EVALUATING MODEL FREE POLICY OPTIMIZATION STRATEGIES FOR NON LINEAR SYSTEMS

BY ADITYA H CHUKKA

A thesis submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Computer Science

Written under the direction of

Kostas E Bekris

and approved by

_____

_____

_____

New Brunswick, New Jersey

May, 2017

ABSTRACT OF THE THESIS

Evaluating Model Free Policy Optimization Strategies for Non Linear Systems

by Aditya H Chukka

Thesis Director: Kostas E Bekris

The **Iterative Linear Quadratic Regulator** (ILQR), a variant of **Differential Dynamic Programming** (DDP) is a tool for optimizing both open-loop trajectories and guiding feedback controllers using dynamics information that can be inferred from data. This technique assumes linear dynamics and quadratic cost functions and improves the control policy iteratively until convergence. We demonstrate the capabilities of this framework in designing controllers for regulating both natural and custom behavior on a simple pendulum, the primitive non linear system. The method's assumptions limit its validity to smaller regions of the state space. Direct Policy Search methods use **Reinforcement Learning** to develop controllers for such scenarios. Nevertheless, these methods require numerous samples to generate an optimal policy and often converge to poor local optima.

Guided Policy Search (GPS) is a new technique that optimizes complex non-linear policies, such as those represented through deep neural networks, without computing policy gradients in high dimensional parameter space. It trains the policy in a "supervised" fashion using numerous locally valid controllers produced by ILQR. GPS provides appealing improvement and convergence guarantees in simple convex and linear settings and bounds the error in a non-linear setting. We apply Guided Policy Search

to generate control policies for locomotion of a tensegrity robot, producing closed-loop motion that could not be achieved with previous methods.

Acknowledgements

First of all, I would like to thank my parents who supported me during all my studies, without whom I would not have reached this stage of my life. I'm also obliged to my family who were always there to assist me and my parents whenever needed.

I'm greatly indebted to my supervisor **Kostas E Bekris**, Ph.D. whose support was instrumental for this thesis and his valuable advices through out our journey. My gratitude goes also to my teachers at Rutgers and Indian Institute of Technology, Guwahati for their precious advices and help.

Special thanks to **David Surovik** for mentoring me and being there through out the project. I couldn't have completed the report and the presentation without your assistance. I would also like to thank **Zakary Littlefield** for helping in understanding and writing the software, **Thanasis Krontiris** for all the discussions we had and the great journey during the Port Authority project, **Andrew Dobson** for being patient at my constant blabbering, **Andrew Kimmel**, **Shaojun Zhu**, **Zacharias**, **Rahul Shome** and all the members of **PRACSYS** for the wonderful moments at the lab. This is one of my best journeys and I have lots of moments to cherish.

I'm also grateful to my friends at Rutgers University and Indian Institute of Technology, Guwahati who played a pivotal in my transition of a normal kid to an Engineer and a better human being.

I would like to use this opportunity to thank the Rutgers Support and Staff Members who were always there to assist me in my administrative work. My greetings go to every person I interacted with and had an opportunity to exchange my ideas.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Traditionally robots have been developed with the ability to perform repetitive tasks in restrained environments. More recently, the demand for robots that can execute non-trivial tasks and operate in generalized settings has increased. This lead to growth in complexity and dimensionality of the robots which pushed the need for advanced structures and algorithms. Tensegrity robots such as SUPERball, designed by NASA for planetary exploration [1], [2] belong to a class of high dimensional complex systems. They are light weight, impact tolerant systems made up of metallic rods joined by elastic cables that can be actuated to generate locomotion.

Several approaches have been formulated in the field of motion planning to handle such systems. Littlefield et al 2016 [3] have developed a motion planner known as Stable Sparse-RRT (SST) which can produce locomotion in SUPERball. However, such planners mostly serve as open-loop controllers that do not have a clear fall-back mechanism and are not robust. Exploring terrestrial objects often involves unforeseen terrains or rocky surfaces where the systems need a feedback mechanism to counteract deviations from the original trajectory. This accentuates the need for designing feedback controllers.

Classical approaches designed feedback controllers using the exact model of the system. Model-based approaches are shown to provide near optimal controllers using an accurate model of the system. However, systems like SUPERball with highly nonlinear dynamics and complex contact forces (ex:friction) are difficult to accurately express using a mathematical model. Using an inaccurate models of the system might lead to disastrous results incurring heavy losses.

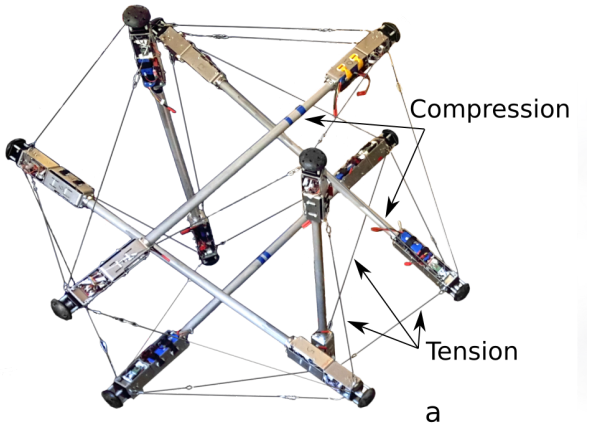


Figure 1.1: Tensegrity Robot [4]

Differential Dynamic Programming (DDP) is an indirect-method that determines the optimal policy in an unconstrained control space and is fast enough to be evaluated on modern computers. Although, it provide a good feedback policies it mostly relies on higher order gradients. These gradients might be intractable or harder to estimate in cases where the system is complex or model is not completely known.

Control theory offers a mechanism known as the Linear Quadratic Regulator (LQR) to estimate policy for system's with linear dynamics and quadratic cost (or reward) function. We use variant of DDP known as Iterative Linear Quadratic Regulator (ILQR) to compute the local optimum policies. There are few policy search methods can stitch these local policies to find a global policies. However, they often run into issues like convergence time and often land in poor local optima.

Guided Policy Search provides an efficient method to combine the local policies into global policies while bounding the information loss and minimizing the cost functions. In this method, we can represent the high dimensional complex policies using deep neural networks and train them in a supervised fashion.

Note: Most of the content and knowledge is inspired from Abbeel's lecture on Advanced Robotics [5] and Boyd's lecture on Dynamical Systems [6].

1.1.1 Optimal Control

Control theory is the science that deals with the behavior of dynamical systems 5.1 and their influence on the environment. In general we have two types of controls loops: open loop control and closed loop (feedback) control.

Open Loop Control: The action from the controller is independent of the output of the process. For ex: A conventional washing machine with constant voltage. In general the duration of washing time(voltage) is decided by the human. So the machine runs with constant voltage independent of the load of the system. Or alarm system that triggers in-case of external disturbance. However, it keeps on beeping until manual intervention.

Closed Loop Control: The action from the controller depends on the output of the process. For ex: A Thermostat monitoring the building temperature. A feedback signal ensures that the thermostat maintains the temperature of the building.

Closed Loop controllers have the following advantages over Open Loop controllers

- less sensitivity to changes in parameters
- better performance even with model uncertainties
- greater stability etc.

In some systems we use open-loop and closed-loop control simultaneously. In such systems, the open-loop is termed as *feedforward* control. A common closed-loop architecture in industrial systems is the **proportional-integral-derivative** controller commonly referred as **PID** Controller [7]. It continuously calculates the error value $e(t)$ as the difference between the desired and the measure process states. Though they are applicable to many systems, they need not give an optimal control in general due to its constant parameters and no knowledge of the process. They cannot be extended to systems with high nonlinearity and typically involve noise in computation of the derivative and integral terms [8] [9]. Linear Quadratic Regulator (LQR) is the primitive feedback controller that overcomes the shortcomings of PID Controllers by tunings its parameters using the full knowledge of the system.

Chapter 2

Linear Quadratic Regulator

If the dynamics of the system can be reported by a set of *linear* differential equations and the cost can be described by a *quadratic* function, the problem can be poised as a **LQ problem**. The solution for such a problem can be provided by a **feedback** controller known as **Linear-Quadratic Regulator (LQR)**. A stochastic variant of LQR is known as **Linear Quadratic Gaussian (LQG)**.

2.1 Linear Systems

Depending on the horizon setting (finite or infinite) [5.1] and system status (continuous or discrete). LQR operates in four different phases.

2.1.1 Finite Horizon Discrete System

We have a discrete system, defined on a time interval $t \ \epsilon \ [t_0, t_N]$. In other words, it runs for a horizon of length N time steps. The dynamics of the system are expressed as:

$$\mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}_t \tag{2.1}$$

where $\mathbf{x}_t$ is the current state, $\mathbf{u}_t$ is the current control and $\mathbf{x}_{t+1}$ is the next state. The above equation[2.1] is also referred to as *state update equation* where matrices A and B capture the effect of state and input(control) of the state derivative and control derivative respectively.

We define a cost function on the system as

$$J(\mathbf{x}_0, \mathbf{U}) = \sum_{t=t_0}^{t_N-1} (\mathbf{x}_t^T Q \mathbf{x}_t + \mathbf{u}_t^T R \mathbf{u}_t) + \mathbf{x}_N^T Q_N \mathbf{x}_N \tag{2.2}$$

where $\mathbf{U} = (\mathbf{u}_0, \mathbf{u}_1, .., \mathbf{u}_{N-1})$ denotes the sequence of control steps that are applied and $\mathbf{X} = (\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, .., \mathbf{x}_N)$ represents the trajectory (or sequence of states). $Q = Q^T \geq 0$ is the *state* cost matrix, $Q_N = Q_N^T \geq 0$ is the *final state* cost matrix and $R = R^T \geq 0$ is the *input* or *control* cost matrix. The first term in the cost J measure the **deviation** of the state while second term measures the **control authority** and final term measures the final state deviation. Note that cost is non-zero for all non-zero states and controls. [5.2].

Observation: At $Q = Q_N = 0$ and $R = \frac{1}{2}$, *LQR* turns out be *least squares regression.*

Problem : Given a finite horizon discrete LQR system, find the sequence of controls $\mathbf{U} = (\mathbf{u}_0, \mathbf{u}_1, .., \mathbf{u}_{N-1})$ that lead the system from a state $\mathbf{x}_0$ to a target state $\mathbf{x}_N$ [2.2] in N time steps.

The above problem can be solve using the principle of Dynamic Programming.

2.1.1.1 Value Function

It defines the minimum **cost-to-go** value to reach a target state $\mathbf{x}_N$ from a given state x in t steps.

$$V(\mathbf{x}, t) = \min_{\hat{\mathbf{U}}} J_t(\mathbf{x}, \hat{\mathbf{U}}) \tag{2.3}$$

where $\hat{\mathbf{U}} = \{\mathbf{u}_{t_N-t}, \mathbf{u}_{t_N-t+1}, \cdots, \mathbf{u}_{t_N-1}\}$ is the sequence of controls from $\mathbf{x}$ to $\mathbf{x}_N$

Observation: $V(\mathbf{x}, t)$ is the minimum LQR cost if $\mathbf{x} = \mathbf{x}_0$ and time $t = t_N - t_0$.

2.1.1.2 Dynamic Programming

Dynamic programming is an optimization method to solve the complex problems by splitting them into smaller sub problems, solving them individually and stitching them together. Applying the same approach to value function gives the Bellman "Principle of Optimality" to solve the optimal control problem. It states that value function with $t+1$ steps to go at state $\mathbf{x}$ is equal to the sum of minimum cost incurred at current state $\mathbf{x}$ and the value function with t steps to go.

$$V(\mathbf{x}, t+1) = \min_{\mathbf{u}} \quad [\mathbf{x}^T Q x + \mathbf{u}^T R u + V(A\mathbf{x} + B\mathbf{u}, t)] \tag{2.4}$$

This often referred to as Bellman Equation 2.4.
At the final state, i.e with $\mathbf{0}$ steps to go, $V(\mathbf{x}, 0) = \mathbf{x}^T P_0 \mathbf{x}$ where $P_0 = Q_N$. So the value function with $t = 1$ steps to go turns out to be

$$V(\mathbf{x}, 1) = \min_{\mathbf{u}} \quad [\mathbf{x}^T Q \mathbf{x} + \mathbf{u}^T R \mathbf{u} + (A\mathbf{x} + B\mathbf{u})^T P_0 (A\mathbf{x} + B\mathbf{u})] \tag{2.5}$$

In order to find the minimum over $\mathbf{u}$, we simply set the gradient of the above equation to zero. So

$$\frac{\partial}{\partial \mathbf{u}} V = 0$$
$$2R\mathbf{u} + 2B^T P_0 (A\mathbf{x} + B\mathbf{u}) = 0$$

Rearranging the terms we get

$$\mathbf{u} = -(R + B^T P_0 B)^{-1} B^T P_0 A \mathbf{x} \tag{2.6}$$

Substituting eq[2.6] in the equation above we get

$$V(\mathbf{x}, 1) = \mathbf{x}^T P_1 \mathbf{x}$$
$$\text{where } P_1 = Q + {K_1}^T R K_1 + (A + BK_1)^T P_0 (A + BK_1)$$
$$\text{and } K_1 = -(R + B^T P_0 B)^{-1} B^T P_0 A.$$

Solving for the next value functions recursively, we would get

$$V(\mathbf{x}, 2) = \mathbf{x}^T P_2 \mathbf{x}$$
$$\text{where } P_2 = Q + {K_2}^T R K_2 + (A + BK_2)^T P_1 (A + BK_2)$$
$$\text{and } K_2 = -(R + B^T P_1 B)^{-1} B^T P_1 A.$$

Continuing in this manner(backwards), till we reach the initial state would yield the following solution to LQR problem

Algorithm 1 Solution to Linear Quadratic Regulator (Discrete System)

1: Set $P_0 = Q_N$
2: **for** $i = 1, 2, 3, \cdots, N$ **do**
3: $\quad K_i = -(R + B^T P_{i-1} B)^{-1} B^T P_{i-1} A$
4: $\quad P_i = Q + {K_i}^T R K_i + (A + BK_i)^T P_{i-1} (A + BK_i)$
5: **end for**

The optimal policy for any horizon of length t is given by:

$$\pi(\mathbf{x}) = K_t \quad \mathbf{x} \tag{2.7}$$

The cost-to-go value for any horizon of length t is given by:

$$V(\mathbf{x}, t) = \mathbf{x}^T P_t \mathbf{x} \tag{2.8}$$

The matrix K determines the feedback *gain* of the controller.

2.1.2 Infinite-horizon Discrete System

In this case, the system runs indefinitely in discrete time steps i.e it is defined in the time interval $t \epsilon [t_0, \infty)$. The dynamics are same as eqn[2.1]:

$$\mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}_t$$

where $\mathbf{x}_t$ is the current state, $\mathbf{u}_t$ is the current control and $\mathbf{x}_{t+1}$ is the immediate next state. The cost function spans indefinitely

$$J(\mathbf{X}, \mathbf{U}) = \sum_{t=t_0}^{\inf} (\mathbf{x}_t^T Q \mathbf{x}_t + \mathbf{u}_t^T R \mathbf{u}_t) \tag{2.9}$$

where $\mathbf{U} = (\mathbf{u}_0, \mathbf{u}_1, \cdots)$ denotes the sequence of control steps that are applied and $\mathbf{X} = (\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \cdots)$ represents the trajectory (or sequence of states). Again, the weight matrices are chosen such that $Q = Q^T \geq 0$ is the *state* cost matrix and $R = R^T \geq 0$ is the *input* cost matrix. This makes sure that we have a non-negative cost at all stages.

2.1.2.1 Solution

The solution to this system is similar to the 1previous solution except that we drop the indices of the feedback terms since the horizon is infinite. Initialize P to a random value.

$$K = -(R + B^T P B)^{-1} B^T P A$$
$$P = Q + K^T R K + (A + BK)^T P (A + BK)$$

The optimal control ($\mathbf{u} = \pi(\mathbf{x})$) for any state is given by:

$$\pi(\mathbf{x}) = K\mathbf{x} \tag{2.10}$$

2.1.3 Finite-horizon Continuous System

Here we have a continuous linear system P, defined on a time interval $t \epsilon [t_0, \mathbf{t_N}]$. The dynamics of the system are expressed as update equation is:

$$\dot{\mathbf{x}}_t = A\mathbf{x}_t + B\mathbf{u}_t \tag{2.11}$$

where $\mathbf{x}_t$ is the current state, $\dot{\mathbf{x}_t}$ is its time derivative and $\mathbf{u}_t$ is the current control. A and B capture the effect of state and control on the time derivate of the state. We define the cost function on the system as

$$\mathbf{J}(\mathbf{X}, \mathbf{U}) = \int_{t=t_0}^{\mathbf{t_N}-\mathbf{1}} (\mathbf{x}_t^T Q \mathbf{x}_t + \mathbf{u}_t^T R \mathbf{u}_t) + \mathbf{x}_N^T \mathbf{Q}_N \mathbf{x}_N \tag{2.12}$$

where $U = (\mathbf{u}_0, \mathbf{u}_1, .., \mathbf{u}_{N-1})$ are the sequence of control steps that are applied, $Q = Q^T \geq 0$ is the *state* cost matrix, $\mathbf{Q}_N = \mathbf{Q}_N^T \geq 0$ is the *final state* cost matrix and $R = R^T \geq 0$ is the *input* cost matrix.

Algorithm 2 Solution to Linear Quadratic Regulator (Continuous System)

1: Set $P_0 = Q_N$
2: **for** $i = 1, 2, 3, \cdots, N$ **do**
3: $\quad K_i = -(R + B^T P_{i-1} B)^{-1} B^T P_{i-1} A$ where P_i is solution to the equation below.
4: $\quad -\dot{P}_i = A^T P_i + P_i A - P_i B R^{-1} B^T P_i + Q$
5: **end for**

The optimal policy for any horizon of length t is given by:

$$\pi(\mathbf{x}) = K_t \mathbf{x} \tag{2.13}$$

The cost-to-go value for any horizon of length t is given by:

$$\mathbf{V}(\mathbf{x}, t) = \mathbf{x}^T \mathbf{P_t} \mathbf{x} \tag{2.14}$$

2.1.4 Infinite-horizon Continuous System

This setting is similar to the previous except for the fact that the system runs indefinitely. In other words, we have a continuous linear system P, defined on a time interval $t \, \epsilon \, [t_0, \infty]$. The dynamics of the system are expressed as update equation is:

$$\dot{\mathbf{x_t}} = A\mathbf{x}_t + B\mathbf{u}_t \tag{2.15}$$

where $\mathbf{x}_t$ is the current state, $\dot{\mathbf{x_t}}$ is its time derivative and $\mathbf{u}_t$ is the current control. A and B capture the effect of state and control on the time derivate of the state. We define the cost function on the system as

$$\mathbf{J}(\mathbf{X}, \mathbf{U}) = \int_{t=t_0}^{t_N - 1} (\mathbf{x}_t^T Q \mathbf{x}_t + \mathbf{u}_t^T R \mathbf{u}_t) \tag{2.16}$$

where $U = \{\mathbf{u_0}, \mathbf{u_1}, \cdots\}$ are the sequence of control steps that are applied, $Q = Q^T \geq 0$ is the *state* cost matrix, $Q_N = {Q_N}^T \geq 0$ is the *final state* cost matrix and $R = R^T \geq 0$ is the *input* cost matrix.

The solution is same as the 2previous one except that we drop the indices sine the horizon is not limited. Set $P_0 = Q_N$ and iterate over the following equations until convergence.

$$K = -(R + B^T P B)^{-1} B^T P A$$

where P is solution to the equation

$$A^T P + P A - P B R^{-1} B^T P + Q = 0$$

The optimal policy for any state is given by:

$$\pi(\mathbf{x}) = K\mathbf{x} \tag{2.17}$$

2.2 Extensions

LQR can also be extended to

- Affine Systems
- Stochastic Systems
- Linear Time Varying Systems
- Tracking a state for a non-linear system
- Trajectory following for a non-linear system etc.

2.2.1 Affine Systems

- In this case we deviate the state by a constant amount i.e add a constant noise.
- So the dynamics are represented as

$$\mathbf{x_{t+1}} = A\mathbf{x}_t + B\mathbf{u}_t + c \tag{2.18}$$

- The cost function is described as

$$g(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{x}_t^T Q \mathbf{x}_t + \mathbf{u}_t^T R \mathbf{u}_t \tag{2.19}$$

- Replace $\mathbf{x}_t$ with $\mathbf{z}_t$ where $\mathbf{z}_t = [\mathbf{x}_t; 1]$,.
  - Redefining the dynamics we get

$$\mathbf{z_{t+1}} = \begin{bmatrix} \mathbf{x_{t+1}} \\ 1 \end{bmatrix} = \begin{bmatrix} A & \mathbf{c} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x}_t \\ 1 \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} \mathbf{u}_t = A'\mathbf{z}_t + B'\mathbf{u}_t$$

- The optimal policy remains linear in the state ($\mathbf{z}_t$).
- The optimal cost-to-go remains quadratic in state ($\mathbf{z}_t$) and the control.

2.2.2 Stochastic Systems

- In this case, we add dynamic noise w_t to the system for $t = 0, 1, \cdots, N-1$.
- w_t are Independent and Identically distributed random variables, i.e $\mathbf{E}[w_t] = 0$ and $\mathbf{E}[w_t^T w_t] = \mathbf{W}$ where $\mathbf{W}$ is a given matrix.
- The dynamics of the system are represented as

$$\mathbf{x_{t+1}} = A\mathbf{x}_t + B\mathbf{u}_t + \mathbf{w}_t \tag{2.20}$$

- The cost function is described by

$$g(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{x}_t^T Q \mathbf{x}_t + \mathbf{u}_t^T R \mathbf{u}_t \tag{2.21}$$

- We can solve the system in the same manner using the value functions and dynamic programming principle.
- The optimal policy for any horizon of length t is given by:

$$\begin{aligned} \pi(\mathbf{x}) &= K_t \mathbf{x} \\ \text{where } K_t &= -(R + B^T P_{t-1} B)^{-1} B^T P_{t-1} A \\ \text{and } P_t &= Q + {K_t}^T R K_t + (A + BK_t)^T P_{t-1} (A + BK_t) \end{aligned}$$

- The cost-to-go value for any horizon of length t is given by:

$$\begin{aligned} V(\mathbf{x}, t) &= \mathbf{x}^T P_t \mathbf{x} + \kappa_t \\ \text{where } \kappa_t &= \kappa_{t-1} + Tr(\mathbf{W} P_{t-1}) \end{aligned}$$

- Notice that the optimal policy is same as the deterministic case and the feedback terms are identical.
- The cost-to-go function remains quadratic and the extra term doesn't depend on control.

2.2.3 Linear Time Varying Systems

- The only difference in this case, is that the weights in both the dynamics and cost functions depend on time.
- The dynamics of the system are represented as

$$\mathbf{x}_{t+1} = A_t\mathbf{x}_t + B_t\mathbf{u}_t \tag{2.22}$$

- The cost function is described by

$$g(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{x}_t^T Q_t \mathbf{x}_t + \mathbf{u}_t^T R_t \mathbf{u}_t \tag{2.23}$$

- So the solution is pretty much similar to that of deterministic case, with the exception of time varying weights.
  - Set $P_0 = Q_N$
  - for $\mathbf{i = 1}$, $\mathbf{2}$, $\mathbf{3}$,$\cdots$, N

$$K_i = -(R_{N-i} + {B_{N-i}}^T P_{i-1} B_{N-i})^{-1} {B_{N-i}}^T P_{i-1} A_{N-i}$$
$$P_i = Q_{N-i} + {K_i}^T R_{N-i} K_i + (A_{N-i} + B_{N-i}K_i)^T P_{i-1}(A_{N-i} + B_{N-i}K_i)$$

- The optimal policy for any horizon of length t is given by:

$$\pi(\mathbf{x}) = K_t\mathbf{x}$$

- The cost-to-go value for any horizon of length t is given by:

$$V(\mathbf{x}, t) = \mathbf{x}^T P_t \mathbf{x}$$

- Again the optimal policy and cost-to-go function are similar to the deterministic case as expected.

2.2.4 Tracking a state for a non-linear system

- Given a target state $\mathbf{x}_G$, find the optimal policy that positions the non-linear system at $\mathbf{x}_G$.
- The dynamics are represented as

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) \tag{2.24}$$

- We can keep the system at a state $\mathbf{x}^*$ if and only if $\exists\ \mathbf{u}^*$ such that $\mathbf{x}^* = f(\mathbf{x}^*, \mathbf{u}^*)$
- Linearizing the dynamics around $\mathbf{x}^*$ will give us

$$\mathbf{x}_{t+1} \approx f(\mathbf{x}^*, \mathbf{u}^*) + \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}^*, \mathbf{u}^*)(\mathbf{x}_t - \mathbf{x}^*) + \frac{\partial f}{\partial \mathbf{u}}(\mathbf{x}^*, \mathbf{u}^*)(\mathbf{u}_t - \mathbf{u}^*)$$

- The above equation is equivalent to:

$$\mathbf{x}_{t+1} - \mathbf{x}^* \approx A(\mathbf{x}_t - \mathbf{x}^*) + B(\mathbf{u}_t - \mathbf{u}^*) \tag{2.25}$$

- Define variables $\mathbf{z}_t$, $\mathbf{v}_t$ such that $\mathbf{z}_t = \mathbf{x}_t - \mathbf{x}^*$ and $\mathbf{v}_t = \mathbf{u}_t - \mathbf{u}^*$
- Now we can re-write the dynamics eq[2.25] as follows

$$\mathbf{z}_{t+1} = A\mathbf{z}_t + B\mathbf{v}_t \tag{2.26}$$

- The updated cost function is:

$$g(\mathbf{z}_t, \mathbf{v_t}) = \mathbf{z}_t^T Q \mathbf{z}_t + \mathbf{v}_t^T R \mathbf{v}_t \tag{2.27}$$

- Solving the equations [2.25, 2.27] via the *standard LQR* yields us the following optimal policy.

$$\begin{aligned}\mathbf{v}_t &= \mathbf{K}_t \mathbf{z}_t \\ &= \mathbf{K}_t(\mathbf{x}_t - \mathbf{x}^*)\end{aligned}$$

- Replace $\mathbf{v}_t$ with $(\mathbf{u}_t - \mathbf{u}^*)$ would give

$$\begin{aligned}\mathbf{u}_t - \mathbf{u}^* &= \mathbf{K}_t(\mathbf{x}_t - \mathbf{x}^*) \\ \mathbf{u}_t &= \mathbf{u}^* + \mathbf{K}_t(\mathbf{x}_t - \mathbf{x}^*)\end{aligned}$$

- The optimal policy for tracking a state is given by

$$\mathbf{u}_t = \mathbf{u}^* + \mathbf{K}_t(\mathbf{x}_t - \mathbf{x}^*) \tag{2.28}$$

- The cost-to-go value for any horizon of length t is be quadratic in $(\mathbf{x}_t - \mathbf{x}^*)$.

2.2.5 Tracking a trajectory for a non-linear system

- Find the Optimal policy that follows a given trajectory (or sequence of states) $\{\mathbf{x}_0^*, \mathbf{x}_1^*, \cdots, \mathbf{x}_N^*\}$ for the non linear system.
- The dynamics are similar to eq2.24.

$$\mathbf{x_{t+1}} = f(\mathbf{x}_t, \mathbf{u}_t) \tag{2.29}$$

- A trajectory $\{\mathbf{x}_0^*, \mathbf{x}_1^*, \cdots, \mathbf{x}_N^*\}$ is feasible if and only if $\exists\ \{\mathbf{u}_0^*, \mathbf{u}_1^*, \cdots, \mathbf{u}_{N-1}^*\}$: $\forall t \epsilon \{0, 1, \cdots, N-1\}$ such that $\mathbf{x}_{t+1}^* = f(\mathbf{x}_t^*, \mathbf{u}_t^*)$
- So linearizing the dynamics around $(\mathbf{x}_t^*, \mathbf{u}_t^*)$ will give us

$$\mathbf{x_{t+1}} \approx f(\mathbf{x}_t^*, \mathbf{u}_\mathbf{t}^*) + \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_\mathbf{t}^*, \mathbf{u}_\mathbf{t}^*)(\mathbf{x}_t - \mathbf{x}_\mathbf{t}^*) + \frac{\partial f}{\partial \mathbf{u}}(\mathbf{x}_\mathbf{t}^*, \mathbf{u}_\mathbf{t}^*)(\mathbf{u}_t - \mathbf{u}_\mathbf{t}^*)$$

- The above equation is equivalent to:

$$\mathbf{x_{t+1}} - \mathbf{x}_{\mathbf{t+1}}^* \approx \mathbf{A_t}(\mathbf{x_t} - \mathbf{x}_\mathbf{t}^*) + \mathbf{B_t}(\mathbf{u_t} - \mathbf{u}_\mathbf{t}^*) \tag{2.30}$$

- Define variables $\mathbf{z_t}$, $\mathbf{v_t}$ such that $\mathbf{z}_t = \mathbf{x}_t - \mathbf{x}^*$ and $\mathbf{v}_t = \mathbf{u}_t - \mathbf{u}^*$.
- Now we can re-write the dynamics eq[2.30] as follows

$$\mathbf{z}_{t+1} = A_t \mathbf{z}_t + B_t \mathbf{v}_t \tag{2.31}$$

- The updated cost function is:

$$g(\mathbf{z}_t, \mathbf{v}_t) = \mathbf{z}_t^T Q \mathbf{z}_t + \mathbf{v}_t^T R \mathbf{v_t} \tag{2.32}$$

- Running the standard LQR back ups for the equations [2.30, 2.32] results in the

following optimal policy for t time steps to go.

$$\begin{aligned}\mathbf{v}_{N-t} &= K_t \mathbf{z}_{N-t} \\ &= K_t(\mathbf{x}_{N-t} - \mathbf{x}^*_{N-t})\end{aligned}$$

- Replace $\mathbf{v}_{N-t}$ with $(\mathbf{u}_{N-t} - \mathbf{u}^*_{N-t})$ would give

$$\begin{aligned}\mathbf{u}_{N-t} - \mathbf{u}^* &= K_t(\mathbf{x}_{N-t} - \mathbf{x}^*_{N-t}) \\ \mathbf{u}_{N-t} &= \mathbf{u}^*_{N-t} + K_t(\mathbf{x}_{N-t} - \mathbf{x}^*_{N-t})\end{aligned}$$

- The optimal policy for tracking the trajectory with t time-steps to-go is given by

$$\pi(\mathbf{x}_{N-t}) = \mathbf{u}^*_{\mathbf{N-t}} + K_t(\mathbf{x}_{N-t} - \mathbf{x}^*_{N-t}) \tag{2.33}$$

- The cost-to-go value with t time-steps is again quadratic in $(\mathbf{x}_{N-t} - \mathbf{x}^*_{N-t})$.
- Remarks
  - Note that the target trajectory need not be feasible to apply this technique.
  - If the trajectory is infeasible, then the linearizations will not be valid around the (state, control) pairs that are visited.

2.2.6 General Scenario

- Let us define the most generic optimal control problem

$$\min_{\mathbf{U}} \sum_{t=0}^{N} (\mathbf{x}_t, \mathbf{u}_t)$$

$$\text{subject to } \mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) \quad \forall t$$

- One way to solve the above system is by iteratively approximating the system to LQ problem and using the standard LQR output.
- Initialize the algorithm with either of the following
  (a) A Policy $\pi^{(\mathbf{0})}$ (or)
  (b) A Sequence of states $\mathbf{X} = \{\mathbf{x_0}^{(\mathbf{0})}, \mathbf{x_1}^{(\mathbf{0})}, \cdots, \mathbf{x_N}^{(\mathbf{0})}\}$ and
  Sequence of controls $\mathbf{U} = \{\mathbf{u_0}^{(\mathbf{0})}, \mathbf{u_1}^{(\mathbf{0})}, \cdots, \mathbf{u_{N-1}}^{(\mathbf{0})}\}$.
- If the algorithm is initialized by policy (a) move to Step (1) else go to Step (2).
- Iterate until convergence:
  1. Execute the current policy $\pi^{(\mathbf{i})}$ on the system and note the sequence of states and $\mathbf{X}^{(\mathbf{i})} = \{\mathbf{x_0}^{(\mathbf{i})}, \mathbf{x_1}^{(\mathbf{i})}, \cdots, \mathbf{x_N}^{(\mathbf{i})}\}$ and controls $\mathbf{U}^{(\mathbf{i})} = \{\mathbf{u_0}^{(\mathbf{i})}, \mathbf{u_1}^{(\mathbf{i})}, \cdots, \mathbf{u_{N-1}}^{(\mathbf{i})}\}$.
  2. Compute the Linear Quadratic approximation of the optimal control problem i.e linearize the dynamics by the first-order Taylor expansion and limit the cost function to order 2 by a second-order Taylor expansion around the noted sequence of states $\mathbf{X}^{(\mathbf{i})}$ and controls $\mathbf{U}^{(\mathbf{i})}$.
  3. Estimate the optimal policy $\pi^{(\mathbf{i+1})}$ using the one of the previous LQR back ups from the LQ approximation in Step (2).
  4. Set $i = i + 1$ and go to Step (1).
- Issues:
  - The system might not converge as expected since the optimal policy might not stay close to the points around which we computed the LQ approximations using Taylor Expansions.

- To mitigate this issue we can adjust cost function to force the policy to stay close to LQ approximation points i.e use the cost function

$$'(\mathbf{x}_t, \mathbf{u}_t) = (1-\alpha)'(\mathbf{x}_t, \mathbf{u}_t) + \alpha(\|\mathbf{x}_t - \mathbf{x}_t^{(\mathbf{i})})\|_2^2 + \|\mathbf{u}_t - \mathbf{u}_t^{(\mathbf{i})})\|_2^2)$$

  where α decides the strength of linearizations

- Since the dynamics f are non linear, we are dealing with a non-convex optimization problem that can get stuck in a local optima. Hence we need to have *good initialization*.
- The cost function can be non-convex in which case the weight matrices for the cost $\mathbf{Q_t}$, $\mathbf{R_t}$ fail to be positive-definite. We have to increase the penalties for the deviations in current state $\mathbf{x}_t$ and control $\mathbf{u}_t$ until the matrices $\mathbf{Q_t}$, $\mathbf{R_t}$ turn positive-definite.

The goodness (or optimality) of the solution depends on

1. How much non-linearity are we throwing away? i.e How good are the dynamics approximated?
2. How frequently we update the feedback gain matrix ($\mathbf{K}$).

2.3 Differential Dynamic Programming

It is not always easy to estimate the actual dynamics model for robotic systems especially for the ones that are highly non linear. Establishing the effects of contact dynamics in the mathematical models always provide a challenge and inaccurate representations might lead to disastrous results. To mitigate this issues, Differential Dynamic Programming has been introduced that removes the dependencies on the actual dynamics f and the cost functions g with their derivatives.

Differential Dynamic programming (DDP) is an optimal control algorithm of the trajectory optimization 5.1[10] class introduced by Mayne [11]. It uses locally-quadratic models of the dynamics and cost functions and shows quadratic convergence.

Instead of linearizing the dynamics and quadratizing (approximate it till the second order) the cost function, DDP [12] directly performs second order Taylor expansion of the Bellman back up equation 2.4. The value function update would require the second order derivatives of dynamics model and the cost function.

With the increasing complexity of the robotic systems the second order derivatives of the dynamic are not always tractable especially for nonlinear systems. We do not a mathematical model that accurately estimate the second order derivatives of function from it's raw data. However, we can estimate the first order derivatives using finite differences method. Ex: $f_x = \frac{f(x+\delta)-f(x-\delta)}{2\delta}$. So we use a variant of DDP known you Iterative Linear Quadratic Regulator [13] which needs the first order derivatives of the dynamics and second order derivatives of the cost function. Note that they are other more accurate methods for estimating the dynamics and its derivatives apart from ordinary finite differences as you'll see in the upcoming sections.

Chapter 3

Iterative Linear Quadratic Regulator

Iterative Linear Quadratic Regulator is a trajectory optimization method that could be seen as the control analog of the Gauss-Newton method for nonlinear least-squares optimization.

Consider a system with discrete-time dynamics, but the similar derivation works for continuous case. The dynamics are same as the equation 2.24 i.e

$$x_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) \tag{3.1}$$

Let us represent the cost function for each state and control pair $(\mathbf{x}_t, \mathbf{u}_t)$ by
$g(\mathbf{x}_t, \mathbf{u}_t)$ $\forall t = 0, 1, \cdots, N-1$, and $\hat{g}(\mathbf{x}_N)$ as the cost of the final state.
The *total cost* on the system is defined the *sum* of all possible state and control pairs and cost of the final state.

$$J(\mathbf{X}, \mathbf{U}) = \sum_{t=t_0}^{\mathbf{N}-1} g(\mathbf{x}_t, \mathbf{u}_t) + \hat{g}(\mathbf{x_N}) \tag{3.2}$$

where $\mathbf{U} = \{\mathbf{u}_0, \mathbf{u}_1, \cdots, \mathbf{u}_{N-1}\}$ is the sequence of controls that are applied for the sequence of states $\mathbf{X} = \{\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_N\}$.

3.1 Estimating the dynamics

- We simulating the system by small perturbations in state ($\delta\mathbf{x}$) and control ($\delta\mathbf{u}$) and approximate the first order derivatives of the dynamics using finite-differences.
- The derivate of the dynamics w.r.t to the state ($\mathbf{f_x}$) is given by

$$\mathbf{f_x} \approx \frac{f(\mathbf{x}+\delta\mathbf{x},\mathbf{u}) - f(\mathbf{x}-\delta\mathbf{x},\mathbf{u})}{2\delta\mathbf{x}} \tag{3.3}$$

- The derivate of the dynamics w.r.t to the control ($\mathbf{f_x}$) is given by

$$\mathbf{f_u} \approx \frac{f(\mathbf{x},\mathbf{u}+\delta\mathbf{u}) - f(\mathbf{x},\mathbf{u}+\delta\mathbf{u})}{2\delta\mathbf{u}} \tag{3.4}$$

- We do not have efficient mathematical models to estimate higher order derivatives of the dynamics (like $\mathbf{f_{xx}}, \mathbf{f_{uu}}$ etc.) with the raw data.
- Note that in case of a continuous system the dynamics depend on velocity of the state $\dot{\mathbf{x}}_t = f(\mathbf{x}_t, \mathbf{u}_t)$

3.2 Computing the Deterministic Optimal Policy

The *Value* for a horizon of length t at a state $\mathbf{x}$ is the minimum total cost-to-go to reach a target state $\mathbf{x_N}$ in t time steps

$$\mathbf{V}(\mathbf{x},t) = \min_{\mathbf{U}^t} \quad \mathbf{J}(\mathbf{x},\mathbf{U}^t) \tag{3.5}$$

where $\mathbf{U}^t = \{\mathbf{u_{N-t}}, \mathbf{u_{N-t+1}}, \cdots, \mathbf{u_{N-1}}\}$ is the sequence of controls applied.
Expanding the above eqn[3.5] using the dynamic programming principle yields us

$$\mathbf{V}(\mathbf{x},\mathbf{t+1}) = \min_{\mathbf{u}} \quad [\quad g(\mathbf{x},\mathbf{u}) + \mathbf{V}(f(\mathbf{x},\mathbf{u}),t) \quad] \tag{3.6}$$

In the standard LQR setup we would substitute the dynamics (linearized) and solve the system backwards in time. However, in the case of ILQR, we would find the smallest

perturbation in control δu that minimizes the change in the value function between subsequent time steps. This would ensure that the algorithm is not deviating from its path to optima.

Define the argument in the expansion of the above equation [3.6] using the second-order Taylor expansion. Let $Q(\partial\mathbf{x}, \partial\mathbf{u})$ denote the smallest perturbation around the i^{th} ($\mathbf{x}$, $\mathbf{u}$) to the **RHS** of the equation (3.6)

$$\begin{aligned} Q(\partial\mathbf{x}, \partial\mathbf{u}) = \quad & g(\mathbf{x} + \partial\mathbf{x}, \mathbf{u}\partial\mathbf{x}) - g(\mathbf{x}, \mathbf{u}) \\ + \quad & \mathbf{V}(f(\mathbf{x} + \partial\mathbf{x}, \mathbf{u} + \partial\mathbf{u}), t) - \mathbf{V}(f(\mathbf{x}, \mathbf{u}), t) \end{aligned} \tag{3.7}$$

The subscripts denote differentiation, for example: $\mathbf{f_x} = \frac{\partial f}{\partial \mathbf{x}}$ implies differentiation of dynamics w.r.t to the state. If we expand the $Q(\partial\mathbf{x}, \partial\mathbf{u})$ using Taylor's expansion to the second order, we get

$$Q(\partial\mathbf{x}, \partial\mathbf{u}) \approx \frac{1}{2} \begin{bmatrix} 1 \\ \partial\mathbf{x} \\ \partial\mathbf{u} \end{bmatrix}^T \begin{bmatrix} 0 & \mathbf{Q_x}^T & \mathbf{Q_u}^T \\ \mathbf{Q_x} & \mathbf{Q_{xx}} & \mathbf{Q_{xu}} \\ \mathbf{Q_u} & \mathbf{Q_{ux}} & \mathbf{Q_{uu}} \end{bmatrix} \begin{bmatrix} 1 \\ \partial\mathbf{x} \\ \partial\mathbf{u} \end{bmatrix} \tag{3.8}$$

Equating the above two equations [3.7, 3.8] we get the following set of equations.

$$\mathbf{Q_x} = \mathbf{g_x} + \mathbf{f_x}^T \mathbf{V_x}' \tag{3.9}$$

$$\mathbf{Q_u} = \mathbf{g_u} + \mathbf{f_u}^T \mathbf{V_x}' \tag{3.10}$$

$$\mathbf{Q_{xx}} = \mathbf{g_{xx}} + \mathbf{f_x}^T \mathbf{V_{xx}}' \mathbf{f_x} + \mathbf{V_x}' \dot{\mathbf{f}}_{\mathbf{xx}} \tag{3.11}$$

$$\mathbf{Q_{uu}} = \mathbf{g_{uu}} + \mathbf{f_u}^T \mathbf{V_{xx}}' \mathbf{f_u} + \mathbf{V_x}' \dot{\mathbf{f}}_{\mathbf{uu}} \tag{3.12}$$

$$\mathbf{Q_{ux}} = \mathbf{g_{ux}} + \mathbf{f_u}^T \mathbf{V_{xx}}' \mathbf{f_x} + \mathbf{V_x}' \dot{\mathbf{f}}_{\mathbf{ux}} \tag{3.13}$$

The terms marked in red are retained in DDP and ignored in ILQR. This property makes ILQR applicable to systems with complex dynamics since we can directly estimate all required terms using raw data from the system.

Minimizing equation (3.8) w.r.t to $\partial \mathbf{u}$

$$\delta \mathbf{u}^* = \arg\min_{\delta \mathbf{u}} Q(\delta \mathbf{x}, \delta \mathbf{u}) = -Q_{\mathbf{uu}}^{-1}(Q_{\mathbf{u}} + Q_{\mathbf{ux}}\delta \mathbf{x}) \tag{3.14}$$

The above policy gives us an open-loop term $\mathbf{k} = -Q_{\mathbf{uu}}^{-1}Q_{\mathbf{u}}$ and a feed-back term $K = -Q_{\mathbf{uu}}^{-1}Q_{\mathbf{ux}}$. Plugging these back into Q-function [equation (3.8)] results in

$$\Delta \mathbf{V}(i) = \quad -\tfrac{1}{2}Q_{\mathbf{u}}Q_{\mathbf{uu}}^{-1}Q_{\mathbf{u}} \tag{3.15}$$

$$\mathbf{V}_{\mathbf{x}}(i) = Q_{\mathbf{x}} - Q_{\mathbf{u}}Q_{\mathbf{uu}}^{-1}Q_{\mathbf{ux}} \tag{3.16}$$

$$\mathbf{V}_{\mathbf{xx}}(i) = Q_{\mathbf{xx}} - Q_{\mathbf{xu}}Q_{\mathbf{uu}}^{-1}Q_{\mathbf{ux}} \tag{3.17}$$

We recursively compute the **value** function ($\mathbf{V}(t)$) and its derivatives and control terms $\mathbf{k}(t)$, $K(t)$ from $t = \mathbf{N} - \mathbf{1}$ down to $t = 0$. At the final state the value function is final cost, i.e $\mathbf{V}(\mathbf{x}, 0) = \gamma(\mathbf{x})$. This is known as a **backward pass**. (eqns 3.14, to 3.15).
Once the backward pass is completed, we perform a **forward pass** to compute a new sequence of states and controls from $t = 0$ up to $t = N - 1$.

$$\begin{aligned} \mathbf{x}_0 &= \bar{\mathbf{x}_0} \\ \mathbf{u}_t &= \bar{\mathbf{u}_t} + \mathbf{k}_t + \mathbf{K}_t(\mathbf{x}_t - \bar{\mathbf{x}_t}) \\ \mathbf{x}_{t+1} &= \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t) \end{aligned} \tag{3.18}$$

Both the backward (3.15) and forward passes (3.18) are repeated until convergence.

3.3 Computing Inverses

Liao and Shoemaker [14] have shown that the steps taken by DDP are comparable to or better than a full Newton step for the entire control sequence. Since the control terms depend on ${\mathbf{Q}_{\mathbf{uu}}}^{-1}$, it is essential to ensure that $\mathbf{Q}_{\mathbf{uu}}$ is invertible and its computation is not expensive. Therefore, we use a regularization parameter (say control parameter) λ that ensures positive semi definite nature of $\mathbf{Q}_{\mathbf{uu}}$ [15]. λ plays the role of **Levenberg-Marquardt** [16] heuristic which in a interpolation between **Newton's Method** [17] and **Gradient Descent**. The following section details a method

to compute the inverse of $\mathbf{Q_{uu}}$.

Algorithm:

(a) Compute the eigen values (Q_{uu_evals}) and eigen vectors ($\mathbf{Q_{uu\_evecs}}$) of $\mathbf{Q_{uu}}$

(b) Remove all negative eigen values ($Q_{uu\_evals}[Q_{uu\_evals} \leq 0] = 0$) - This ensures that the matrix is positive semi-definite.

(c) Add the regularization term λ to all the eigen values. $[Q_{uu\_evals} += \lambda]$.

(d) Determine the inverse of $\mathbf{Q_{uu}}$ matrix using the below formulation

$$\mathbf{Q_{uu}}^{-1} = \mathbf{Q_{uu\_evecs}} \cdot diag(\frac{1}{Q_{uu\_evals}}) \cdot Q^T_{\mathbf{uu\_evecs}}$$

where *diag* gives the diagonal matrix

3.4 Convergence

At the end of each ilqr iteration, check if we have a better trajectory than the previous one in terms of the cost function i.e ($cost\_new < cost\_old$).

- If the current cost is better than the previous, we will decrease λ by a constant and be a bit relaxed in terms of the our gradient step.This step mimics Newton's method of optimization.
- Otherwise i.e current trajectory is costlier than previous, we will be increase λ by a constant and be a bit more cautious in gradient approach. This approach is closer to Gradient Descent in optimization theory.

If the parameter λ reaches a certain threshold ($\lambda_{\max}$) or all the iterations are completed, we say that the algorithm has converged and stop it. Yuval et.al 2012 has shown that this method works for controlled limited systems [15], .

3.5 Experiments

We have evaluated Iterative Linear Quadratic Regulator (ILQR) using *simple pendulum*, the primitive non linear system on different cost functions and targets. The idea was to check whether this method would be feasible to guide the system to achieve the desired targets under limited controls and apply them to real robotic systems. Regulating natural motion includes designing an controller (open-loop) for guiding systems with no or low input to the systems and achieve desired behavior. Custom behavior experiments include constructing feedback policies that help the nonlinear systems in attaining the targeted performance.

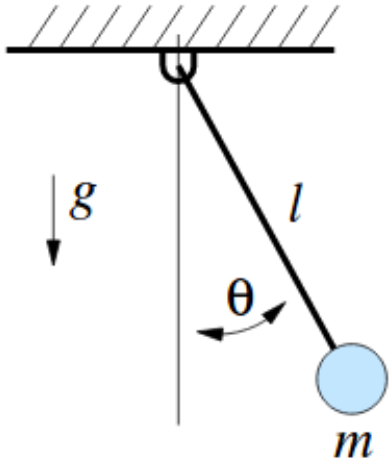


Figure 3.1: Simple Pendulum [1]

3.5.1 Settings for the simple pendulum

- The state representation of the simple pendulum consists of angular_displacement and angular_velocity pair i.e $\mathbf{x} = (\theta, \dot{\theta})$.
- The control is the torque applied at the pivot $\mathbf{u} = \tau$

[1] Image Courtesy: Russ Tedrake's Notes on Under Actuated Robotics 2009

- Any physical system with Kinetic Energy T and Potential Energy U can be represented using the Lagrangian Mechanics $L = T - U$

$$Q = \frac{d}{dt}\frac{\partial L}{\partial \dot{\mathbf{q}}_i} - \frac{\partial L}{\partial \mathbf{q}_i}$$

- In a canonical pendulum, $\mathbf{q} = \theta$, $\dot{\mathbf{q}} = \dot{\theta}$ and $Q = \tau - b\dot{\theta}$ is the generalized force.

- Solving the above equations results in the following dynamics for the pendulum

$$\tau = ml^{\mathbf{2}}\ddot{\theta} + b\dot{\theta} + mgl\sin\theta \tag{3.19}$$

- In our specific setup, we are using an undamped ($b = 0$) pendulum of unit mass ($m = 1$) and unit length ($l = 1$). The simulation (or time) step is **0.01**.

- The limits of angular displacement (θ) are between $[-\pi, \pi]$ where as the angular velocity ($\dot{\theta}$) ranges from -7 to 7.

- In all our experiments the start state is the horizontal position $(0, 0)$ [3.2] of the pendulum and final state is the upright position with minimum velocity $(\frac{\pi}{2}, 0)$.

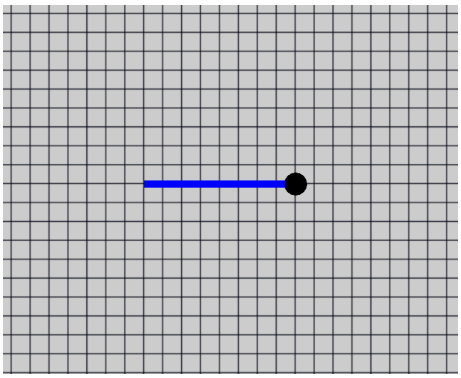


Figure 3.2: Simple Pendulum in PRACSYS [2]

[2]PRACSYS LAB: http://pracsyslab.org/

3.5.2 Tracking a state

- In this setting, we require the simple pendulum to reach an upright position with minimum speed i.e $\mathbf{x} = (\pi, 0)$ from a given initial state $\mathbf{x} = (\frac{\pi}{2}, 0)$.
- Algorithm is initialized with a policy of constant torque=**2** for a horizon of length **240** time steps.
- Following figure shows the phase plot for the pendulum where the x-axis represents angular displacement θ and y-axis shows the angular velocity $\dot{\theta}$.

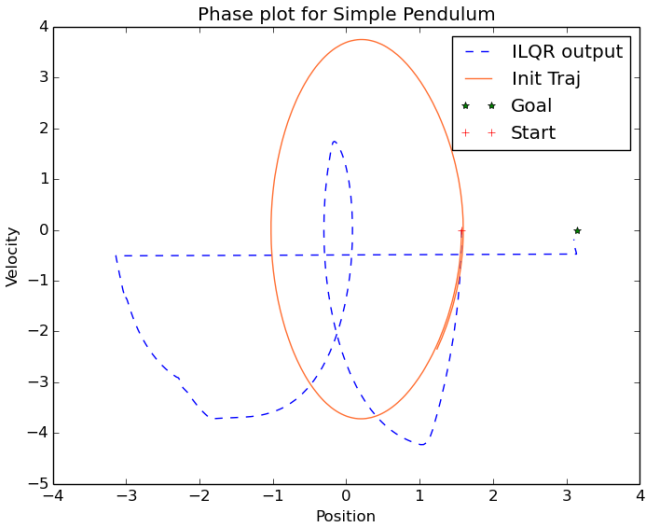


Figure 3.3: Phase plot for Tracking the Goal State

- Please note that the angular displacement is clamped between $-\pi$ to π, and hence the straight line or the jump in the plot from negative x-axis to the positive. In other terms the angular displacement wraps over $-\pi$ to π and vice-versa.
- The control (torque) ranges from $-\mathbf{10}$ to $+\mathbf{10}$ and the horizon consists of 240 simulation steps.
- The following figure compares the original control sequence with the output of ILQR algorithm.

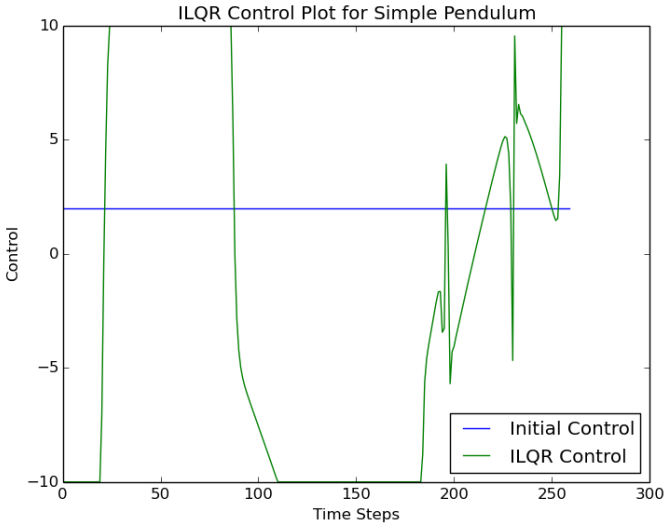


Figure 3.4: Control plot for Tracking the Goal State

- **Observations**
  - The start state is indicated by + symbol and the target state by * icon.
  - The initial trajectory is almost close to the trajectory governed by open loop setting and ILQR does a neat job in getting the pendulum close to the goal state (3.3).
  - However, the output policy is not smooth and makes rapid changes in control as indicated by the shakiness in the above plots. This might not be appropriate on a real robot though it is locally optimal.
  - This behavior is quite expected since the capability of ILQR is limited by the fixed horizon and bounded controls.

3.5.2.1 Extensions

Current Section includes the for tracking the target state with different cost functions.

(a) **Extension 1**

- In this case the ILQR algorithm is initialized with zero plan i.e open loop trajectory.
- The cost function tracks only the final state and all other intermediate costs are zero, i.e

$$g_N(\mathbf{x}_N) = w_p(\theta_N - \theta_{goal})^2 + w_v(\dot{\theta}_N - \dot{\theta_{goal}})^2$$
$$g(\mathbf{x}_t, \mathbf{u}_t) = w_c\|\mathbf{u}_t\|^2 \quad \text{where } t \neq N$$

where w_p, w_v, w_c are constants.

  - A Horizon of 137 simulation steps yielded the policy with lowest cost (0.62) result. In this experiment that control range has been shrunk down to $[-5, 5]$.

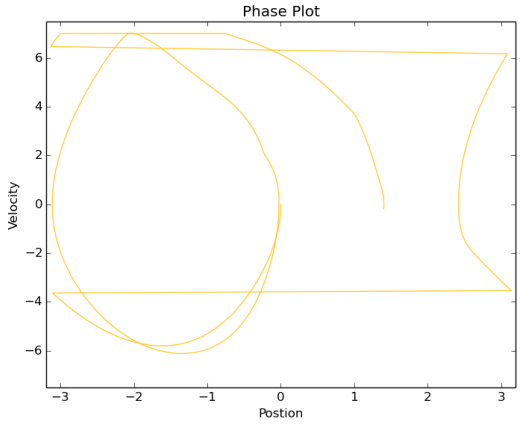


Figure 3.5: Phase plot for Tracking the Goal State with final state cost only

– The control plot for the same setting

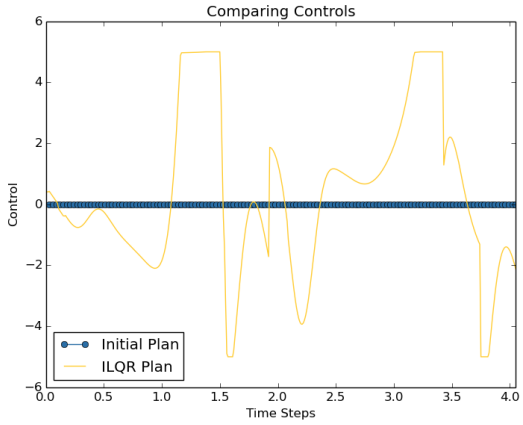


Figure 3.6: Control plot for Tracking the Goal State with final state cost only

– **Observations**

* It is good to see that ILQR is able to find a suitable plan with no initial control.
* Also the output policy looks a bit smoother compared to the previous case.
* Tracking only the final state helps ILQR to get relaxed at the intermediate states and allows the pendulum to prefer its natural motion over constrained behavior which would not have been possible in previous case.

– A video for the same setting can be found in the additional files section with name *Track_State_2.mp4*.

(b) **Extension 2**

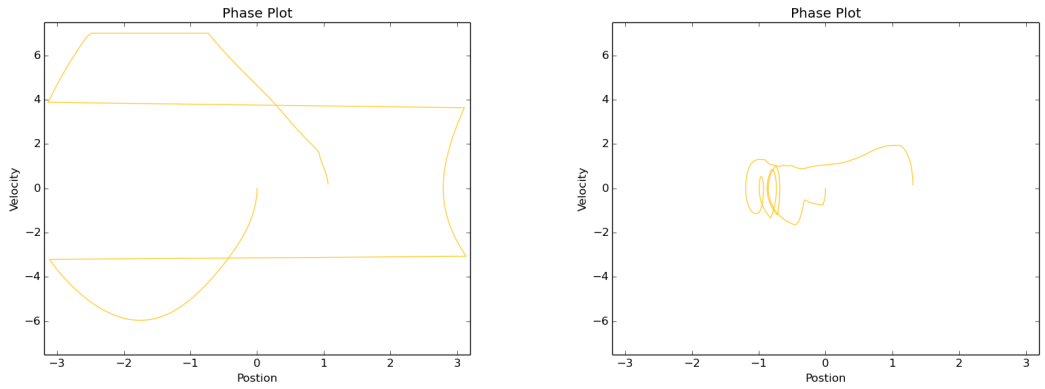
- In this case, the initialization is same as previous but the cost function now includes state deviation for intermediate states.

$$g_N(\mathbf{x}_N) = w_p(\theta_N - \theta_{goal})^2 + w_v(\dot{\theta}_N - \dot{\theta_{goal}})^2$$

$$g(\mathbf{x}_t, \mathbf{u}_t) = w_c * \|\mathbf{u}_t\|^2 + w'_p(\theta_N - \theta_{goal})^2 + w'_v(\dot{\theta}_N - \dot{\theta}_{goal})^2 \quad t \neq N$$

where $w_p, w_v, w'_p, w'_v, w_c$ are constants.

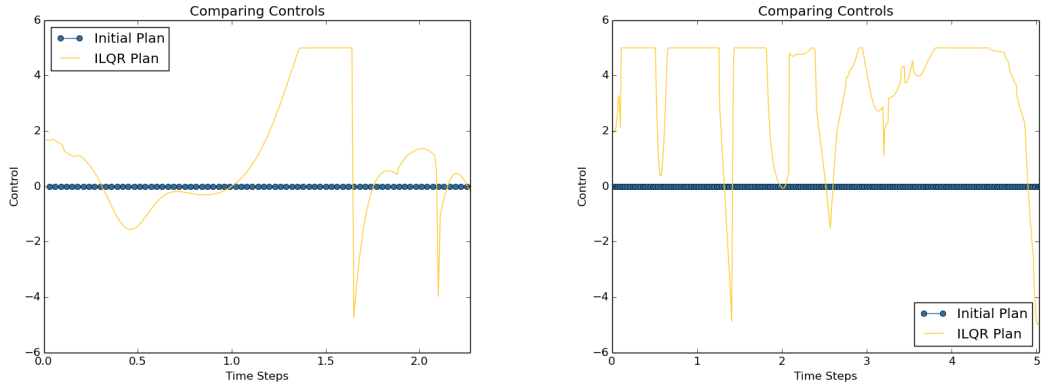
– Tracking states with intermediate costs



(a) Horizon 78 (b) Horizon 179

Figure 3.7: Phase plot for Tracking the Goal State with with Intermediate Costs

– The control plot for the same setting



(a) Horizon 78 (b) Horizon 179

Figure 3.8: Control plot for Tracking a State with Intermediate Costs

– **Observations**

∗ Though the phase plot for Horizon 78 looks smoother compared to Horizon 179, the latter tries to build momentum initially to reach the target state as dictated by the control plot.

∗ In case of Horizon 78, ILQR prefers to perform natural motion i.e use the force of gravity to guide the system towards the target due

to limited time. Hence, we do not see too many peeks in the control plot.

- ∗ However, in case of Horizon 179, it tries to move away from natural motion, i.e it fights the gravity (go against it) since it has more time to reach the goal. This is the reason for initial peeks in the control plot. Note that control is clamped between -5 and 5.
- ∗ As the horizon increases, we would expect the pendulum to fight gravity and reach the target. Due to the periodicity of the pendulum, there would be a set of horizons where the system fights gravity or allows natural motion.
- ∗ In the additional files section, you can find the videos for with names *Track_State_Horizon_78.mp4* and *Track_State_Horizon_179.mp4*.

(c) In this case, the initialization is same as previous but the cost function includes time varying state deviation for intermediate states.

$$g_N(\mathbf{x}_N) = w_p(\theta_{\mathbf{N}} - \theta_{\mathbf{goal}})^2 + w_v(\dot{\theta}_N - \dot{\theta_{\mathbf{goal}}})^2$$
$$g(\mathbf{x}_t, \mathbf{u}_t) = w_c * \|\mathbf{u}_t\|^2 + w'_p * \left(\frac{t}{N}\right)^2 * (\theta_N - \theta_{goal})^2 + w'_v * \left(\frac{t}{N}\right)^2 * (\dot{\theta}_f - \dot{\theta}_{goal})^2 \quad t \neq N$$

where $w_p, w_v, w'_p, w'_v, w_c$ are constants.

- Phase plot for tracking the goal state with time varying intermediate state costs

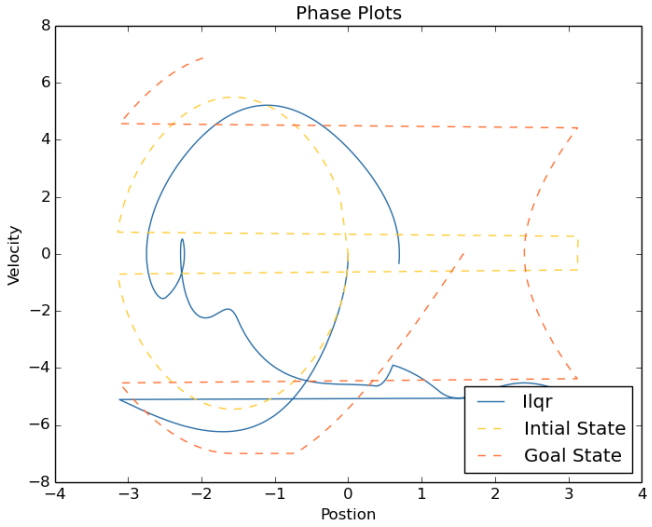


Figure 3.9: Phase plot for Tracking a State with Time Varying Intermediate Costs

- The control plot for the same setting

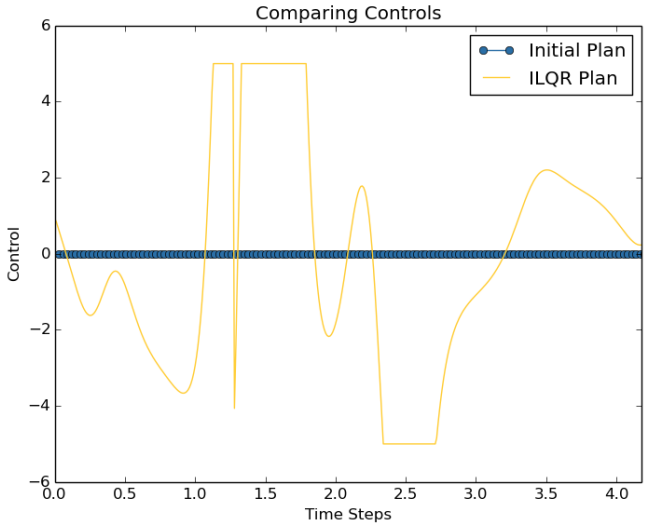


Figure 3.10: Phase plot for Tracking Goal State with Time Varying Intermediate Costs

- **Observations**
  - In the previous case, all the intermediate states were constrained by the same weight. This would lead the system to reach the system to the goal state as soon as possible which might in-turn rapid changes in control.
  - Hence, we constrain the intermediate states that are closer to target

state more than the initial states.

  - The orange dotted lines shows the natural behavior of the pendulum at the initial state and the red dotted lines shows the natural motion at the goal state. The goal of ILQR is to bring this input trajectory to the goal sequence which is marked in thick lines marked in blue.
  - We can see smoother transitions in control and phase plots compared to the previous case, especially during the initial states.
  - Note that this specific behavior is achieved with no input policy (zero policy) which would be hard for other types of controllers to give an optimal solution.

3.5.3 Tracking a Target State With Initialization

We were able to get an output policy with no input for tracking a target state for a system like pendulum. This might not work for other systems with more dimensional systems and complex dynamics. In addition to this, tracking a state deals with manual tuning of horizon length and control limits. These issues can be mitigated by using a sampling planner to give a good initial guess of the policy (+horizon and control limits) and apply ILQR on top of it.

- In this setup, we initialize the ilqr algorithm with the output policy of *RRT* planner with 3 different initial trajectory.
  - RRT - **Randomly Exploring Random** tree is a probabilistic complete algorithm to efficiently search in high dimensional and space.
  - It is show to generate probabilistically-optimal Open loop trajectories for complex nonlinear systems with additional state constraints.
  - It's key properties include randomly building a space filling tree and the search is more biased towards large unsearched areas.

- The cost function is of the form

$$g_N(\mathbf{x}_N) = w_p(\theta_\mathbf{N} - \bar{\theta}_{goal})^2 + w_v(\dot{\theta}_N - \bar{\dot{\theta}}_{goal})^2$$
$$g(\mathbf{x}_t, \mathbf{u}_t) = w_c * \|\mathbf{u}_t\|^2 + w'_p * \left(\frac{t}{N}\right)^2 * (\theta_t - \theta_{goal})^2 + w'_v * \left(\frac{t}{N}\right)^2 * (\dot{\theta}_t - \dot{\theta}_{goal})^2 \quad , t \neq N$$

  where $w_p, w_v, w'_p, w'_v, w_c$ are constants.

- Before running the algorithm, we simulate the system with policy and determine the sequence of trajectories $\bar{\mathbf{X}} = \{\dot{\mathbf{x}}_0, \dot{\mathbf{x}}_1, \cdots, \dot{\mathbf{x}}_N\}$ and the sequence of controls $\bar{\mathbf{U}} = \{\dot{\mathbf{u}}_0, \dot{\mathbf{u}}_1, \cdots, \dot{\mathbf{u}}_{N-1}\}$.

- Case 1:

  – The following figure shows the phase plot for tracking trajectory where the initial state is at $\mathbf{x_0} = (\pi/2, 0)$ and the final state is $\mathbf{x_N} = (\pi, 0)$.

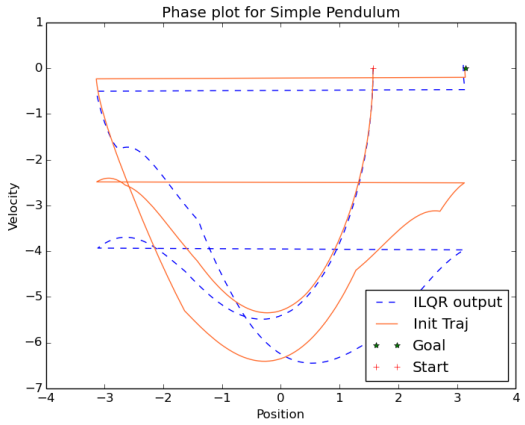


Figure 3.11: Phase plot for Tracking State with Initialization

  – The following plot compares the initial policy with the output of ILQR.

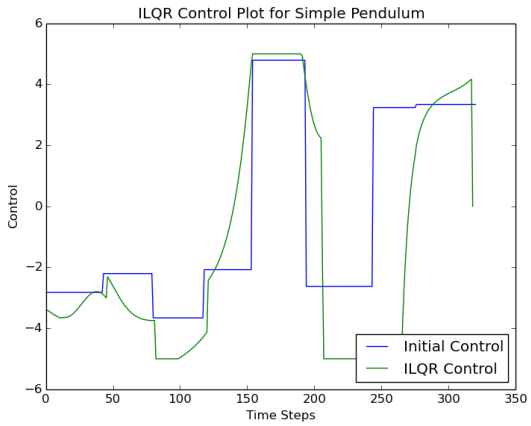


Figure 3.12: Control plot for Tracking State with Initialization

- **Observations** From the plot, you can see ILQR tries to make the changes in control compared to the initial policy from RRT maintaining with low cost.

- Case 2:
  - The following figure shows the phase plot for tracking target state with a different initialization where the initial state is at $\mathbf{x_0} = (\pi/2, 0)$ and the final state is $\mathbf{x_N} = (\pi, 0)$.

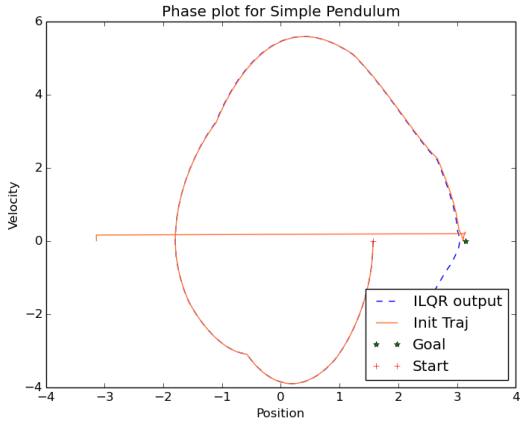


Figure 3.13: Phase plot for Tracking State with Initialization II

  - The following plot compares the initial policy with the output of ILQR.

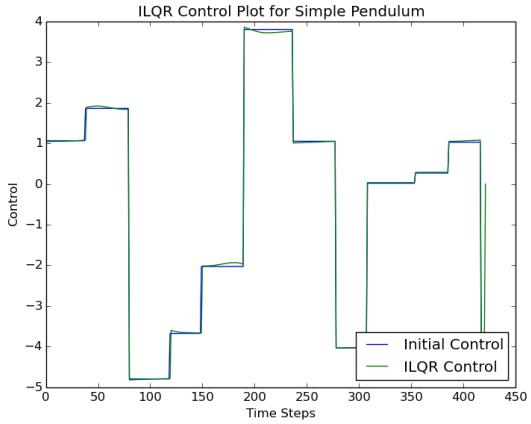


Figure 3.14: Control plot for Tracking State with Initialization II

– In this case, ILQR does not change the initial trajectory and control by much implying that we already have a local optima solution w.r.t to the cost function.

- Case 3

– The following figure shows the phase plot for tracking trajectory where the initial state is at $\mathbf{x_0} = (\pi/2, 0)$ and the final state is $\mathbf{x_N} = (\pi, 0)$.

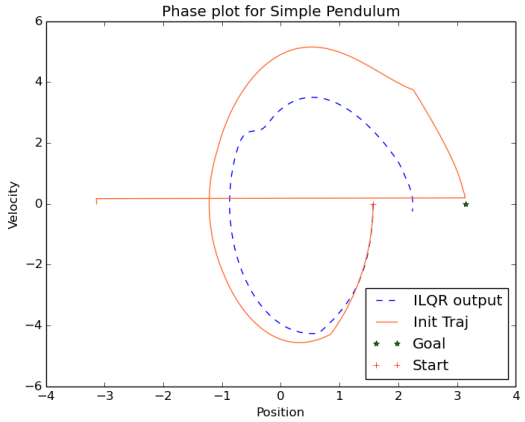


Figure 3.15: Phase plot for Tracking State with Initialization III

– The following plot compares the initial policy with the output of ILQR.

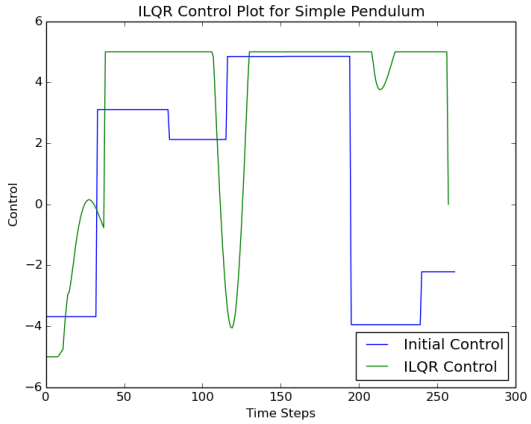


Figure 3.16: Control plot for Tracking State with Initialization III

- In this case, ILQR tries to the initial trajectory by a large amount yielding an output policy that does not comply with our target state. This implies that our initialization is too weak and limits the capabilities of ILQR for this specific cost function.

- **Observations**

  - ILQR performs better at reaching a Target State given a good initial guess since it provides good samples for policy learning and removes manual parameter tuning.
  - However, it is too sensitive to initialization as seen from the previous three sections [3.11, 3.13. 3.15].
  - If the initial trajectory is too good, it makes minor changes to the policy and if the trajectory is too bad, it cannot find the local optima though the initial policy works for the system.

3.5.4 Tracking an energy manifold

The idea behind this setup is to track the natural motion of the system with help of ILQR. If the system does not disseminate energy, periodic systems like pendulum tend to repeat similar behavior in regular intervals with no external feedback or force. In systems like Tensegrity Structures [1], [2] this setup might help us in generating continuous locomotion gaits like rolling or jumping.

- The objective is to displace the pendulum from its initial manifold to energy manifold of the goal state.
- The following shows the phase plot shows the transition of pendulum towards the energy manifold of the goal state.

(a) Control = -1.22 (b) Control = -1.09

Figure 3.17: Phase plot for Tracking Energy of Goal State

- Motivation
  - If we are able to move the system from one energy level to the target energy level, the system would eventually reach the goal state. If the systems are periodic, the goal can be reached in finite time whereas non-periodic systems might get there in in-finite time.
  - The energy manifold has higher dimension than a target point (1 versus 0), so we're always closer, so our linearized updates are always better. Also, in this case it has a special relationship to the upper equilibrium.
- Observations
  - From the phase plot, you can see that we are able to move away from the original energy level and towards the goal state energy manifold. Once we

reach this level, we can let the pendulum run for at most one time period to reach the target state.

3.5.5 Tracking a custom behavior

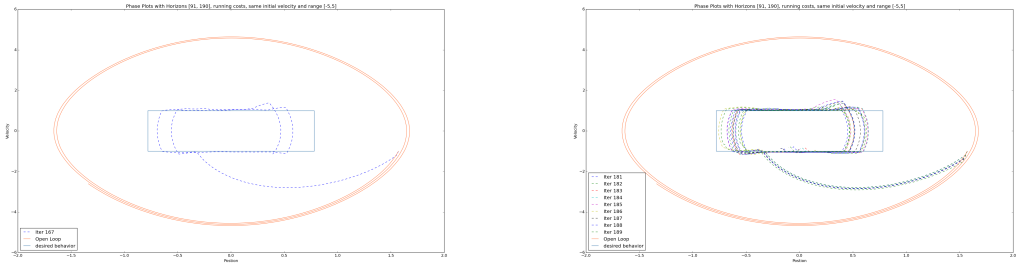
We can call the tracking energy system as an open-loop controller since it's objective is more inclined towards allowing natural behavior for the system. However, it might not always work for achieving other behaviors like crawling, bending in tensegrity systems. So, we wanted to design a feed-back controller that makes the pendulum swing with a desired velocity using ILQR.

- We want the pendulum to track a particular behavior, for example swing with constant angular velocity between both the horizontal ends, i.e $\frac{\pi}{4} \leq \theta \leq \frac{\pi}{4}$ and $\dot{\theta} = 1$.

- The custom behavior is determined by its cost function

$$g(\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{2} w_p * \frac{t}{N} \min(0, (\theta_t^2 - \theta_{goal}^2)^2) + \frac{1}{4} w_v \frac{t}{N} ((\dot{\theta}_t^2 - \dot{\theta}_{goal}^2)^2)$$

- The following plot shows the phase plot for tracking the above behavior.



(a) Horizon 167 (b) Horizons 180 to 190

Figure 3.18: Phase plot for Tracking Custom Behavior

- The following plot shows the control plot for tracking the above behavior.

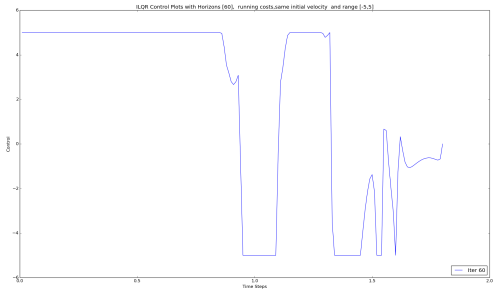


Figure 3.19: Control plot for Tracking Custom Behavior

- Observations:
  - From the phase and control plots, you can see that we are able to generate a controller that keeps the pendulum with in the desired limits.
  - The orange ellipse shows the natural motion of the pendulum, i.e behavior without any input and the blue rectangle shows the custom behavior that has to be tracked with ILQR.
  - The blue dotted lines depicts the output trajectory of ILQR which limits the pendulum to $-\frac{\pi}{4}$ and $\frac{\pi}{4}$ with constant angular velocity of 1. The adjacent phase plot shows the output of ILQR over few iterations.
  - The initial jumps in the control plot were mainly due to sudden change in the direction of velocities at the boundaries. However, the system attains enough momentum gradually so that controls need not be high as the horizon increases.
  - This shows that ILQR can be used to track various kinds of behavior by tuning the cost function appropriately which might not be achieved via other controllers.

3.6 Remarks

- **Advantages**
  - ILQR is a locally optimal feedback controller.
  - It does not involve computing complex gradients.
  - It can be extended to higher dimensional systems.
- **Disadvantages**
  - It is locally valid and need not work for other parts of the state space.
  - Cannot be extended for broader regions on the state space.
  - Sensitive to initialization as seen in the previous examples.
- Iterative LQR is capable of generating locally optimal (sometimes close to optimal) by estimating dynamics till the first order and cost function up-to the second order.
- However, linearizations at one part of the state space might not be directly applicable to other regions of the state space. This limits the strength of Iterative LQR to smaller or local validity.
- Hence, we need a mechanism to generate a global policy (need not be optimal) that has locally optimal properties. This can be achieved by a technique known as Guided Policy Search GPS by training the global policy on locally optimal controllers.

Chapter 4

Guided Policy Search

As seen in the previous section, local controllers cannot work for parts of the state space that is not covered by the initial controller (policy). There optimality is limited only to initial setting and they need not provide a fall back mechanism, if the system lends in an unknown space due to sensor noise or other irregularities. Since our objective is to generate locomotion in Tensegrity Robot, there are high chances that the system lands in parts of space that is out of context for the local policies. Hence, we need a mechanism to design globally valid controllers with feedback in case of unseen circumstances or issues.

Value function based Reinforcement Learning is powerful framework for controlling physical robots. Direct Policy Search methods are often employed for goal directed robotic motion and can easily be extended to higher dimensions with appealing convergence guarantees [18]. However, they require expert policy initialization to guarantee convergence otherwise they end up in poor local optima. Specialized innovative classes [19] have been developed to determine the initial policies. Nevertheless, these classes restrict the behaviors that can be learned. For example, a policy that tracks a single trajectory cannot pick different trajectories based on the state.

Guided Policy Search (GPS) can learn policies that are general and flexible representations, such as deep neural networks which can represent broad range of behaviors. GPS [20] uses Iterative Linear Quadratic Regulator [21] to generate "guiding samples". These samples assists policy search by exploring regions of high rewards with locally optimal behavior. Primitive GPS uses A variant of likelihood estimator known as important sampling incorporates these samples and guides the policy learning. The later versions of GPS [22], [20], [23] over come the need for importance sampling using

clustering and the concept of duality.

4.1 Preliminaries

Reinforcement Learning finds a parametrized policy π_θ to control an agent in a stochastic environment. At every time step, it observes a state $\mathbf{x}_t$, selects a control $\mathbf{u}_t$ based on the current policy $\pi(\mathbf{x}_t|\mathbf{u}_t)$ and produces a state transition $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$. The goal would be provided by reward function $r(\mathbf{x}_t, \mathbf{u}_t)$ and the objective would be maximize the expected sum of all the rewards for the horizon. We will use ζ to denote the sequence of state and control ($\zeta = \{(\mathbf{x}_t, \mathbf{u}_t)\}$) and $\pi(\zeta)$ denotes the probability under π.

Policy Gradients methods [18] optimize the expected return given by $E[J(\theta))]$ w.r.t to parameter θ to the learn the optimal policy. They do so by estimating the gradient of the return $E[\nabla J(\theta))]$ using the samples $\zeta_1, \zeta_2, \cdots, \zeta_m$ (where "m" denotes the number of sampled trajectories) drawn from the current policy π_θ and improve policy by taking a step in this gradient.

$$E[\nabla J(\theta))] = E[r(\zeta)\nabla \log \pi_\theta(\zeta)] \approx \frac{1}{N}\sum_{i=1}^{N} r(\zeta_i)\nabla \log \pi_\theta(\zeta_i)$$

In the above equation $\nabla \log \pi_\theta(\zeta_i)$ decomposes to $\sum_t \log \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ since the transition model $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ does not depend on θ. Standard likelihood ratio methods require samples from current policy at each gradient step. However, they cannot accept off-policy samples and require carefully chosen learning rate to ensure convergence.

Guided Policy Search generates samples from effective guiding distributions that guides search towards regions with high reward regions and at the same it favors regions with broader distributions. These samples make policy search less optimal to suboptimal experts. It uses a technique known as Kullback Leibler Divergence [24], which measures the distance between two probability distributions p and q. Since we are trying to estimate a global policy π from locally valid individual controllers p_i, we can use KL-Divergence to minimize their differences.

4.1.1 Kullback Leibler Divergences

Kullback-Leibler divergence (Kullback 1951) is an information-based measure of disparity among probability distributions. There are two types of KL measures that are usually minimized in probability theory and information theory.

1. **Moment Projection**[25]:

$$\arg\min_p KL(q||p) = \arg\min_p \sum_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})} \tag{4.1}$$

The above equation 4.1 tries to match the moments of q with moments of p which is almost similar to **Maximum Likelihood** estimate. p would be large where ever q is large.

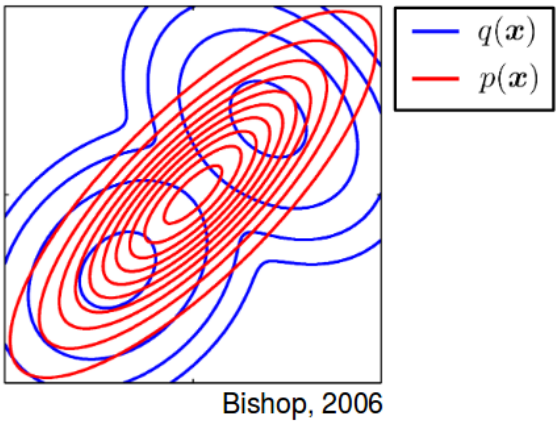


Figure 4.1: Plot showing moment projection of distribution q [1]

2. **Information Projection**(I-projection)[25][26]:

$$\arg\min_p KL(p||q) = \arg\min_p \sum_{\mathbf{x}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} \tag{4.2}$$

The above equation 4.2 forces p to be zero wherever q is zero. This guarantees **no wild exploration**. However, the I-projection is not unique for most of the distributions. The most interesting aspect of I-projection is that it contains the entropy of the distribution we are trying to estimate (here p) which is **important for exploration**.

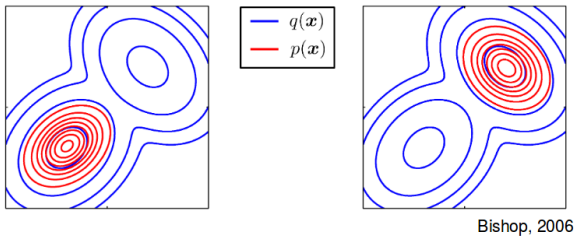


Figure 4.2: Plot showing I-Projection of distribution q [2]

4.1.2 Constructing Guiding Distributions

An effective guiding distribution would cover high reward regions while avoiding large $q(\zeta)$ densities, since they result in low importance weights. In this case a good guiding distribution would be an I-projection of $p(\zeta) \propto \exp(r(\zeta))$. An I-projection q of p minimizes the KL Divergence $\mathcal{D}_{KL}(q||p) = E_q[-r(\zeta)] - \mathcal{H}(q)$. The first term forces q to be high only in regions where reward is high and the second term maximizes the entropy favoring broader distributions. In the following section, we show that a good guiding distribution is an I-projection of $p(\zeta) \propto \exp(r(\zeta))$ which can be computed by Iterative LQR [15].

As we have seen previously, ILQR [15] outputs the deterministic optimal policy of the form

$$\pi_d(\mathbf{x}_t) = \bar{\mathbf{u}}_t + \mathbf{k}_t + \mathbf{K}_t(\mathbf{x}_t - \bar{\mathbf{x}}_t)$$

Under the framework of solvable MDPs [27] and the concept of maximum entropy control [28], we will use ILQR to build an approximate Gaussian I-projection of $p(\zeta) \propto \exp(r(\zeta))$. Let us denote the optimal policy in this case by π_G and augment the reward to include the KL Divergence of p and π_G.

$$\tilde{r}(\mathbf{x}_t, \mathbf{u}_t) = r(\mathbf{x}_t, \mathbf{u}_t) - \mathcal{D}_{KL}(\pi_G(.|\mathbf{x}_t)||p(.|\mathbf{x}_t)) \tag{4.3}$$

[2] Image Courtesy: Pattern Recognition and Machine Learning, Bishop 2006

If p is uniform the expected return of policy π_G is given by

$$E_{\pi_G}[\tilde{r}(\zeta)] = E_{\pi_G}[r(\zeta)] + \mathcal{H}(\pi_G). \tag{4.4}$$

The above eqn implies that if π_G maximizes the return, then it is an I-projection of p (if p is uniform). Ziebart (2010) [28] that the optimal policy under "uniform passive dynamics" is given by

$$\pi_G(.|\mathbf{x}_t)||p(.|\mathbf{x}_t)) = \exp(Q_t(\mathbf{x}_t, \mathbf{u}_t) - V_t(\mathbf{x}_t)). \tag{4.5}$$

where V is modified value function given by

$$V_t(\mathbf{x}_t) = \log \int \exp(Q_t(\mathbf{x}_t, \mathbf{u}_t) d\mathbf{u}_t.$$

Under linear dynamics under quadratic rewards, the optimal policy equation 4.6 is shown to be a linear Gaussian with mean $\pi_d(\mathbf{x}_t)$ and the covariance is given by $-Q^{-1}_{\mathbf{uu}t}$ i.e

$$\pi_G(.|\mathbf{x}_t)||p(.|\mathbf{x}_t)) = \mathcal{G}(\mathbf{u}_t; \pi_d(\mathbf{x}_t), -Q^{-1}_{\mathbf{uu}t}). \tag{4.6}$$

The above policy can be used to generate guiding samples. It should be noted that $\pi_G(\zeta)$ is only Gaussian under linear dynamics. When the dynamics are nonlinear, $\pi_G(\zeta)$ approximates a Gaussian around the nominal trajectory. Fortunately, the feedback term usually keeps the samples close to this trajectory, making them suitable guiding samples for the policy search.

4.1.3 Adaptive Guiding Samples

The distribution in the previous section consider only high reward regions often neglecting the current policy π_θ. So GPS optionally generates adaptive guiding samples to consider the current policy. In-order to do this, it runs ILQR with an augmented reward function i.e $\bar{r}(\mathbf{x}_t, \mathbf{u}_t) = r(\mathbf{x}_t, \mathbf{u}_t) + \log \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$. The resulting distribution would then be an I-projection of $p_\theta(\zeta) \propto \exp(\tilde{r}(\zeta))$.

4.2 Estimating Dynamics

- Since, we have established that local optimal policies take the form of linear gaussian controllers . We can estimate the dynamics using clustering.
- Simulate the system and collect $\{\mathbf{x_t^i}, \mathbf{u_t^i}, \mathbf{x_{t+1}}^\mathbf{i}\}$ pairs and cluster them using Gaussian Mixture Models GMM on the vectors.
- The conditionals $\mathbf{c_i}(\mathbf{x_{t+1}}|\mathbf{x_t}, \mathbf{u_t})$ represents linear-Gaussian dynamics while the marginals $\mathbf{c_i}(\mathbf{x_t}, \mathbf{u_t})$ specifies the region of the state-action space where this model is valid.

Algorithm 3 Guided Policy Search with unknown dynamics

1: **while** not converged **do**
2: Generate samples $\mathcal{D}_i = \{\tau_i^j\}$ from each linear gaussian policy $p_i(\tau)$
3: Fit the dynamics $p_i(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ using samples $\mathcal{D}_i$.
4: **C-step**: $p_i \leftarrow \arg\min_{p_i} \mathbf{E}_{p_i(\tau)}[\sum_{t=1}^{N} g(\mathbf{x}_t, \mathbf{u}_t)]$ such that $\mathcal{D}_{KL}(p_i(\tau)||\hat{p}_i(\tau)) \leq \epsilon$.
5: **S-step**: $\pi_\theta \leftarrow \arg\min_\theta \sum_{t,i} \lambda_{i,t} \mathcal{D}_{KL}(p_i(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)||p_i(\mathbf{x}_t, \mathbf{u}_t))$.
6: Increment the dual variables $\lambda_{i,t}$ by $\alpha \mathcal{D}_{KL}(p_i(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)||p_i(\mathbf{x}_t, \mathbf{u}_t))$
7: **end while**
8: **return** optimized policy parameters θ

4.3 Comparison with Previous Methods

Policy gradient methods cannot work with off-policy samples and higher order optimization methods like LBFGS. This makes them harder to find complex parametric policies [18]. Prior works have shown that importance sampling technique [[29], [30], [31]] can overcome these issues. These methods also recycle the samples where as Guided Policy Search would generate additional guiding samples that dramatically reduces the convergence time. Trajectory based Dynamic Programming (TBDP) [32] methods cannot learn arbitrary parametric policies while GPS can generalize on rough terrain.

Imitation methods like DAGGER [33] train a classifier in a supervised fashion on samples generated by expert policies. However, they are sensitive experts since they assume that DDP actions are optimal while they are valid only around the nominal

trajectory. Some methods do not admit off-policy samples while GPS maximizes the actual return using effective guiding distributions.

- **Advantages**
  - Does not involve gradient computations, since we use clustering (GMM) for estimating the dynamics.
  - No learning rate is required.
  - Better convergence time.
  - Less number of samples due to clustering.
- **Disadvantages**
  - Need good parameters for the Gaussian Mixture Models. Ex: Maximum number of clusters, number of samples per cluster, initial guess for mean and covariances.
  - Need sensible parameters for KL Divergence λ and step size ϵ in case of Mirror Descent Guided Policy Search [23].

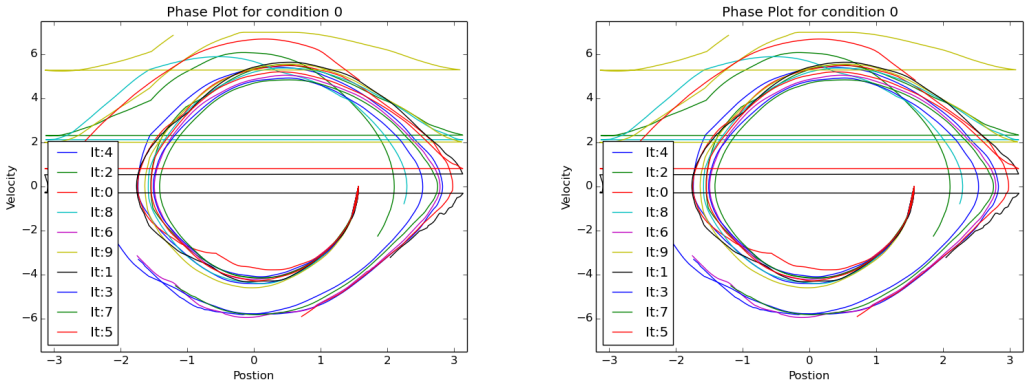
4.4 Experiments

We have evaluated a variant Guided Policy Search (GPS) [34] known as Mirror Descent Guided Policy Search (MDGPS) [23] on simple pendulum and Tensegrity Robots. MDGPS provides good improvement and convergence guarantees in simplified and convex systems and bounds the error in case of complex nonlinear systems. It closely represent Mirror Descent algorithm [35] that provides sub-gradients for non convex optimization. The output of MDGPS is a neural network policy that can be used off-line without any supervision.

4.4.1 Simple Pendulum

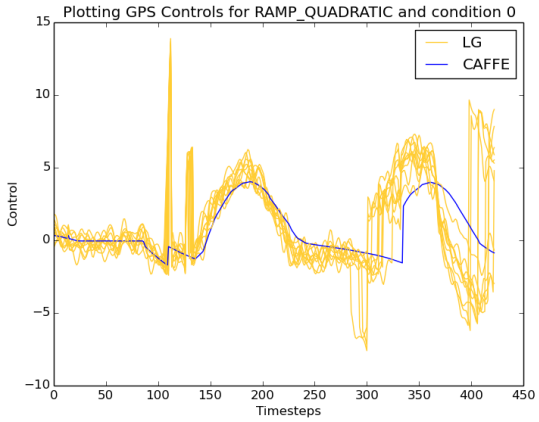
- We have evaluated Mirror Descent Guided Policy Search (MDGPS) framework [36] on a simple pendulum.3.1

- Caffe [37] is used as the neural network framework to learn the global policy.
- The objective is to output a global policy that tracks an input trajectory provided by a planner like **RRT**.

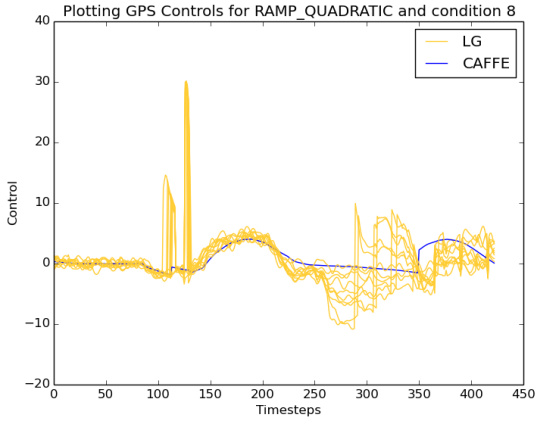


(a) Condition 0 (b) Condition 8

Figure 4.3: Phase Plot for Tracking State with Initialization using MDGPS



(a) Condition 0



(b) Condition 8

Figure 4.4: Control Plot for Tracking State with Initialization using MDGPS

- **Observations**
  - As you can see from the plot, MDGPS gives smoother trajectories and policies which can be applied on real robots.
  - It also bounds the divergence in the initial and final policies.
  - It does not require optimal initial policies and can work even if the policy is suboptimal. The horizons and controls were determined by the initial policy.
  - The final output policy can be off-line, i.e without intervention of planner.

4.4.2 Super Ball (Tensegrity Robot)

- The objective is generate policies that assist the locomotion of a tensegrity robot named SUPERball.
- The SUPERball is simulated using NASA Tensegrity Robotics Toolkit (NTRT) [38] which uses bullet (a Physics Engine) [39] for physics simulation (especially gravity and forces due to friction on individual rods and joints).
- The initial states of the robot correspond to 6 equilateral triangles (faces) that are found by randomly simulating the robot until stable equilibrium is reached.
- So we simulated the system using NTRT interface to such that robot moves from one equilibrium state to another until it reaches a state (isosceles face) that was not significantly represented in the training samples (i.e has not found a locally optimal solution for that specific state).

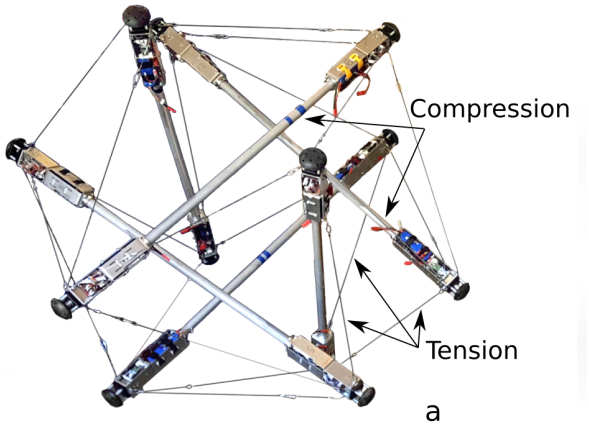


Figure 4.5: Super Ball - Tensegrity Robot [3]

[3]PRACSYS LAB: http://www.pracsyslab.org/tensegrity_planning

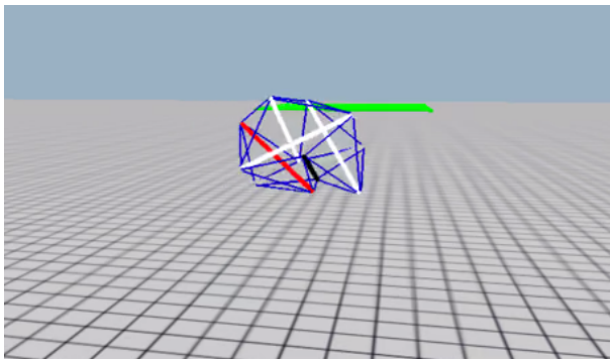


Figure 4.6: SuperBall Off-line Policy

- **Note**: Please check the additional files section to find the video on SUPERball titled *superball_offline.mp4*
- This specific controller (Neural Network) cover broader areas of the state space compared to the older methods.

4.5 Conclusion

- We were able to show the Iterative Linear Quadratic Regulators are effective in regulating both natural and custom behavior through fine tuned cost functions. The experiments include
  - Tracking a State
    - With policy initialization .
    - Without policy initialization.
    - Different Cost functions.
  - Tracking Natural motion using Energy levels.
  - Tracking Custom behavior.
- Generating smooth policies that guarantee no wild explorations for pendulum using Mirror Descent Guided Policy Search (MDGPS).
- Generate a policy for locomotion in SUPERball that can be used off-line without any supervision.

4.6 Future Work

Since the structure of SUPERball is highly symmetrical i.e many degrees of symmetry, we aim to take advantage of this information to learn policies with less data. Another line of work includes generating non-trivial gaits like crawling and jumping etc. An exciting avenue incorporates generalizing locomotion for multiple environments and or using larger neural networks and state space.

Chapter 5

Appendix

5.1 Taxonomy

- **Dynamical Systems**: A system where the position can be determined as a function of time.
- **State**: The internal representation of the system.
- **Policy**: A Policy represented by π is a mapping from state to control.
  - Deterministic Policy: $u = \pi(x)$
  - Stochastic Policy: $\pi(u|x) = \mathtt{P}[u|x]$ where $\mathtt{P}$ gives the probability of the control u given the state x .
- **Trajectory**: A sequence of states the system has to be follow.
- **Horizon**: The amount of time the system is defined for.
- **Time step (Simulation Step)**: The smallest incremental change in time.
- **Trajectory optimization**: The process of designing a trajectory that minimizes (or maximizes) some measure of performance while satisfying a set of constraints.

5.2 Theorems

- **Lemma 1**: For a square matrix M we have $M \geq 0$ if and only if for all vectors a, we have $a^T M a \geq 0$.

References

[1] K. Caluwaerts, J. Despraz, A. Işçen, A. P. Sabelhaus, J. Bruce, B. Schrauwen, and V. SunSpiral, "Design and control of compliant tensegrity robots through simulation and hardware validation," *Journal of The Royal Society Interface*, vol. 11, no. 98, p. 20140520, 2014.

[2] R. E. Skelton and M. C. de Oliveira, *Tensegrity systems*, vol. 1. Springer, 2009.

[3] Z. Littlefield, K. Caluwaerts, J. Bruce, V. SunSpiral, and K. E. Bekris, "Integrating simulated tensegrity models with efficient motion planning for planetary navigation," 2016.

[4] A. P. Sabelhaus, J. Bruce, K. Caluwaerts, P. Manovi, R. F. Firoozi, S. Dobi, A. M. Agogino, and V. SunSpiral, "System design and locomotion of superball, an untethered tensegrity robot," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pp. 2867–2873, IEEE, 2015.

[5] P. Abbeel, "Cs 287: Advanced robotics," 2012.

[6] S. Boyd, "Ee363: Linear dynamical systems," 2008.

[7] A. O'Dwyer, *Handbook of PI and PID controller tuning rules*. World Scientific, 2009.

[8] D. P. Atherton and S. Majhi, "Limitations of pid controllers," in *American Control Conference, 1999. Proceedings of the 1999*, vol. 6, pp. 3843–3847, IEEE, 1999.

[9] S. W. Sung and I.-B. Lee, "Limitations and countermeasures of pid controllers," *Industrial & engineering chemistry research*, vol. 35, no. 8, pp. 2596–2610, 1996.

[10] O. Von Stryk and R. Bulirsch, "Direct and indirect methods for trajectory optimization," *Annals of operations research*, vol. 37, no. 1, pp. 357–373, 1992.

[11] D. Mayne, "A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems," *International Journal of Control*, vol. 3, no. 1, pp. 85–95, 1966.

[12] D. H. Jacobson and D. Q. Mayne, "Differential dynamic programming," 1970.

[13] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems.," in *ICINCO (1)*, pp. 222–229, 2004.

[14] L.-z. Liao and C. A. Shoemaker, "Advantages of differential dynamic programming over newton's method for discrete-time optimal control problems," tech. rep., Cornell University, 1992.

[15] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 4906–4913, IEEE, 2012.

[16] J. J. Moré, "The levenberg-marquardt algorithm: implementation and theory," in *Numerical analysis*, pp. 105–116, Springer, 1978.

[17] D. P. Bertsekas, *Nonlinear programming*. Athena scientific Belmont, 1999.

[18] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural networks*, vol. 21, no. 4, pp. 682–697, 2008.

[19] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Movement imitation with nonlinear dynamical systems in humanoid robots," in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, vol. 2, pp. 1398–1403, IEEE, 2002.

[20] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.

[21] E. Todorov and W. Li, "A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *American Control Conference, 2005. Proceedings of the 2005*, pp. 300–306, IEEE, 2005.

[22] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," in *Advances in Neural Information Processing Systems*, pp. 1071–1079, 2014.

[23] W. H. Montgomery and S. Levine, "Guided policy search via approximate mirror descent," in *Advances in Neural Information Processing Systems*, pp. 4008–4016, 2016.

[24] J. M. Joyce, "Kullback-leibler divergence," in *International Encyclopedia of Statistical Science*, pp. 720–722, Springer, 2011.

[25] J. Peters and G. Neumann, "Policy search: Methods and applications," 2015.

[26] C. M. Bishop, "Pattern recognition," *Machine Learning*, vol. 128, pp. 1–58, 2006.

[27] K. Dvijotham and E. Todorov, "Inverse optimal control with linearly-solvable mdps," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)* (J. Fürnkranz and T. Joachims, eds.), pp. 335–342, Omnipress, 2010.

[28] B. D. Ziebart, "Modeling purposeful adaptive behavior with the principle of maximum causal entropy," 2010.

[29] L. Peshkin and C. R. Shelton, "Learning from scarce experience," *arXiv preprint cs/0204043*, 2002.

[30] J. Kober and J. Peters, "Learning motor primitives for robotics," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pp. 2112–2118, IEEE, 2009.

[31] T. Jie and P. Abbeel, "On a connection between importance sampling and the likelihood ratio policy gradient," in *Advances in Neural Information Processing Systems*, pp. 1000–1008, 2010.

[32] C. G. Atkeson and C. Liu, "Trajectory-based dynamic programming," in *Modeling, Simulation and Optimization of Bipedal Walking*, pp. 1–15, Springer, 2013.

[33] S. Ross, G. J. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning.," in *AISTATS*, vol. 1, p. 6, 2011.

[34] S. Levine and V. Koltun, "Guided policy search.," in *ICML (3)*, pp. 1–9, 2013.

[35] A. Beck and M. Teboulle, "Mirror descent and nonlinear projected subgradient methods for convex optimization," *Operations Research Letters*, vol. 31, no. 3, pp. 167–175, 2003.

[36] C. Finn, M. Zhang, J. Fu, X. Tan, Z. McCarthy, E. Scharff, and S. Levine, "Guided policy search code implementation." `http://rll.berkeley.edu/gps`, 2016.

[37] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[38] NASA, "Nasa tensegrity robotics toolkit (ntrt)." `http://ti.arc.nasa.gov/tech/asr/intelligent-robotics/tensegrity/ntrt`.

[39] A. Boeing and T. Bräunl, "Evaluation of real-time physics simulation systems," in *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, pp. 281–288, ACM, 2007.