Tuning online algorithms for autonomous agents

Marius Seritan
LinkedIn
Mountain View, CA 94043
`mseritan@linkedin.com`

Abstract

We[1] use existing open source tools to build an autonomous agent, more specifically a Roomba robot tasked with exiting a room with a known map. We start by describing the project, introduce the POMDP formalism and explain how it is applied to this particular case. We use a set of open source libraries for the implementation and then evaluate different hyper parameters. We show that it is important to validate the reward function definition against the real life outcomes of the problem.

1 Introduction

Building intelligent agents (Russell and Norvig [2003]) is one of the major goals of Artificial Intelligence. The agent performs autonomously a task in a given environment, by taking actions and observing changes through its sensors. The agent relies on a policy indicating which actions should be taken next. Since the state of the system is not know the planning process is challenging and can be addressed using the principled mathematical framework Partially observable Markov decision processes (POMDP) ((Littman et al. [1995])).

In this paper we evaluate ways to provide such policy for a Roomba robot. The robot can take two actions, rotate and move forward, and is equipped with either lidar or bump sensors. The robot is positioned in a room defined by walls, an exit and stairs. The map of the room is known, however the exact position and orientation are unknown. The robot must exit the room while minimizing the amount of motion and contact with the walls and of course not falling down the stairs. We are focussing on using only the bump sensor.

2 Modeling the Roomba as a POMDP problem

We start by defining the underlying Markov Decision Process through the 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$. The state space $\mathcal{S}$ contains the position and orientation of the robot. This space is continuous and it is not directly observable. The action space $\mathcal{A}$ contains the rotation angle and forward speed. The transition model $\mathcal{T}$ is defined as the probability $\mathcal{T}(s' | \, s, a)$ of transitioning to state s' given that the system is in state s and we take action a. For our case the transition model $\mathcal{T}$ is deterministic since it is based on simple motion equations. The reward function $\mathcal{R}$ needs to be designed for this problem.

We cannot observe the state s so in order to specify the Partially Observable MDP we need to define $\mathcal{O}$ and $\mathcal{Z}$. $\mathcal{O}$ is observation space and $\mathcal{Z}(o | \, s, a, s')$ is the observation model, i.e. the probability of receiving observation o in state s' given that the previous was s and we took action a.

[1]One human and multiple computers

Table 1: Observed reward given various hyper parameters

Exit state Reward	Tree depth	Timestep limit	Mean reward	Success rate $*100$	Success reward	Failure rate $*100$	Failure reward
10	20	400	-2.56	46.1	-1.78	33.6	-3.55
100	20	400	0.71	44.6	5.79	30.0	-3.99
10	200	400	-2.54	39.4	-1.34	30.4	-3.99
10	200	4000	-2.58	61.9	-1.80	38.1	-3.85
Rewards for other states: stairs -10, time step -0.1, wall -1.							

3 Methods

We provide an implementations for the 7-tuple defining the POMDP, as described in Section 2. We start with the code provided to us as part of the class. We use the POMDPs.jl package (Egorov et al. [2017]) and additional code to implement a simulator for the Roomba agent, a belief implementation using Particle Filtering (Del Moral and Doucet [2009]) and other glue functions. One of the benefits of the POMDPs.jl library is that it provides integration to multiple solvers. For this problem we evaluate the `POMCPOW`(Sunberg and Kochenderfer [2017]) algorithm. `POMCPOW` is an online algorithm which provides quick feedback to help us increase our understanding of the problem. It also supports continuous states which makes it a good fit for our problem.

Typically planning algorithms are compared using the average total discounted reward achieved, for example in the paper introducing `DESPOT`(Somani et al. [2013]). Defining the reward function is a subjective process and we have devised an experiment to evaluate reward functions against the actual outcomes of different runs. We define successful outcomes the simulations that reach the exit and failed runs the one falling down the stairs. Comparing the ratios of these outcomes allows the practitioner to evaluate various reward functions and hyper parameters in a more objective manner.

The evaluation algorithm is the following:

1. For each hyper parameter change
   (a) generate an uniform grid of initial states
   (b) run a simulation from each state
   (c) generate a gif image with the trajectory of the robot for visual inspection
   (d) save a log line with the initial state, discounted reward, last state reward and length of the path

The code for this project is available here: `https://github.com/winding-lines/AA228FinalProject`.

We selected hyper parameters sequentially based on the results at the earlier step. For example we noticed that the reward for successful and failed runs were very close together and we tried to tune the reward for the exit state. We then increased the depth of the tree search. Since the number of "hang" states was relatively high we then increased the timestamp limit.

4 Results

Table 1 presents the experiments conducted, each grid evaluation takes a couple of days to run on available hardware. All of the result files are available on github, in the order of experiments `https://bit.ly/2RFyYOB`, `https://bit.ly/2RAsNdX`, `https://bit.ly/2Umlybv`, `https://bit.ly/2G1Rdfl`.

We find that the discounted reward for a particular simulation is a poor predictor of the actual outcome. Table 2 illustrates two representative paths. In the first run (id 266, Figure 1) Roomba successfully exits the room while the second run (id 267, Figure 2) falls down the stairs.

We increased the reward for the success state in order to increase the correlation between higher rewards and successful outcomes. By comparing rows 1 and 2 of Table 1 we can see that the

Table 2: Similar discounted rewards can have widely different outcomes

Id	X	Y	θ	Mean Reward	Last Reward	Path length	Outcome
266	0.460	2.340	1.571	-2.937	8.9	203	positive
267	0.460	2.340	3.142	-2.523	-11.1	88	negative

mean reward increases however the ratio of successful and failed experiences decrease slightly from $46\% \rightarrow 45\%$ and $34\% \rightarrow 30\%$.

Another finding is that increasing the search depth does not lead to improvements, see row 3 for a depth search of 200.

Lastly we notice that by allowing the simulation to complete (row 4, simulation limit of 4000) shows a marked increase in the success ratio $46\% \rightarrow 62\%$ a smaller increase in the failure ratio $34\% \rightarrow 38\%$.

5 Discussion

While we have been able to increase the ratio of positive outcomes our approach has some limitations. Notably, it was impossible to avoid hitting the stairs, specially when the robot started by pointing towards the stairs. This suggests that we may want to improve the sensors on the Roomba. Some low cost alternatives could be to eliminate the cost of hitting the stairs by adding an edge sensor or to reduce the uncertainty in the orientation by adding a magnetic based sensor.

We have achieved a near zero ratio of plans reaching the run limit. This is impressive given that our earlier attempts at building a manual policy lead mostly to stuck states.

By looking at the trajectory plots, for example Figure 3, we see that the robot seems to discover some patterns: goes in straight lines when possible, minimally bounces from the wall. In order to improve the run time a future direction is to try to express the action space using macro actions (Theocharous and Kaelbling [2004]).

6 Conclusions

We introduced a more objective method to compare policies for the Roomba project. Our method evaluates the overall quality of the policy by comparing the ratio of successful and failure outcomes. This method is an additional hierarchical measure on top of the reward function which is still the foundational measure behind the planning process. For example, the success of the reward based planning process can be seen in the minimal amount of contacts with the wall, a state with a reward of -1.

References

Pierre Del Moral and Arnaud Doucet. Particle methods: An introduction with applications. Research Report RR-6991, INRIA, 2009. URL `https://hal.inria.fr/inria-00403917`.

Maxim Egorov, Zachary N. Sunberg, Edward Balaban, Tim A. Wheeler, Jayesh K. Gupta, and Mykel J. Kochenderfer. POMDPs.jl: A framework for sequential decision making under uncertainty. *Journal of Machine Learning Research*, 18(26):1–5, 2017. URL `http://jmlr.org/papers/v18/16-300.html`.

Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In *International Conference on Machine Learning (ICML)*. Morgan Kaufmann, 1995. URL `http://people.csail.mit.edu/lpk/papers/ml95.ps`.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003. ISBN 0137903952.

Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee. Despot: Online pomdp planning with regularization. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *NIPS*, pages 1772–1780, 2013. URL `http://dblp.uni-trier.de/db/conf/nips/nips2013.html#SomaniYHL13`.

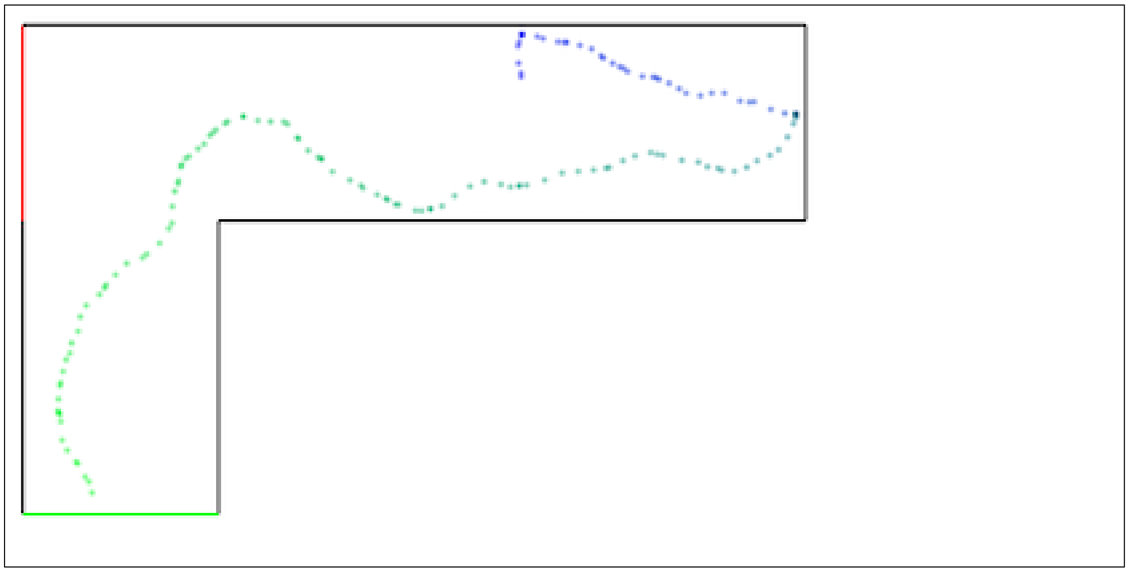

Figure 1: Run id 266, successful run with reward -2.937

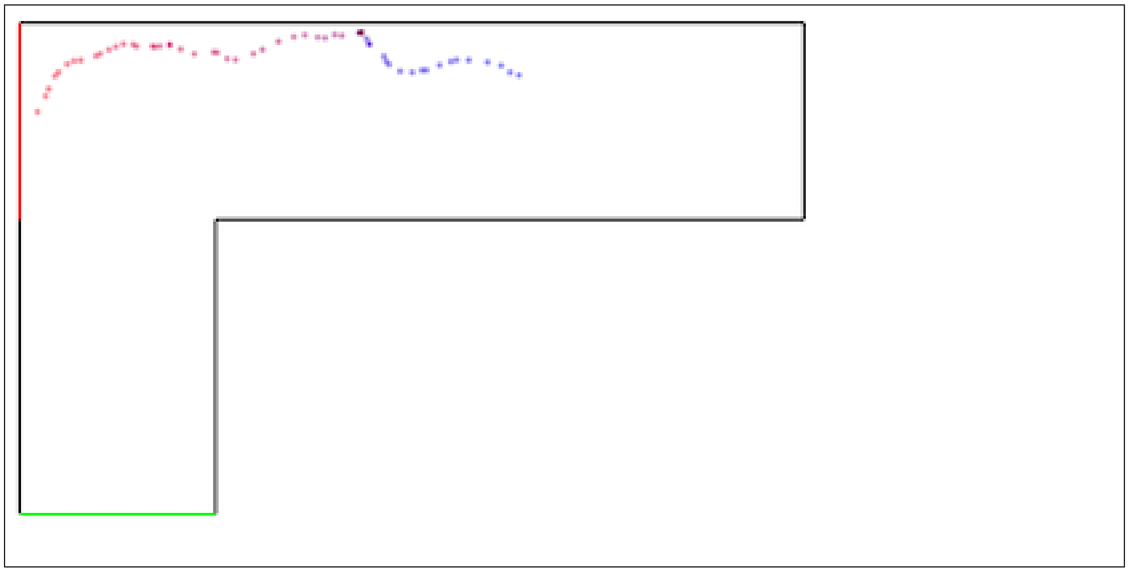

Figure 2: Run id 267, failed run with reward -2.523

Zachary Sunberg and Mykel J. Kochenderfer. POMCPOW: an online algorithm for pomdps with continuous state, action, and observation spaces. *CoRR*, abs/1709.06196, 2017. URL `http://arxiv.org/abs/1709.06196`.

Georgios Theocharous and Leslie Pack Kaelbling. Approximate planning in POMDPs with macro-actions. In *Advances in Neural Information Processing Systems 16 (NIPS03)*, 2004. URL `http://people.csail.mit.edu/lpk/papers/theochar-nips03.pdf`.

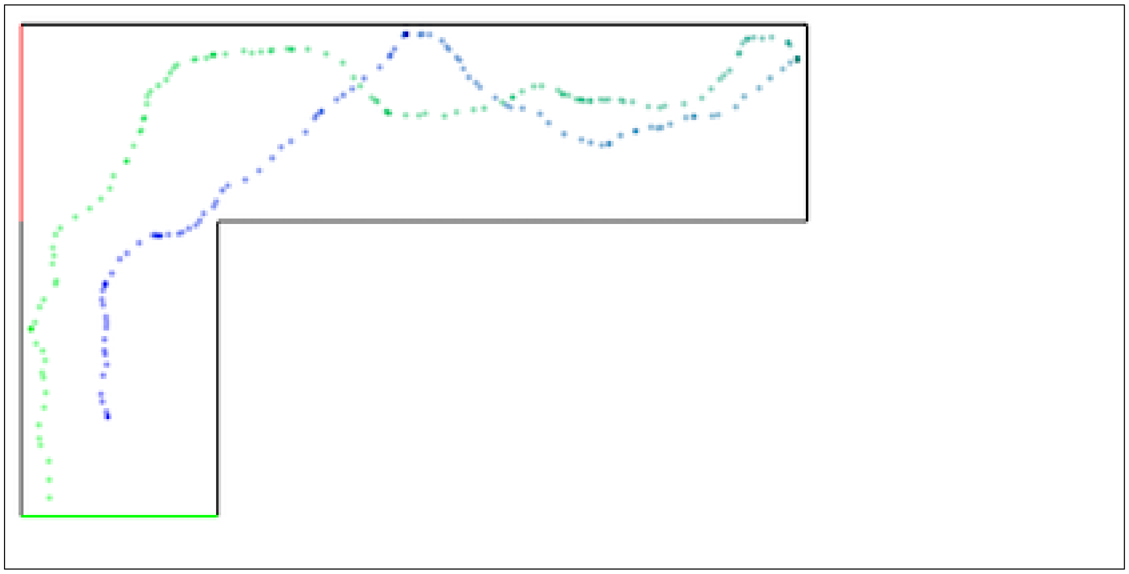

Figure 3: Macro/hierarchical actions learned at tree depth 200