Reinforcement Learning for Control Systems

Luis Pimentel
ECE 4803 Mathematical Foundations of Data Science
Professor Mark Davenport
Georgia Institute of Technology
Atlanta, Georgia
lpimentel3@gatech.edu

***Abstract*—In this report we cover the optimization techniques that are use to solve problems related to Control Systems. These type of problems are found in the field of Reinforcement Learning which is branch of the more broad domain of Machine Learning and Data Science. To solve these problems, classical techniques such as Least Squares Regression and Gradient Ascent are applied to a Control Systems problem that is reformulated as an optimization problem. In this report, we introduce the the field of Reinforcement Learning and how these type of problems are formulated. We then procceed by introducing the general Control Systems problem and the classical Linear Quadratic Regulator (LQR). We show how this can be reformulated as a Reinforment Learning problem and apply optimization techniques to the system of an Inverted Cart Pendulum and analyze the results.**

***Index Terms*—reinforcement learning, control systems, linear quadratic regulator, policy gradient, finite difference, episodic reinforce**

I. Reinforcement Learning

Reinforcement Learning is a branch of the more broad field of Machine Learning. To begin understanding Reinforcement Learning it is important to describe the other branches and how Reinforcement Learning is different. The other branches of Machine Learning are supervised learning and unsupervised learning. In supervised learning data is carefully labeled and the goal is to use this labeled data to solve problems involving regression and classification. Applications of these techniques are task driven with the goal of being able to predict values or classify values into groups. In unsupervised learning, data is not labeled and the goal is learn the relationship of data and how it is clustered or associated. Reinforcement Learning is different from these areas in that its goal is to learn how an agent should behave in an unknown and uncertain environment by iteracting with that environment. This formulation is different from supervised learning in that in that the data it takes in is not labeled but rather feedback come from the formulation of reward functions based on its interactions. This is also different from unsupervised learning as its goal is to learn a model of behavior from data, rather than finding relationships in that data.

We can more formally define the vocabulary used in the Reinforcement Learning problem formulation:

- **Agent**: An agent is a system that performs actions affecting the state of that system.
- **Environment**: An environment is the world with which the agent interacts with.
- **State** S_t: A state is a particular condition of an agent at a specific time.
- **Action** A_t: An action is activity that is performed by the agent.
- **Policy** π_t: A policy is a mapping from state to action. This policy is what Reinforcement Learning tries to learn.
- **Reward** R_t: Reward is feedback from the environment as a result of an action that the agent takes.

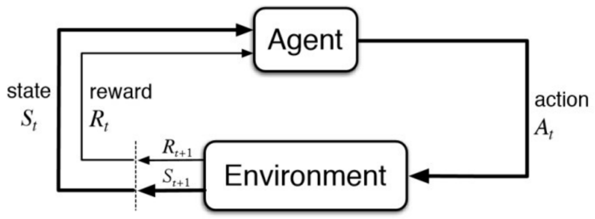

Fig. 1. Diagram of the Reinforcement Learning problem. [source]

In Reinforcement Learning an agent is able to learn through the formulation of the reward function. It is important to design this reward function appropriately depending on the context of the problem so that when an agent takes an action that leads to a "good" state, this action is considered "good" and the agent receives a reward. In the otherhand if an agent takes an action that leads to a "bad" state, this action is considered "bad" and the agent is penalized instead. This allows us to formulate the Reinforcement Learning problem as an optimization problem that seeks to maximize the accumulation of reward.

Reinforcement Learning is suited for many applications that require interaction betwen a system and an enviroment. It has been used to solve problems in natural language processing, neuroscience, and particularly robotics where one can see the clear connection to Control Systems.

A. Reinforcement Learning as an Optimization

In this section we formulate Reinforcement Learning as an optimization problem as described in the previous section. We begin by defining a trajectory τ as a series of state-action pairs, indexed by time, that our agent follows such that

$$\boldsymbol{\tau} = [\boldsymbol{s}_0, \boldsymbol{a}_0, \boldsymbol{s}_1, \boldsymbol{a}_1, \ldots; \boldsymbol{s}_{N-1}, \boldsymbol{a}_{N-1}, \boldsymbol{s}_N]$$

Furthermore, we define a constraint on the optimization problem such that our agent must follow the dynamics of the system

$$f(\boldsymbol{s}_t, \boldsymbol{a}_t, e_t)$$

where our dynamics are parameterized by Gaussian noise e_t, representing uncertainty in the environment. We then define the Reward of a trajectory $R(\boldsymbol{\tau})$ as the sum of the reward for a certain state-action pair:

$$R(\boldsymbol{\tau}) = \sum_{t=0}^{N} r(\boldsymbol{s}_t, \boldsymbol{a}_t, t)$$

We can then formulate the objective function of our optimization problem as the expectation of the reward over many trajectories. In this problem we want to maximize this expectation with respect to the policy $\pi(\boldsymbol{\tau})$. We write down our optimization probem as follows

$$\begin{aligned} &\underset{\pi}{\text{maximize}} \quad \mathbb{E}\Big[R(\boldsymbol{\tau})\Big] \\ &\text{subject to} \\ &\boldsymbol{s}_{t+1} = f(\boldsymbol{s}_t, \boldsymbol{a}_t, e_t) \\ &\boldsymbol{a}_t = \pi(\boldsymbol{\tau}) \end{aligned} \tag{1}$$

This optimization problem can be solved using the classical optimization technique of gradient ascent. In a following section this algorithm is detailed once we reformulate the Control Systems problem as a Reinforcement Learning problem.

II. Control Systems

In Control Systems problems the goal is to design a controller for a system. The system which is being controlled is referred to as the *plant* and an output of the system can be a state of that plant. The goal of the controller is provide an actuating signal to the input of the plant, based on an input to the controller with the ultimate goal of driving the plant to a desired state. In the formulation of a feedback control system, one or more states of the plant are fed to the controller to calculate the error between the current state of the plant and the desired state.

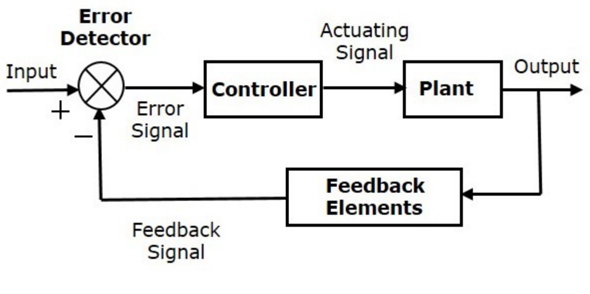

Fig. 2. Diagram of the feedbac control systems problem. [source]

This report focuses on the case of a linear controller in which the dynamics of the plant can be mathematically described through a linear system of equations. This system of equations is derived from a set of differential equation describing the dynamics of the plant. We begin our mathematical formulation by describing the elements of this control system:

- $\dot{\boldsymbol{x}}$: Set of differential equations describing how the state changes.
- $\boldsymbol{x}$: The state of the sytem
- $\boldsymbol{u}$: The input to the system
- $\boldsymbol{y}$: The output of the sytem respectively.

With this we define the control system as:

$$\begin{aligned} \dot{\boldsymbol{x}}(t) &= \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t) \\ \boldsymbol{y}(t) &= \boldsymbol{C}\boldsymbol{x}(t) \end{aligned} \tag{2}$$

Where the state transition matrix $\boldsymbol{A}$ and control transition matrix $\boldsymbol{B}$ describes how the state of the plant transitions from one state to the other and the observation matrix $\boldsymbol{C}$ describes which state of the plant we are observing.

A. *Inverted Cart Pendulum*

The inverted cart pendulum is a classical control problem that can be used as a benchmark for testing different controllers. In this problem a pendulum is attached to the top of a moving cart and goal is apply a force on the cart in the horizontal direction such that the pendulum will remain balanced on top of the cart in equilibrium.

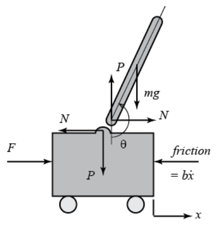

Fig. 3. Diagram of inverted cart pendulum problem. [source]

We define the states of the system as follows:

- $\boldsymbol{x}$, $\dot{\boldsymbol{x}}$, $\ddot{\boldsymbol{x}}$: The position, velocity, and acceleration of the cart.
- $\boldsymbol{\phi}$, $\dot{\boldsymbol{\phi}}$, $\ddot{\boldsymbol{\phi}}$: The angle, angular velocity, and angular acceleration deviation from equilibrium point of the pendulum.

With this state representation, the dynamics of this sytem can be described by the following linearized set of differential equations:

$$\begin{aligned} (I + ml^2)\ddot{\phi} - mgl\phi &= ml\ddot{x} \\ (M + m)\ddot{x} + b\dot{x} - ml\ddot{\phi} &= u \end{aligned} \tag{3}$$

The state space representation of this system, following the representation in (2), can then be described as follows:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\phi} \\ \ddot{\phi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-(I+ml^2)b}{I(M+m)+Mml^2} & \frac{m^2gl^2}{I(M+m)+Mml^2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-mlb}{I(M+m)+Mml^2} & \frac{mgl(M+m)}{I(M+m)+Mml^2} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \phi \\ \dot{\phi} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{I+ml^2}{I(M+m)+Mml^2} \\ 0 \\ \frac{ml}{I(M+m)+Mnl^2} \end{bmatrix} u$$

In the experiments section we implement the Reinforcment Learning algorithms on this plant model.

B. Linear Quadratic Regulator

As described in the previous section, the goal of the control system problem is to design a controller that provides the actuating input signal $\boldsymbol{u}$. A classical controller often used in Optimal Control problems is the Linear Quadratic Regulator (LQR). In an Optimal Control problem, a controller is designed with respect to a cost function that depends on the control input and the state of the system. The Optimal Controller minimizes this cost. For the LQR controller we the define the controller $\boldsymbol{u}$ and the cost function $\boldsymbol{J(x,u)}$ as follows:

$$\boldsymbol{u} = -\boldsymbol{K}\boldsymbol{x} \tag{4}$$

$$J(\boldsymbol{x},\boldsymbol{u}) = \boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{u}^T\boldsymbol{R}\boldsymbol{u} \tag{5}$$

Where the $\boldsymbol{Q}$ and $\boldsymbol{R}$ matrices are designed to describe the cost of a control input and state of the system. The $\boldsymbol{K}$ term is the optimal feedback gain which in LQR is solved analytically through the following equations:

$$\boldsymbol{K} = \boldsymbol{R}^{-1}\boldsymbol{B}^T\boldsymbol{P}$$
$$\boldsymbol{PA} + \boldsymbol{A}^T\boldsymbol{P} + \boldsymbol{Q} - \boldsymbol{PBR}^{-1}\boldsymbol{B}^T\boldsymbol{P} = \dot{\boldsymbol{P}}$$

From this we can gather that using LQR the optimal control problem can be solved with just knowledge of the plant model and the designed cost function.

III. Reinforcement Learning and Control

As seen in the previous section, LQR provides an optimal controller that can solve a feedback control system, as long as you have knowledge of the sytem. However, it is interesting to think about scenerios where we do not have a detailed mathematical model of the physics of the system. If we do not have a model, is it possible to design a controller that can operate optimally and solve a control system? To solve this problem we turn to Reinforcement Learning. As described in Section I, in Reinforcement Learning an agent can learn a policy by interacting with the world. We use this inspiration to reformulate the Control Systems problem as a Reinforcement Learning problem were we instead learn the controller by providing inputs to the sytem, observing the outputs and assigning reward based on a designed reward function.

A. Reformulation as an Optimization

Again we can formulate this as an optimization problem. First, we define a trajectory $\boldsymbol{\tau}$ as a series of state-control pairs:

$$\boldsymbol{\tau} = [\boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{x}_1, \boldsymbol{u}_1, \ldots; \boldsymbol{x}_{N-1}, \boldsymbol{u}_{N-1}, \boldsymbol{x}_N]$$

Furthermore, the system is constrained by the dynamics:

$$\dot{f}(\boldsymbol{x}_t, \boldsymbol{u}_t, e_t)$$

While this is a constraint on the system, it is not necessary to know or learn the dynamics. This is simply a constraint that describes how a fed input to the sytem will effect and output of the sytem. We then define our controller to follow a policy π parameterized by a parameter $\boldsymbol{\theta}$:

$$\boldsymbol{u}_t = \pi(\boldsymbol{\tau}, \boldsymbol{\theta})$$

Again, we define the Reward of a trajectory $R(\boldsymbol{\tau})$ as the sum of the rewards for a certain state-control pair:

$$R(\boldsymbol{\tau}) = \sum_{t=0}^{N} r(\boldsymbol{x}_t, \boldsymbol{u}_t, t)$$

We again formulate the objective function of our optimization problem as the expectation of the reward over many trajectories. In this problem we want to maximize this expectation with respect to the parameter $\boldsymbol{\theta}$. We write down our optimization probem as follows:

$$\begin{aligned} &\underset{\boldsymbol{\theta}}{\text{maximize}} \quad \mathbb{E}\Big[R(\boldsymbol{\tau})\Big] \\ &\text{subject to} \\ &\boldsymbol{x}_{t+1} = \dot{f}(\boldsymbol{x}_t, \boldsymbol{u}_t, e_t) \\ &\boldsymbol{u}_t = \pi(\boldsymbol{\tau}, \boldsymbol{\theta}) \end{aligned}$$

We can now present the gradient ascent algorithm used to solve this problem by first simplifying our optimization problem as:

$$\underset{\boldsymbol{\theta}}{\text{maximize}} \quad J(\boldsymbol{\theta}) \tag{6}$$

```
Algorithm 1: Gradient Ascent for Policy Parameterization
Result: $\boldsymbol{\theta}$
initialize: $\boldsymbol{\theta} = \boldsymbol{\theta}_0$, $\gamma$: learning rate
while unconverged do
    $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_i)$ = compute_policy_gradient($\boldsymbol{\theta}_i$)
    $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \gamma \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_i)$
```

Fig. 4. Gradient Ascent Algorithm for Policy Parameterization.

Solving this optimization problem now becomes a problem of computing the policy gradient which is the gradient of the objective function with respect to $\boldsymbol{\theta}$. At glance, computing this gradient would require us to know the physics of the system. However, in the following sections we show different techniques that can be used to create an estimation of this gradient that we can use in our gradient ascent algorithm. These techniques are called "model free" as a model of the dynamics are unknown.

B. Finite Difference Method

The Finite Difference Method is one of the oldest methods of policy gradient estimation used and is quite simple to implement and easy to understand. The underlying optimization technique used in this method to compute the policy gradient is Least Squares Regression. The first step in this algorithm is to apply many random perturbations $\delta\boldsymbol{\theta}$, over M rollout trajectories, to our parameter of interest $\boldsymbol{\theta}$. Using these perturbed parameters $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta} + \delta\boldsymbol{\theta}$ we then recompute our objective function such that:

$$J(\boldsymbol{\theta}+\delta\boldsymbol{\theta}_1) = J(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})^T \delta\boldsymbol{\theta}_1$$
$$J(\boldsymbol{\theta}+\delta\boldsymbol{\theta}_2) = J(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})^T \delta\boldsymbol{\theta}_2$$
$$\vdots$$
$$J(\boldsymbol{\theta}+\delta\boldsymbol{\theta}_m) = J(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})^T \delta\boldsymbol{\theta}_m$$

Our new objective function implicitly contains the policy gradient we are trying to estimate $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})^T$. We can now rearrange these equations to compute the change in our objective function as a function of this policy gradient and our perturbation such that $\Delta\hat{J} = J(\boldsymbol{\theta}+\delta\boldsymbol{\theta}) - J(\boldsymbol{\theta})$:

$$\Delta\hat{J}_1 = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})^T \delta\boldsymbol{\theta}_m$$
$$\Delta\hat{J}_2 = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})^T \delta\boldsymbol{\theta}_m$$
$$\vdots$$
$$\Delta\hat{J}_m = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})^T \delta\boldsymbol{\theta}_m$$

Using this we can set up the following Least-Squares Regression Problem such that:

$$\begin{bmatrix} \Delta\hat{J}_1 \\ \Delta\hat{J}_2 \\ \cdots \\ \Delta\hat{J}_m \end{bmatrix} = \begin{bmatrix} \delta\boldsymbol{\theta}_1^T \\ \delta\boldsymbol{\theta}_2^T \\ \cdots \\ \delta\boldsymbol{\theta}_m^T \end{bmatrix} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

$$\Delta\hat{\boldsymbol{J}} = \Delta\Theta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

Therefore, the solution to our policy gradient estimation is:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \left(\Delta\Theta^T\Delta\Theta\right)^{-1}\Delta\Theta^T\Delta\hat{\boldsymbol{J}}$$

The Finite Difference Method is a very simple policy gradient that has been applied to many robotic systems and has been show to be very efficient in deterministic systems. There are many different configurations to this algorithm in that designer can choose:

- How to generate the perturbations $\delta\boldsymbol{\theta}$.
- How many perturbations to generate.
- How to compute $\Delta\hat{J}$.
- How to tune hyperparameters of gradient ascent such as choosing the convergence criterion and learning rates.

It is important to note that this method does however require close supervision as badly generated $\delta\boldsymbol{\theta}$ can lead to destabilization and thus failure in the optimization. Creating many perturbations over many rollout trajectories can also be computationally expensive. In the next section we discuss an alternate method for policy gradient estimation that when implemented can be more robust and efficient.

C. Episodic REINFORCE

The Episodic REINFORCE method uses techniques of random sampling over probability distributions to find the estimation of the policy gradient. To begin we define the path probability of a trajectory $\boldsymbol{\tau}$ as follows:

$$p(\boldsymbol{\tau}) = \underbrace{p(\boldsymbol{x}_0)}_{init.dist.} \prod_{i=0}^{N} \underbrace{p(\boldsymbol{x}_{i+1}|\boldsymbol{x}_i,\boldsymbol{u}_i)}_{\text{state trans. dist.}} \underbrace{p(\boldsymbol{u}_i|\boldsymbol{x}_i;\boldsymbol{\theta})}_{\text{policy } \boldsymbol{\pi} \text{ dist.}} \tag{7}$$

From this we can rewrite our objective function 6 as the expecion of the reward of a path probability as follows:

$$\underset{\boldsymbol{\theta}}{\text{maximize}} \int p(\boldsymbol{\tau})R(\boldsymbol{\tau})d\boldsymbol{\tau}$$

We can now compute our policy gradient as:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \int \nabla_{\boldsymbol{\theta}} p(\boldsymbol{\tau})R(\boldsymbol{\tau})d\boldsymbol{\tau}$$

Using the log likelihood trick $\nabla_{\boldsymbol{\theta}} log(p(\boldsymbol{\tau})) = \frac{1}{p(\boldsymbol{\tau})}\nabla_{\boldsymbol{\theta}} p(\boldsymbol{\tau})$ we can rewrite our policy gradient as an expectation:

$$\int p(\boldsymbol{\tau})\nabla_{\boldsymbol{\theta}} log(p(\boldsymbol{\tau}))R(\boldsymbol{\tau})d\boldsymbol{\tau}$$

And rewrite this expression as an expectation:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = E_{p(\boldsymbol{\tau})}\Big[\nabla_{\boldsymbol{\theta}} log(p(\boldsymbol{\tau}))R(\boldsymbol{\tau})\Big]$$

This now becomes a problem of computing $\nabla_{\boldsymbol{\theta}} log(p(\boldsymbol{\tau}))$. At glance, it seems that we will need to understand the dynamics of our system to compute this gradient as the path probability depends on the state-control pairs in a trajectory. However, we can show that is not the case by rewriting our path probability using our definition in 7 and further simplifying:

$$\nabla_{\boldsymbol{\theta}} log(p(\boldsymbol{\tau})) = \nabla_{\boldsymbol{\theta}} log\left(p(\boldsymbol{x}_0)\prod_{i=0}^{N-1} p(\boldsymbol{x}_{i+1}|\boldsymbol{x}_i,\boldsymbol{u}_i)p(\boldsymbol{u}_i|\boldsymbol{x}_i;\boldsymbol{\theta})\right)$$
$$= \nabla_{\boldsymbol{\theta}} log(p(\boldsymbol{x}_0)) + \nabla_{\boldsymbol{\theta}} log\left(\prod_{i=0}^{N-1} p(\boldsymbol{x}_{i+1}|\boldsymbol{x}_i,\boldsymbol{u}_i)\right) + \nabla_{\boldsymbol{\theta}} log\left(\prod_{i=0}^{N-1} p(\boldsymbol{u}_i|\boldsymbol{x}_i;\boldsymbol{\theta})\right)$$
$$= \nabla_{\boldsymbol{\theta}} log(p(\boldsymbol{x}_0)) + \nabla_{\boldsymbol{\theta}} \sum_{i=0}^{N-1} log(p(\boldsymbol{x}_{i+1}|\boldsymbol{x}_i,\boldsymbol{u}_i)) + \nabla_{\boldsymbol{\theta}} \sum_{i=0}^{N-1} log(p(\boldsymbol{u}_i|\boldsymbol{x}_i;\boldsymbol{\theta}))$$
$$= \sum_{i=0}^{N-1} \nabla_{\boldsymbol{\theta}} log(p(\boldsymbol{u}_i|\boldsymbol{x}_i;\boldsymbol{\theta}))$$

Finally we can substitute this expression and and rewrite our policy gradient as:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = E_{p(\boldsymbol{\tau})}\left[\sum_{i=0}^{N} \nabla_{\boldsymbol{\theta}} log(\underbrace{p(\boldsymbol{u}_i|\boldsymbol{x}_i;\boldsymbol{\theta})}_{\text{policy } \boldsymbol{\pi} \text{ dist.}})R(\boldsymbol{\tau})\right]$$

This final expression is important as it shows the the expected reward only depends on the expected reward over a trajectory and the log probability of the parameterized control policy. Therefore, the derivatives with respecte to the dynamics do not need to be computed to the estimate the policy gradient.

Episodic REINFORCE is considered as a benchmark in developing Reinfocement Learning applications.It has good convergence guarantees in that it it can converge to the true gradient at the fastest theoritical rate. It can also be more computationaly inexpesive than the Finite Difference method as you no longer need to compute policy paramter perturbations and is more robust as this eliminates the potential to endanger the policy gradient estimation.

IV. Application to LQR and the Inverted Cart Pendulum

In this section we develop an experiment where we can apply the Finite Difference Method to a control systems problem. The purpose of this is to find out if we can use Reinforcement Learning to compute the optimal control gain found in the LQR optimal control formulation in Section II-B. For our system plant we used the inverted cart pendulum presented in Section II-A. This experiment was developed using MATLAB.

A. Problem Formulation

We begin reformulating the LQR optimal control problem into a Reinforcement Learning problem and expressing this as an optimization:

$$\begin{aligned}&\underset{\theta}{\text{maximize}} \quad \mathbb{E}\Big[R(\boldsymbol{\tau})\Big]\\ &R(\boldsymbol{\tau}) = \sum_{t=0}^{N} r(\boldsymbol{x}_t, \boldsymbol{u}_t, t)\\ &r(\boldsymbol{x}_t, \boldsymbol{u}_t, t) = -\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} - \boldsymbol{u}^T\boldsymbol{R}\boldsymbol{u}\\ &\boldsymbol{u} = -\boldsymbol{\theta}\boldsymbol{x}\end{aligned}$$

Since the goal of our Reinforcement Learning problem is to find the optimal feedback gains in 4 of LQR we parameterize our control policy such that $\boldsymbol{K} = \boldsymbol{\theta}$. In the LQR problem, the solution minimizes the cost function in 5. In our Reinforcement Learning formulation we want to maximize the reward, therefore we design our cost function so the we maximize the negative of the cost.

In Reinforcement Learning we do not know the physics of our system, however since we are using a simulation and not a real system, we use a forward-euler approximation of the system in 3 to sample our trajectories and compute our rewards.

B. Experiments and Results

We design our cost/reward functions such that:

$$\boldsymbol{Q} = \begin{bmatrix} 1e-2 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 1e-2 & 0\\ 0 & 0 & 0 & 0\end{bmatrix}$$
$$\boldsymbol{R} = \begin{bmatrix}1e-3\end{bmatrix}$$

Our function is scaled this way so that during the optimization when we sample trajectories using unstable $\boldsymbol{\theta}$ estimations, our cost/reward does not blow up quickly and cause the optimization to fail.

In our experiment we compare our Reinforcement Learning solution to the LQR solutions provided by MATLAB's function **lqr(A,B,Q,R)**. This results in the optimal control gains of

$$K = [-3.162, -3.42, 23.97, 4.49]^T$$

It was found through experimentation that having our initial $\boldsymbol{\theta}_0$ at unstable gains leads to an unstable system and the optimization fails. For this reason we use our known optimnal gains from LQR and move our initial guess away from the truth enough so that the system remains stable but far from optimal. To achieve this we use:

$$\boldsymbol{\theta}_0 = [-4.16, -4.42, 18.97, 3.49]^T$$

For our physics simulation, we begin at the initial state of

$$[\boldsymbol{x}, \dot{\boldsymbol{x}}, \boldsymbol{\psi}, \dot{\boldsymbol{\psi}}] = [0.5, 0, -0.5, 0]^T$$

The following figures illustrate the results of our experiment.

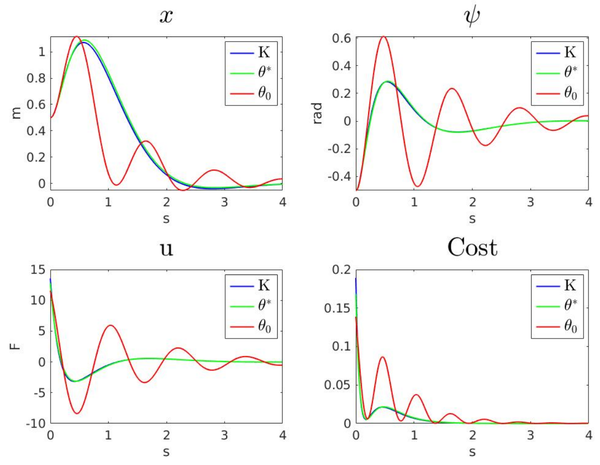

Fig. 5. Simulated trajectory using the known LQR control gain K, the solution to the Reinforcement Learning optimization $\boldsymbol{\theta}^*$, and the initial guess $\boldsymbol{\theta}_0$. Top left shows the position of the car x; top right shows the angle deviation from equilibrium of the pendulum ψ; bottom left shows the control input force on the cart u; bottom right shows the cost along the trajectory.

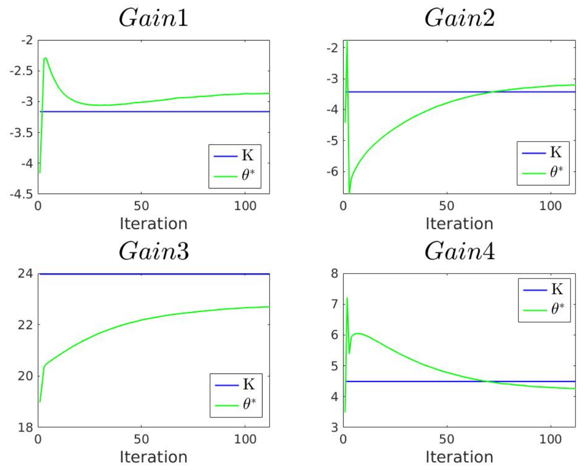

Fig. 6. $\boldsymbol{\theta}$ solutions at each iteration over the optimization and how they compare to the known LQR solution.

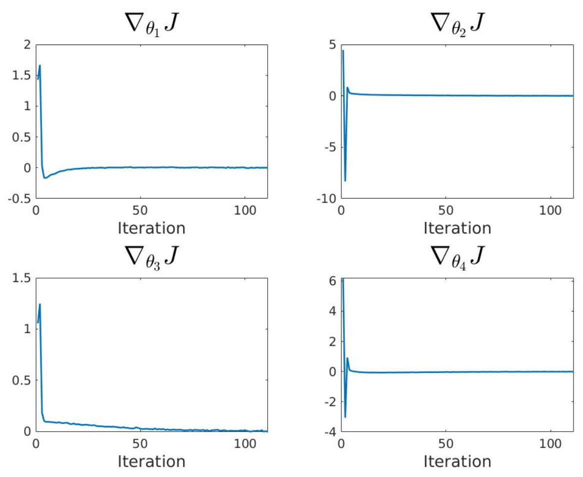

Fig. 7. The gradient estimates of with respect to our $\boldsymbol{\theta}$ solutions at each iteration over the optimization.

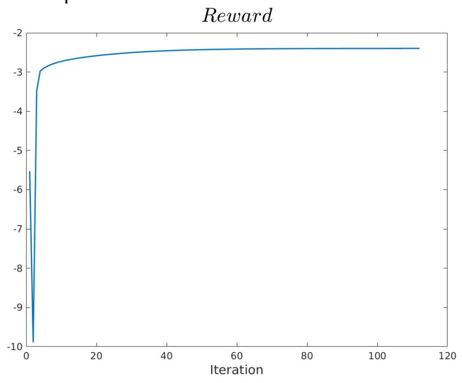

Fig. 8. The accumulated reward of our $\boldsymbol{\theta}$ solutions at each iteration over the optimization.

Our Reinforcement Learning optimization results in optimal feedback gains of:

$$\boldsymbol{\theta}^* = [-2.86, -3.19, 22.69, 4.26]^T$$

These feedback gains do not exactly equal result the $\boldsymbol{K}$ gains computed using LQR, as Figure 5 illustrates that some gains overshoot and some do not reach the optimal LQR gain. Despite this, Figure IV-B shows that these gains $\boldsymbol{\theta}^*$ result in a very similiar trajectory, control input, and cost as the LQR gains. Furthermore, this figure shows how much less optimal our initial guess gains $\boldsymbol{\theta}_0$ were and that we were able to improve these gains through Reinforcement Learning.

V. Conclusions

In this report we have given an introductory overview of the fields of Reinforcement Learning and Control Systems. We shown how there is clear connection between the kind of problems that these two fields try to solve. This drives the motivation to apply techniques from Reinforcement Learning to Control Systems in situtations where Control Systems is unable to solve the problem, such as situtations where a model of the dynamics is not available. Finally, we have given an introductory overview these ”model free” techniques and have show how can we apply one of these more simple techniques to a simple system such as an Inverted Cart Pendulum. We have shown promising initial results in our experiments that can likely be improved with more work in understanding how to deal with instability in a system where the formulation requires us to have no knowledge of the model.

References

[1] B. Recht, “A Tour of Reinforcement Learning: The View from Continous Control”, June 25, 2018.
[2] L. P. Kaelbling, M. L. Littman, A. W. Moore, “Reinforcement Learning: A Survey”, May 1, 1996.
[3] “5 Things You Need to Know about Reinforcement Learning.” KDnuggets, 10 Dec. 2020, www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html.
[4] Richard Sutton and Andrew Barto. Reinforcement Learning: An Introduction. Chapter 13. http://incompleteideas.net/book/bookdraft2017nov5.pdf.
[5] J. Peters, S. Schaal, “Reinforcement learning of motor skills with policy gradients”, February 24, 2008.
[6] J. Peters, S. Schaal, “Policy Gradient Methods for Robotics”, October, 2006.
[7] “Control Tutorials for MATLAB and Simulink”, University of Michigan, 10 Dec. 2020, ctms.engin.umich.edu/CTMS/example=InvertedPendulum
[8] “Linear Quadratic Regulator.” Wikipedia, Wikimedia Foundation, 6 Dec. 2020, en.wikipedia.org/wiki/Linear%E2%80%93quadratic_regulator.
[9] “Control Systems - Introduction.” Tutorialspoint, www.tutorialspoint.com/control_systems/control_systems_introduction.htm.