Evaluating Optimality of Legged Robot Controllers

Siddharth Trehan

February 16, 2019

Abstract

In nature and in robotics, is not clear what makes a gait optimal. It it known that animals take energy efficiency and stability into consideration, but there are often unknown objective functions that make a particular gait suitable for a particular task and a particular animal. Roboticists face similar challenges when choosing an objective function for robot behavior, and in this proposal I present a new idea which combines reinforcement learning with force control, which I propose to expand on and implement on high-dimensional robots such as the Cheetah robot. This idea not only allows the calculation of optimal robot controllers based on particular objective functions, but also the evaluation of specific robot controllers in terms of which objectives they optimize for and how well they achieve their intended goal.

1 Introduction

1.1 Optimal Gaits

Although it may not be clear what objective functions produce the best robot gaits, a good controller for robot locomotion should have the following key characteristics:

1. *Stability in stochastic environments* Modeling errors or external disturbances should be not cause the robot to diverge in behavior, and the controller should maintain the robot's gait even in the presence of noise.

2. *Long-term evaluation in decision-making* The controller should evaluate the impact of the control signal in a long-term sense and incorporate some degree of motion planning.

3. *Adaptability* The controller should to some degree be able to operate in environments and for tasks other than those for which it was conceived to operate in, or the model for computing the optimal controller should be simple enough to work with that a programmer can modify it to suit different environments and tasks

4. *Simplicity of formulation* The controller should be understandable and intuitive, and the program for deriving the controller for a task and an environment should have parameters that are intuitive and easy to determine

5. *Algorithmic efficiency* The controller should be able to run in real-time on the hardware available for the robot

Currently, the Cheetah robot has modules for model-predictive control, impedance control, position control, and leg utility evaluation running at 1 kHz. It also has programs for computing ground reactive forces and optimizing robot motion for a particular forward velocity. As a result, it has achieved feats such as being able to jump over obstacles and climb stairs.

However, what it lacks is the simplicity of formulation: all the Cheetah controllers do not fit under a single elegant framework. It is difficult to troubleshoot bugs in its behavior and to choose the correct parameters for different environments and tasks. All paramters must be fine-tuned and it is not clear how optimal a given controller is without testing on the physical robot.

Finally, it is not understood which objective functions to choose for our controllers and why the objective functions we use that produce good gaits indeed work. I hope to frame these objective functions in a probabilistic context and provide good justifications for the objective functions we use. In nature, the ultimate objective function is the survival of the animal– objective functions in robotics should be just as simple.

1.2 Roadmap

In this proposal, I first introduce reinforcement learning, a framework that can produce controllers with all the aforementioned qualities. It integrates a reward function over a trajectory and learns a non-linear controller that exactly optimizes a user-specified reward function. My focus is on keeping the reward function as simple as possible so that the behavior you want out of a robot can be predictably specified in the reward function.

The advantage of reinforcement learning is that theoretically, it guarantees total optimization of the reward function over the entire trajectory given any starting point or perturbation. It also allows the programmer to account for stochasticity in both measurement noise and process noise. However, its two main disadvantages are that it is not typically very intuitive to interpret the resulting controller and it can be unfeasible to train controllers in very high-dimensional spaces. Therefore, I introduce two new ideas: the connection of reinforcement learning to Lagrangian mechanics so that controllers can be understood in a more traditional form, and the formulation of reinforcement learning as a differential equation so that the controller can be learned much more quickly and with already-existing techniques.

Finally, I describe what I propose to do with my reinforcement learning idea. It may or may not turn out to be effective on the physical Cheetah robot, but it is likely to be an effective tool in simulation, where an optimal gait for a given reward function can be absolutely determined and can be compared with other proposed gaits. I also plan to incorporate my work on vision and gait planning in complex environments into the reinforcement learning algorithm.

2 Discrete-Space Reinforcement Learning

2.1 Setup

Reinforcement learning is a very general form of unsupervised learning in which an agent can learn to how to optimize a long-term reward. In robotics, it can be used to learn trajectories which optimize an objective function over the entire trajectory. The following assumptions are made when using reinforcement learning on a system:

1. The system is characterized by state $\vec{x}$.
2. There exists a function $A(\vec{x})$ that outputs the set of all possible actions you can take at state $\vec{x}$. For discrete systems this is a finite set, and for continuous systems an infinite set.
3. There exists a stochastic transition function $T(\vec{x}, a)$ that outputs the state the system will take in the next timestep given the current state is $\vec{x}$ and the action taken in the current state is a.
4. There exists a deterministic reward function $R(\vec{x}, a)$ that outputs a reward for a system's transition given the current state is $\vec{x}$ and the action taken in the current state is a.
5. We wish to find a controller function $F(\vec{x})$ that maps the state $\vec{x}$ to the optimal action that state, such that this function optimizes the expected value of the long-term sum of rewards over the entire trajectory $S = \sum_{t=0}^{\infty} \gamma^t R(\vec{x}(t), F(\vec{x}(t)))$. Here, γ is a discount factor $(0 < \gamma \leq 1)$ that determines how much the agent should consider immediate payoff versus longer-term payoff.

The problem here is that an infinite number of potential controller functions must be tried to see which one maximizes the expectation of the reward sum S. Even when we assume a particular shape for the controller function (e.g. linear controller), it is still computationally infeasible for S to be computed.

Q learning exploits the structure of S to find the optimal controller function F efficiently. The premise of Q learning is to learn a function $Q(\vec{x})$, the expected value of the reward sum S given you run the optimal controller F starting at point $\vec{x}$. To learn it efficiently, we re-arrange it so that its definition is recursive:

$$Q(\vec{x}) = R(\vec{x}, F(\vec{x})) + \gamma \mathbf{E}\left[Q\left(T(\vec{x}, F(\vec{x}))\right)\right] \tag{1}$$

Typically, the learning is made more efficient by discretization of the state space and the action space, and because Q is recursively defined, it is learned iteratively. The result of Q learning is that F can then be defined based on the Q function– the optimal action to take at every state is the one that leads to the highest value of Q. Intuitively, this transforms optimization over an entire trajectory to a local gradient ascent.

It is worth noting that the definition of Q depends on F, but F depends on Q. One solution is to initialize F as a completely stochastic function, and then using a variable learning rate and exploration rate, iteratively and alternatively update Q and F. This method is very commonly used and is guaranteed to converge to the optimal value of Q, but its computational requirements are often infeasible for high-dimensional spaces like for the MIT Cheetah Robot. Since we have more information about the physics of the transition function, it may be possible to determine Q in an approach that is partially numerical and partially analytical.

2.2 Application to Force Control

The frameworks of force control and impedence control can be combined into a more general approach, which will then allow us to find optimal trajectories more efficiently.

First, we can choose as our state vector the combined configuration and momentum of our robot: $\vec{x} = \begin{pmatrix} \vec{q} \\ \vec{p} \end{pmatrix}$, where $\vec{q}$ is the configuration and $\vec{p}$ is the momentum. These two vectors include both linear and angular components. We will let the action set be the range of possible forces and torques (which we will generally denote $\vec{F}$, encompassing both the linear and angular components) be the action set at every state. Then, our transition function can be defined based on mechanics:

$$T(\vec{x}, \vec{F}) = \begin{pmatrix} 1 & A \\ 0 & 1 \end{pmatrix} \vec{x} + \begin{pmatrix} 0 \\ B \end{pmatrix} \vec{F} + \begin{pmatrix} 0 \\ B \end{pmatrix} \vec{F}_{ext} + \Delta\vec{x} \tag{2}$$

In this equation:

1. A and B are matrices which convert momentum to a change in position and force to a change in momentum (they are dependent on configuration because configuration, momentum, and force include both linear and angular components)
2. $\vec{F}_{ext}$ is the vector of external forces (for the MIT Cheetah robot, gravity, joint frictions, and ground reaction force)
3. $\Delta\vec{x}$ is a zero-mean random-variable that takes into account disturbances we may not have accounted for

The reward function, of course, depends on the task we wish to perform. For complex tasks, there is likely no easily definable reward function. For many tasks that are programmed on the MIT Cheetah robot, however, the reward function can take a particularly nice form:

$$R(\vec{x}, \vec{F}) = -(\vec{x} - \vec{x}_d)^T H_X (\vec{x} - \vec{x}_d) - \vec{F}^T H_F \vec{F} \tag{3}$$

The main things to note about this equation are:

1. It is a quadratic in $\vec{x}$ and in $\vec{F}$
2. It enforces a sum-of-squared error between $\vec{x}$ and some constant desired state $\vec{x_d}$ based on a covariance matrix defined by H_X.
3. It enforces a regularization on $\vec{F}$ based on a covariance matrix defined by H_F.
4. It has a term that only depends on state $\vec{x}$ and a term that only depends on applied force $\vec{F}$.

One common task for which a reward function of such a form can be conceived of is moving the robot at a specific forward velocity. The MIT Cheetah robot is controlled primarily through an X-Box controller in which the operator can command a specific forward and horizontal velocity. A reward function which penalizes squared deviations between current and commanded robot momentum allows the robot to optimize its motion for a particular velocity while minimizing deviations from it throughout its trajectory.

The regularization term on force has three functions. One is to add an energy minimization term, which may or may not be useful in a particular application. The parameters of the H_F matrix in that case can be interpreted as conversion factors between force squared and energy, which can be computed from motor constants and armature resistances. The second function is to incorporate force and torque limits as penalties without having to add constraints. In this case, the H_F matrix is a covariance matrix with deviations proportional to these force and torque limits. This is related to energy minimization because the fundamental reason there are force and torque limits is because motors have power output limitations above which, integrated over time, increased temperature will cause mechanical damage to the motors. This approach may therefore be better than constraints because we are actually allowed to exceed these limits as long as the result of its integration is not too large. Finally, the third reason for regularization is that it produces a nicer connection between the reinforcement learning framework of robot control and the Lagrangian augmentation approach discussed in the following section.

We can define a nice version of the Q function by splitting the reward function into its state-dependent and force-dependent terms, $R(\vec{x}, \vec{F}) = R_X(\vec{x}) - \vec{F}^T H_F \vec{F}$:

$$Q(\vec{x}) = R_X(\vec{x}) + \max_{\vec{F}} \left(-\vec{F}^T H_F \vec{F} + \gamma \mathbf{E}\left[Q(T(\vec{x}, \vec{F})) \right] \right) \tag{4}$$

And the optimal controller can be defined as:

$$\vec{F}(\vec{x}) = \operatorname*{argmax}_{\vec{F}} \left(-\vec{F}^T H_F \vec{F} + \gamma \mathbf{E}\left[Q(T(\vec{x}, \vec{F})) \right] \right) \tag{5}$$

3 Continuous-Space Reinforcement Learning

3.1 Connection with Lagrangian Augmentation

In typical feedback control we augment the Lagrangian with a term that causes the robot to converge on a particular trajectory. It can be shown that the Q function is proportional to the augmentation term that produces the most optimal limit cycles, without any assumptions on the form of the reward function other than that its only dependence on $\vec{F}$ is quadratic.

To show this, consider Q learning in the continuous domain, where the size of a timestep δt is infinitesimally small and the state space and action space are no longer discrete. Then:

$$\begin{aligned} \frac{d\vec{x}}{dt} &\approx \frac{T(\vec{x}, \vec{F}) - \vec{x}}{\Delta t} \\ &\approx \begin{pmatrix} 0 & A/\Delta t \\ 0 & 0 \end{pmatrix} \vec{x} + \begin{pmatrix} 0 \\ B/\Delta t \end{pmatrix} \left(\vec{F} + \vec{F}_{ext} \right) + \Delta\vec{x}/\Delta t \end{aligned} \tag{6}$$

In the continuous domain, the control law is to choose the force that maximizes the quantity $-\vec{F}^T H_F \vec{F} + \mathbf{E}\left[\frac{dQ}{dt}\right]$, which can be expanded as follows:

$$\begin{aligned} -\vec{F}^T H_F \vec{F} + \mathbf{E}\left[\frac{dQ}{dt}\right] &= -\vec{F}^T H_F \vec{F} + \left(\frac{\partial Q}{\partial \vec{x}}\right) \mathbf{E}\left[\frac{d\vec{x}}{dt}\right] \\ &= -\vec{F}^T H_F \vec{F} + \left(\frac{\partial Q}{\partial \vec{x}}\right) \left(\begin{pmatrix} 0 & A/\Delta t \\ 0 & 0 \end{pmatrix} \vec{x} + \begin{pmatrix} 0 \\ B/\Delta t \end{pmatrix} \left(\vec{F} + \vec{F}_{ext} \right) \right) \\ &= -\vec{F}^T H_F \vec{F} + \left(\frac{\partial Q}{\partial \vec{q}}\right) \left(\frac{A}{\Delta t}\right) \vec{q} + \left(\frac{\partial Q}{\partial \vec{p}}\right) \left(\frac{B}{\Delta t}\right) \left(\vec{F} + \vec{F}_{ext} \right) \end{aligned} \tag{7}$$

Removing all terms that do not depend on $\vec{F}$ and solving the quadratic for the optimum, we find the optimal force is:

$$\vec{F}(\vec{x}) = \left(\frac{1}{2\Delta t} {H_F}^{-1} B^T \right) \left(\frac{\partial Q}{\partial \vec{p}} \right)^T \tag{8}$$

The first portion of the equation is constant, so this shows that if Δt is small enough and the reward function's dependence on $\vec{F}$ is only a quadratic regularization term, the optimum force

is proportional to a partial derivative of Q. This suggests that Q can be interpreted as a sort of energy function that can augment the robot's Lagrangian to produce the optimum controller for a particular reward function. More specifically, though a portion of it can be a conservative potential, it is more generally a Rayleigh dissipation function.

3.2 Formulation as Differential Equation

The continuous domain approximation also allows for a more efficient computation of Q. Consider the definition of Q in the continuous domain:

$$Q(\vec{x}(t_0)) = \frac{1}{\Delta t}\int_{t_0}^{\infty} e^{-t/\tau}\,\mathbf{E}\left[R_X\left(\vec{x}(t)\right) - \vec{F}(t)^T H_F \vec{F}(t)\right] dt \tag{9}$$

In this equation, we've redefined $\gamma = e^{-1/\tau}$ for more intuitive understanding: τ is the time constant of the window over which we're integrating the reward. We can then determine:

$$\mathbf{E}\left[\frac{dQ}{dt}\right] = \frac{\partial Q}{\partial \vec{x}}\mathbf{E}\left[\frac{d\vec{x}}{dt}\right] = \frac{1}{\tau}Q(\vec{x}) - \frac{1}{\Delta t}\mathbf{E}\left[R_X\left(\vec{x}\right) - \vec{F}(\vec{x})^T H_F \vec{F}(\vec{x})\right] \tag{10}$$

We can substitute in the optimal value of $\vec{F}(\vec{x})$ derived in Equation 8 and the value of $\frac{\partial Q}{\partial \vec{x}}\mathbf{E}\left[\frac{d\vec{x}}{dt}\right]$ derived in Equation 7. Furthermore, we use the chain rule in combination with Equation 6 to substitute $\frac{\partial Q}{\partial \vec{q}} = \left(\frac{\partial Q}{\partial \vec{p}}\right) A^{-1}$. The result is the following non-linear differential equation for Q:

$$Q(\vec{x}) = \frac{\tau}{4\Delta t^2}\left(\frac{\partial Q}{\partial \vec{p}}\right)\left(BH_F B^T\right)\left(\frac{\partial Q}{\partial \vec{p}}\right)^T + \frac{\tau}{\Delta t}\left(\frac{\partial Q}{\partial \vec{p}}\right)\vec{p} + \frac{\tau}{\Delta t}\left(\frac{\partial Q}{\partial \vec{p}}\right)B\vec{F}_{ext} + \frac{\tau}{\Delta t}R_X(\vec{x}) \tag{11}$$

Using Equation 11, we can learn Q using either discretization or value approximation without having to do any trajectory simulations.

4 Proposed Implementation

For my implementation of reinforcement learning for the Cheetah robot, I propose the following:

1. Test Q learning on a simple low-dimensional robot, such as a two-link SLIP model walking in a two-dimensional space
2. Find ways to reduce the dimensionality of high-dimensionality robots and find efficient discretizations and value approximators of the Q function
3. Simulate and evaluate the gaits produced by different reward functions in simulation and determine a good reward function for different tasks and different pre-determined environments (e.g. walking on stairs)
4. Evaluate the effectiveness of existing controllers by converting them to Q functions and seeing which reward functions they optimize for
5. Efficiently incorporate more complex, unknown environments and vision into the Q learning process
6. Get the most optimal controller to work on the physical Cheetah robot

I will leverage the existing simulation environment for the Cheetah and build my optimal gait controller on top of it.