EE 290-005 Integrated Perception, Learning, and Control **Spring 2021**

Lecture 6: Geometric Nonlinear Control with RL; Koopman Operator

Scribes: Tiffany Cappellari, Jay Monga, Valmik Prabhu

6.1 Combining Geometric Nonlinear Control with RL-Enabled Control (Tyler Westenbroek)

6.1.1 Geometric Nonlinear Control

Main Idea: Leverage underlying structure in your problem to design feedback controller

- Exploits the underlying structures in the system to systematically design feed-back controllers
- *Example*: By designing a Lyapunov function, etc, you can take a global problem like stabilization to a local problem (reducing the energy).
- Your controllers give you fine-grain control and are amenable to formal analysis.
- *Problem*: They require high-fidelity models and perform poorly under non-parametric uncertainty

6.1.2 Deep Reinforcement Learning

Main Idea: sample the system trajectories to approximately find the optimal feedback controller.

- Success in complex manipulation tasks
- By sampling the system trajectories, you're essentially "discovering" connections between 'local' and 'global' structures.
- *Problem*: Hard to define the reward that gets the system to do what you want. Need reward shaping and lots of data to work well.

6.1.3 Thesis Proposal and Research

High level overview of Tyler's work:

1. Start with nominal model based control and add RL based correction term

- Lyapunov stabilizing controller, feedback linearization, and other architectures for global control
- Local control assisted through a learned correction
- Main focus of this presentation

2. Develop correctness and safety guarantees for learning algorithms
3. Future questions about fundamental use of control
   - Explore necessary and sufficient conditions for these control strategies

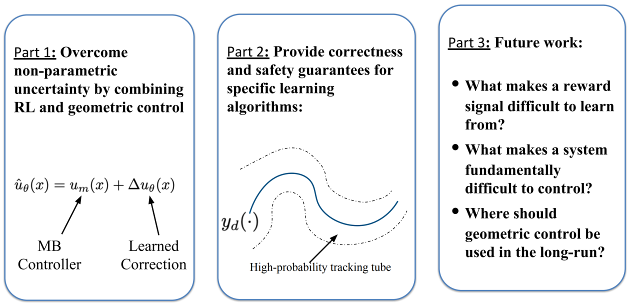
Part 1: Overcome non-parametric uncertainty by combining RL and geometric control
$\hat{u}_\theta(x) = u_m(x) + \Delta u_\theta(x)$
MB Controller
Learned Correction
Part 2: Provide correctness and safety guarantees for specific learning algorithms:
$y_d(\cdot)$
High-probability tracking tube
Part 3: Future work:
What makes a reward signal difficult to learn from?
What makes a system fundamentally difficult to control?
Where should geometric control be used in the long-run?

Figure 6.1: A diagram of the project flow.

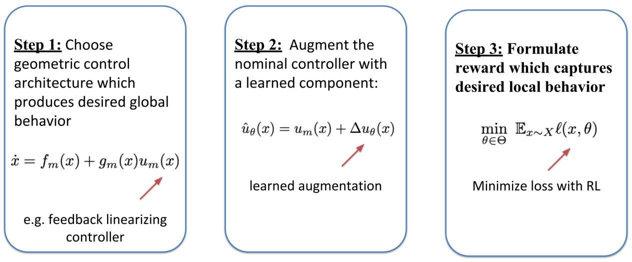
Step 1: Choose geometric control architecture which produces desired global behavior
$\dot{x} = f_m(x) + g_m(x)u_m(x)$
e.g. feedback linearizing controller
Step 2: Augment the nominal controller with a learned component:
$\hat{u}_\theta(x) = u_m(x) + \Delta u_\theta(x)$
learned augmentation
Step 3: Formulate reward which captures desired local behavior
$\min_{\theta \in \Theta} \mathbb{E}_{x \sim X} \ell(x, \theta)$
Minimize loss with RL

Figure 6.2: A diagram of the design process.

6.1.4 Combining Control Architectures and Model Based Learning

The idea is to start with some model of our system and augment it with a learning-based term to increase performance

1. Start with some geometric control architecture which has desired performance. Standard classical nonlinear control techniques

$$\dot{x} = f_m + g_m(x)u_m(x)$$

2. Augment the controller with a learned component to increase performance

$$\hat{u}_\theta(x) = u_m(x) + \Delta u_\theta(x)$$

3. Shape reward to penalize deviation from model (

$$\min_{\theta \in \Theta} \mathbb{E}_{x \sim X} l(x, \theta)$$

6.1.5 Feedback Linearization

The idea is to "invert" the nonlinearities out of our dynamics

- Choose a control input that makes the dynamics behave like a linear system. Once we have a linear model, we can leverage classical technique of linear control. Given dynamics of the form

  $$\dot{x} = f(x) + g(x)u$$

  and a set of outputs of the form

  $$y = h(x)$$

  We want to keep differentiating y until the input appears: Take γ derivatives of the output and "invert" the dynamics to get

  $$\begin{bmatrix} y_1^{(\gamma_1)} \\ \vdots \\ y_n^{(\gamma_n)} \end{bmatrix} = b(x) + A(x)u$$

  Then apply the control law

  $$u(x, v) = \frac{1}{A_\gamma(x)}[-b_\gamma(x) + v]$$

  which yields

  $$y^\gamma = v$$

  to get

  $$y^{(\gamma)} \triangleq \begin{bmatrix} y_1^{(\gamma_1)} \\ \vdots \\ y_n^{(\gamma_n)} \end{bmatrix} = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$$

- *Problems*: If you have model mismatch, you can't properly invert these dynamics. However, if your overall architectural assumptions hold (relative degree), there does exist a true feedback linearizing controller, so you can learn an error term between the nominal controller and this "true" linearizing controller. Then by penalizing deviations from the desired linear behavior, you can learn this error term. Basically, you're learning a controller to *make your system act like a linear system you know how to control*. In this case, (if you have an orthogonal basis representing your learned controller) you end up getting a strongly convex optimization problem. Using a neural network negates this guarantee, but it still works well in practice.

6.1.5.1 Directly Learning the Linearizing Controller

- We know the controller is going to take some specific form

$$u_p(x,v) = \beta_p(x) + \alpha_p(x)v$$

$$u_m(x,v) = \beta_m(x) + \alpha_m(x)v$$

- Mismatch in controllers due to mismatch in model creates a "gap" between the controllers

$$u_p(x,v) = [\beta_m(x) + \Delta\beta(x)] + [\alpha_m(x) + \Delta\alpha(x)]v$$

  To overcome this we approximate

$$u_p(x,v) \approx \hat{u}_\theta(x,v) = [\underbrace{\beta_m(x)}_{\text{Model Based Components}} + \underbrace{\beta_{\theta_1}(x)}_{\text{Learned Components}}] + [\underbrace{\alpha_m(x)}_{\text{Model Based Components}} + \underbrace{\alpha_\theta(x)}_{\text{Learned Components}}]v$$

- We will design a reward that penalizes deviations for our desired linear system behaviour

Assuming that that our learned parameters are of the form,

$$\beta_{\theta_1}(x) = \sum_{k=1}^{K_1} \theta_1^k \beta_k(x) \qquad \alpha_{\theta_2}(x) = \sum_{k=1}^{K_2} \theta_2^k \alpha_k(x)$$

where $\{\beta_k\}_{k=1}^{K_1}, \{\alpha_k\}_{k=1}^{K_2}$ are linearly independent sets of features, our optimization problem is actually strongly convex and has a unique solution. It's important to note that our learned parameters are continuous functions that we are approximating with some basis, and the choice of basis can determine how well that approximation is.

In practice, we will take the reward signal we generate, discretize it, and feed that into RL solvers optimized for these kinds of problems.

6.1.6 Results

This control methodology has been successfully used in several platforms, both hardware and simulation.

6.1.6.1 12D Quadcopter

- Highly nonlinear system controlled under a learned feedback linearized model
- Immense improvement in 1 hour of training, great improvement with relatively low amount of data/training compared to other RL based approaches

6.1.6.2 Augmented Lyapunov Controllers

- Design controller using control lyapunov function with learned parameters. Constrained optimization problem looks like

$$u^*(x) = \min_{u \in U} \mathbb{E}_{x\sim X} ||u||_2^2$$
$$\text{s.t. } \underbrace{\Delta V(x)[f_p(x) + g_p(x)\hat{u}_\theta(x)] + \sigma(x)}_{\Delta(x,\theta)} \leq 0, \forall x$$

but we will work with the unconstrained problem

$$\min_{\theta \in \Theta} \mathbb{E}_{x\sim X} [\, ||\hat{u}_\theta(x)||_2^2 + \lambda \underbrace{H(\Delta(x,\theta))}_{\text{penalty function}} \,]$$

Under the assumption of a controller linear in its parameters, this optimization problem is also strongly convex.

This technique has been successful in simulation environments. Highlighted in the presentation was achieving desired performance for control of a double pendulum and a stable walking gait on a bipedal robot with only ~ 20 seconds of training data.

6.1.7 Q&A

6.1.7.1 Comments from Professors

- Learning to improve feedback linearization ties into general role of deep learning in approximating dynamics of system
- How would performance work on real world walking robot?
  - Shouldn't expect 20 seconds of data to work
  - Richness of data available in simulation may also make things much easier
- We have many choices when choosing where to employ learning. It's very important to recognize where learning is employed, which Tyler explained well
  - Learn the entire model, or start with approximation

  - Learn deviation from model
  - Basis functions to be used in formulating model deviation

- Tyler asked about how to effectively handle learning in multiple stages as proposed
  - Professor Ma pushes for alternating approach
  - Low-dimensional structure of problem should salient
  - Should converge to global optimum under broad conditions
- Professor Sastry wanted to bring up robustness in control
  - Needs more tests on non-simulator environments
  - Geometry of systems can make control inherently complex
- Professor Ma brings up that we don't have a theory to describe computational complexity or sampling complexity to achieve desired performance

6.2 Feedback Linearization vs Koopman Operator

Presentation by Jason Choi, Fernando Castaneda Garcia-Rozas, Ayush Agarwal

6.2.1 Intro

6.2.1.1 Motivations:

- Find a linear representation of a nonlinear system, so we can use linear control techniques
  - Class of effective techniques for analysis and control of linear systems
  - Much more developed, established techniques vs nonlinear case
- Identify modes of a nonlinear system (natural/dominant frequencies). Remember, we can do this easily with linear systems by looking at the eigenvectors/eigenvalues.

6.2.1.2 A Simple Example:

Given the system:

$$\dot{x}_1 = \mu x_1$$
$$\dot{x}_2 = \lambda(x_2 - x_1^2)$$

Define the set of observables

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \end{bmatrix}$$

By setting these as the states (lifting the system to this higher dimensional space), we can get the following linear system

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \dot{y}_3 \end{bmatrix} = \begin{bmatrix} \mu & 0 & 0 \\ 0 & \lambda & -\lambda \\ 0 & 0 & 2\mu \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

This is great! Can we do this more generally?

6.2.1.3 Koopman Operator

Unfortunately, it's not always possible to find finite linear basis representations for arbitrary nonlinear functions. So the general theory here is that we are lifting a *finite* dimensional *nonlinear* system into an *infinite* dimensional *linear system.* The map between the original system and the lifted one is called the **Koopman Operator**.

Then we can take some finite subset of this infinite dimensional basis and get a linear approximation of the nonlinear dynamics. There are two ways to do this: *Dynamic Mode Decomposition* and finding *Koopman Eigenfunctions.*

6.2.1.4 Dynamic Mode Decomposition (DMD)

Dynamic Mode Decomposition essentially approximates the infinite-dimensional linear system with data using linear least squares. Just take a lot of data, each point with an arbitrarily high number of orthogonal observables X (often one million). Then calculate the A matrix $X^T X$. You then get a giant linear matrix describing the dynamics.

By examining the eigenvectors and their corresponding eigenvalues, you can observe the modes of the Koopman-linearized system. By including only those observables which correspond to high amplitude modes (have high-magnitude eigenvalues), you can get a smaller finite dimensional approximation which still preserves most of the dynamics.

6.2.1.5 Identifying Observables

- The other way to approximate systems is to find the "right" observables which efficiently capture the dynamics.
- *Example*:

$$\dot{x} = x^2$$

We can define some candidate observables:

$$y = \begin{bmatrix} x \\ x^2 \end{bmatrix}$$

The problem is that taking the dynamics gives you

$$\dot{y} = \begin{bmatrix} x^2 \\ 2x^3 \end{bmatrix} = \begin{bmatrix} y_2 \\ 2y_1y_2 \end{bmatrix}$$

Which leads you to expand the number of observables, which has the same problems, etc. Using this method, you need an infinite number of observables. However, if you use the function

$$y = e^{-\frac{1}{x}}$$

You get

$$\dot{y} = e^{-\frac{1}{x}} \frac{1}{x^2} \dot{x} = e^{-\frac{1}{x}} \frac{1}{x^2} x^2 = y$$

And this gives you a linear system in one dimension, *though your original state x is not in the set of observables.* This is an example of an eigenfunction.

6.2.1.6 Finding Eigenfunctions

An eigenfunction has two very useful properties

- Guarantees closure of update
- Linear system

Solving for such an eigenfunction requires solving some PDE and getting some eigenvalues, but in fact much better methods exist. Recent work has shown success with auto-encoder decoder neural network structure.

Given the discrete time dynamics

$$x_{k+1} = F_t(x_k)$$

we want to find an eigenfunction $\phi(x)$ such that

$$\lambda\phi(x) = \phi(F_t(x))$$

In continuous time the definition is

$$\frac{d}{dt}\phi(x) = \lambda\phi(x)$$

In continuous time, this is a partial differential equation. What you do here is parameterize the eigenfunction using a set of orthogonal basis functions (such as polynomials), and solve the PDE. This gets you *one* of the observables you'll be using to represent the system (the different solutions to the PDE are the different eigenfunctions).

6.2.1.7 Control Inputs

By lifting the system, you *sometimes* get a linear system. In this case, control is easy. But for arbitrary control-affine nonlinear systems, you get bilinear systems, which we still don't know how to control. Most people just linearize the bilinear system at that point to eliminate the bilinear term.

6.2.1.8 Key Takeaways

Koopman analysis has some main benefits

- Allows us to use linear systems techniques on nonlinear systems
- Lets us identify the dominant modes of the dynamics
- Reduction to finite dimensional system is still a linear approximation

6.2.2 Q&A

6.2.2.1 Comments from Professors

- Would be good to see under what conditions such approximations (finite-dimesional linear approximation) work
  - Still get guarantees on qualitative performance
  - See performance on chaotic systems
- Ties back into fundamental questions of data analysis - how to represent system with linear approximation of low rank. Professor Ma calls this a common punchline
- Sastry brings up the how classical linearization looks at embedding into the space of monomials but that will not generalize well enough; motivating departure from analytic functions and usage of neural networks.
- Yi thinks it would be nice to look into
  - What family of functions we allow for approximation?
  - How compact we allow our representation to be?
  - To what precision can original space be encoded?

6.2.2.2 Questions from Students

- Valmik: How does Koopman analysis to control a bilinear system compare to normal linearization of nonlinear system? Answer: See Advantages of Bilinear Koopman Realizations for the Modeling and Control of Systems with Unknown Dynamics Bruder et al

- Kshitij: Will our learned transformation ϕ always be perfectly reversable by ϕ^{-1}? Answer: No, there is no guarentee. In most systems, the eignefunctions do not perfectly recover the state. In EDMD, get around this by encoding state in observation but then we can lose invariance