# Real Time Optimal Control for Uncertain Systems using Reinforcement Learning

By

**Abdul Sami**

Supervisor

**Dr. Attaullah Memon**

Department of Electronics and Power Engineering

Pakistan Navy Engineering College (PNEC)

National University of Sciences and Technology (NUST)

Islamabad, Pakistan

July 2019

# Real Time Optimal Control for Uncertain Systems using Reinforcement Learning

By

**Abdul Sami**

Supervisor

**Dr. Attaullah Memon**

A thesis submitted to conform the requirements of

a *Master of Science* degree in

Electrical (Control) Engineering

Department of Electronics and Power Engineering

Pakistan Navy Engineering College (PNEC)

National University of Sciences and Technology (NUST)

Islamabad, Pakistan

July 2019

# Declaration

I, *Abdul Sami* declare that this thesis work bearing the title "Real Time Optimal Control for Uncertain Systems using Reinforcement Learning" is the result of my genuine research and any material or content in it, unless clearly referred, is my work solely written and presented by myself.

I confirm that:

1. This work is mainly done while in candidature for a MS Degree in Electrical Engineering at National University of Sciences and Technology.
2. No part of this thesis has previously been submitted for any degree at any other institution including this one.
3. Wherever I consulted from other published work, I have clearly attributed it.
4. Where I have quoted from the work of others, the source is always given.
5. I have acknowledged all main sources of help.
6. I have made clear exactly the work done by others and the contribution of this work.

Abdul Sami,

# Copyright Notice

This thesis is dedicated to *my beloved parents*

# Abstract

The thesis explores the area of Reinforcement Learning (RL) that is emerging as an elegant tool in solving the control problems that require optimality and robustness simultaneously. The task of stabilization of an inverted pendulum system with known/unknown internal dynamics is discussed to reveal the advantages of RL approach over conventional approach and is demonstrated using simulations. Also discussed are the algorithmic challenges faced by the designer in using the RL approach for online robust optimal control of unknown systems.

**Keywords:** *Reinforcement Learning, Optimal Control, Inverted Pendulum System, Real Time Control*

# Acknowledgments

First and the foremost, I would like to thank ALLAH Almighty for everything, for His endless bounties and providing me the strength as well as abilities to accomplish this task, that would not have been achieved without His blessings.

I would like to thanks my supervisor *Capt. Dr. Attaullah Y. Memon PN* for his invaluable advice, sincere unconditional support and guidance all the way to complete this task successfully. I will be extremely grateful for his encouragement.

I would also like to thank my Guidance and Evaluation Committee members *Capt. Dr. Sajjad Haider Zaidi PN* and *Cdr. Dr. Rana Hammad Raza Khan PN* for their consent to work as my GEC members, reviewing my thesis and providing with vital comments.

Finally I feel gratitude to thank my parents and my family especially my grant parents for their love, support and encouragement.

# Contents

# List of Figures

# List of Abbreviations and Symbols

## Abbreviations

| | |
|---|---|
| **RL** | Reinforcement Learning |
| **PI** | Policy Iteration |
| **VI** | Value Iteration |
| **ARE** | Algebraic Riccati Equation |
| **TD** | Temporal Difference |
| **DOF** | Degree of Freedom |
| **CT** | Continuous Time |
| **LS** | Least Squares |
| **VFA** | Value Function Approximation |
| **SDP** | Sequential Decision Problem |

Chapter 1

# Introduction

## 1.1 Background

Control Systems are designed on the basis of mathematical model of some real physical system whereas they are implemented on the physical system itself to achieve the desired and up to the mark system behavior.

However, it is then obvious that any errors in the mathematical model will result in degraded performance of the designed controller when applied for the control of the actual physical system.

Despite the high level of efforts that is put up into system modeling, some degree of error is always present due to the following reasons.

- The physical system may have complicated and/or hidden dynamics that are difficult to be incorporated into the mathematical model.
- Even if in the mathematical model, all the system dynamics of the physical system have been accounted for, yet some or all of those dynamics may not remain the same due to their time varying nature or due to their degradation/deration over long lengths of time.
- Real systems, unlike mathematical models, are subjected to external unknown disturbances that may change the dynamics of the system in an unpredictable manner and thus the mathematical model may not be a good representation of the system working in a real environment.

Considering the above, it is absolute that one has to put enough flexibility into the controller itself that can tackle the above hidden or varying system dynamics, yet achieve the desired overall control performance. This is where the field of *Robust Control* plays its role in the development of such controllers. Robust control aims to develop control systems that can perform the desired control tasks with bounded uncertainties in system models and/or bounded disturbances offered by the system environments.

However, a *Robust Control's* main task is to make sure that the plant remains stable despite the constraint that the plant dynamics are not known and that the controller is authorized to use as much control resources as is necessary to handle the unknown system. This is where the designers who are much more concerned about the utilization of control resources don't find the domain of Robust Control an all appealing solution. Instead, they are interested in solutions that give the optimal system performance with utilization of minimal control resources.

This is where the field of *Optimal Control* comes into play in which such controls are sought that minimize user defined cost functions while achieving desired optimal system performance.

However, the dilemma is that Optimal Control design requires the use of absolute plant dynamics i.e. one must have an exact mathematical model of the real system to design the best (optimal) control w.r.t the user defined cost function that can achieve the desired control objective.

It is therefore pretty clear that one must sacrifice the Robustness property in the design if optimal solutions are sought. On the contrary, the designer must be willing to pay all the control costs if a fully robust design is needed.

This is where Reinforcement Learning solves the dilemma.

These are control methodologies developed from the work of Computational Intelligence in which the control systems are intelligent enough to adapt to changing environment/systems and learn to optimize the control design with more and more training.

Control Design using Reinforcement Learning concepts is not new in the Control Systems community. However, as we surveyed the various literature in our research, we felt that the idea requires more explanations as to how useful is the concept in a more practical manner. Further, the algorithms that have already been developed utilizing this concept require demonstrations with concrete examples.

## 1.2 Literature Review

The optimal control problem in the domain of control theory has been studied for decades. However, it has been an offline design with full knowledge of system dynamics and there did not exist concept of online and real time design of optimal control.

In this thesis, we are primarily focusing the work of *F. Lewis* & *D. Vrabie* in their work [1], [2] where the concept of Reinforcement Learning has been used to allow online optimal control design with the added advantage that system dynamics are not needed and all requisite information can be derived from system real time data in an online fashion.

The algorithms to find the solutions of *The Optimal Control Problem* have already been developed such as *Policy Iteration* [3],[4],[5], *Value Iteration* [6], *Generalized Policy Iteration* [4].

However, the optimal control design in the control systems community was an offline procedure until the work of [2] wherein the above mentioned algorithms were extended to allow an online real time design of optimal controllers. Various researches have been conducted to implement these algorithms for the online optimal control of systems such as *AUVs* [7], Hypersonic Vehicle Tracking Control [8] and others.

However, to the best of our knowledge, non have approached the problem from a cost effective point of view. Majority have harnessed the data driven adaptable nature of Reinforcement Learning Control to improve the controller performance in terms of computational efficiency, zero steady state error and high quality transient performance compared to the conventional and traditional methods.

They claim and show that the solutions they have found using the RL approach are optimal and are indeed optimal when compared with the identical results of conventional methods. However, as we show in this thesis that offline planning is not the same as online learning. That an optimal control found using a poor algorithm may be more non-optimal than the regular non-optimal control itself. That is, the high performance achieved using the RL concepts is just one aspect and equally important is the consideration of the cost of the algorithm that is used to find the optimal control.

## 1.3 Publication resulted from this work

The work was submitted in and accepted by *ANZCC 2018*, was published in the Conference Proceedings and made available on *IEEE Xplore Digital Library*

A. Sami, Attaullah Y. Memon, "Robust Optimal Control of Continuous Time Linear System Using Reinforcemeng Learning", *Australian & New Zealand Control Conference (ANZCC) Swinburne University of Technology Melbourne, Australia, Dec 7-8, 2018*

## 1.4 Thesis Objectives

The objectives of this thesis are,

- We have demonstrated and discussed in detail the Policy Iteration method of solving the optimal control problem for a benchmark system namely, the stabilization of the Inverted Pendulum system.
- The problem has been approached in both the offline and online manners, the former requiring the knowledge of system's internal dynamics whereas the latter does not. A comparison is given between the offline and online designs highlighting the pros and cons of both and the importance of the costing of online algorithms.
- Challenges that a designers needs to face while designing online RL algorithms are discussed and form a major contribution of this work.

## 1.5 Thesis Outline

- Chapter 2 gives a basic introduction on Reinforcement Learning defining various terminologies, concepts and algorithms from the developed literature necessary to get a sound understanding of the work that follows in the subsequent chapters.
- Chapter 3 starts with the definition of The Optimal Control Problem followed by conventional and offline/online Reinforcement Learning methods for the solution of this problem.
- Chapter 4 gives our simulations of the conventional and Reinforcement Learning methods applied for the stabilization of the Inverted Pendulum System.

- Chapter 5 gives a discussion on the main extracts of this research followed by a conclusion of our work.

Chapter 2

# Concept Building

## 2.1 Defining Reinforcement Learning...

Living organisms tend to interact with their environment and use those interactions to improve their own actions in order to survive and increase. This "Modification of Actions" is an indicator that the organisms have learned something and are now inclined for such interactions that result in more favorable outcomes. The more the interactions, the more the organisms get to know their environment and it becomes very likely that the organism learn the best actions that result in outcomes in their best interest in that particular environment. However, as soon as the environment changes for the organisms, the entire learning cycle starts all over again.

This behavior and method of living organisms to survive is the source of inspiration to develop intelligent man made systems that learn and modify their behavior the same way living organisms do in order to achieve the desired objectives in most favorable manner in the worst environmental conditions. The *Modification of Actions* by the organisms (*Agent*) based upon receiving some stimulus (*reward* or *punishment* collectively known as *reinforcement*) from the environment is defined as *Reinforcement Learning.* Fig. 2.1 summarizes this definition of RL.

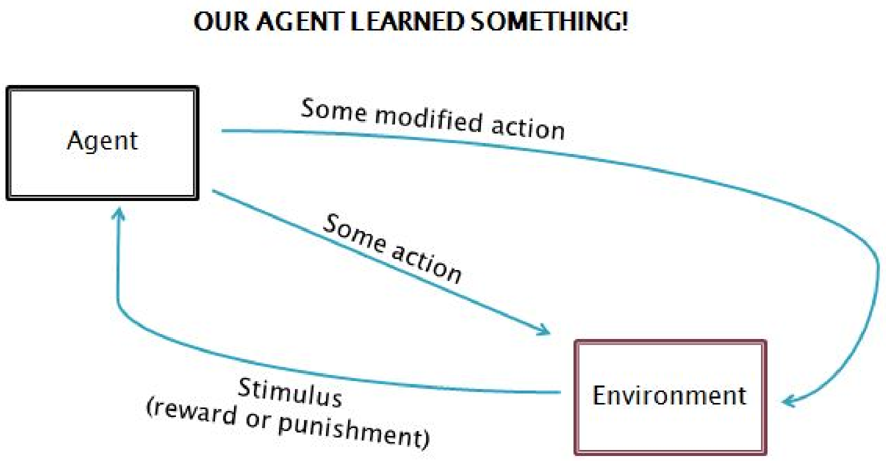

**Figure 2.1:** Reinforcement Learning Defined

## 2.2 More Defining Terms

### 2.2.1 Sequential Decision Problem

Suppose we have an agent that has been placed in a 4 x 3 environment, has 11 number of accessible states with two of them being the terminal states as shown in Fig. 2.2

The agent may start from any state; say $(1,1)$ and the task of this agent is to reach the goal state of $+1$. The agent can select any of the four available actions namely; UP, LEFT, DOWN, RIGHT. This defines a Sequential Decision Problem $(SDP)$ in which an action is to be selected at each time step by the agent in order to reach the goal state from the starting position.

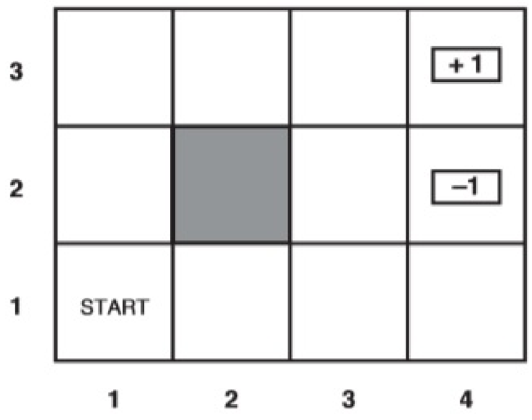

**Figure 2.2:** 4 x 3 Environment

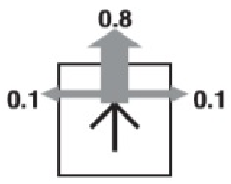

**Figure 2.3:** Transition Model

### 2.2.2 Policy

For a deterministic environment, easy solutions are readily available for the SDP, one of which in our example can be $(Up, Up, Right, Right, Right)$. However, the environment won't usually be that friendly to such solutions. That is, the actions are unreliable and an action may result in a different outcome other than the intended outcome. There might be some model of stochastic motion such as the one in Fig. 2.3. This particular model of stochastic motion reveals that any action taken to move the agent in a particular direction has a 0.8 probability that the agent will move in the desired direction and a 0.1 respective probability that the agent will move sideways. The agent might reach states that were not part of the proposed solution in which it started from a specific state and was meant to reach the goal by following a pre-defined number and sequence of states. For any state other than the intended states, the agent will have no clue on what action to take to reach the goal. That is, assuming the worst case scenario, an action for every possible state must be defined. A solution of this kind is known as a *Policy*. Stated formally,

*"A policy is a functional map that assigns an action to every state."*

$$u = h(s) \tag{2.2.1}$$

where u is the applied action, h represents the policy and s is the state of the agent

Fig. 2.4 is an example of a policy.

### 2.2.3 Reward

As mentioned in the definition of section 2.1, the modification of the agent's actions is to result in favorable outcomes. However, the agent has no knowledge about what

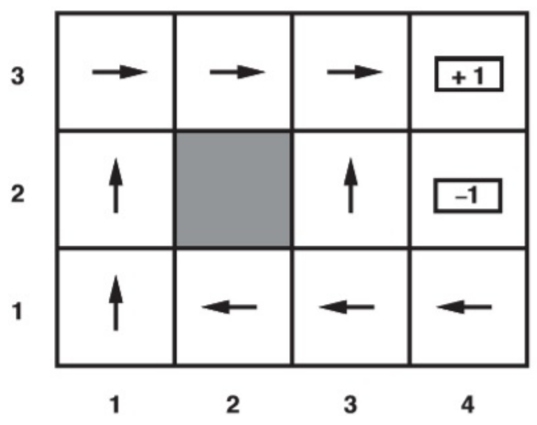

**Figure 2.4:** Policy

its actions do. A reinforcement learning agent will select at random from a set of available controls until it has accidentally selected the control that results in a favorable outcome upon which the agent receives some good stimulus from the environment telling the agent that the particular action has produced a favorable outcome and is likely to be selected again in the future for that specific state of the agent. Likewise, all other actions that resulted in unfavorable outcomes result in reception of bad stimulus by the agent from the environment and are less likely to be used again for the same states. That is, given appropriate stimulus, one can force the agent/controller to select the actions/controls that produce the most favorable outcomes. Such stimulus that is given to the agent to characterize its action as good or bad is known as *Reward* or *Reinforcement.* It is up to the designer whether the rewards are to be given at each step and direct the agents behavior throughout the course or a single reward to be given in the end and leave the agent on its own to decide its course for reaching the goal. This is however, different than supervised learning in which an agent is told the correct action/control for every encountered state. Rewards may be positive or negative respectively representing a favorable or unfavorable outcome. Negative rewards may also be considered as punishments.

One may also consider using a *Reward Function* that assigns a reward for every state. That is,

$$Reward = RewardFunction(State)$$

The reward function will be entirely responsible for shaping the behavior of the agent in the specific environment and any required control performance from the agent should be directly or indirectly incorporated in the reward function.

As an example, let us again consider the Fig. 2.2 in which rewards of +1 and -1 respectively are given to the agent on the terminal states. Let us further consider that the agent receives a negative reward of -0.04 for every other non-terminal state. We can program the agent to look for positive rewards and try to avoid negative rewards. This will motivate an agent in the START state of (1,1) to go into other states to avoid the negative rewards until it accidentally reaches the state of +1 representing success or -1 representing failure.

### 2.2.4 Utility

As mentioned in section 2.2.3, the negative reward of the non-terminal states forces the agent to go into other states since it has been programmed to avoid negative rewards. However, it is very obvious that there is no guarantee if the agent will ever end up in the terminal state or how many states will it go into before ending up in the terminal state. Even if the agent ends up in the terminal state after exploring a large number of random states, such reaching of the goal state is not an achievement at all since much control resources/actions have been expended in doing so and every action taken to go to next state counts as a control resource expenditure. Rather, a designer would be interested if the agent can reach the terminal state as soon as possible or at least if it reached the goal state after going through a large number of states the first time, it should be intelligent enough to learn from any previous false attempts and avoid repeating them and reach the goal state in a comparatively lesser number of states the second time.

Such a programming of the agent can be possible if one introduces the concept of *Utility*, which is stated as,

> *"Utility is the total reward accumulated by an agent that is using a specific policy from a given start state to the end state."*

Then, success or failure of the agent is based upon the total reward accumulated for all the states it went through including the start and the terminal state and the agent is said to have learned if it accumulates a greater utility the second time.

In our example of section 2.2.3, the longer the agent stays in the non-terminal states, the more negative reward it accumulates resulting in an overall less value of total reward.

The agent will be programmed to remember any actions in a specific state that resulted in more accumulation of negative reward and will avoid repeating the same actions in the same states again the second time. This way the agent's behavior will keep on improving with every same situation it encounters, that is, it will start to learn the optimal action for every state that results in an overall increased total reward. That is, with more and more *training*, the agent will be able to maximize this value of utility by learning an *Optimal Policy* such as the one in Fig. 2.4 for our example.

There are two ways in which accumulation of rewards takes place, that is, *Additive Rewards* & *Discounted Rewards*.

In **Additive Rewards**, one simply takes the sum of all the rewards for all the states the agent went through to calculate the utility. That is,

$$Utility(State_0) = Reward(State_0) + Reward(State_1) + Reward(State_2) + ... \tag{2.2.2}$$

In **Discounted Rewards**, one introduces a Geometric Series of a discounting factor, say $\gamma$, where $(0 < \gamma < 1)$, such that the terms of the additive reward series are weighted by the corresponding terms of the geometric series to form a compound convergent series. That is,

$$Utility(State_0) = Reward(State_0) + \gamma.Reward(State_1) + \gamma^2.Reward(State_2) + ... \tag{2.2.3}$$

It is important here to highlight that the utility for a given state sequence depends upon the specific policy followed, since this policy is directly responsible for generating this state sequence. In other words, the utility is a representation of the *Quality of a Policy* and can be used to *evaluate* a policy. However, if one assumes a model of stochastic motion such as the one in Fig. 2.3, then the same policy can yield different state sequences and different values of utility in different attempts by the agent to reach the goal, in which case, one can use the *Expected Value* of utility to characterize the policy. The utility of a policy can be calculated for every state of the environment that the agent has access into considering it to be the start state and from then onwards till the end state. Fig. 2.5 gives an example.

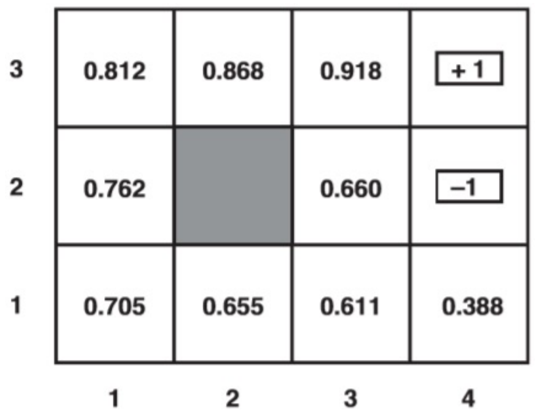

**Figure 2.5:** Utility

The utility of a policy may be finite or infinite depending upon the type of utility (additive or discounted) and the type of horizon (finite or infinite) considered.

For a finite horizon case, the agent has a limited time in which it has to reach the goal state and the expiration of this time results in an immediate failure of the agent. With such time limits, the optimal action from a given state to reach the goal may change with the remaining available time.

That is, the agent will be willing to take all the risks to reach the goal quickly if very short time is remaining and modify its actions to take the shortest path and vice versa. That is, the optimal policy is dependent upon time in the finite horizon case.

On the contrary, for an infinite horizon, the agent may reach the goal in an indefinite amount of time. That is, the agents's preferences to choose among the actions and resulting next states are time independent and so is the optimal policy. However, as mentioned previously in the beginning of this section, there is no guarantee as to how many number of states the agent will go through before reaching the goal state and the utility values may diverge towards infinity for infinite state sequences in case of additive rewards. This is where the concept of discounted rewards becomes more useful and one can get finite values of utility even for infinite state sequences since the constraint on the discounting factor ($0 < \gamma < 1$) forces the series to be a convergent series.

In this work, we consider the infinite horizon case with discounted reward method of calculation of utilities. Further, we also assumes that actions do result in intended outcomes and no such model of stochastic motion is considered such as that in Fig. 2.3.

### 2.2.5 Dynamic Programming

Originally coined by Richard Bellman in the 1950s, the word Dynamic Programming was first used to describe a multistage decision process. Unrealted to computer programming, dynamic programming instead was used to represent *planning* and the word *dynamic* indicated a time varying process since it occurred in multiple stages.

Dynamic Programming ia powerful algorithmic problem solving technique suitable for problems that exhibit two characteristics,

1. Overlapping Sub-problems
2. Optimality

A problem is said to exhibit overlapping sub-problem characteristic if it can be broken down into simpler sub-problems, which are not independent from each other, however which may be solved and the solutions combined to obtain the solution of the overall problem. The idea here is that smaller solutions of the sub-problems may be obtained once and then stored for a later reference for the solution of the overall initial problem. This is what is meant by dynamic programming that the solutions of the sub-problems may be used as many times as needed in the solution of the overall problem, however, no re-calculation of the solution of any sub-problem takes place. Clearly, this will allow a much efficient and fast solution of the overall problem.

Similarly, if a recurrence relation can express the solution of an overall problem, then the problem is said to exhibit the characteristic of optimality. The following sections describe more on these recursions.

## 2.3 The Bellman Equation

As mentioned in the previous section, when the agent is using a specific policy, one can calculate the value of utility for every state the agent has access into, however, this calculation will require multiple attempts/training by the agent in which it has to start from a specific state and reach the goal state by using the specific policy. And this needs to be done for all the 11 states as shown in Fig 2.5 considering each state as the start state in the specific training episode. It is then pretty obvious that the problem of calculating the utility of a policy is multiplied with the increase in the possible number of states of the environment and the number of training attempts to calculate the utilities for all the states will be as many as the number states of the environment.

This problem can be simplified if one keenly observes a close connection between utilities of states. That is,

> The utility of a state is the sum of the reward for that state plus the discounted utility of the subsequent state when the agent is using a fixed policy.

If this is kept under consideration, then one can calculate the utilities of the current states by knowing the reward for that state and the utilities of the next states. That is, we can rewrite Eq. 2.2.3 as,

$$Utility(State_0) = Reward(State_0) + \gamma.Utility(State_1) \tag{2.3.1}$$

or more generally, if we denote the present state by $s$, the next state by $s$', $V$ represents the utility or value function and $r$ denotes the reward function, then, it becomes,

$$V(s) = r(s) + \gamma.V(s') \tag{2.3.2}$$

This is known as the *Bellman Equation.* Calculating the values of utilities using Bellman Equation is much more easier since one only need to know one equation for each state and a simultaneous solution of all the equations gives the utilities of all the states together at once.

As mentioned before, the purpose of introducing the concepts of rewards and utilities is to modify agent's actions so as to result in favorable outcomes. The purpose to evaluate a policy by the utility function is to know if there exist any better policy than the one currently known. That is, we are looking for the policy that can give the greatest utility (maximum desired performance). Mathematically, this means that we are looking for,

$$V^*(s) = \max_{h(.)}(r(s) + \gamma.V(s')) \tag{2.3.3}$$

At this point, we make another bold assumption for our work that rewards are perceived in terms of costs. This means that the agent is receiving negative values of rewards that depend upon an action (u) taken in a particular state (s) and our task is to actually maximize the overall negative utility (which means that we are trying to reduce the costs). With this assumption, Eq. 2.3.3 becomes,

$$V^*(s) = \max_{h(.)}(r(s,u) + \gamma.V(s')) \tag{2.3.4}$$

where; r(s,u) is known as the *one step cost of control* and the utility or value is known as the *cost to go* or total cost.

This maximization problem is generally difficult to solve for general non-linear systems. See [1]. A further simplification can be made if we recall the famous *Bellman Optimality Principle* which states,

> An optimal policy has the property that no matter what the previous decisions (i.e. controls) have been, the remaining decisions must constitute an optimal policy with regard to the state resulting from those previous decisions.

A direct consequence of this principle is that Eq. 2.3.4 becomes,

$$V^*(s) = \max_{h(.)}(r(s,u) + \gamma.V^*(s')) \tag{2.3.5}$$

known as the *Bellman Optimality Equation* and gives us the *Optimal Value* of states. That is, the utilities of states of an optimal policy are connected. The corresponding *Optimal Policy* is,

$$u(s) = \arg\max_{h(.)}(r(s,u) + \gamma.V^*(s')) \tag{2.3.6}$$

It is much easier to determine optimal controllers using Eq. 2.3.5 and 2.3.6 compared to Eq. 2.3.4. However, one thing to now observe is that Eq.2.3.2 had a linear structure that allowed a simultaneous solution of the Bellman Equations to obtain a linear solution. However, Eq. 2.3.5 has a non-linear structure since the *max* operator is a non-linear operator and the simultaneous solution of the Bellman Equations is not possible. Instead, well known *iterative algorithms* do exits for the solutions of the Bellman Optimality Equations known as the *Value Iteration* and *Policy Iteration* which we discuss next.

## 2.4 The Value Iteration Algorithm

The value iteration algorithm belongs to a family of iterative algorithms that allow the solution of non-linear equations such as the Bellman Optimality Equations. As stated in the previous section, Eq. 2.3.5 is used to find the optimal value. In the value iteration algorithm, one can start with arbitrary initial values of the utilities of states in Eq. 2.3.5, plug them on the R.H.S of the equations and obtain updated values of utilities on the L.H.S. Then the updated values obtained on the L.H.S are again plugged in on the R.H.S to obtain new estimates of utility values. This procedure is repeated until the utility values on the L.H.S reach an equilibrium at which point, the optimal values of the utility of states has been found. Each step of the utility update is known as the *Bellman Update*.

The above iterative procedure is guaranteed to converge to optimal values of utilities of states. This is because the Bellman Equation and the Bellman Optimality equation are *fixed point* equations and form a *contraction* that is guaranteed to reach a fixed point whenever the constraint on the discounting factor $(0 < \gamma < 1)$ is taken care of. The following sub-section gives a proof of this claim.

### 2.4.1 Convergence of the Bellman Equation

To support our proof of the claim that Bellman Equation and Bellman Optimality Equation are fixed point equations, Let us consider two arbitrary functions $f$ & $g$ and let $\max f(a)$ & $\max g(a)$ represent the maximum values of these functions at some arbitrary input arguments $a$. Also let us assume that the maximum value of $f$ is greater than that of $g$, then,

$$|\max f(a) - \max g(a)| = \max f(a) - \max g(a)$$

Suppose the maximum value of $f$ occurs at some input argument, say; $a^*$, then,

$$\max f(a) = f(a^*)$$

Then, we can write,

$$|\max f(a) - \max g(a)| = f(a^*) - \max g(a)$$

If we put the same $a^*$ in the input argument of $g$ as well, then the R.H.S of above will be less than or equal to it. That is,

$$\because g(a^*) \leqslant \max g(a)$$
$$\therefore |\max f(a) - \max g(a)| \leqslant f(a^*) - g(a^*)$$

And,

$$\because f(a^*) - g(a^*) \leqslant \max |f(a) - g(a)|$$
$$\therefore |\max f(a) - \max g(a)| \leqslant \max |f(a) - g(a)|$$

In English, this means that,

> The difference between the maxima of two functions is less than or equal to the maximum difference between the functions.

Secondly, Let us view the Bellman Equation 2.3.5 as an operator $B$ which when applied to a utility estimate $V_i$ gives an updated estimate $V_{i+1}$. That is,

$$V_{i+1} \leftarrow BV_i$$

Let $V_i$ and $V_i'$ be any two utility estimates and $BV_i$ and $BV_i'$ respectively be their updated estimates, then, from the notion of a contraction,

$$||BV_i - BV_i'|| \leqslant \gamma.||V_i - V_i'||$$

where; $||.||$ represents the *maxnorm*.

That is, the utility estimates must converge to a fixed point with repeated applications of the Bellman Operator.

Proof:

$$
\begin{aligned}
|BV_i(s) - BV_i'(s)| &= |\max_u(r(s,u) + \gamma.V_i(s')) - \max_u(r(s,u) + \gamma.V_i'(s'))| \\
|BV_i(s) - BV_i'(s)| &\leqslant \max_u |r(s,u) + \gamma.V_i(s') - (r(s,u) + \gamma.V_i'(s'))| \\
|BV_i(s) - BV_i'(s)| &\leqslant |r(s,u^*) + \gamma.V_i(s') - (r(s,u^*) + \gamma.V_i'(s'))| \\
|BV_i(s) - BV_i'(s)| &\leqslant |r(s,u^*) + \gamma.V_i(s') - r(s,u^*) - \gamma.V_i'(s'))| \\
|BV_i(s) - BV_i'(s)| &\leqslant |\gamma.V_i(s') - \gamma.V_i'(s')| \\
|BV_i(s) - BV_i'(s)| &\leqslant \gamma.|V_i(s') - V_i'(s')|
\end{aligned}
$$

From the definition of maxnorm we have,

$$
\begin{aligned}
||BV_i - BV_i'|| &= \max_s |BV_i(s) - BV_i'(s)| \\
||BV_i - BV_i'|| &\leqslant \gamma.\max_s |V_i(s') - V_i'(s')| \\
||BV_i - BV_i'|| &\leqslant \gamma.||V_i - V_i'||
\end{aligned}
$$

That is,

> "The Bellman Operator is a contraction by a factor of $\gamma$. Whenever this operator is applied on two vectors spaced by some distance, then the spacing between their Bellman Updated versions is reduced by a factor of at least $\gamma$. This implies that repeated applications of the Bellman operator cause the vectors to converge at a common point. Likewise, if one of the vectors is held fixed, known as the *fixed point*, then with the repeated applications of the Bellman Operator, the other vector will converge to the same fixed point."

In our case, the optimal value utility vector is the desired fixed point and with the repeated application of the Bellman Equations on the initial utility estimates in an iterative manner as described in the beginning of this section, we can cause the utility estimates to approach the optimal values.

This concludes the proof.

## 2.5 The Policy Iteration Algorithm

The Value Iteration algorithm of the previous section is focused on finding the Optimal Value Function of Eq. 2.3.5 and then the corresponding optimal policy using the *one step lookahead* of Eq. 2.3.6.

However, we must also keep in mind that as to how accurate estimates of the optimal utilities we are looking for. That is, how many iterations of the Bellman operator are necessary to be applied on the initial utility estimates to get a reasonable estimate of the actual optimal utilities and how important is it actually to obtain an accurate estimate of the actual optimal values. If the estimates lack accuracy and one does use these estimates in the one step lookahead of Eq. 2.3.6 and execute the policy, then, will the resulting behavior be as optimal.

To answer the above questions, let us define the notion of *Policy Loss* as the difference between the true optimal utility and the utility acquired by executing the above said policy, then, as is established in the literature, see [9], one does find that the Policy Loss becomes zero long before our utility estimates reach the exact optimal utilities. That is, whilst looking for an optimal policy, it is not important that we have an accurate estimate of the true optimal utilities, rather, what is important is that the Policy Loss becomes zero or decreases below a user defined threshold. This suggests an alternative of the Value Iteration Algorithm, namely; *The Policy Iteration Algorithm.*

Before stating the Policy Iteration Algorithm, let us also define an *Admissible Policy* as one that is *stabilizing* and yields a *finite cost.* Let us also borrow the concept of sec. 2.2.4 that the utility of a policy can be used to evaluate the policy and this evaluation of the policy can be done using the Bellman Equation of sec. 2.3 which we re-write here as,

$$V(s) = r(s,u) + \gamma . V(s') \tag{2.5.1}$$

Notice that we have retained the same assumptions of our work as those in sec. 2.3 that rewards are dependent upon an action (u) taken in a particular state (s). Having these constructs with us, we are ready to state the Policy Iteration Algorithm as follows.

In the Policy Iteration algorithm, one starts with an arbitrary initializing admissible policy that is evaluated using the Bellman Equation 2.5.1. This value of the current policy can be used to find a better policy using the one step look-ahead of Eq. 2.3.6. This new policy found is again evaluated by 2.5.1 and re-updated by 2.3.6. The entire process is repeated until the policy update step of 2.3.6 yields no more better policies at which point, we have reached the optimal policy.

One may ponder as to how this one step lookahead can give us a better policy, however, if one closely observes the equation, then, it is obvious that the equation is similar to the one step Bellman Update that brings the utility and ultimately the new policy one step closer to the optimal policy. If the policy evaluation and updating is repeated in an iterative manner, then, one can reach the optimal policy much faster than in value iteration algorithm. Why this happens? The value iteration algorithm is focused on finding the optimal utilities and iterates through a sequence of utilities in order to reach the optimal utility. However, we must realize that if the Policy Loss has become zero or has decreased below a user defined threshold, then, it is of no use to find the optimal utility because it will give us no better policy than the one already found. In other words, if one action is clearly better than others, then, the utility estimates need not be accurate. The Policy Iteration algorithm takes advantage of this fact and is focused on finding the optimal policy instead and thus switches between values and policies in an iterative manner as described above, and, getting closer to the optimal policy with every iteration.

It is this Policy Iteration Algorithm that we have used to gather the results of our work.

Chapter 3

# The Optimal Control Problem

In this chapter, we formally lay the mathematical foundations of the control problem we intend to pursue using the concepts and methods of Reinforcement Learning that were described in detail in the previous chapter. As mentioned in the introductory section of this work, the concept of Reinforcement Learning originally belongs to the world of Computational Intelligence that has found its applications in the Control Systems community as well. The previous chapter was more inclined on defining the terms and concepts as is described in the computational intelligence community. However, from this chapter and onwards, we will be using the terminologies and naming conventions that are familiar to the control systems engineers.

## 3.1 Defining The Optimal Control Problem

Let us consider the following state space model of a continuous time linear system,

$$\dot{x} = Ax + Bu \tag{3.1.1}$$

with $x \in \Re^n$ represents the space of states, $A \in \Re^{n\times n}$ represents the inherent (internal) dynamics of our system and $B \in \Re^{n\times m}$ represents the input coupling function of the control input with our system. The above linear system may also be the result of linearization of a non-linear system with the assurance that the non-linear system should operate closely to a specified point of equilibrium. The $u \in \Re^m$ acts as the controller for our system.

Let us also define the following functional as a measure of cost,

$$V(x) = \int_t^\infty r(x,u)dt \tag{3.1.2}$$

where $V(x) : \Re^n \to \Re^+$ represents the total acquired cost gathered over time for an infinity horizon and $r(x,u) : \Re^n \times \Re^m \to \Re$ represents the cost incurred at every instant along the horizon.

Notice that we have considered a cost functional instead of a reward functional since utilization of a controller's resources are considered as costs incurred. Also, note that we have assumed additive costs instead of discounted costs and that the summation has been replaced by integration since we are dealing with continuous state space.

Having said that, let us find the state dependent feedback control for solving The Optimal Control Problem as,

$$u(x) = \arg\min_{u\to\Omega} V(x) \tag{3.1.3}$$

Surely this control has to satisfy the dynamics (3.1.1) in order to be a valid control for our system and making our closed loop system globally asymptotically stable.

Notice the use of the *min* operator instead of *max* since the control objective is to find the minimum cost control.

In the sections to come, we will be describing both the conventional and RL methods for the solution of this Optimal Control Problem so we may have a better insight of the advantages of using the RL approach over the conventional approach.

## 3.2 The Conventional Approach

One conventional way to solve this optimal control problem is to actually solve the continuous time Algebraic Riccati Equation (ARE),

$$A^TP + PA - PBR^{-1}B^TPQ = 0 \tag{3.2.1}$$

where $P$, a positive definite matrix, is our solution required. Having this positive definite matrix, the optimal control is available to us as,

$$K = R^{-1}B^TP \tag{3.2.2}$$

Solving (3.2.1) is a routine task and one can refer to these references to find a solution, [10], [11] and [12].

However, when solving for a class of non-linear systems, things get complicated and computation becomes difficult unlike the policy iteration algorithm discussed in the following sections.

## 3.3 The Policy Iteration Algorithm (Offline Version)

One of the main reinforcement learning methods, namely; The Policy Iteration (PI) algorithm can be used to solve our optimal control problem. The idea is that a control,

$$u = h(x) \tag{3.3.1}$$

has a cost (3.1.2) that must be paid if the control is to be used. We can be find this cost if we solve the following equation,

$$V(x(t)) = \int_t^{t+T} r(x(\tau), h(x(\tau))d\tau + V(x(t+T)) \tag{3.3.2}$$

where $T$, a positive number, internal for each reinforcement and (3.3.2) is the called as the interval reinforcement version of Bellman Equation. See [1],[13]

We might want to consider that (3.3.2) is a fixed point equation with the fixed point at $V(x)$, therefore, we can use the control policy (3.3.1) and obtain $V(x)$ in an iterative manner.

Once this initial control policy (3.3.1) is classed as admissible, that is, it has a finite cost (3.1.2) and is stabilizing the system (3.1.1), then we may also want to find a better policy by,

$$u^{'} = \arg\min_u(r(x,u) + (\nabla V^u)^T(Ax + Bu)) \tag{3.3.3}$$

The claim that the policy $u^{'}$ is better in terms of cost (that is, it has lesser or at most equal cost but not more) can be referred in [14],[15] And because of this, one gets further motivated to evaluate this new control by (3.3.2) and again find an even better control policy by (3.3.3).

If this procedure is repeated in an iterative manner, then, upon convergence, the converged values of $V(x)$ and $u$ will represent the optimal values and the optimal control respectively. This algorithm to find the optimal control is known as the Policy Iteration Algorithm. We begin with an admissible control and then iterate between (3.3.2) and (3.3.3) until the cost and control variables converge to optimal values.

To have a better grasp of the above concepts, let us explain the things with an example. Considering a state feedback control as,

$$u = -Kx \tag{3.3.4}$$

This policy makes our closed loop system (3.1.1) stable for all values of initial conditions of the system.

Also, let us consider the following cost function that is given in quadratic form,

$$r(x,u) = x^T Q x + u^T R u \tag{3.3.5}$$

Then, (3.1.2) becomes,

$$V(x(t)) = \int_t^{\infty} x^T(\tau) Q x(\tau) + u^T(\tau) R u(\tau) d(\tau) = x^T P x \tag{3.3.6}$$

where,

$$P = \begin{bmatrix} p_{11} & p_{12} \\ p_{12} & p_{22} \end{bmatrix}$$

This $P$ is the real symmetric and positive definite solution of the following Lyapunov Equation,

$$(A - BK)^T P (A - BK) = -(K^T R K + Q) \tag{3.3.7}$$

Once we are able to find $P$, our current control policy has been evaluated and a new control policy can be found by,

$$K = R^{-1}B^TP \tag{3.3.8}$$

Now, as said before, we begin with an initial control $K$ that is able to make our system stable and then we iterate between (3.3.7) and (3.3.8) until convergence is achieved. This iterative procedure will then be an offline PI algorithm for the system (3.1.1).

We also consider it important to tell that compared to the conventional method, it is much easier as well as computationally cheap to evaluate a control policy using the Lyapunov Equation (3.3.7) that can be solved with a mere recursion method rather than solving the conventional Riccati Equation using non-linear methods. See [1].The PI algorithm is fact an algorithm containing iterations within iterations. The inner iterations of (3.3.7) is for evaluating the current control and the outer iterations between (3.3.7) and (3.3.8) is to find the optimal control.

## 3.4 The Policy Iteration Algorithm (Online Version)

Despite the simple nature of PI RL algorithms, their real power is in their overall framework allowing these algorithms to be applied in an online manner .i.e. with a non-optimal control policy with us, an optimal control can also be found in an online manner by traversing through the system locus.

Having said that, one can retrieve the information of a system's behavior through its states, unlike the $A$ or $f(x)$ matrix used by the offline PI to obtain the same. That is, we can make algorithms with the capability to acquire system dynamics in an online fashion by traversing through the system locus.

This is known as Online Reinforcement Learning in the literature.

With the concepts such as TD Learning, (3.3.6) can serve to form the basis for an online RL version.

One can re-write the term $x^TPx$ in (3.3.6) as,

$$x^TPx = \bar{p}^T\bar{x} \tag{3.4.1}$$

where $\bar{x}$ representing the Kroknecker Product and giving a columnized vector that has repeated terms removed and $\bar{p}$ representing the columnized P that has repeated terms omitted.

Assuming (3.3.4) and (3.3.5), and with the above formulation, we can re-write (3.3.2) as,

$$\bar{p}^T(\bar{x}(t) - \bar{x}(t+T)) = \int_t^{t+T} x^T(\tau)(Q + K^T RK)x(\tau)d(\tau) \tag{3.4.2}$$

where $(\bar{x}(t) - \bar{x}(t+T))$ represents a regression and the Right Hand Side is the interval reinforcement as,

$$r(x, K) = \int_t^{t+T} x^T(\tau)(Q + K^T RK)x(\tau)d(\tau) \tag{3.4.3}$$

After an initial control policy is chosen to be applied on the system (3.1.1), (3.4.2) will be used to find the value function parameters. This is going to require us to measure the previous state $x(t)$, the current state $x(t+T)$, the interval reinforcement $r(x(t), K)$ after each time interval $T$.

Once sufficient time intervals have passed, we will have enough data to set up a LS problem and find the value function parameters so that it satisfies (3.4.2).

And with $\bar{p}$ available to us, we would have actually evaluated the current control $K$ like as we did in (3.3.7).

Then , as before, we can update this control using (3.3.8) and our algorithm can switch to the newer control while travesing through the real time state locii.

And if we iterate on this repeatedly, that is, we evaluate each newer control available to us using (3.4.2) and find even better control from this evaluation using (3.3.8), then we can obtain the same optimal control as found before.

However, we must explicitly mention the tings necessary to be kept under consideration when trying to achieve our objective to find the optimal control using online algorithms.

The first one is, our system needs to have sufficient initial excitation. This is because any information about the dynamics of a system are contained in the system states and the more excitation the system possesses, the more easily and truly the information (parameters) regarding the system dynamics can be extracted from the states. Second, our algorithm should be designed so that it switches to the newer controls online as fast as possible. This is because, the longer the non-optimal controls are kept applied, the more the system would have reached the stabilization point through the non-opitmal path and therefore if an opitmal control is found after the system is almost stable is of no beneficial purpose. Also, when the system is closer to the equilibrium point, that is, it is almost stable, then, we can not evaluate the inverse in (4.4.2) which is needed for sure for online RL.

Chapter 4

# Simulation Examples

We can now begin our simulation example with all the design tools of control developed and discussed in chapter 3. We begin by considering a swing up task of a pendulum having $1 - DegreeOfFreedom$. The following state space model defines our system,

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= 120.42 \sin x_1 - 1.597 x_2 + 29.53 u \end{aligned} \tag{4.0.1}$$

Since our system is an inverted pendulum system, it has only two number of points of equilibrium which are $(0, 0)$ $(\pi, 0)$ one of them being an unstable equilibrium point and the other being stable respectively. And we actually have to make our inverted pendulum system stable on the $(0, 0)$ point.

We will be using all the tools to do this as discussed in chapter 3. But in addition, we will also be computing a non-optimal conventional control first to be able to give a meaningful comparison and plus points of using RL technique. Besides, the non-optimal control that we find first will also be useful as an initial value necessary to begin the PI algorithm.

The state space model (4.0.1) is a non-linear model and its linearized version about the unstable equilibrium point $(0, 0)$ is,

$$A = \begin{bmatrix} 0 & 1 \\ 120.42 & -1.597 \end{bmatrix}$$

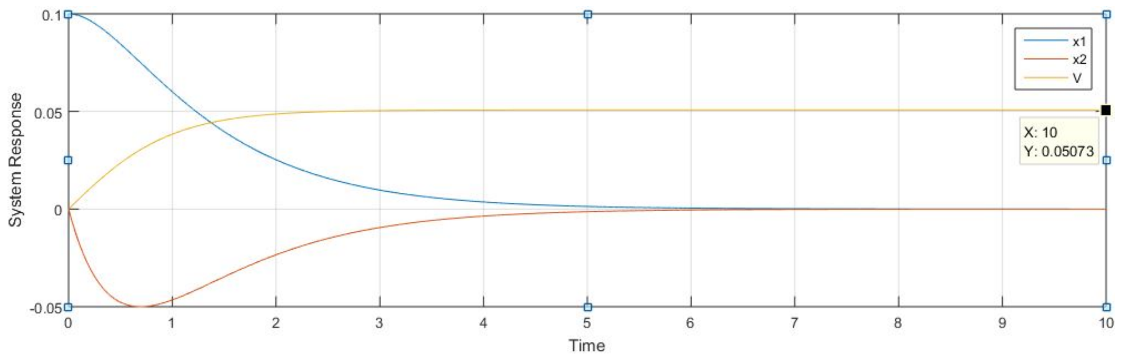

**Figure 4.1:** Non-Optimal Control Response of Pendulum System

$$B = \begin{bmatrix} 0 \\ 29.53 \end{bmatrix}$$

Also let,

$$Q = \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix}$$

and $R = 0.01$

## 4.1 The Non-Optimal Control

We have an unstable eigen value in the $A$ matrix. However, we can make the closed loop system (3.1.1) stable by making $(A - BK)$ Hurwitz assuming the control (3.3.4).

Using the *Ackerman Formula*, we can find an arbitrary stabilizing controller $K$ for small initial perturbations from the unstable equilibrium point, the response of which can be seen in Fig. 4.1.

We can see that this control accumulates a total cost of $V = 0.051$ for stabilizing the system.

## 4.2 The Optimal Control by ARE

As discussed in the previous chapter, we can solve the CT Algebraic Riccati Equation as a conventional approach for solving the Optimal Control Problem.

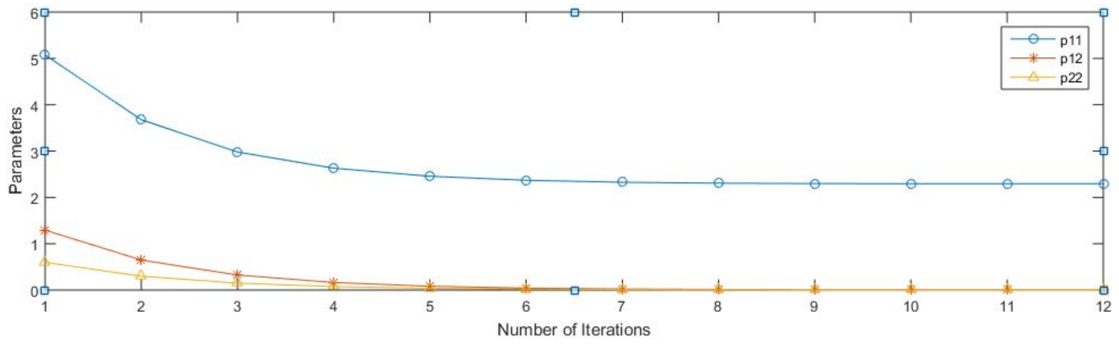

**Figure 4.2:** Value Function Parameters Converged using Offline Policy Iteration

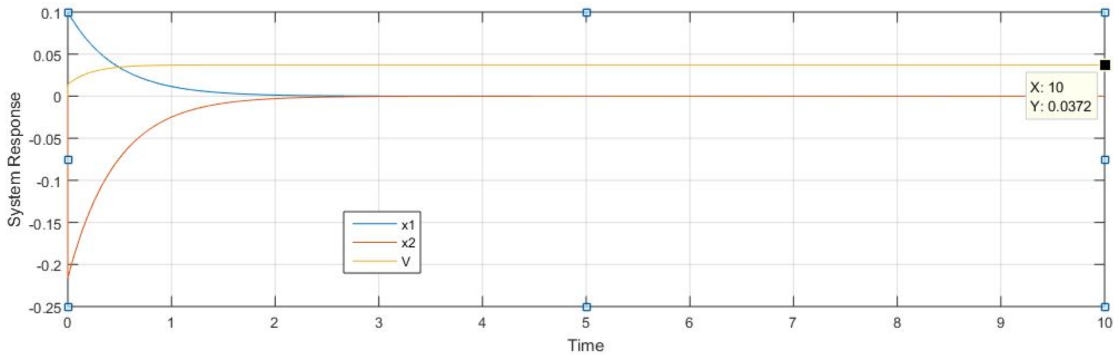

**Figure 4.3:** Offline Policy Iteration Control System Response at 1st Iteration

For the linearized version of our pendulum system, we find the optimal value as,

$$P = \begin{bmatrix} 2.2957 & 0.0091 \\ 0.0091 & 0.0034 \end{bmatrix}$$

and we find the optimal control using (3.2.2) as,

$$K = \begin{bmatrix} 26.8074 & 10.0364 \end{bmatrix}$$

## 4.3 The Optimal Control by Offline PI

Using as a seed value, the arbitrary initial stabilizing controller of section 4.1, a sequence of policy evaluations using (3.3.7) and updates using (3.3.8) allow us to find the solution of the Optimal Control Problem by reaching the optimal value and optimal control upon convergence.

As can be seen in Fig.4.2, it took only 12 number of iterations for the Value Function Parameters to converge at the optimal values. Fig.4.3 shows the total cost of 0.037 of the system response accumulated along the system trajectories at first iteration (3.3.8)

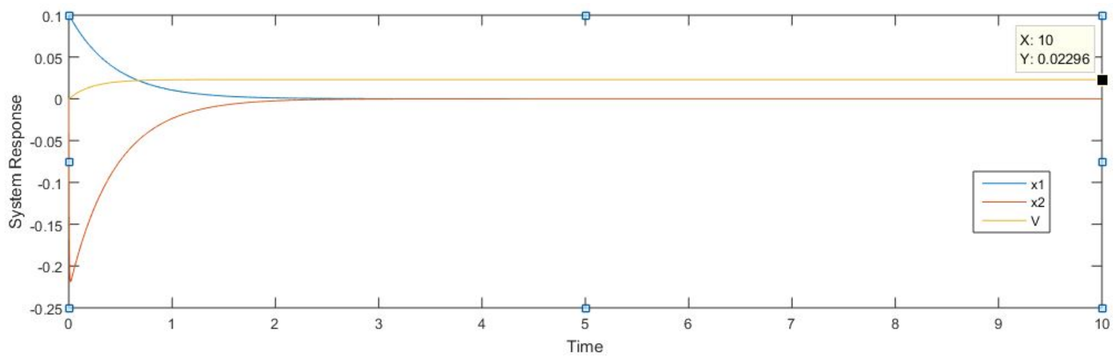

**Figure 4.4:** Offline Policy Iteration Optimal Control System Response

of the Offline PI algorithm. If we compare this cost incurred with that of Fig.(4.1), we can clearly see that this updated control is better than the initial one. Fig. 4.4 shows the system response. The optimal control policy is obtained at the 12th iteration of the offline PI Algorithm. Also note that minimal control cost for stabilizing our system is $V = 0.023$, that is, the optimal control cost or value. We must also take notice that the optimal control that we found for our system using the Offline Policy Iteration Algorithm is exactly the same as that found by solving the Algebraic Riccati Equation in the conventional approach. However, the former has a less computational cost whereas the latter does not.

And with this, we are done with our simulation of the offline version of the Policy Iteration Algorithm.

## 4.4 The Optimal Control by Online PI

Let us redo the same simulation example of the previous section to show the applicability of online reinforcement learning algorithms and also to allow an offline vs online comparison, however, this time, we will not be using the info of the plant matrix $A$ but that about the systems will be obtained from the real time state trajectories in an online fashion.

For us to use (3.4.2) and setting up the LS problem, let us first make an evaluation of the Right Hand Side of (3.4.1),

$$x^T P x = \bar{p}^T \bar{x}$$

with

$$\bar{x} = x \otimes x = \begin{bmatrix} x_1^2 \\ \cancel{x_1 x_2} \\ x_1 x_2 \\ x_1^2 \end{bmatrix}$$

and

$$\bar{p} = Vec(P) = \begin{bmatrix} p_{11} \\ \cancel{p_{12}} \\ p_{12} \\ p_{22} \end{bmatrix}$$

Also for the sake of simplicity,let,

$$x(t) = x_o$$

$$x(t+T) = x_1$$

$$r(x_o, K) = b$$

$$\bar{p} = s$$

Then a simplification of (3.4.2) can be,

$$As = b \tag{4.4.1}$$

The Least Squares solution to (4.4.1) is,

$$s = (A^T A)^{-1} A^T b \tag{4.4.2}$$

With this, we can now begin our simulation example of the Online Policy Iteration Algorithm using the same values of the system, that is, $B$, $Q$, $R$, $K_{ini}$, however; $A$ will not be included in our calculations.

Further let us arbitrarily select,

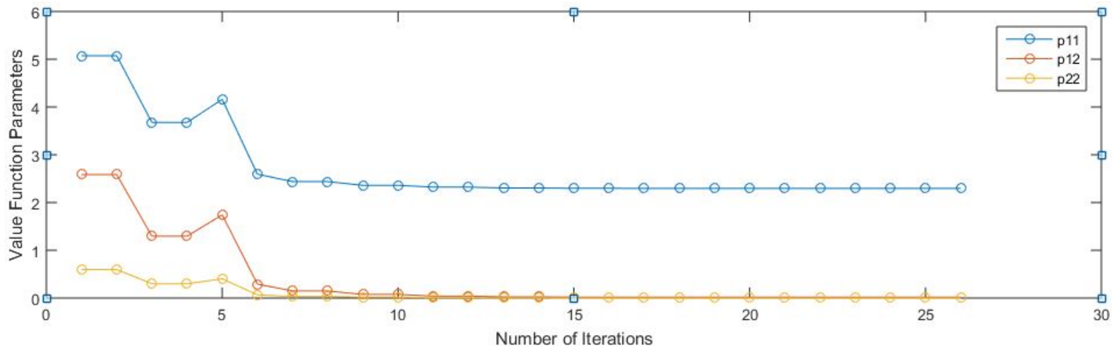

**Figure 4.5:** Value Function Parameters Converged using Online Policy Iteration

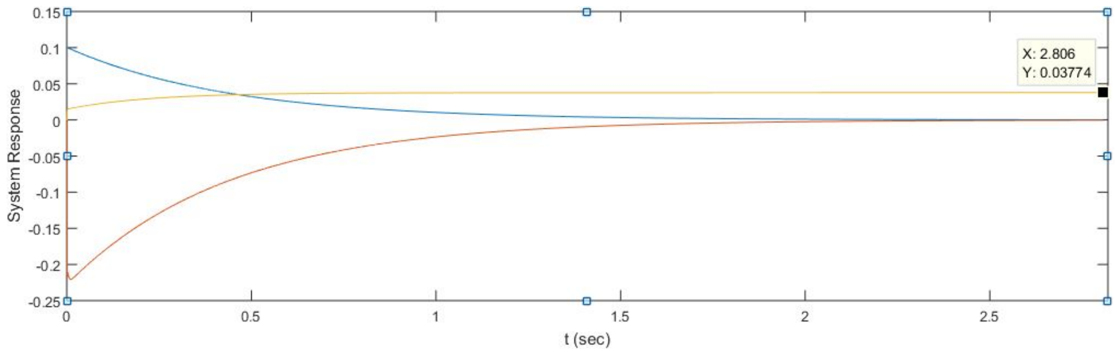

**Figure 4.6:** Online Policy Iteration Optimal Control System Response

$$T = 0.0001, x(0) = \begin{bmatrix} 0.1 & 0 \end{bmatrix}$$

In the online version, (4.4.2) will be used to make an evaluation of the control. That is, for the online PI algorithm, we iterate between (4.4.2) and (3.3.8) till convergence is achieved.

As is told in the previous chapter, one must have enough data to acquire a solution of the Least Squares problem (4.4.1). The inverse in (4.4.2) in our simulation example can only exist, if the matrix A possesses a rank of 3 atleast. This means that we need to let at least 3 number of time intervals $T$ to pass in order to acquire an initial solution $s$.

But we must also take notice of the fact that acquiring an initial solution $s$ is necessary but not sufficient. What is more important is that we do have a converged LS solution using the data received after the passage of every single $T$ time interval. Once the $LS$ solution seems to have converged or more specifically has fallen below a user defined error threshold, then, we can imply that the control $K$ has been evaluated and we can now update this control to obtain and apply the newer updated control.

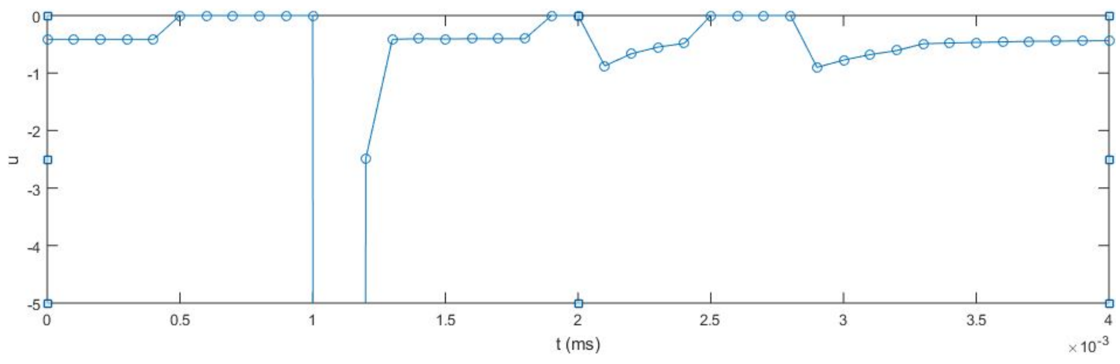

**Figure 4.7:** Online Control Applied on the System (First four milliseconds)

The simulations graphs of our Inverted Pendulum System can be seen in Fig 4.6. The system learns the optimal control in an online manner without using any info about the system dynamics.

Indeed the same optimal values of the Value Function Parameters and the same optimal gains of the state feedback control are achieved from the Online PI Algorithm, however, one should not take for granted the Offline Policy Iteration Algorithm just because the same results are achieved for the Online PI Algorithm in the end and with the added advantage that system dynamics are not needed. This is because the offline and online versions of the PI Algorithm are principally different, that is, with an accurate info about the system dynamics available to us, the offline PI will always outperform its online counterpart. Why?

The Online version of the PI Algorithm gathers information about the system's behavior by traversing trough the system trajectories in real time. And having said that, learning is not possible in a stationary system. Even for a non-stationary system, it is important that the system possesses sufficient excitation, only then the sates will carry true and accurate information about the behavior of a system. Now, let us suppose that a control is applied at the initial condition and is simultaneously being evaluated as well, however, we must keep in mind that the longer we keep applying this control for its evaluation, the more the system dynamics die out with time because the system will get closer and closer to the stabilization point. If an optimal control is not found and the system dynamics die out, then, no further learning/evaluations can take place and we will not be able to find the optimal control. This problem can be dealt with if we restart the entire process of stabilization again from the initial condition as is done in conventional Reinforcement Learning Algorithms.

We are going to take an approach that is different from the conventional one. We are not going to start the whole stabilization process again, instead, why can we not revive the dynamics of our system in the same process through the application of a carefully measured non-stabilizing control or by turning off the control momentarily. In our simulations, it is clearly obvious that for reviving the system dynamics, the applied control was totally turned off at three points in time as can be seen in Fig. 4.7

Also, when dealing with offline algorithms, the control cost that needs to be paid is what is called as *exploitation*), that is the cost for exploiting or applying a control on the system whereas in online algorithms, besides the exploitation cost of using a control, another control cost that must be paid is what is known as *exploration*), that is the cost for finding a control.

Having said that, it is pretty clear that for online algorithms, the system will accumulate more and more cost as it is progressing through the system in search of better controls and ultimately the optimal control. That is, compared with its offline counterpart that has its optimal control (minimum cost control) available to it from the very beginning of state trajectories, the online algorithm will first bear the costs of applying the initial non-optimal (expensive) controls by traversing through the expensive states to be able to find the minimum cost optimal control and then using it to stabilize the system. In short, the total cost of offline RL algorithms is always less than the total cost of online RL algorithms.

Having said that, in order to keep the learning costs as low as possible for online algorithms, we stress on reviving the system dynamics during the same state trajectories rather than initializing the entire stabilization process from the very beginning.

The cost of offline control as can be seen in Fig. 4.4 is $V = 0.022$ whereas that for the online algorithm can be seen in Fig. 4.6 which is $V = 0.037$. The difference of 0.015 highlights the cost of the exploration part in our algorithm. It is pretty clear that the online control we found is better than the non-optimal control obtained through a conventional method as can be seen in Fig. 4.1 which has a cost of 0.050.

However, if we intend to harness the potential of online algorithms, then care must be given to maintain a balance between exploration part and the exploitation part to avoid the costs from exceeding that of the non-optimal controls in which case the entire purpose of finding an optimal control through RL techniques becomes useless.

Chapter 5

# Discussions, Future Work and Conclusion

## 5.1 Discussion

Our main findings of this work are that despite the stability and convergence properties of PI algorithms that are established in the literature, one must give an equally important consideration to the following points if optimal controls are to be found using online RL algorithms.

- A system must possess sufficient excitation if a control is to be evaluated online on that system.
- Online RL requires the data to be gathered really fast and quickly.
- One should not apply a non-optimal control on a system for longer durations as this will cause the system excitation to die out and the learning costs will increase.
- One should have enough samples gathered to have an accurate and converged estimate of the Value Function Parameters that represent the value of a control.
- The Least Squares solution method should be once selected and kept the same throughout since different solution methods can lead to different value function parameters for the same applied control.
- A system should not be subjected to very high excitation neither initially nor

during the phase of reviving the system dynamics as this can lead to huge costs making the control non-optimal.

- Unnecessary excitation of a system should be avoided since every re-excitation adds to the total cost.
- It is possible that a system or a particular state or set of states cannot be re-excited repeatedly to allow learning and thus a tradeoff must be made to settle for an almost opitmal control rather than an optimal control.

## 5.2 Future Work

The future directions of our work are as follows

- An online RL algorithm design, that can despite learn the system dynamics in real time, will require a designer's experience about the particular system's behavior in deciding its control strategy. Like in our work, the designer will only program the system to perform re-excitations only if it is possible for that particular system/state. The algorithm has no knowledge about this in advance and will rely on the designer's programming to take a decision on this. Similarly, how many number of re-excitations will suffice to find the optimal control while not becoming the major reason of increased costs. It will be the programmer's skill on how he incorporates this knowledge into the algorithm design. If such smart algorithms can be designed for various systems, then we can expect to be able to develop some real intelligent algorithms possessing the knowledge for a class of systems smart enough to decide from a set of control strategies in real time.
- Every online algorithm design when implemented on a system will reveal its cost and the fact if it is actually better than a non-optimal conventional control. We think that there exists a need of a systematic procedure that can perform this costing of an algorithm allowing the designer to tweak his design at the time of designing and suggesting ways to develop the most optimal online algorithm.
- We also feel the need of a systematic procedure on selecting the specific cost function (like in our work, the values of Q and R) that while achieving the user

desired control task will also make sure the user desired control performance is achieved.

## 5.3 Conclusion

This thesis discusses on the potential of Reinforcement Learning Control in solving the online optimal control problems. We approached the task of the stabilization of an inverted pendulum system on its unstable equilibrium point and demonstrated it with the help of many simulation examples of the conventional and RL methods. An extensive comparison between the online and offline Reinforcement Learning Algorithms has been given. Also highlighted were the main challenges that a designer needs to face for designing online algorithms. We concluded this work with highlighting our findings in this work followed by specifying the future directions of our work.

# References

[1] Frank Lewis and Draguna Vrabie. Reinforcement learning and adaptive dynamic programming for feedback control. *IEEE Circuits and Systems Magazine*, 3rd Quarter:32–50, 2009.

[2] Draguna Vrabie. Online adaptive optimal control for continuous-time systems. *Presented to the Faculty of the Graduate School of The University of Texas at Arlington*, pages 1–183, 2009.

[3] R.J. Leake and Ruey-Wen Liu. Construction of suboptimal control sequences. *J. SIAM Control*, Vol. 5th(1):54–63, 1967.

[4] R.S. Sutton and A.G. Barto. Reinforcement learning, an introduction. *Cambridge, MA: MIT Press*, 1998.

[5] D.P. Bertsekas and J.N. Tsitsiklis. Neuro-dynamic programmingma: Athena scientific. *MA: Athena Scientific*, 1996.

[6] Gary A. Hewer. An iterative technique for the computation of the steady state gains for the discrete optimal regulator. *IEEE Trans. on Automatic Control*, pages 382–384, 1971.

[7] Rongxin Cui, Chenguang Yang, Yang Li, and Sanjay Sharma. Adaptive neural network control of $AUVs$ with control input nonlinearities using reinforcement learning. *IEEE Trans. on Systems, Man and Cybernetics: Systems*, pages 1–11, 2017.

[8] Chaoxu Mu, Zhen Ni, Changyin Sun, and Haibo He. Air-breathing hypersonic vehicle tracking control based on adaptive dynamic programming. *IEEE Trans. on Neural Networks and Learning Systems*, pages 1–15, 2016.

[9] Peter Norvig and Stuart Russel. *Artificial Intelligence, A Modern Approach.*

[10] Balzer L. A. Accelerated convergence of the matrix sign function method of solving lyapunov, riccati and other equations. *Int. J. Control*, 32(6):1076–1078, 1980.

[11] Byers R. Solving the algebraic riccati equation with the matrix sign. *Linear Algebra and Its Applications*, 85:267–279, 1987.

[12] Laub A. J. A schur method for solving algebraic riccati equations. *IEEE Trans. on Automatic Control*, 24(6):913–921, 1979.

[13] D. Vrabie, O. Pastravanu, M. Abu-Khalaf, and F. L. Lewis. Adaptive optimal control for continuous-time linear systems based on policy iteration. *Automatica*, 45(2):477–484, 2009.

[14] F. Lewis, D. Vrabie, and K. G. Vamvoudakis. Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers. *IEEE Control Systems*, 32(6):76–105, 2012.

[15] D.P Bertsekas and J.N. Tsitsiklis. Neuro-dynamic programming. *MA: Athena Scientific*, 1996.