoolbox/index_category.html

[4] pymdptoolbox info http://www7.inra.fr/mia/T/MDPtoolbox/index_category.html

[5] pymdptoolbox info https://github.com/sawcordwell/pymdptoolbox

[6] http://mnemstudio.org/path-finding-q-learning-tutorial.htm