Markov models in motion planning and sequential decision problems

Weifu Wang Belhal Karimi

1 Introduction

Here, I briefly note down some associations between MPD models and planning scenarios. Associated with the MDP model, I will also add some other probability-based models in the planning domain, hoping to link all together. We can make notes here, and find a common interesting topic that we can manage, and work towards our first collaboration.

One interesting paper: NeurIPS 2020: Multi-Robot collision avoidance under Uncertainty with Probabilistic Safety Barrier Certificates.

This is not directly a MDP model, but the probability settings, if can be associated with MDP, even with some interesting look-ahead policies / strategies, may provide better certificates or guarantees;

Motion planning: In an environment, given a start and goal, find a *path* that connects from start to goal. Challenge: what are legal path segments? Here, the concept of *control* is introduced. An agent applies control to move itself, but the control have some limits, so the agent may not always be able to move towards all directions at the same velocity from different configurations. Therefore, the planning problem, is also a sequential decision problem: at time-step i, what decision / control do I choose so that eventually, the agent can reach goal.

In many cases, MDP is used to model sequential decisions. Especially, when some states, i.e. the results of some actions / controls are not fully understood or observable, POMDP is used. The action forms a transition probability on states, but also depends on states, even though action set is limited, the result of the same action from different states can be different. Also, the number of states is usually unlimited, or to be more precisely, a discretization of the entire space (continuous).

Normally, the states of planning is the configuration, i.e. location and other parameters describing the status of the robot. However, in different applications, one can model states differently, making some problem more complex or more interesting. For example, in a situation with planning multiple robots, the state can be the relative relation among different robots. Then, the MDP states can be of a bounded set, rather than samples from a continuous space.

POMDP is also used when there's uncertainty, such as when the obstacles in the space are moving.

Often, sequential planning is often modeled as temporal logic as well. I believe such models can integrate with MDP naturally.

I will post papers on this link periodically, and if you find interesting models, please do the same, and message each other. If we have some thoughts or ideas, we can put them in the below section.

2 Ideas

2.1 Weifu

2.2 Belhal

- Data augmentation for Control via MDP
- Efficient latent states simulation in POMDP (how can we improve the simulation of unobserved states)
- Energy Based Models for Motion Planning

3 Notations and background

BK: so the goal is to output the vector $(u_1, \cdots, u_T)$ of controls for each timestep $t \in [1, T]$?

Weifu: In a sense, yes. We want the sequence of controls that can lead the robot to the goal. However, it is often complicated to get the exact controls, so usually the first step is to find sequence of intermediate configurations between start and goal, so that from the start, we can connect through the intermediate configurations to the goal.

Define the robot configuration as the minimum set of parameters needed to fully describe the robot's state. For example, given a car on the plane, we can describe its configuration as (x, y, θ), so that every point on the car can be described or computed based on a given (x, y, θ) value. Similarly, the configuration of a robot arm is usually described as a sequence of joint angles between adjacent links.

BK: Obstacles also come with their triplets (x, y, θ) to know where they are in advance?

Weifu: Obstacle descriptions are complicated, sometimes they are polyhedrons, with known geometries, or sometimes they are described as point clouds, or meshes.

BK: Ok, if it's that complicated, we should formalize the state transition probability conditioned on the obstacles. Looking at what this object looks like will be useful. Is there any references where this has been formalized?

- Challenge 1: collision detection. Since the obstacles are not always the same, in order to know whether a control or a configuration of a robot is valid or not, we need to perform collision detection, to make sure whatever action we choose to use or configuration we choose to move to is valid. **Collision detection** is a necessary subroutine in robot motion planning.

- Challenge 2: Local planner. It is usually not straight forward to compute what control a robot needs to perform to reach a nearby configuration even without obstacles, so we need to rely on *local planners*, which is another subroutine that computes the controls needed or way-points needed to pass to reach the goal configuration. In extreme case, we cannot even have local planner to compute how to move to a given configuration. What we have is called *steering method*, only able of simulating where the robot can be after apply a given control for a small duration.

MDP and POMDP modeling: Given an environment, a robot, and a start and goal pair, find a sequence of actions / transitions that lead to the goal.

BK: That is where we can use Energy-Based Modeling. Learning the transitions that lead to the desired goal can be seen as a probability distribution learning problem. We should start by checking what methods are used to learn those transitions: MPII (Model Predictive Path Integral), Maximum Entropy?

Weifu: It is true, we can model it like that. Question is, the environment may change after a task is performed. So, a long learning process is usually not acceptable, the whole planning should be done in a relatively short time, unless the distribution can be reused in different environments, with the fast updates. So, can you let me know usually how long will these methods learn the distribution for a task like this? Some pre-computation is ok in planning, but usually not too heavy.

BK: At each state, hence at each model update θ, the learning of EBM can take few MCMC transitions (can be small) and a single gradient step to update the model parameter.

In rare cases, if we are not sure if we will collide with the obstacles, we have an partially observed model, POMDP, simulating uncertainties of whether collision happens. Similar arguments can be made with moving obstacles, or from an unknown environment with unknown obstacles.

Motion planning Problem description: Let there be an environment of $\mathbb{R}^d$ where $d = 2$ or 3; obstacles $\mathcal{O}_i \in \mathbb{R}^d$, $i = \{1, 2, \ldots, n\}$, a robot of geometry $\mathcal{B}$, denote the configuration of the robot as $q \in \mathbb{R}^n$, where n is the number of Degree of Freedom (DoF) of the robot. Let the robot has controls $u \in \mathcal{U}$, and let there be start $s \in \mathbb{R}^n$ and goal $g \in \mathbb{R}^n$; find a sequence of robot configurations or a sequence of controls so that the robot can go from s to g. We can assume there is an oracle function $\mathcal{F} : (q \times \cup\mathcal{O}) \to \{0, 1\}$ that can return collision detection result in a given environment for any given configuration of the robot.

One common approach is called sampling based motion planning, which is achieved through placing samples in $\mathcal{R}^n$, the configuration space, and retain valid non-collision samples in the configuration space by inquiring $\mathcal{F}$. The invalid samples are discarded (or retained in some cases), and the valid samples are connected if the path connecting the configurations is valid (pass the validity check after inquiring $\mathcal{F}$). A good set of samples will lead to solutions much faster compared to random samples, though may not always be optimal.

If we want to use a learning method to create the samples, with bias, we may be able to find paths faster.

3.1 Model

Let us consider a state space $\mathcal{S} = \mathbb{R}^n$ which represents the configuration of the robot, and action space $\mathcal{A} = \mathbb{R}^m$ which represent the actions one robot can take to move between different states, and a reward function $\mathcal{R} : \mathbb{R}^n \to \mathbb{R}$, a transition function $T : \mathcal{S} \to \mathcal{S} \times \mathcal{A}$.

- Find a better suitable accumulated reward function $\mathcal{R}_\rfloor$ that represents the (approximately correct) real cost to goal;
- Find a policy π such that starting from the state s_0, one can find an approximate *shortest path* to goal s_g; or report no path with high confidence;

Here, the policy π, especially the transition probabilities can be learned, as a sampling process. The action can be long or short, as different resolutions, maybe actions of different resolutions can be sampled at different stages to help with the exploration and exploitation, to first search for the goal, and then improve the path qualities.

On the other hand, the reward function for each state can represent a cost to goal, and is some sort of heuristic, often used in A* algorithms. A good heuristic can be very helpful in the search or exploration. Assuming a continuous reward function over the state space, can we quickly find the appropriate cost-to-goal heuristics?

Here is another problem, that may also be suitable for MDP and learning. The problem is the planning of tensegrity robot, one example is the model shown in NASA's SuperBallBot, with six bars, and 24 cables. This robot can move by changing the length of the cables (which would result in the change of conter of mass (COM)), and topples onto a different supporting face. The problem of controlling the given robot is still a challenge. In particular, for this SuperBallBot

Given a state space $\mathcal{S} = \mathbb{R}^3 6$ which represents the location of end-points for the six-bars, and action space $\mathcal{A} = \mathbb{R}^2 4$ which represents the cable lengths (or the change of cable lengths), a transition function $T : \mathcal{S} \to \mathcal{S} \times \mathcal{A}$, and a reward function $\mathcal{R} : \mathbb{R}^2 4 \to \mathbb{R}$, representing how good is a configuration is at moving the robot towards a specific direction / location. We would like to find a policy π so that the robot can follow to move to the desired direction / location with most confidence. The advantage of this problem is that, the number of policies are limited, and can be reused, compared to the more generic settings above. We can learn the policy using EBM, and utilize it for any planning task, as a lower level planner.

4 Related papers