

QUESTION: Observe what you see with the agent's behavior as it takes random actions. Does the **smartcab** eventually make it to the destination? Are there any other interesting observations to note?

1. **Prior** to implementing the random choice among possible actions (`None`, `'forward'`, `'left'`, `'right'`), the red car stayed in place. This makes sense, because previously the action for each iteration was always `None`.
2. **After** implementing the random choice among possible actions, as expected, the car began to move, and yes, it usually doesn't eventually make it to its destination, even if `enforce_deadline` is set to `False`, since the `hard_time_limit` in `environment.py` terminates that trial after 100 moves.
3. After listening to the lectures on Q-Learning, I admit I **expected** for the car to reach its destination "better" (e.g. more directly) on subsequent iterations, but then I noticed `# TODO: Learn policy based on state, action, reward`, and realized that learning will come later.
4. It's a little hard to tell, but it also appears like the agent is, at this point, doing wrong things (e.g. disobeying traffic laws in ways that are likely to bring harm to itself and others). From what I could tell in `environment.py`, such harmful actions are classified as `Invalid Moves`, and result in a reward of `-1.0`. There were definitely a few rewards of `-1.0` in the simulation.
5. So, at this point, it seems like we've successfully given movement to an actor which is likely to (unintentionally) inflict harm. Yikes!

QUESTION: What states have you identified that are appropriate for modeling the **smartcab** and environment? Why do you believe each of these states to be appropriate for this problem?

1. Many, but not all, of the elements in `inputs` (that is, in `self.env.sense(self)`) are important for modeling the **smartcab** and its environment.
 - i. The state of the light (e.g. `red`, `green`) is required to know whether the desired next action is presently allowed.
 - ii. Similarly, the location and heading of other cars is important for knowing whether or not our agent needs to adjust its desired heading. Without knowing this information, our agent will be unable to perform **collision avoidance**. Note however that this information is concisely conveyed by the value stored at the `left` and `oncoming` indexes of `inputs`, and does not need to be expressed explicitly as separate location and heading variables.
 - iii. However, it's important to note that the information provided by `right` is redundant to the information provided by `light`. Specifically, if the `light` is `green`, then traffic which is to the `smartcab`'s `right` should yield to the `smartcab` in all circumstances (i.e. should not turn `left` or go `forward`, because their light is `red`, and it should only turn `right` when there is no traffic going `forward` from its `left`).

In reality, it is perhaps paramount for a `smartcab` to pay attention to whether there is traffic to its `right`. Actors in the wild don't always yield (e.g. sometimes turn `right` into a far `left` lane to miss traffic from the `left`, sometimes turn `right` without noticing traffic from the `left`), and a `smartcab` shouldn't ignore the possibility of a collision when right-of-way dictates that none should occur.

Nevertheless, for the purposes of this exercise, it is appropriate to assume that other actors will obey traffic laws at all times. That being the case, information about cars to the `smartcab`'s `right` is **redundant**, and is therefore not an appropriate state to consider.

2. Additionally, it is practically important to include `self.planner.next_waypoint()`, since the reinforcement provided by `Environment.act()` considers whether the action taken is or isn't the action which `RoutePlanner.next_waypoint()` expects.

In one sense, we want the smartcab to have access to all of the information it's being graded on, so that it can learn for itself the way to get the best grade.

In another, broader sense, reinforcement learning is a *slow* process, unless feedback can be provided fairly regularly during the process. To use an analogy, it's going to take someone a long time to learn the rules of football if you only tell them 'good job' after every game they play, hide the scoreboard from them, and don't let them know when they or anyone else got a penalty.

Accordingly, we can imagine that grading the smartcab according to `RoutePlanner.next_waypoint()` as injecting domain knowledge into our Q-Learning algorithm (where it *should* go, except in certain circumstances). Certainly we hope that the smartcab will improve on this suggestion (specifically, by not crashing into other cars or disobeying traffic signals), but this frequent feedback is crucial to rapidly developing a sensible policy.

3. It is neither practical nor desirable, however, to include the `deadline` as a state about which the smartcab should care. Firstly, if the deadline only reaches to 20, then the **size** of the state space increases by a factor of 20. This means that it will take 20 times as long to explore the state space, and therefore ~20 times as long to converge to a sensible Q-Learning policy. Secondly, it's unclear what information the `deadline` would be conveying to the smartcab beyond the information that's already conveyed by merely missing the deadline; should it behave differently when the `deadline` value is smaller? Should it start disregarding traffic laws in order to reach its destination? That's probably not the result we would want here, and so it would be inappropriate to include `deadline` in the state vector.

OPTIONAL: How many states in total exist for the **smartcab** in this environment? Does this number seem reasonable given that the goal of Q-Learning is to learn and make informed decisions about each state? Why or why not?

1. There are 96 distinct states which our **smartcab** should concern itself with. This is because there are two states for `light`, and four for both `oncoming` and `left` (specifically, `None`, `forward`, `right`, and `left`). (Note from the above question that `right` is being ignored, as it contains only redundant information). Also, from `RoutePlanner.next_waypoint()`, there are three additional, distinct states (`right`, `left`, and `forward`).

Accordingly, there are a total of $2 \times 4 \times 4 \times 3 = 96$ **distinct states** in this smartcab environment.

```
{'next_waypoint': 'forward', 'green': 'green', 'oncoming': None, 'left': None}
{'next_waypoint': 'left', 'red': 'red', 'oncoming': None, 'left': 'left'}
{'next_waypoint': 'right', 'red': 'red', 'oncoming': 'right', 'left': 'forward'}
```

2. This high number of states seems correct for the agent to have a full understanding of the intersection and its possible next actions. Without an understanding of this full, it seems unlikely that a **smartcab** would be able to explore alternative routes in case the chosen one is blocked.
 - For instance, consider a smartcab approaching a red light, and intends to turn right at the intersection. If another car approaches the intersection from the smartcab's left and intends to go forward, the smartcab must correctly ascertain that it must wait for the car to pass, even if the smartcab can legally turn right from a red light after having stopped first.
 - Even though there are 96 distinct possible states, many of these states are invalid, since they would cause an accident even in the absence of the smartcab. However, even strange accidents happen occasionally, so while the likelihood of many of the potential states is low, it's not zero, so the smartcab can't assume they'll never occur. But, note that within the confines of this assignment, other actors are presumed to make only valid decisions, so

there will be some of the 96 states which our **smartcab** never sees (e.g. perpendicular traffic disobeying their red lights).

QUESTION: *What changes do you notice in the agent's behavior when compared to the basic driving agent when random actions were always taken? Why is this behavior occurring?*

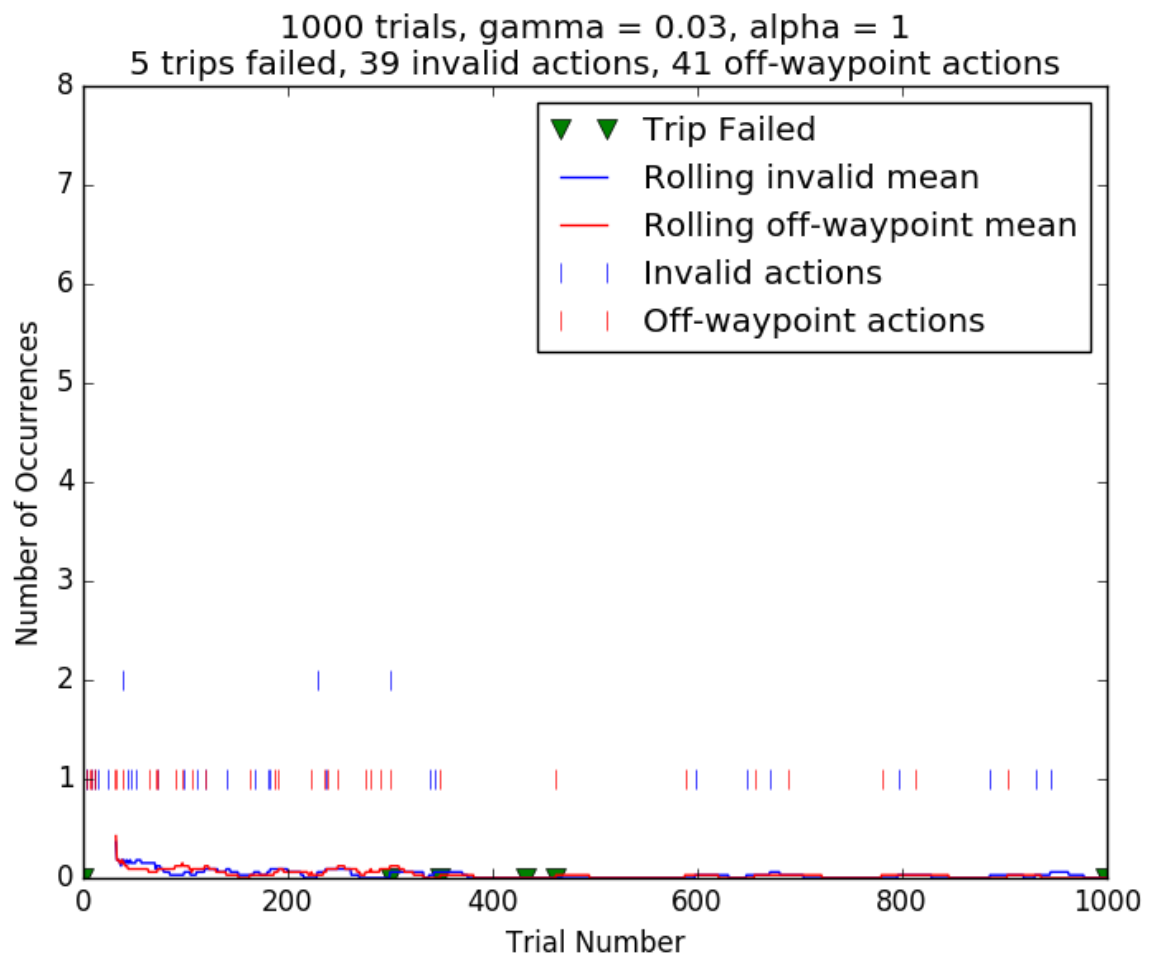
1. Previously, the smartcab was only taking random actions, and unfortunately no mechanism was in place to choose better on subsequent trials. Now, however, although the smartcab begins by taking random actions (many of which are invalid), by the third or fourth trial, the number of invalid actions approaches 0, and the smartcab appears to take a fairly direct route to the destination.

In short, this is because the smartcab is seeking rewards and remembering consequences, and the rewards have been constructed in a way that leads to the observed behavior. In more depth:

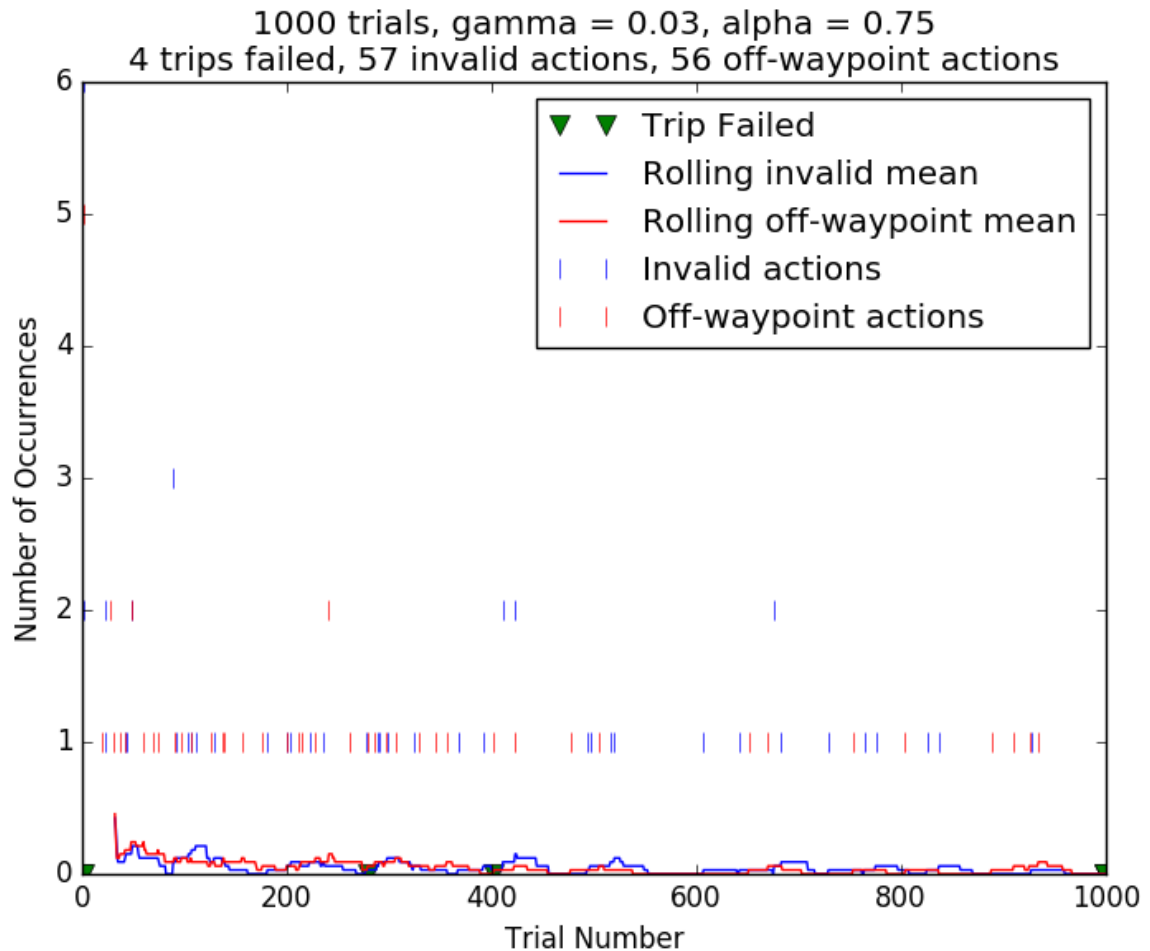
- i. The smartcab is determining its state (i.e. suggested next move, the color of the traffic light, and the location and heading of nearby cars).
 - ii. The smartcab is updating its Q-table (think of it like its memory for all scenarios) with the expected reward for any of the actions it could take in the current state (i.e. left, forward, right, or remain stationary). If the smartcab hasn't been in this state before, it assigns the actions a high reward, so that it explores things it doesn't know well yet.
 - iii. The smartcab takes the action with the highest Q-value in its table for this state. It breaks ties randomly, because there isn't a sensible way to break them systematically.
 - iv. After having acted, the smartcab again determines its state and updates its Q-table, except this time it doesn't care about any one action in particular, but rather about whether there is some high-Q action available to it now. (In essence, it knows it can make decisions later, but it's wondering if the last action it took it to a land of meaningful opportunity, and updates the value of that previous decision accordingly).
 - v. As discussed above, there are 192 distinct states, and 4 possible actions for each of these states. Accordingly, it takes a while for the Q-table in this simulation to converge to consistent, optimal behavior.
2. Interestingly, by the 7th or eighth trial (though perhaps sooner), the smartcab develops a tendency to loop, and never remains at an intersection. While comical, this is probably also not an ideal policy in the abstract, since it's wasteful of gas, increases wear on the vehicle, and greatly confuses the passengers. However, perhaps that behavior can be eliminated by tweaking variables in the Q-Learning algorithm.

QUESTION: *Report the different values for the parameters tuned in your basic implementation of Q-Learning. For which set of parameters does the agent perform best? How well does the final driving agent perform?*

1. `epsilon`, interestingly enough, was not used (discussion follows).
2. `alpha` was dynamic in this algorithm, always being the inverse of the number of times the Q-Learning algorithm had encountered the current state-action pair. Thus, initially, `alpha` is 1, then 0.5, then 0.33, etc. for a particular state in which that action has already been chosen. This has the effect of replacing `epsilon`'s simulated annealing effect, as over time, the learning rate "cools" just the same, though by a different calculation. For comparison, static `alpha` values were used as below:
 - i. For a static `alpha` of 1, the difference in performance between a dynamic `alpha` is negligible.



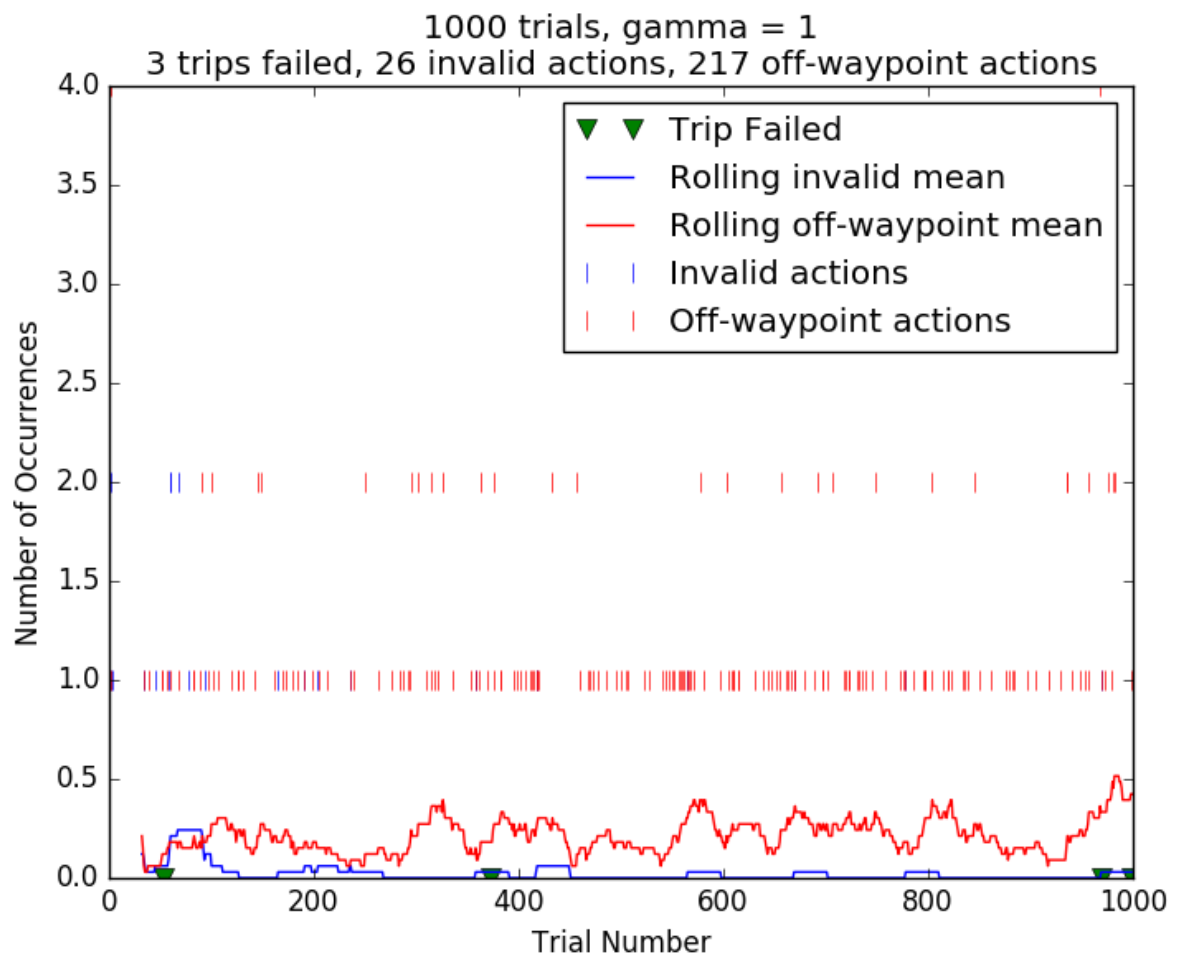
- ii. For a static alpha of 0.75, the number of invalid and off-waypoint actions increased somewhat.



iii. This trend continued as alpha decreased to 0.5, 0.25, 0.1, 0.02, and 0.01.

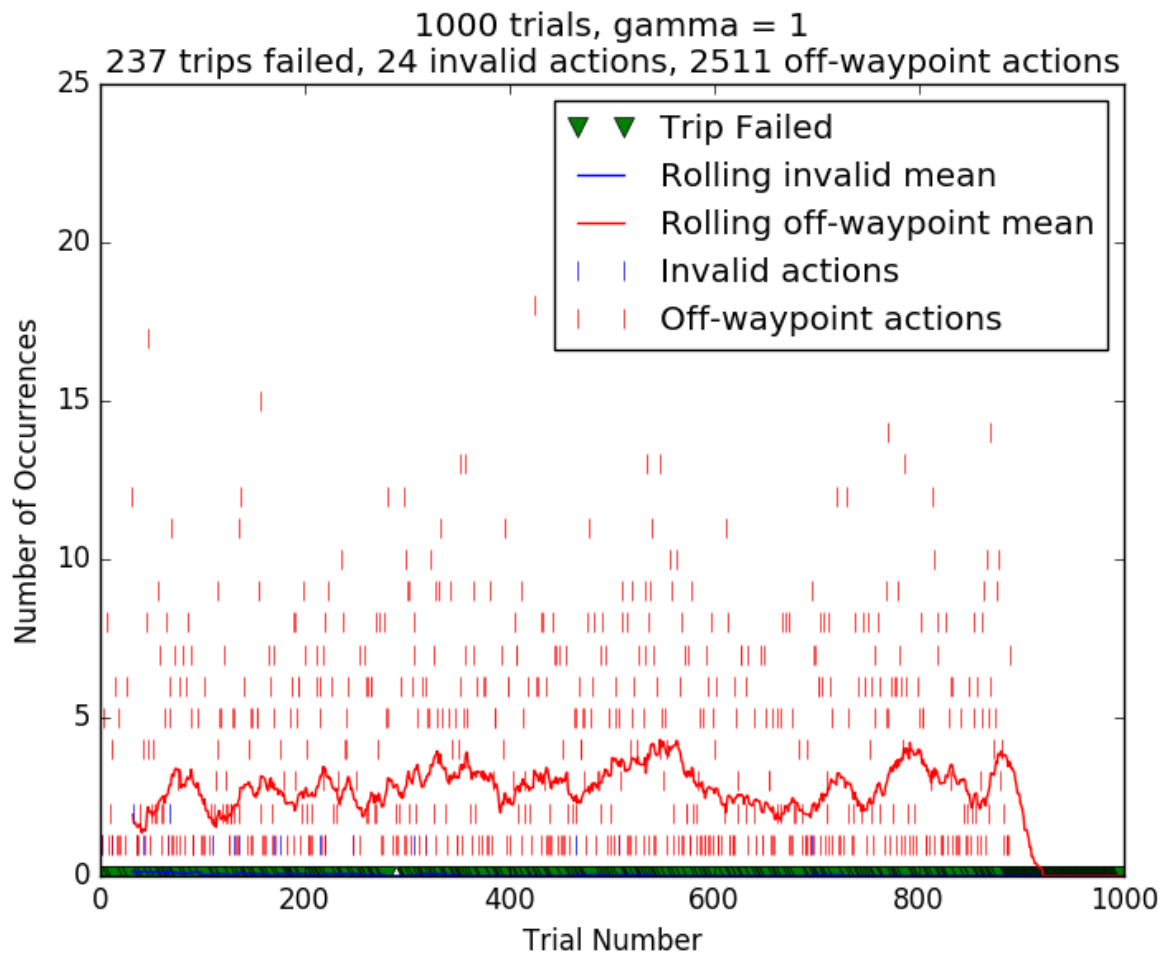
3. gamma (as hinted by the graphics above) was 0.03. This is primarily an attempt to suppress the "looping" tendency identified in the question above. Interestingly, a gamma this low still allowed the smartcab to "learn" a good driving policy within a few trials.

i. At a gamma of 1, the smartcab's behavior tends depend on its very early encounters. Sometimes, it reaches the target fairly frequently, but it takes unnecessary actions.

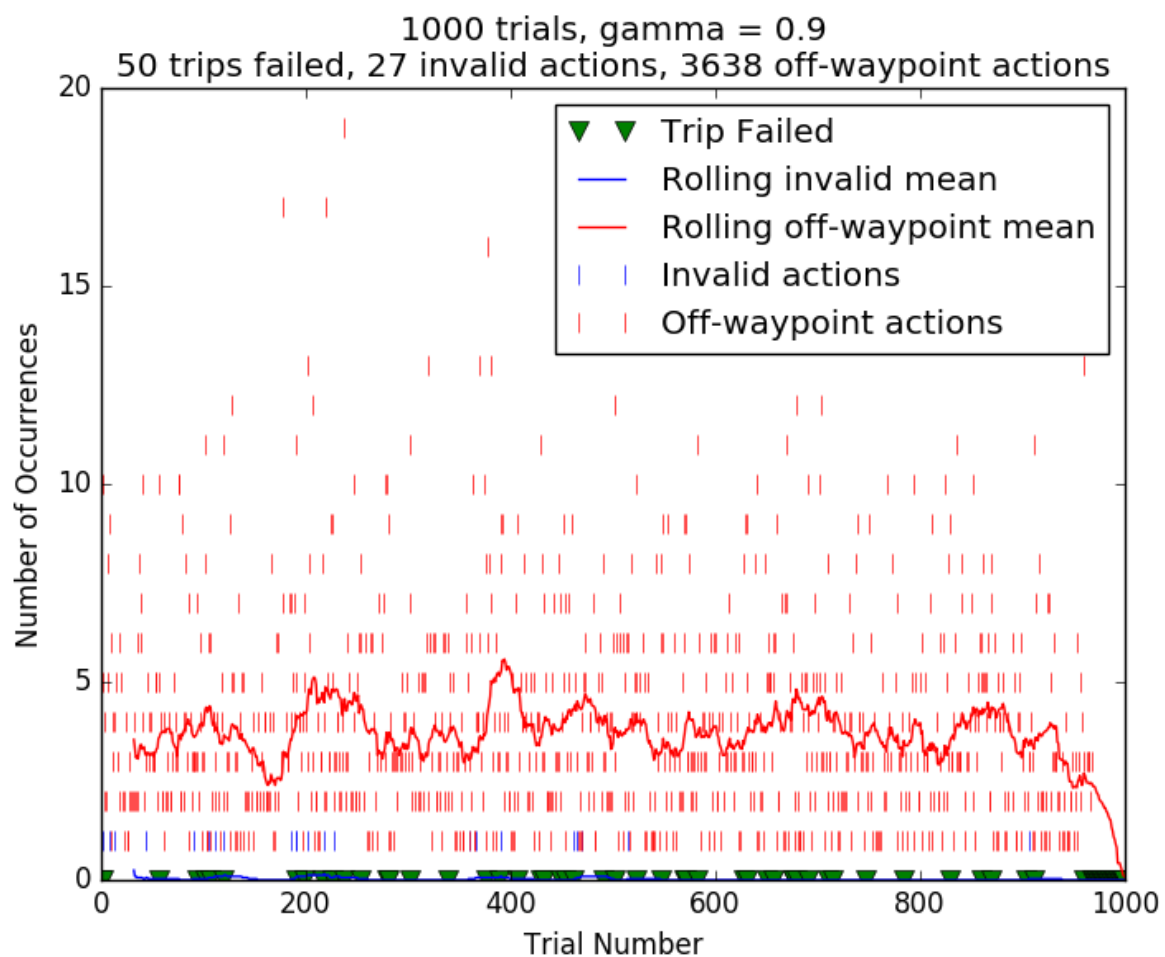


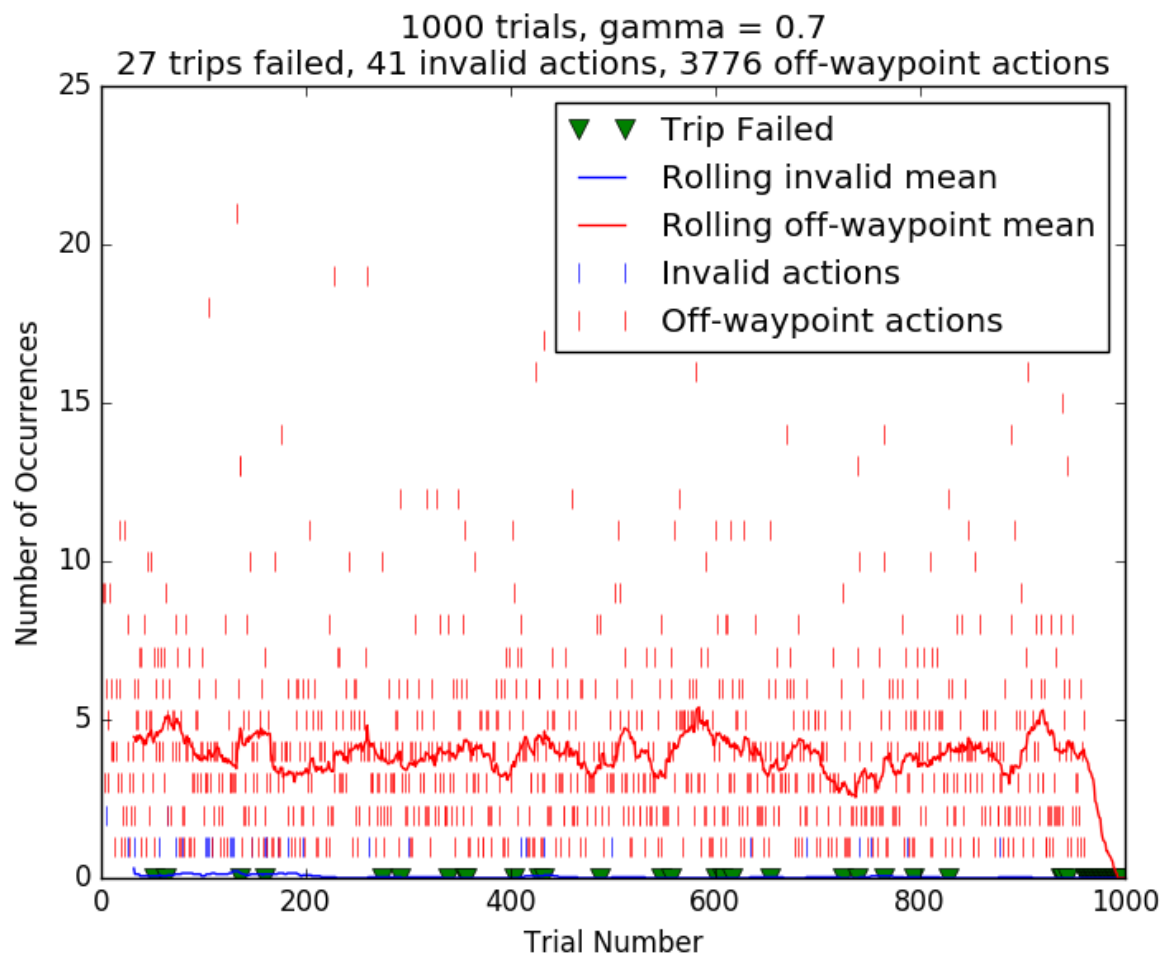
At other times, it prefers taking many unnecessary actions, and reaches the target far less

frequently.

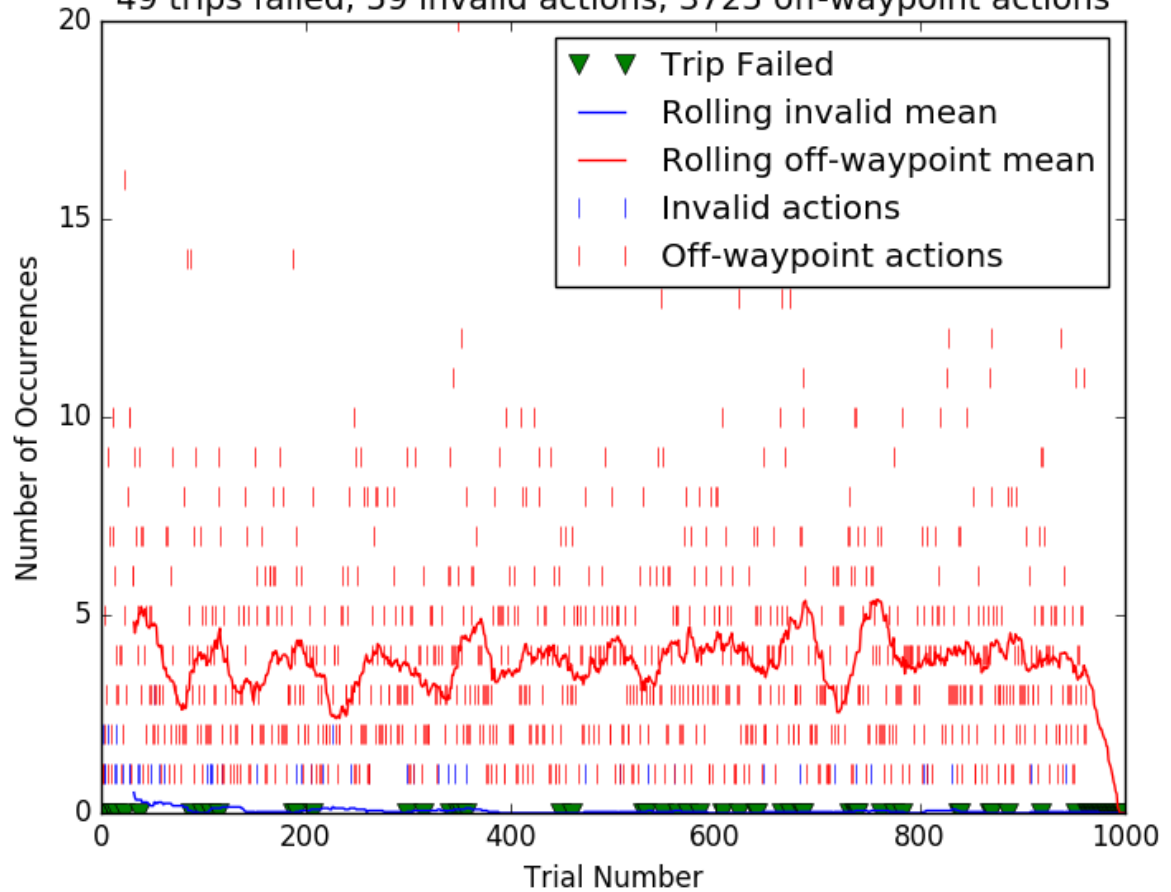


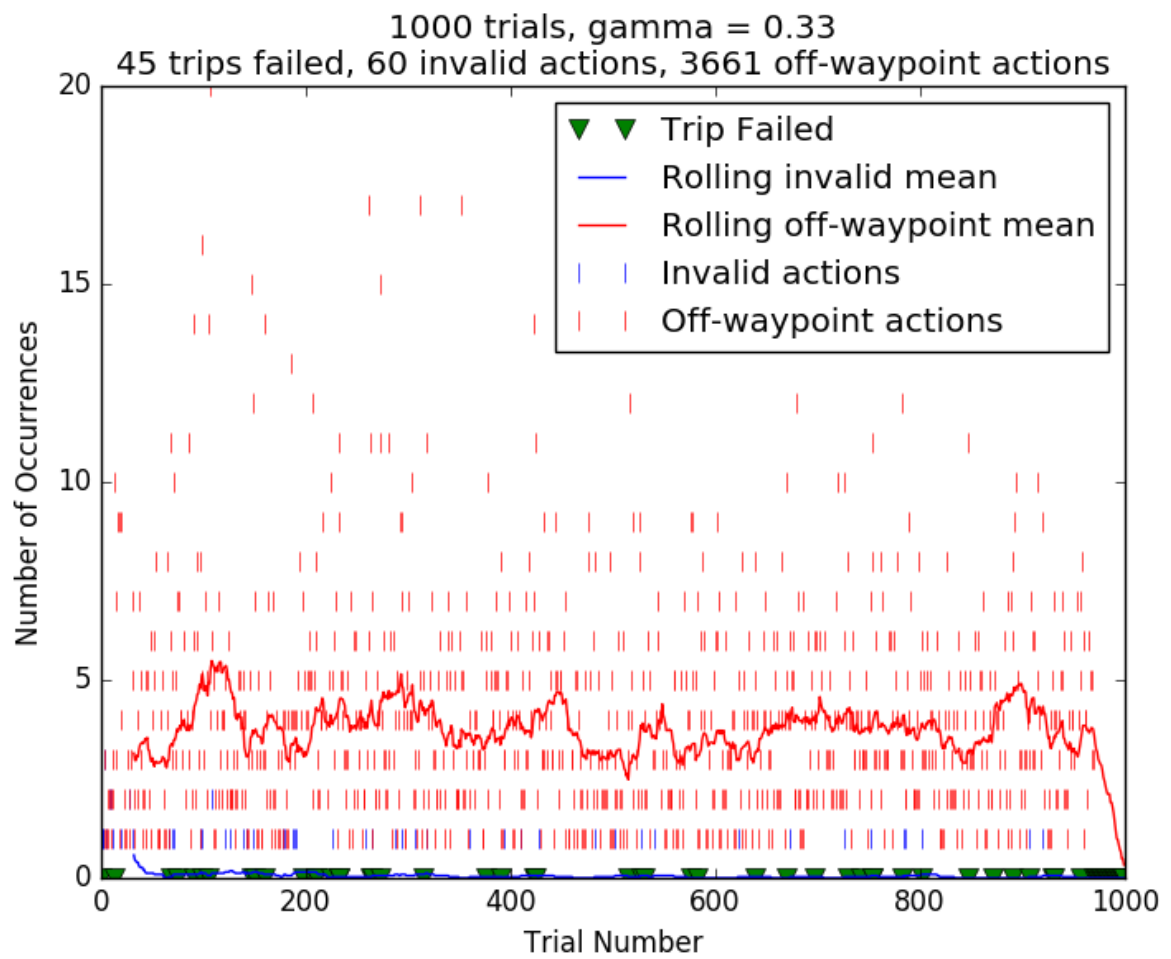
- ii. For gammas between 0.9 and 0.2, the car very quickly favors looping, and thus takes many unnecessary actions.



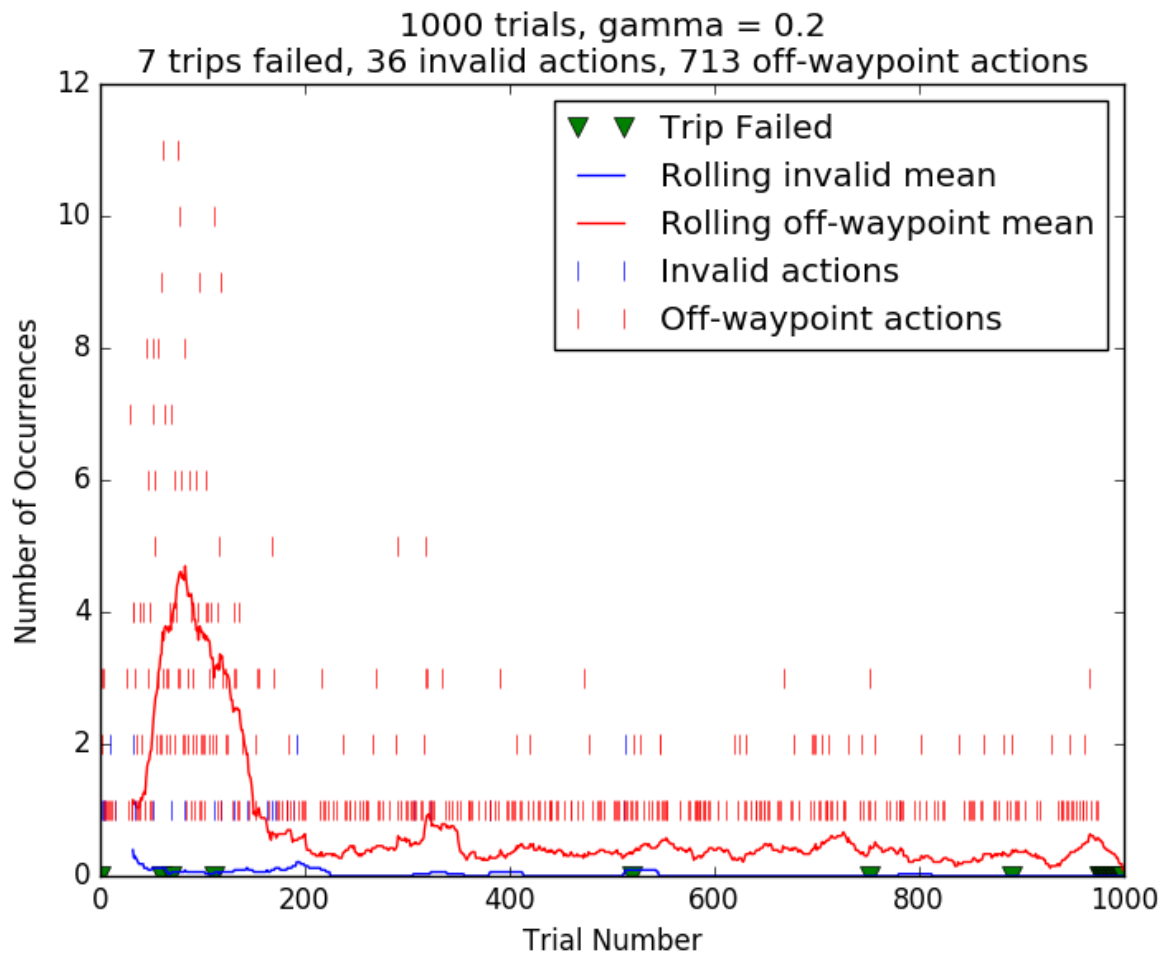


20

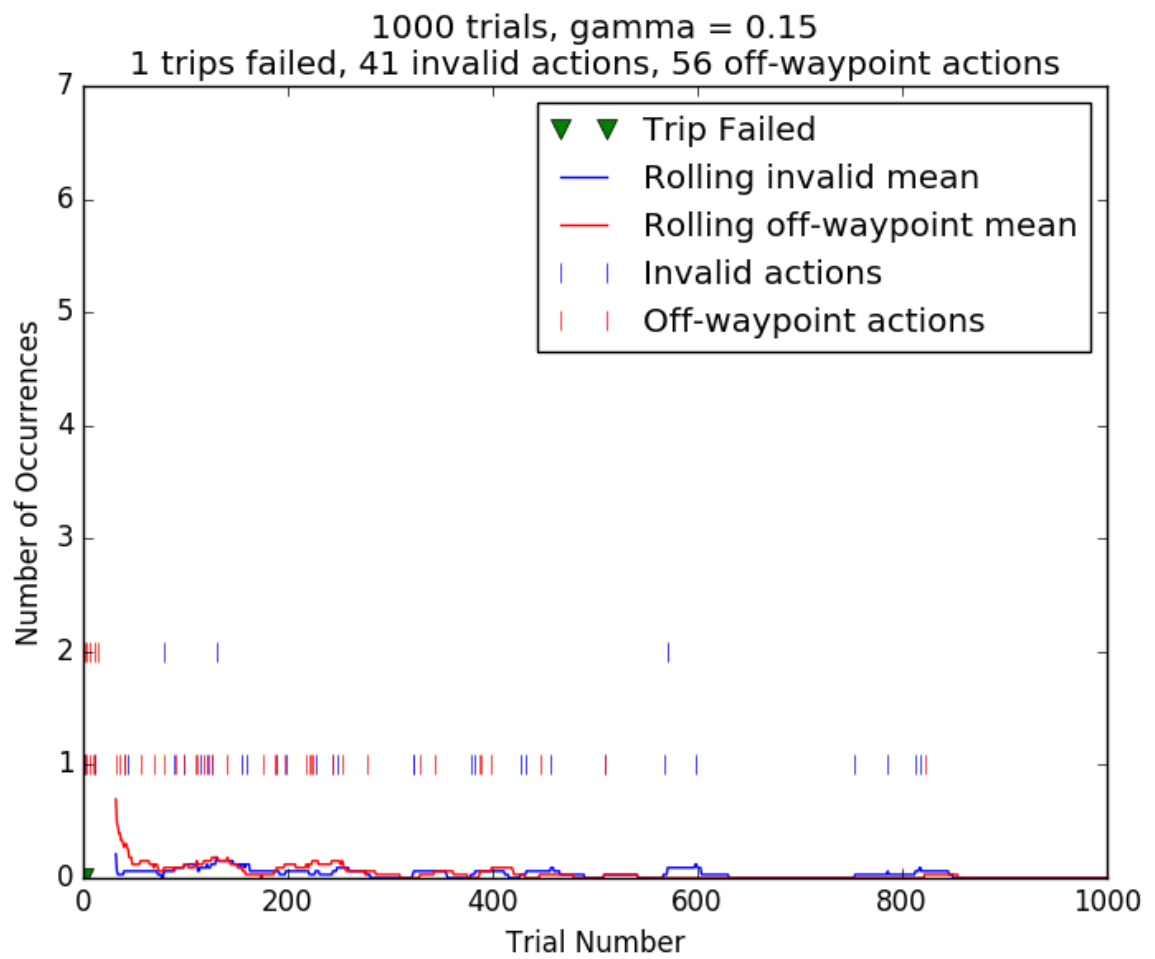


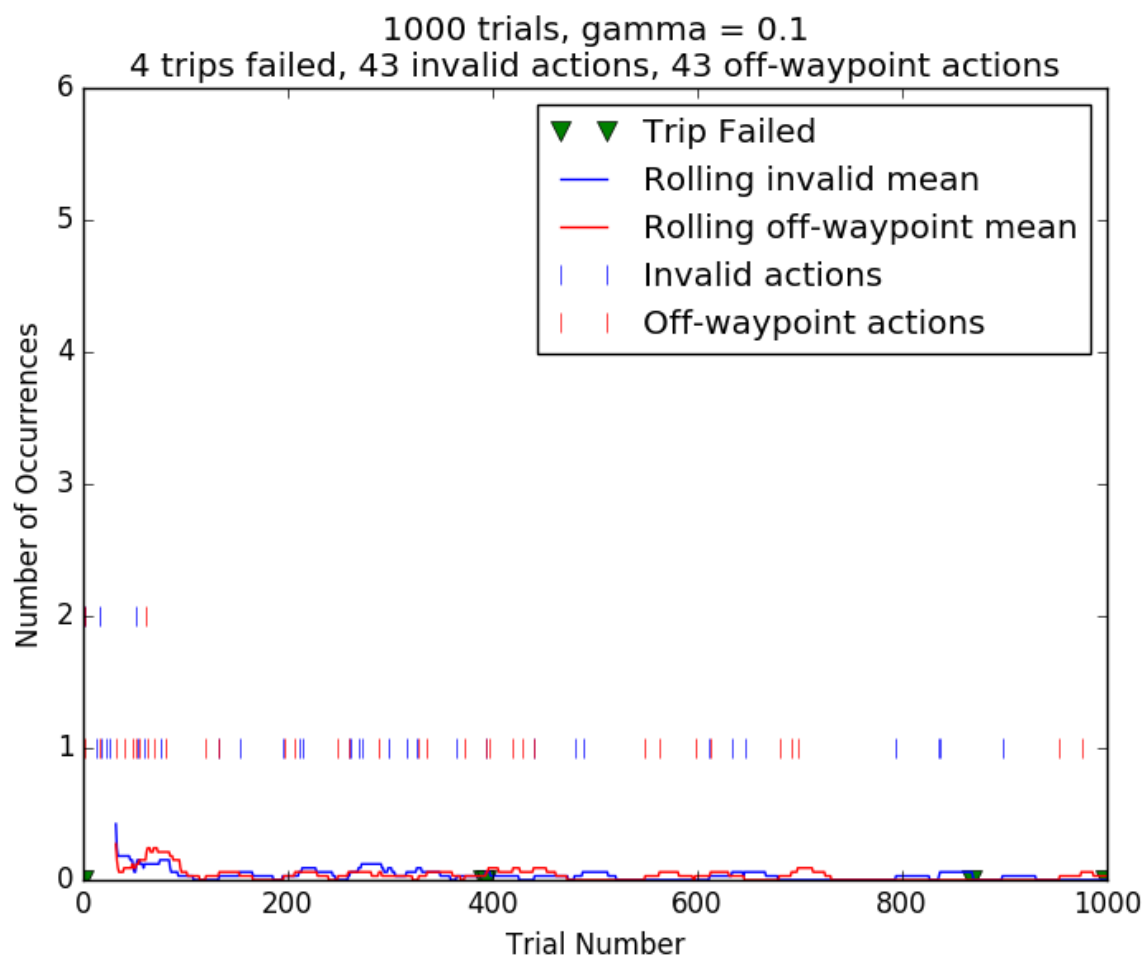


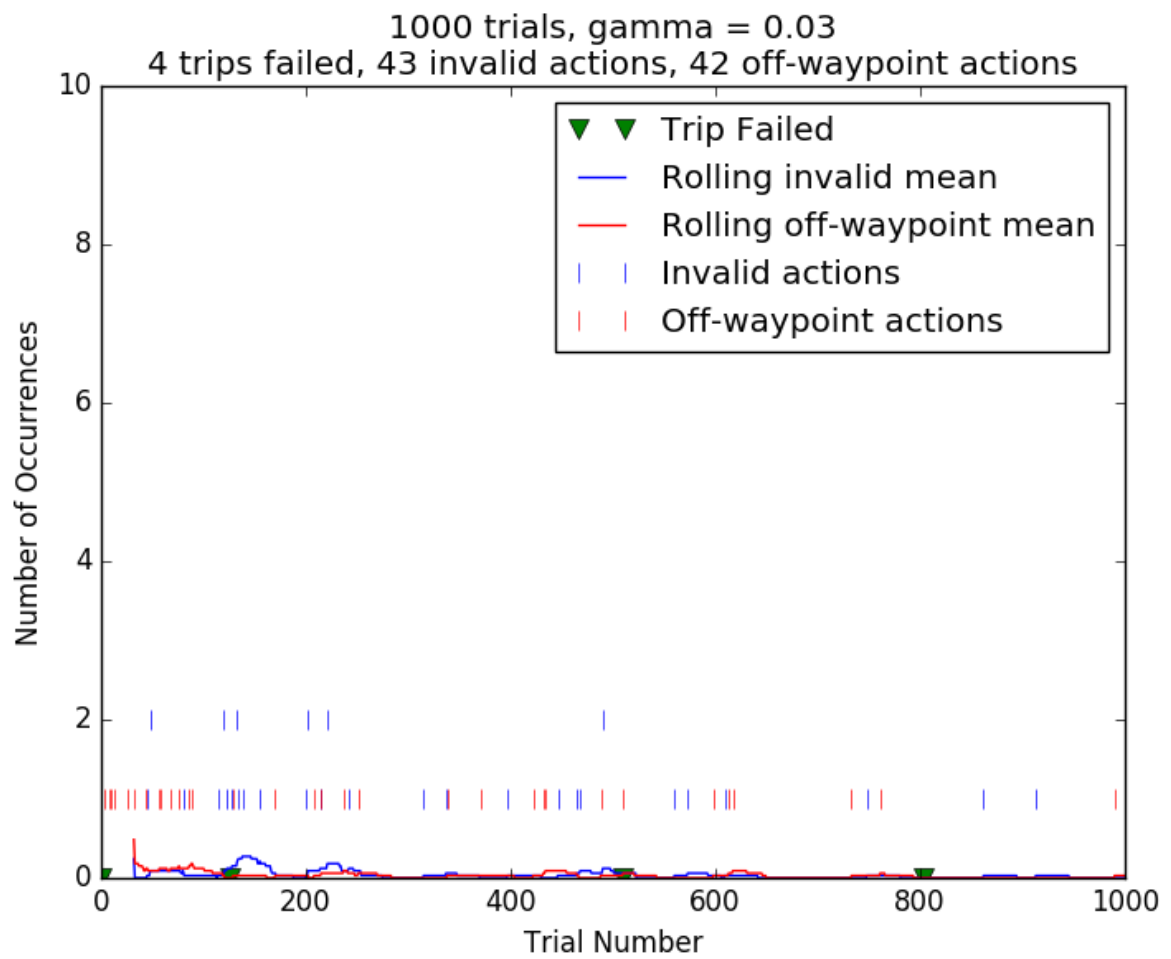
iii. At a gamma of 0.2, the tendency to loop begins to diminish.

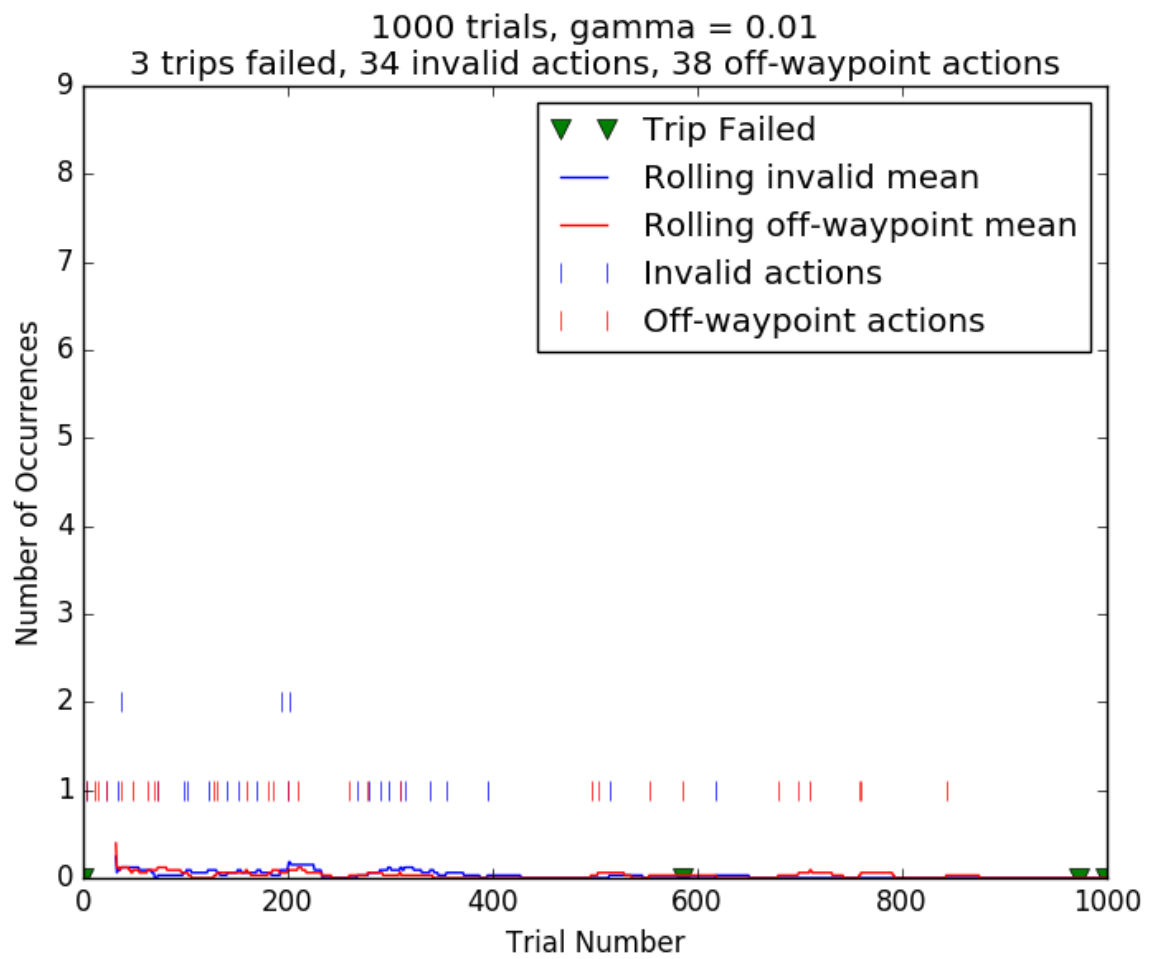


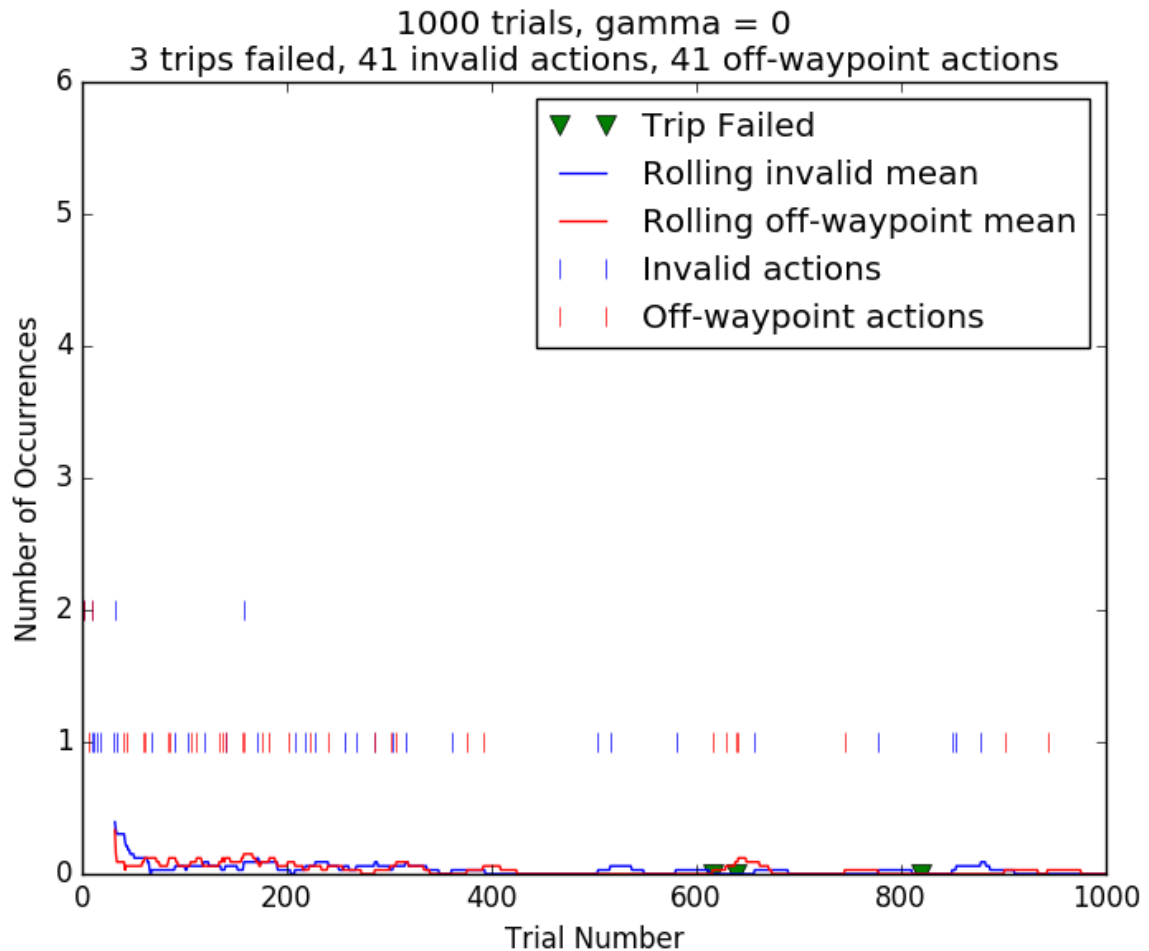
iv. For gammas between 0.15 and 0, there is no noticeable difference in behavior. Taking unnecessary actions and taking invalid actions are minimized.











4. The resulting driving agent prefers to identify a straight-line (Manhattan) path and follow it to the destination. This agent does not opportunistically divert to side streets when it finds itself at a red light (the "looping" behaviour discussed above). This agent obeys traffic signals, but occasionally acts in a way that would **cause a collision** with other vehicles on the roadway (yikes!). The success rate of this agent rapidly approaches 1.0 over 100 trials.
5. The occasional collisions, I assert, are because of a low number of trials relative to the probability of encountering any given other-vehicle configuration. Since encountering other vehicles is relatively rare, the smartcab doesn't have time in only 100 trials to learn not to ram into them.
6. By increasing the number of trials to 1000, the smartcab eventually encounters enough other-vehicle state-action pairs to "learn" the correct actions to take in those cases.

QUESTION: Does your agent get close to finding an optimal policy, i.e. reach the destination in the minimum possible time, and not incur any penalties? How would you describe an optimal policy for this problem?

1. To me, an **optimal policy** is one that mirrors Asimov:
 - i. The smartcab never collides with another vehicle or allows another vehicle to collide with it,
 - ii. it never disobeys traffic laws, and
 - iii. it always reaches its destination, so long as doing so is consistent with the above two.
2. The resulting agent quickly approaches, but does not quite achieve, an optimal policy.
 - i. After the first few trials, the agent generally attempts to head toward the destination along the straight-line Manhattan path. This is because, in the absence of other vehicles, it learns which actions can be taken at green lights and at red lights.
 - ii. For the remaining trials, the agent will occasionally encounter other vehicles. As it encounters other vehicles rarely, it is uncertain of the action it should take in these cases.

This usually results in collisions.

- iii. After each of these collisions, the agent learns not to act in that particular way (e.g. turn left) in that particular state (e.g. on a red light with no other cars around) again. After about 100 trials, the agent has learned enough to mostly avoid colliding with other vehicles while on the roadway.
3. While none of the `alpha` or `gamma` values used in this exercise resulted in the above **optimal policy**, when `alpha = 1` and `gamma = 0.03`, the smartcab came the closest to the optimal policy. In those trials, over 100 runs, only about 3-4 trips were failed, only about 20 invalid actions were taken (i.e. disobeying traffic laws), and only about 20 off-waypoint actions (i.e. actions which didn't match the Route Planner's suggested next action) were taken. These rates place it well below human drivers in terms of performance, so such an actor should *definitely* not be set loose on the roads. Nevertheless, it is an impressive accomplishment for such a simple algorithm to produce.
4. The following is a chronological ordering of the errors which the **smartcab** makes. Note that, for the most part, the smartcab first explores what to do at empty intersections (i.e. based solely off of `light` and `waypoint` information). The smartcab next makes periodic incorrect actions at intersections with other vehicles.

This makes sense, since there are only **24** possibilities for an empty intersection (2 states for `light` and 3 states for `waypoint` yields 6 total states, and there are 4 possible actions to take in each of those states), and since the field is relatively empty of other vehicles, the **smartcab** learns a lot about those states quickly.

Since encountering other vehicles is relatively infrequent, and allows for comparatively more choices (from the above calculation, 90 of the 96 states involve other vehicles), a smartcab seeking the destination does not learn enough about other vehicles to know when not to make invalid or off-waypoint actions around them.

Trial	Next Waypoint	Light	Left	Oncoming	Action	Reward
0	left	red	None	None	forward	-1
0	left	red	None	None	left	-1
0	left	green	None	None	forward	-0.5
0	left	red	None	None	right	-0.5
0	right	red	None	None	forward	-1
0	right	green	None	None	forward	-0.5
0	right	green	None	None	left	-0.5
0	forward	green	None	None	right	-0.5
0	left	green	None	None	right	-0.5
0	forward	green	None	None	left	-0.5
0	forward	red	None	None	left	-1
0	forward	red	None	None	right	-0.5
1	right	red	None	None	left	-1
2	forward	red	forward	None	left	-1
2	forward	red	None	None	forward	-1
7	forward	red	None	left	forward	-1
15	forward	red	None	left	right	-0.5
18	forward	red	None	left	left	-1
20	forward	green	forward	None	right	-0.5
21	left	red	None	forward	right	-0.5
22	forward	green	None	forward	left	-1
23	forward	red	right	None	forward	-1
24	left	red	right	None	right	-0.5
34	forward	green	left	None	left	-0.5
36	right	red	None	forward	forward	-1
39	forward	red	right	None	left	-1
41	left	green	forward	None	forward	-0.5
44	forward	green	None	forward	right	-0.5
49	forward	red	None	forward	left	-1
49	forward	red	None	forward	right	-0.5
54	forward	red	right	None	right	-0.5
61	left	red	left	None	right	-0.5
67	forward	red	None	forward	forward	-1