Learning Control in Robotics

Trajectory-Based Optimal Control Techniques

Saloni Shah - N15267450
(sds730@nyu.edu)

Abstract – Robots are slowly and steadily becoming part of our everyday lives. Sooner there will be a situation where our will start and end with the help of robots. They need to adapt the path of optimality to uncertainty changes among environment. Reinforcement learning (RL) enables a robot to autonomously discover an optimal behavior through trial-and-error interactions with its environment. This paper gives us the idea of the basics of the learning control with various techniques and different approaches to find the optimality of the complex robots. It also describes the comparison with the strategies taught during Optimal and Learning Control for Robotics by *professor Ludovic Righetti.*

Summary of the paper

By going through this, it will give us the more details to what was in the paper and what are the approaches they used for finding the optimal trajectory path.

What and How it was done in the paper?

- ***Basics of learning control***

The flow of the paper starts with the basics where they talk about the control policy π that maps continuous state vector of the controlled system in time t to continuous control action given by:

$$u = \pi (x, t, \theta)$$

which further can be written as non-linear dynamics function followed by the observation equations

$$\dot{x} = f(x, u, t, ex)$$
$$y = h(x, u, t, ey)$$

where ex and ey are the noise. However, the control policy can be proceeded in 2 ways:

1. model-based learning, indirect learning, or internal model learning
2. model-free learning or direct learning

It is useful to have an idea related to the class of task. The approaches to robot learning can be categorized using three dimensions: direct versus indirect control, the learning method used, and the class of tasks in question (Fig1).

- ***Approaches to Robot Learning(three)***

<u>*Learning Internal Models for Control*</u>

The most well-known are kinematics and dynamic models. For instance, the forward direct kinematics of a robot relates joint variables q to end-effector variables y, i.e., y = g(q) but we

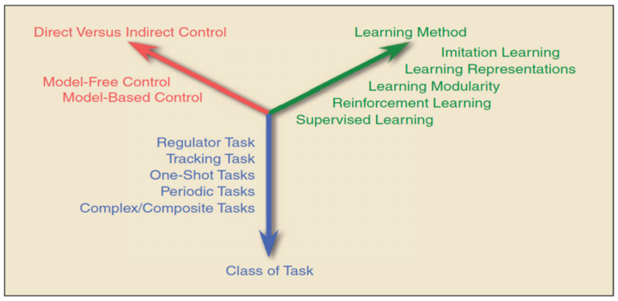

Figure 1. *Classification of robot learning along three dimensions. Topics further out on the arrows can be considered more complex research topics than topics closer to the center.*

need the inverse kinematics $q = g^{-1}(y)$ or the inverse dynamics $u = f^{-1}(q, \dot{q}, t)$. Thus, nonlinear function approximation is needed to learn internal models. One way is with piecewise linear models. It also talks about new approach; Gaussian process regression (GPR) to function where it requires an iterative optimization that needs to invert a matrix of size (N x N); N is the no. of training data points, GPR quickly saturates the computational resources with moderately many data points.

But any arbitrary nonlinear function approximation method to the inverse model problem can lead to unpredictably bad performance. So, valid approach is linearization of a forward model and learning an inverse mapping within the local region of the forward model. From the given data a differential forward kinematics model $\dot{y} = J(q)\dot{q}$ was learned, where J is the Jacobian matrix. The transformation from $\dot{q}$ to $\dot{y}$ can be assumed to be locally linear at a particular configuration q of the robot arm. BLWR (Bayesian locally weighted regression) is used to learn the forward model in a piecewise linear fashion and the goal of the robot task is to track a desired trajectory $(y, \dot{y})$ specified only in terms of x, z Cartesian positions and velocities.(Fig2)shows performance of learned inverse model.

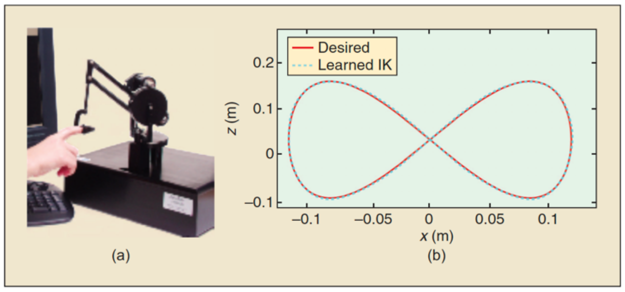

Figure 2. *(a) Phantom robot. (b) Learned-inverse kinematics solution; the difference between the actual and desired trajectory is small.*

<u>*Model-Based Learning*</u>

In this they talked about conventional dynamic programming and focuses on offline planning of the nonlinear policies required for control problems with continuous states and actions, deterministic time invariant discrete time dynamics, $X_{k+1} = f(X_k, U_k)$, and a time-invariant one-step cost or reward function L (x, u). It approximates the value function V(x) repeatedly solving the Bellman equation $V(x) = \min_u \{L(x, u) + V(f(x, u))\}$. Some type of interpolation is used which leads to curse of dimensionality. One way to avoid it by *Decomposing Problem;* performing dynamic programming on the sagittal system and produce a value function $V_s(x_s)$ and same with the lateral system $V_l(x_l)$ and minimizing L ((x, u) + V (f (x, u)) w.r.t u, with V(x) estimated by $V_s(x_s) + V_l(x_l)$. This ignores the linking of the two systems and improved by adding elements for each sub-system. Another way is by *Trajectory Optimization and Trajectory Libraries*; it finds local optima if we do interpolation which can be done by using a neighboring trajectory as the initial trajectory that can be done in

1) use the optimized action corresponding to the nearest state in the library at each time state
2) store the derivative of the optimized action w.r.t state
3) search states from numerous trajectories and generate a weighted blend of the suggested actions.

In the e.g. (Fig3) mentioned the implementing of the DDP wasn't successful and thus implemented direct reinforcement learning where controller must try to minimize total cost of trajectory. The one-step cost function for this e.g. is a weighted sum of the squared position errors and the squared torques: L (x, u) = $0.1\theta^2 T + \tau^2 T$ where 0.1 weights the position error relative to the torque penalty and T is time step of the simulation (0.01s).

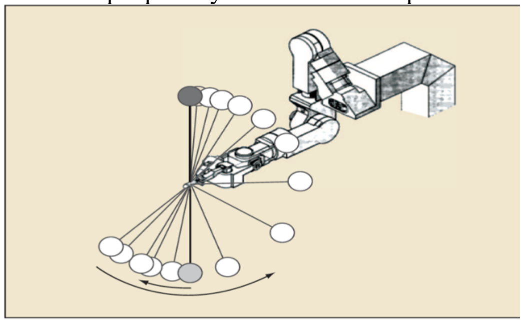

Figure 3. *The robot swinging up an inverted pendulum.*

We use one link pendulum swing up as an e.g. in (Fig4). Due to the dynamics and cost function are time invariant, there is a steady state control law and value function in fig.

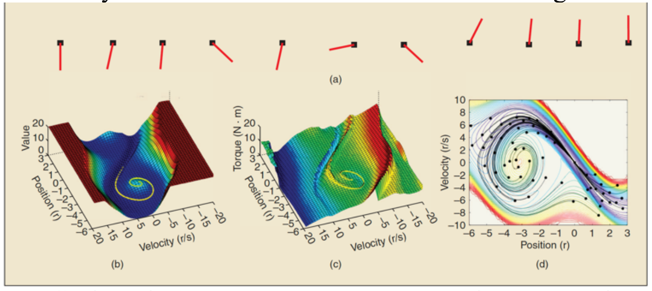

Figure 4. *(a) Configurations from the simulated one link pendulum optimal trajectory every half second and at the end of the trajectory. (b) Value function for one-link example. (c) Policy for one-link example. (d) Trajectory-based approach: random states (dots) and trajectories (black lines) used to plan one-link swing-up, superimposed on a contour map of the value function [33].*

__Model-Free Learning__

Value function-based methods are estimated with actor-critic methods, temporal difference (TD) learning, and Q-learning instead of DP and works well Instead V(x) we will use Q (x, u) defined as:

$$Q(\mathbf{x},\mathbf{u}) = L(\mathbf{x}_0,\mathbf{u}_0) + \min_{\mathbf{u_k}} \sum_{k=1}^{\infty} L(\mathbf{x_k},\mathbf{u_k})$$

One more way is learning the control policy from trajectory rollouts; *Policy Gradient Methods*:

$$J(\mathbf{x}_0) = E_\tau\left\{\sum_{k=0}^{N}\gamma^k L(\mathbf{x}_k,\mathbf{u}_k)\right\},$$

where, expectation E {} is taken over all trajectories τ that start in state x_0. The agenda is to determine the motor commands u_k that optimize this cost function. The principle of policy gradient methods is to compute the gradient $\partial J / \partial\theta$ and optimize updates. A range of algorithms exist to compute the gradient namely Finite difference gradients, reinforce algorithm, second-order gradient method which is the fastest gradient-learning approaches. It can scale to high-dimensional state-action spaces, at the cost of finding only locally optimal control policies. Another is *Probabilistic Direct Policy Learning* which can be done in 2 ways (1) where RL was formulated as an expectation–maximization (EM) algorithm and to treat the reward L(x, u) as a pseudoprobability, i.e., strictly positive, and the integral of the reward has to result in a finite number which modifies the previous equation to:

$$\log J(\mathbf{x}) = \log \int_\tau p_\theta(\tau) R(\tau) d\tau, \text{ where } R(\tau) = \sum_{k=0}^{N}\gamma^k L(\mathbf{x}_k,\mathbf{u}_k).$$

(2) Bellman equations can be transformed into a path-integral estimation problem which is based on value functions and requires a model-based approach. E.g. (Fig5) application of path-integral RL to a robot-learning problem where it uses the forward progress as a reward and slightly penalized the squared acceleration of each DOF. This algorithm is very simple, and manual tuning only focused on generate a good cost function.

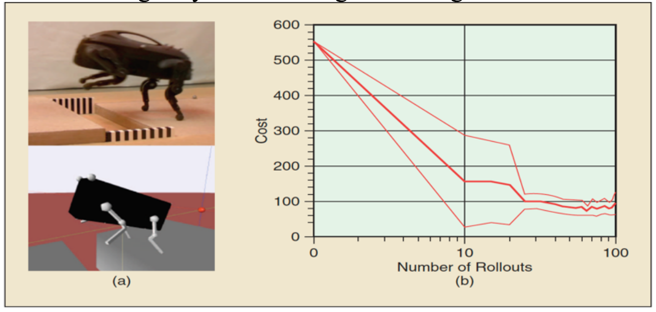

Figure 5. *(a) Actual and simulated robot dog. (b) Learning curve of optimizing the jump behavior with path-integral reinforcement learning.*

__Imitation Learning, Policy Parameterizations, and Inverse RL__

IL has become a popular topic to initialize and speed up robot learning. Finding useful parameterizations for control policies is a topic that is often discussed along with IL. Finally process of designing useful reward functions remains one of the most time-consuming. Thus, obtaining the reward function from observed behavior is a topic of great importance.

Why was it worth doing?

It tells us about the recent trend in the robot learning where trajectory-based optimal control techniques and RL to scale complex robotic systems. We can get more clear idea where they combined local models, and trajectory optimization to create an approach to practical dynamic programming for robot control problems. We will now be able to solve problems we couldn't solve before using tabular or global function approximation approaches. It also tells us more about how probabilistic RL methods and function approximation, have contributed to a steadily increasing interest in robot learning.

What are the results?

Starting from the beginning let's discuss the failure of the methods described in the paper; Gaussian process regression (GPR) to function approximation will require further research developments for scalability to continuous and real-time learning in complex robots. Therefore, with the further development of MBL: DDP, this dynamic method and one-step cost function are useful and have an accurate state estimation because it can generate a local model of the value function and strategy. The newer methods can optimize trajectories faster than DDP and that we can use a combination of methods to accomplish our goals. Parametric trajectory optimization based on sequential quadratic programming (SQP) overlooks work in aerospace and animation. We have used SQP methods to optimize trajectories and pass of DDP to produce local models of value functions and policies. Later in the last discussion, it talked about robustness which has not been addressed well in robot learning. Designing robust controllers with additive noise and performing stochastic dynamic programming does not capture the effect of correlated errors hence, creating a large increase in the number of states, which is not practical for dynamic programming. Finally, moving ahead with the last topic of discussion; MFL: Policy gradient methods can scale to high-dimensional state-action spaces, but it has a drawback that it requires manual tuning of gradient parameters, which can be tedious for which we use the probabilistic methods which is widely used.

Relation of the paper with algorithms in the class

As seen in the (Fig6), we can get the overall idea of the Methods used in the paper and what was done in the class. We know that for MBL once we learn the model, we can use it with optimal control techniques i.e. all the DP solutions under the global methods which include value and policy iterations and the trajectories which are all mentioned in the paper. Learning about the (SQP) and the robustness was new to us which was helpful as can be considered for future updates. But for MFL we cannot use OC directly and hence we need TD, Q-learning, and gradient. These were covered in the paper but dealing with the log-likelihood can be optimized with the EM algorithm was a new and useful thing to learn. We learned the policy gradient theorem using various approaches where we directly compute the policy without knowing the Q- or value functions, then policy parameterization during the lecture. The path integrals along with the derivations and studied the same e.g. as that of the paper for our learning purpose. Along with the Q-learning with a neural network. we went through Deep Q-network (DQN) and its approach towards finding the optimal which was very useful but was not discussed in the paper. It only covered Q-learning: off-policy TD control. Lastly, the paper just includes a synopsis of inverse RL; most reliable and convenient but it was fully taught and covered during the course along with the derivations and policies. But what was lit in the paper was imitation learning which is a new approach to learn and carry out further research.

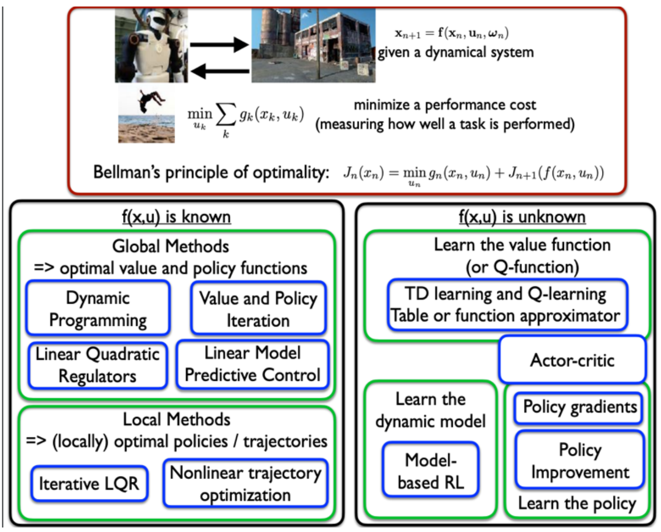
$x_{n+1} = f(x_n, u_n, \omega_n)$
given a dynamical system
$\min_{u_k} \sum_k g_k(x_k, u_k)$
minimize a performance cost
(measuring how well a task is performed)
Bellman's principle of optimality: $J_n(x_n) = \min_{u_n} g_n(x_n, u_n) + J_{n+1}(f(x_n, u_n))$
f(x,u) is known
Global Methods
=> optimal value and policy functions
Dynamic Programming
Value and Policy Iteration
Linear Quadratic Regulators
Linear Model Predictive Control
Local Methods
=> (locally) optimal policies / trajectories
Iterative LQR
Nonlinear trajectory optimization
f(x,u) is unknown
Learn the value function
(or Q-function)
TD learning and Q-learning
Table or function approximator
Actor-critic
Learn the dynamic model
Model-based RL
Policy gradients
Policy Improvement
Learn the policy

FIGURE 6: Overview of the approaches

Discussion on paper: Pros AND Cons/Future Work

The paper only gives us an idea about how we can perform trajectories based RL. The detailed explanation and algorithm of MBL and MFL policies should also be mentioned to make it better. Also, the limitation of the paper is it does not go into the detailing of initial learning and inverse RL, which is the latest methodology that one adapts to for learning control in the robotics for carrying out further research. In RL, use recurrent neural networks to reconstruct the states without measuring all the rules. More research can be done in the inverse RL combined with imitation learning to carry forward as a next step. Using inverse and forward OC for mapping human locomotion measurements to an autonomous generation of human-like locomotion on humanoid robots. Also, mentioning about Monte-Carlo Tree Search using the academic policies in the further research work would be helpful in finding the optimal path for robot.