Research Survey for Robot Learning

Chao Ni

October 2018

Summary of the Survey

In this survey, I want to summarize approaches in robot navigation and robot learning. The survey mainly introduced Markov Decision Process based methods including some variants including partially observable Markov Decision Process, Bayesian learning based POMDP, model predictive control approach and KL divergence upper bound based POMDP. Besides applying other cutting-edge machine learning and deep learning methods, we can also make use of other control methods such model predictive control in robot learning problems.

Markov Decision Process

The robot has to remember something about its history of actions and observations and use this information, together with its knowledge of the underlying dynamics of the world including the given map, to maintain an estimate of its location. Markov Decision is a discrete-time stochastic control process and can serve as a mathematical framework in situations where outcomes are partly random and partly under the control of the decision maker. The decision maker's decision can result in the change of its surrounding environments. Robotics is such a field where the robot is involved in the environment. In Markov Decision Process, an agent will interact with the world directly. The agent takes the state of the world as its input, and generate action as its output, which will affect the state of the outside world. One seemly strong assumption is that the state of the world and the robot itself is known all the time. This is the main difference with Partially Observable Markov Decision Process, and we will address it later. A Markov decision process can be described as a tuple $< S,A,T,R >$ [1], where:

- S is a finite set of states of the world;
- A is a finite set of actions;
- T is the state-transition function, mapping from $S\times A$ to the policy $\Pi(s)$. and we can write as *T(s,a,s')*, where s' is the resulting state after the previous state s and the action a. The s' and s are essentially belonging to the same set.
- R is a reward function, mapping from S and A to a real value. It measures the immediate reward after the action.

The goal is to minimize the expected sum of rewards in the next k steps. In some cases, we will consider infinite steps with a discount rate γ. In the finite step cases, the action should not be independent of the time step. We denote $V_{\pi,t}(s)$ as the expected reward obtained from the state s and taking the policy π for the t^{th} step. Then the expected reward can be described as:

$$v_{\pi,t}(s) = R(s,\pi_t(s)) + \gamma \sum_{s' \in S} T(s,\pi_t(s),s') V_{\pi,t-1}(s') \tag{1}$$

In many cases, we need to find the optimal policy given a value function. Strictly, it only makes sense to do it in infinite-horizon case. A whole sequence of value functions are needed to derive the policy for the finite horizon. We can apply a greedy strategy to obtain the policy in both two cases:

$$\pi_t(s) = \arg\max_a (R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_{\pi_{t-1},t-1}(s') \tag{2}$$

Partial Observable Markov Decision Process

For MDP problems, we assume the current state of the world and the agent are known to the agent. However, what if the agent cannot determine the current state with full confidence? One possible approach is to choose the action with the highest possibility, but this means the agent will do the same thing as it encounters a similar location. It seems not a convincing approach. We can add some randomness to the agents' action, which means, the policy for the agent can be a distribution over actions mapping from observations. Therefore, if we are going to provide a probability distribution over actions, we need to memorize previous actions as well as observations. Apart from the tuple $< S,A,T,R >$ described above, we need to introduce two more elements:

- Ω is a finite set of observations the agent can experience;
- O: $S \times A \rightarrow \Pi(\Omega)$ is the observation function, which gives a possibility distribution over possible observations. we can write as $O(s',a,o)$ for the probability of observation o, where s' is the resulting state and a is the action.

we can divide the problem of controlling a POMDP into two parts; the agent has to make observations and generate actions, as shown in Fig. 1. The agent has an internal belief b, which represents the agents' experience. Based on the belief, the agent will generate action. The belief is a function of the previous belief, observation, and past action.

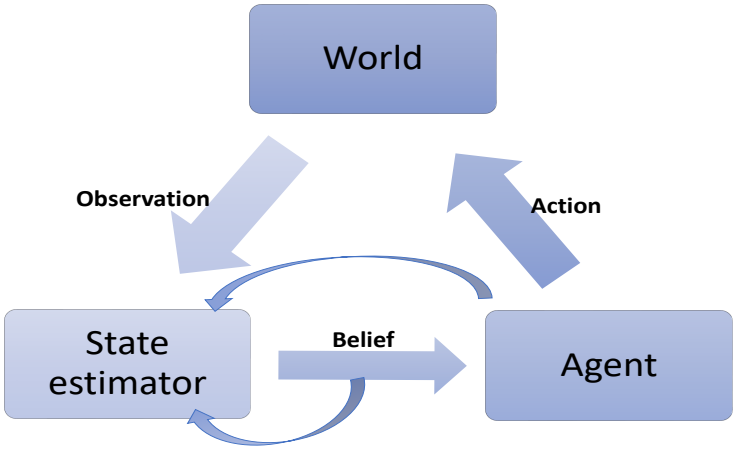

Figure 1: The process of POMDP: the state estimator is introduced, it makes use of the information of previous actions and belief, as well as the observation at the current time step.

0.1 Belief formulation

Just like the observation, which is not a deterministic variable, the belief is also a probability distribution over states. The distribution reflects the subjection about making decisions, which makes the robots more anthropomorphic. What's more, through the formation of belief, we can imagine a belief with possibility distribution comprises information about agent's historical experiences.

As a belief is a probability distribution over state S, we let b(s) represent the probability of world state s in agent's belief. Then they have to meet the requirement:

$$\sum_{s \in S} b(s) = 1 \tag{3}$$

where all probabilities are no less than 0 and no bigger than 1. The new belief b' then must be computed by the state estimator (SE), given an old belief b, an action a and an observation o. The new belief b' can be expressed as follows:

$$\begin{aligned} b'(s') &= P(s'|a,b,o) \\ &= \frac{P(o|s',a,b)P(s'|a,b)}{P(o|a,b)} \\ &= \frac{P(o|s',a,b)\sum_{s\in S}P(s'|a,b,s)P(s|a,b)}{P(o|a,b)} \\ &= \frac{O(s',a,o)\sum_{s\in S}T(s,a,s')b(s)}{P(o|a,b)} \end{aligned} \tag{4}$$

Notice that the observation o is not relevant to the observation and the old belief is not relevant to action to be taken in the next step.

0.2 Policy formulation

If the agent only has one step to go, then the decision is easy. The agent has to maximize the reward it will obtain. If the agent has two steps to go, it can take an action, and then make an observation, then take another action. In general, if the agent has t steps to go, then the whole process can be drawn like a tree. The initial step can be seen as a parent, and it has k observations (k sons). Each son will have an action, based on the observation and previous information, and each son will have k grandsons, too. The expected reward for the final step (a single step) is:

$$V_p(s) = R(s,a(p)) \tag{5}$$

The p denotes the policy tree node. A more general expression for a t step tree is that:

$$V_p(s) = R(s,a(p)) + \gamma \sum_{s'\in S} T(s,a,s') \sum_{o_i\in\Omega} O(s',a(p),o_i)V_{o_i(p)}(s') \tag{6}$$

Because the agent would never know the exact state of the world, it is more meaningful to take an expectation over all possible states:

$$V_p(b) = \sum_{s\in S} b(s)V_p(s) \tag{7}$$

So the question becomes to find the policy tree that the reward will be maximized. One useful approach to solving this problem is the witness algorithm, whose main idea is to choose an action among all possible actions to maximize the reward. If we denote $V_p(b)$ as the optimal reward function at the t step tree and denote $Q_p^a(b)$ as the reward after taking an action in belief b and continuing optimally for t-1 steps, then we will have following formula:

$$Q_t^a(b) = \sum_{s\in S} b(s)R(s,a) + \gamma \sum_{o\in\Omega} P(o|a,b)V_{t-1}(b'_0) \tag{8}$$

where b'_0 is the belief state resulting from taking action a and observation o from belief b. Thus we have:

$$b'_0 = SE(a,b,o) \tag{9}$$

The next step is to find a best action a among all possible actions, which means:

$$V_t(b) = \max_a Q_t^a(b) \tag{10}$$

Like in the Markov Decision Process case, we need to generate action policies given value functions.

$$a_t(b) = \arg\max_a \left(\sum_{s\in S} b(s)R(s,a) + \gamma \sum_{o\in\Omega} P(o|a,b)V_{t-1}(b'_0)\right) \tag{11}$$

0.3 Combine Bayesian learning and POMDP

This methodology bridges the machine learning and POMDP in that it introduces the collision probability and divides the expected reward into two parts: one is relative to the collision, and the other part is not. However, to accurately model the collision distribution over real world is super difficult, and this method uses machine learning to gain the knowledge of collision distribution from training data, which implicitly captures the relevant traits of our training environment rather than to model the map distribution directly.

One major missing element of the POMDP is that it cannot provide an initial belief over the world states. Here we will use the distribution of probability over the collision and update it using Bayesian learning. We first rewrite (11) and (6) as:

$$\begin{aligned} a_t(b) &= \arg\max_a (\sum_{s\in S} b(s)R(s,a) + \gamma \sum_{s\in S} b(s) \sum_{s'\in S} T(s,a,s') \sum_{o_i\in\Omega} O(s',a,o')V_{o_i(p)}(s')) \\ &= \arg\max_a (\sum_{s\in S} b(s)R(s,a) + \gamma \sum_{o\in\Omega} P(o|a,b)V_{t-1}(b'_0)) \\ &= \arg\max_a (\sum_{s\in S} b(s)R(s,a) + \gamma \sum_{s'\in S} P(s'|b,a)V_{t-1}(s')) \end{aligned} \tag{12}$$

The reward term $V_t(s)$ or $V_t(b)$ (they are essentially the same as can be seen from (7)) can be divided into two parts: one can be seen as a function of time spent to describe the efficiency of the robot, and the other one is a collision check function which describes the safety of the current action. [4]

$$V(s) = V^*(s) + J_c \times ICS(s) \tag{13}$$

where $ICS(s)$ equals 1 if the collision is inevitable.

Therefore, the action policy can be rewritten as:

$$a_t(b) = \arg\max_a (\sum_{s\in S} b(s)R(s,a) + \gamma (\sum_{s'\in S} P(s'|b,a)V^*_{t-1}(s') + \sum_{s'\in S} P(s'|b,a)J_c \times ICS(s))) \tag{14}$$

For the reward regarding to time spent, we can use a simple heuristic function to approximate it:

$$h(a_t,b_t) \approx \sum_{s'\in S} P(s'|b,a)V^*_{t-1}(s') \tag{15}$$

and the term regarding to collision is approximated by machine learning methods:

$$f(\phi(a_t,b_t)) \approx \sum_{s'\in S} P(s'|b,a)J_c \times ICS(s) \tag{16}$$

To predict collision probabilities, we need to collect training data $D = (\phi_1,y_1),...,(\phi_N,y_N)$ Here we introduce the y_i as the resulting binary collision state('collision','non-collision'). The goal is to learn the function $f(\phi)$. The update process is based on a Bayesian inference model.

0.4 Model predictive control approach

The model predictive control problems can be formulated as minimizing the sum of expected loss in infinite horizons.

$$\begin{aligned} J^*_{0\to\infty} &= \min_{u_0,u_1,\cdots} \sum_{k=0}^{\infty} h(x_k,u_k) \\ s.t. \quad x_{k+1} &= f(x_k,u_k), \forall k \geq 0 \\ x_0 &= x_S, \\ x_k &\in X, u_k \in U, \forall k \geq 0 \end{aligned} \tag{17}$$

where x and u are system state and input, respectively. f is the update process which represents the dynamic of the system. X and U are state space and input space. x_S is the initial state and $h(x_k,u_k)$ is the corresponding loss at the state x_k after an action u_k is made. However, basing actions on the model predictions introduces issues with robustness due

to the fact that the model is inherently inaccurate, and thus predictions further in the future are more uncertain. So in practice, we will not deal with it in infinite cases, but compute the loss within certain steps.

$$\begin{aligned} J^*_{t\to t+N} &= \min_{u_t,\ldots,u_{t+N-1}} \sum_{k=t}^{t+N-1} h(x_k,u_k)+V(x_N) \\ s.t. \quad x_{k+1} &= f(x_k,u_k), \forall k \in [t,t+1,\ldots,t+N-1] \\ x_0 &= x_S, \\ x_k &\in X, u_k \in U, \forall k \in [t,t+1,\ldots,t+N-1] \end{aligned} \tag{18}$$

The function $V(x_N)$ represents the expectation of future rewards after step N. We call this as the value function. Initially, the value function has lots of uncertainty, but as step increases, the uncertainty will decrease. We propose that when the uncertainty is large, we use the action policy obtained from model predictive control framework, but when uncertainty reaches below a predefined threshold, we utilize Markov Decision Process solutions regarding the value function. Thus the problem of the unknown initial probability distribution can be solved.[3]

The optimization problems in model predictive control can be solved using general solution techniques such as linear programming, quadratic programming, etc. The solution to this problem is a sequence of actions. We perform the first action and update the current state. Then move the current step to the next, and repeat the previous steps.

The value function contains information about the model and the world implicitly. The recursive relationship regarding the value function can be described as:

$$V^*(x_{k_0}) = \sum_{u\in U} P(u|x_{k_0}) \times \left(h(x_{k_0},u) + \gamma \sum_{x'} P(x'|x_k,a) V(x') \right) \tag{19}$$

where $P(u|x_{k_0})$ denotes the probability that the policy select the input/action u in state x_{k_0}.

0.5 PAC-Bayes Control

One assume in the previous analysis is that we have enough data to train our model and obtain a validate belief. However, in the robotics field, the dataset is quite small compared to other machine learning targets, like picture classification. So the problem is how to build a controller which generalize well on novel environments. We let ε denotes the possible environments which can be observed. We assume that there is a distribution D over all possible environments which cannot be observed. In this method, the unknown feature of D is emphasized because we focus on the controller dealing with unknown novel environments.

The data set from E consists of our training data, and we will figure out the difference and relationship with the parameter derived from the dataset and those from the unknown environment D. The technique used in this process is KL divergence in PAC-Bayes framework.

0.5.1 KL divergence

In mathematical statistics, the Kullback-Leibler (KL) divergence is a measure of how one probability distribution is different from a second, reference probability distribution. Given two discrete distribution P and Q defined on a common set, the KL divergence between P and Q is:

$$KL(P||Q) = \sum_i P[i] log(\frac{P[i]}{Q[i]}) \tag{20}$$

For scalars p,q$\in$ [0,1], the KL divergence can be defined as:

$$KL(p||q) = KL(B(p)||B(q)) = p\log\frac{p}{q} + (1-p)\log\frac{1-p}{1-q} \tag{21}$$

where B(x) denotes the Bernoulli distribution on $0,1$ with parameter x. On this basis, we can compute a bound scalar q^* regarding to p with a KL divergence smaller than a given constant c.

$$q^* \le KL^{-1}(p||c) = sup\{q \in [0,1] | KL(p||q) \le c\} \tag{22}$$

0.5.2 PAC-Bayes controller formulation

We denote $l(h_w, z)$ as a loss function regarding to the given observed environment and the hypothesis function parameterized by w. We focus on the probability distribution of the agent belief over parameter w, there is a reward (loss) corresponding to this believed distribution denoted as $V_S(P)$

$$V_S(P) = \mathbb{E}_{z\in E}\mathbb{E}_{w\sim P} l(h_w; z) \tag{23}$$

and a loss corresponding to the true unknown distribution denoted as $V_D(P)$:

$$V_D(P) = \mathbb{E}_{z\in D}\mathbb{E}_{w\sim P} l(h_w; z) \tag{24}$$

We choose a prior distribution Then the PAC-bound can be shown as follows:

$$KL(V_S(P)||V_D(P)) \leq \frac{KL(P||P_0) + log(\frac{2\sqrt{N}}{\delta})}{N} \tag{25}$$

where N is the number of data samples in the training environment and δ is a positive scalar smaller then one, which denotes the probability. Therefore, we can obtain an upper bound for the loss under true distribution:

$$V_D(P) \leq KL^{-1}\left(V_S(P)||\frac{KL(P||P_0) + log(\frac{2\sqrt{N}}{\delta})}{N}\right) \tag{26}$$

Another KL divergence equation easier for optimization is:

$$V_D(P) \leq V_S(P) + \sqrt{\frac{KL(P||P_0) + log(\frac{2\sqrt{N}}{\delta})}{2N}} \tag{27}$$

and the second term can be regarded as a regularizer. We can choose the distribution of P by minimizing the right part of the inequality.[2]

References

[1] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

[2] A. Majumdar and M. Goldstein. PAC-Bayes Control: Synthesizing Controllers that Provably Generalize to Novel Environments. *ArXiv e-prints*, June 2018.

[3] Rudy R Negenborn, Bart De Schutter, Marco A Wiering, and Hans Hellendoorn. Learning-based model predictive control for markov decision processes. *Delft Center for Systems and Control Technical Report 04-021*, 2005.

[4] Charles Richter, William Vega-Brown, and Nicholas Roy. Bayesian learning for safe high-speed navigation in unknown environments. In *Robotics Research*, pages 325–341. Springer, 2018.