Automatic 3D Character Animation Using Inverse Reinforcement Learning

Taesung Park

March 2013

Submitted to the Department of Computer Science of

Stanford University

Primary Advisor

Prof. Vladlen Koltun

Secondary Advisor

Prof. Oussama Khatib

Contents

Abstract

This report presents a framework for learning 3D character animation in the Markov Decision Process (MDP) setting using a reward function learned using Inverse Reinforcement Learning (IRL). Solving the 3D character control problem as an optimization in MDP using reinforcement learning is attractive because it automatically generates the details of the motion and is portable accross different environments. However, this approach has been infeasible due to two drawbacks: the curse of dimensionality and the subtlety of the reward function. This report overcomes the dimensionality problem by using a local iterative LQG method that makes local approximations, and using IRL that learns the precise reward function needed to generate the desired motion. The framework was evaluated on two models, a 2-DOF snake model and a 6-DOF bipedal walker model. Both models succesfully verified that optimal control in combination with IRL can be used to control characters to achieve the desired high-level goal. It was also shown that the reward function is portable to different domains by creating a walking motion under reduced gravity.

1 Introduction

As the industry for computer-generated animations and 3D games grows, the need to efficiently control character motions is getting more important. However, the industry still heavily relies on premade character animations that range from hand-crafted animations to motion captured data. These premade animations cannot adjust to new situations such as different terrains or obstruction by another object. Unfortunately, manually creating a new motion each time the environment changes can be very time consuming.

This report introduces a framework for learning 3D character control tasks in an MDP setting using the learned reward functions from Inverse Reinforcement Learning (IRL), or in other words, Inverse Optimal Control. Treating character animations as MDPs using IRL has advantages in many ways over the existing methods for animating characters, because it lets the user specify only the high level goals and not worry about the details that will be automatically filled in by the reinforcement learning algorithm. Thus, reinforcement learning has very low marginal costs of adapting to new environments. Moreover, as the MDP presented in this report uses a physics engine to give feedback to the motion controller, it can generate realistic motions that are physically valid and can even be used for simulating robots in virtual environments.

We work in a deterministic, fixed-horizon control task space where the states and

actions are continuous but the time steps are discrete. To outline the procedure, we first acquire expert demonstrations using various methods ranging from heuristically generated animations to motion captured trajectories. Then we embed the demonstrated motions as a sequence of state and action pairs over time in our MDP setting. Next, we run IRL to recover the reward function that makes the expert trajectory optimal. Once we derive the reward function, using a forward trajectory optimizer, we can optimize any given trajectory into the one that achieves the same goal that the expert was going for. At a high level, this means that we can create a versatile walking motion from a small set of motion-captured walking demonstrations.

Animating characters using standard reinforcement learning does have some shortcomings. One major challenge is the curse of dimensionality. In most cases, the state space of 3D characters is so large that usual global optimization methods with trajectory optimization become infeasible. To address this issue, this report uses the approach by [9] that made several improvements to the iterative LQG method in both efficiency and robustness. This approach overcomes the curse of dimensionality by approximating the control problem with linear dynamics and quadratic reward landscapes and finding a locally optimal trajectory in the approximation.

Another challenge of character animation using reinforcement learning is that there are many ways to achieve the same high level goal. For example, the user should not expect a human-like walking motion after setting the high level goal to be minimizing the distance between the current position of the character and the destination. The character can get there not only by walking, but also by running, any mix of walking and running, or even crawling, which are all legitimate ways of achieving the high level goal. Therefore, the user needs to carefully design the reward function to weed out the unwanted motions. Inverse reinforcement learning can be very helpful in this case, because we can just create a small set of the desired motions, learn the

reward function using IRL, and generate a wider range of motions on MDP with the reward function. One may wonder why IRL is useful at all if example trajectories must be created anyway: why not just directly use the example trajectories? IRL is superior to directly using the example motions because once the reward function is learned, it can be used across different environments, while manually creating the motions each time the situation changes will be very costly. This relates to the issue of imbuing personality to character motion. It is very difficult to design the reward function that lets the character walk with specific style. In this case, we can use IRL to also learn the stylistic component of the demonstrated motion and carry it to new environments.

Many existing IRL methods work by repeatedly running the forward control problem used for checking the optimality of the reward function ([1], [8], [12]). However, this approach of IRL inevitably makes the running time magnitudes slower than the forward problem. Dvijotham and Todorov addressed this problem by directly learning the value function [4]. However, this approach requires constructing good value function bases that are not portable accross domains. Our approach avoids the problem of repeated calls to a global solver by directly optimizing the reward landscape around the expert's trajectories, and this reward function can be ported to other domains too. Moreover, most previous methods assume that the expert demonstrations are globally optimal. While this can give us more insights if the experts indeed know globally optimal behaviors, in many cases even the experts do not have information about the global optimality. For example, a skilled driver may make perfect individual turns, but still fail in finding the fastest route to the destination.

In summary, our method synthesized character animations by learning the reward function using IRL and then running trajectory optimization. The work of this

report is divided into two parts: IRL and trajectory optimization. Both approaches will be discussed more in the following sections.

2 Background

2.1 Inverse Reinforcement Learning

Inverse Reinforcement Learning presented in this report largely follows part of the work by [6] that optimizes the reward function with respect to linear feature weights. This report differs from [6] in the derivation of the likelihood shown in sec. 2.1.1 in order to reduce the computation complexity by simpler indexing and emphasize the connection between IRL and iterative LQG.

We work in deterministic, finite-horizon control problem where the states $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_T)^T$ and actions $\mathbf{u} = (\mathbf{u}_1, \ldots, \mathbf{u}_T)^T$ are continuous but discrete timesteps are used. The relationship between the states and the actions is defined using the dynamics function $\mathcal{F}$:

$$\mathcal{F}(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{x}_{t+1}.$$

The reward $r(\mathbf{x}_t, \mathbf{u}_t)$ of this control problem is defined as a mapping from the state and action pairs to a real number. The optimal actions are the sequence of actions that maximizes the sum of the rewards encountered along the trajectory that the

actions generate. To be more precise,

$$\mathbf{u} = \arg\max_{\mathbf{u}} \sum_t r(\mathbf{x}_t, \mathbf{u}_t).$$

Inverse Reinforcement Learning tries to find the reward function that makes the expert demonstration "the best trajectory". That is, the optimal action $\mathbf{u}$ coincides with the expert demonstration under the reward function computed by IRL. Expert demonstrations are chosen to be the ones that accomplish the task very well. For example, a marathon runner can run very efficiently without specifically knowing what quantities he is optimizing, such as exact muscle usage on each body part, or the contact forces between the feet and the ground. Therefore, acquiring expert demonstration is often easier than designing the reward function that makes the demonstration optimal.

In our setting, the reward function is represented as a weighted linear combination of feature functions $f_i : (\mathbf{x}_t, \mathbf{u}_t) \to \mathbb{R}$. Then IRL reduces to the problem of finding the feature weights under which the expert demonstration matches the optimal actions. In real world situations, however, expert demonstrations are not perfectly optimal, so we add some uncertainty in our model, following the maximum entropy IRL model by Ziebart et al.([12]). In this model, the probability of taking action $\mathbf{u}$ is proportional to the exponential of the total reward:

$$P(\mathbf{u}|\mathbf{x}_0) = \frac{1}{Z} \exp\left(\sum_t r(\mathbf{x}_t, \mathbf{u}_t)\right), \tag{2.1}$$

where $\mathbf{x}_0$ is the initial state and Z is the partition function that makes sure the probability integrates to 1 over all actions. We see that uniform reward across different actions will produce random behavior while the policy becomes more deterministic when the stakes are high. Computing Z requires exploring all actions, which

becomes infeasible as the dimensionality of the problem increases. Therefore, the Laplace approximation of the probability model around $\mathbf{u}$ was used. This approximation not only allows efficient learning in high dimensional space, but also relaxes the requirement on the expert trajectory to be only locally optimal. This relaxation goes well with the real world situations, because expert demonstrations often lack global optimality. For example, an expert driver may take optimal individual turns but still not take the fastest route to the destination. Denoting $r(\mathbf{u})$ to be the sum of all rewards of the trajectory generated by $\mathbf{u}$, Equation 2.1 becomes

$$P(\mathbf{u}|\mathbf{x}_0) = e^{r(\mathbf{u})} \left[\int e^{r(\tilde{\mathbf{u}})} \mathrm{d}\tilde{\mathbf{u}} \right]^{-1}.$$

We approximate $r(\tilde{\mathbf{u}})$ using the second order Taylor expansion around $\mathbf{u}$.

$$r(\tilde{\mathbf{u}}) \approx r(\mathbf{u}) + (\tilde{\mathbf{u}} - \mathbf{u})^T \frac{\partial r}{\partial \mathbf{u}} + \frac{1}{2}(\tilde{\mathbf{u}} - \mathbf{u})^T \frac{\partial^2 r}{\partial \mathbf{u}^2}(\tilde{\mathbf{u}} - \mathbf{u}).$$

Using notation $\mathbf{g} \equiv \frac{\partial r}{\partial \mathbf{u}}$ and $\mathbf{H} \equiv \frac{\partial^2 r}{\partial \mathbf{u}^2}$ and writing the integrand in multivariate normal distribution form, we can analytically compute the integral.

$$P(\mathbf{u}|\mathbf{x}_0) \approx e^{r(\mathbf{u})} \left[\int e^{r(\mathbf{u}) + (\tilde{\mathbf{u}}-\mathbf{u})^T \mathbf{g} + \frac{1}{2}(\tilde{\mathbf{u}}-\mathbf{u})^T \mathbf{H} (\tilde{\mathbf{u}}-\mathbf{u})} \mathrm{d}\tilde{\mathbf{u}} \right]^{-1} \tag{2.2}$$

$$= \left[\int e^{-\frac{1}{2}\mathbf{g}^T \mathbf{H}^{-1} \mathbf{g} + \frac{1}{2}(\mathbf{H}(\tilde{\mathbf{u}}-\mathbf{u})+\mathbf{g})^T \mathbf{H}^{-1} (\mathbf{H}(\tilde{\mathbf{u}}-\mathbf{u})+\mathbf{g})} \mathrm{d}\tilde{\mathbf{u}} \right]^{-1} \tag{2.3}$$

$$= e^{\frac{1}{2}\mathbf{g}^T \mathbf{H}^{-1} \mathbf{g}} |-\mathbf{H}|^{\frac{1}{2}} (2\pi)^{-\frac{D_\mathbf{u}}{2}}, \tag{2.4}$$

where $D_\mathbf{u}$ is the dimension of the action. Note that the last step that uses the property of multivariate normal distribution is valid only when the Hessian $\mathbf{H}$ is negative definite. Indeed, maintaining $\mathbf{H}$ to be negative definite is not trivial, and it will be discussed later in depth in sec. 2.1.3. Taking log on both sides of the

equations gives us the approximate log likelihood.

$$\mathcal{L} = \frac{1}{2}\mathbf{g}^T\mathbf{H}^{-1}\mathbf{g} + \frac{1}{2}\log|-\mathbf{H}| - \frac{D_{\mathbf{u}}}{2}\log 2\pi. \tag{2.5}$$

The Inverse Reinforcement Learning problem is then the process of maximizing (2.5) with respect to the parametrization of the reward function. In this report, we assume that the reward function is represented as a linear combination of the feature functions $\mathbf{f}_i : (\mathbf{x}_t, \mathbf{u}_t) \to \mathbb{R}$. Therefore, IRL finds the set of weights on the features that maximizes $\mathcal{L}$.

We can gain some intuition behind the approximate log likelihood. Note that $\frac{1}{2}\mathbf{g}^T\mathbf{H}^{-1}\mathbf{g} \leq 0$ for all $\mathbf{g}$ because $\mathbf{H}$ is negative definite and an inverse of a negative definite matrix is also negative definite. Hence, the first term $\frac{1}{2}\mathbf{g}^T\mathbf{H}^{-1}\mathbf{g}$ will attain its maximum when $\mathbf{g} = 0$. This means $\mathcal{L}$ will increase as we get close to the local peak of the reward. The second term $\frac{1}{2}\log|-\mathbf{H}|$ describes how steep the peak is, and we prefer the peak to be steeper, which in general suggests large determinant of the Hessian.

Maximizing $\mathcal{L}$ requires fast evaluation of $\mathcal{L}$. The computation is dominated by the term $\mathbf{H^{-1}g}$, so the cost of calculating $\mathcal{L}$ is cubic in the number of timesteps T and the action dimensionality. Since this is too slow, we introduce a method to calculate $\mathcal{L}$ in time linear in T by linearizing the dynamics. For calculation of the likelihood, we developed a method that resembles the Linear-Quadratic Regulator (LQR) approach, which approximates the control task with linear dynamics and quadratic reward matrices.

2.1.1 Efficient Log Likelihood Estimation

Instead of the method presented in [6], this report introduces another method of likelihood evaluation method following the idea of [9]. Although both methods use the same LQR assumption, the indexing scheme used by [9] simplifies the formula that we can use to derive the log likelihood. Moreover, the identical method can be reused later in the forward trajectory optimization phase, which also requires fast reward estimation.

Let $\bar{\mathbf{x}}_t$ and $\bar{\mathbf{u}}_t$ be the state and action deviation from the example trajectory. That is, $\bar{\mathbf{x}}_t = \mathbf{x}_t - \mathbf{x}_t^*$ and $\bar{\mathbf{u}}_t = \mathbf{u}_t - \mathbf{u}_t^*$, where $\mathbf{x}_t^*$ and $\mathbf{u}_t^*$ are the state and action of the example trajectory. Under the MaxEnt model by [12], the probability of taking action $\bar{\mathbf{u}}_t$ in $\bar{\mathbf{x}}_t$ around the example trajectory is

$$P(\bar{\mathbf{u}}_t|\bar{\mathbf{x}}_t) = \exp\left(Q_t(\bar{\mathbf{x}}_t, \bar{\mathbf{u}}_t) - V_t(\bar{\mathbf{x}}_t)\right),$$

where $Q_t(\bar{\mathbf{x}}_t, \bar{\mathbf{u}}_t) = r(\bar{\mathbf{x}}_t, \bar{\mathbf{u}}_t) + V_{t+1}(\bar{\mathbf{x}}_{t+1})$, whose terms are all defined around the example trajectory, and the value function V_t is

$$V_t(\bar{\mathbf{x}}) = \log \int e^{Q_t(\bar{\mathbf{x}}, \bar{\mathbf{u}})} d\bar{\mathbf{u}}.$$

Now we used second order Taylor expansion of Q_t around the example trajectory.

$$Q_t(\bar{\mathbf{x}}, \bar{\mathbf{u}}) \approx Q_t(\mathbf{0}, \mathbf{0}) + Q_{\mathbf{x}}^T \bar{\mathbf{x}} + Q_{\mathbf{u}}^T \bar{\mathbf{u}} + \bar{\mathbf{x}}^T Q_{\mathbf{xu}} \bar{\mathbf{u}} + \frac{1}{2} \bar{\mathbf{x}}^T Q_{\mathbf{xx}} \bar{\mathbf{x}} + \frac{1}{2} \bar{\mathbf{u}}^T Q_{\mathbf{uu}} \bar{\mathbf{u}}.$$

Now we can compute $V_t(\bar{\mathbf{x}})$ by using the approximated formula for Q_t and the property of Gaussian distribution.

$$
\begin{aligned}
V_t(\bar{\mathbf{x}}) &= \log \int e^{Q_t(\bar{\mathbf{x}},\bar{\mathbf{u}})} d\bar{\mathbf{u}} \\
&\approx \log \int \exp\left(Q_t(\mathbf{0},\mathbf{0}) + Q_{\mathbf{x}}^T\bar{\mathbf{x}} + Q_{\mathbf{u}}^T\bar{\mathbf{u}} + \bar{\mathbf{x}}^T Q_{\mathbf{xu}}\bar{\mathbf{u}} + \frac{1}{2}\bar{\mathbf{x}}^T Q_{\mathbf{xx}}\bar{\mathbf{x}} + \frac{1}{2}\bar{\mathbf{u}}^T Q_{\mathbf{uu}}\bar{\mathbf{u}}\right) d\bar{\mathbf{u}} \\
&= Q_t(\mathbf{0},\mathbf{0}) + Q_{\mathbf{x}}^T\bar{\mathbf{x}} + \frac{1}{2}\bar{\mathbf{x}}^T Q_{\mathbf{xx}}\bar{\mathbf{x}} - \frac{1}{2}(Q_{\mathbf{u}} + Q_{\mathbf{ux}}\bar{\mathbf{x}})^T Q_{\mathbf{uu}}^{-1}(Q_{\mathbf{u}} + Q_{\mathbf{ux}}\bar{\mathbf{x}}) \\
&\quad - \frac{1}{2}\log|-Q_{\mathbf{uu}}| - \frac{D_{\mathbf{u}}}{2}\log 2\pi.
\end{aligned}
\tag{2.6}
$$

The coefficients $Q_{\mathbf{x}}$, $Q_{\mathbf{u}}$, $Q_{\mathbf{ux}}$, $Q_{\mathbf{xx}}$, and $Q_{\mathbf{uu}}$ can be computed by directly differentiating $Q_t(\bar{\mathbf{x}}_t, \bar{\mathbf{u}}_t) = r(\bar{\mathbf{x}}_t, \bar{\mathbf{u}}_t) + V_{t+1}(\bar{\mathbf{x}}_{t+1})$. For example, $Q_{\mathbf{x}}$ is computed

$$
\begin{aligned}
Q_{\mathbf{x}} &= \frac{\partial}{\partial \bar{\mathbf{x}}}\left[r(\bar{\mathbf{x}},\bar{\mathbf{u}}) + V_{t+1}(\mathcal{F}(\bar{\mathbf{x}},\bar{\mathbf{u}}), t+1)\right] && (2.7)\\
&= r_{\mathbf{x}} + \mathcal{F}_{\mathbf{x}}^T V_{\mathbf{x}}' && (2.8)
\end{aligned}
$$

where the prime ($'$) is used for denoting the next timestep $i+1$. The other quantities are computed similarly as below.

$$
\begin{aligned}
Q_{\mathbf{u}} &= r_{\mathbf{u}} + \mathcal{F}_{\mathbf{u}}^T V_{\mathbf{x}}' && (2.9)\\
Q_{\mathbf{xx}} &= r_{\mathbf{xx}} + \mathcal{F}_{\mathbf{x}}^T V_{\mathbf{xx}}' \mathcal{F}_{\mathbf{x}} + V_{\mathbf{x}}' \cdot \mathcal{F}_{\mathbf{xx}} && (2.10)\\
Q_{\mathbf{uu}} &= r_{\mathbf{uu}} + \mathcal{F}_{\mathbf{u}}^T V_{\mathbf{xx}}' \mathcal{F}_{\mathbf{u}} + V_{\mathbf{x}}' \cdot \mathcal{F}_{\mathbf{uu}} && (2.11)\\
Q_{\mathbf{ux}} &= r_{\mathbf{ux}} + \mathcal{F}_{\mathbf{u}}^T V_{\mathbf{xx}}' \mathcal{F}_{\mathbf{x}} + V_{\mathbf{x}}' \cdot \mathcal{F}_{\mathbf{ux}}. && (2.12)
\end{aligned}
$$

In this algorithm, we can drop the last terms involving $\mathcal{F}_{\mathbf{xx}}$, $\mathcal{F}_{\mathbf{ux}}$, and $\mathcal{F}_{\mathbf{uu}}$ since we assumed that the dynamics is linear. To compute the derivatives of the value

function V, we differentiate 2.6 with respect to $\bar{\mathbf{x}}$ at $\mathbf{0}$.

$$V_{\mathbf{x}}(i) = Q_{\mathbf{x}} - Q_{\mathbf{u}}Q_{\mathbf{uu}}^{-1}Q_{\mathbf{ux}} \tag{2.13}$$

$$V_{\mathbf{xx}}(i) = Q_{\mathbf{xx}} - Q_{\mathbf{xu}}Q_{\mathbf{uu}}^{-1}Q_{\mathbf{ux}}. \tag{2.14}$$

Finally, we can iterate Equation 2.8, 2.9, 2.10, 2.11, 2.12, 2.13, and 2.14 backwards in time to compute the value in quadratic model. Moreover, the log likelihood (2.5) can be computed along the way, because $\mathbf{g} \equiv \frac{\partial r}{\partial \mathbf{u}} = Q_{\mathbf{u}}$ and $\mathbf{H} \equiv \frac{\partial^2 r}{\partial \mathbf{u}^2} = Q_{\mathbf{uu}}$, from the definition of Q, and the total log likelihood is the sum of the log likelihoods at all timesteps.

2.1.2 Working with linear reward functions

In this paper, the reward function r is represented as a weighted linear combination of features.

$$r(\mathbf{x}_t, \mathbf{u}_t) = \sum_k \theta_k f^{(k)}(\mathbf{x}_t, \mathbf{u}_t) = \theta^T \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t)$$

Features can be used on both the states and actions, and are manually constructed to characterize the expert demonstration. For example, in a situation where IRL observes a human walking motion, one possible feature would be the absolute velocity of the center of mass of the human. Velocity can be computed from the state vectors of the trajectory, and it can probably characterize the human walking behavior from other motions. Following the linearity, the gradient $\mathbf{g}$ and the Hessian $\mathbf{H}$ can also

be represented as the sum of each feature gradient and Hessian.

$$\mathbf{g} = \sum_k \theta_k \mathbf{g}^{(k)}$$

$$\mathbf{H} = \sum_k \theta_k \mathbf{H}^{(k)},$$

where $\mathbf{g}^{(k)}$ and $\mathbf{H}^{(k)}$ denote the gradient and Hessian of each feature. The gradients and Hessians of the features can be computed analytically, if the features are simple. However, in this report, finite differencing was used to calculate the derivatives of the features to avoid the complexity of having to analytically differentiate each newly designed feature.

2.1.3 Regularization

It was mentioned earlier that the log likelihood $\mathcal{L}$ in Equation 2.5 is defined only when $\mathbf{H}$ is negative definite. This corresponds to the case where the trajectory of the expert demonstration lies near the peak of the reward landscape. While IRL should indeed find a reward function that meets this condition at the end of the algorithm, it is not easy to pick initial θ such that $\mathbf{H}$ is negative definite and $\mathcal{L}$ can be evaluated. Therefore, we add a dummy regularization feature that makes sure $\mathbf{H}$ is negative definite. This feature has gradient zero and Hessian negative identity. Then the new Hessian with the regularizer becomes

$$\mathbf{H} = \sum_k \theta_k \mathbf{H}^{(k)} - \theta_{reg}\mathbf{I},$$

so with a sufficiently large θ_{reg}, for all ξ,

$$\xi^T \mathbf{H} \xi = \xi^T (\sum_k \theta_k \mathbf{H}^{(k)}) \xi - \theta_{reg} \xi^2 < 0,$$

which means the new Hessian is negative definite. We can pick such θ_{reg} just by doubling the value of θ_{reg} until $\mathbf{H}$ becomes negative definite. Then we will maximize $\mathcal{L}$ with respect to θ, θ_{reg}. Note that θ_{reg} should be zero in order to solve the original problem. During the optimization, therefore, we will decrease θ_{reg} to zero in order to solve the original problem. Augmented Lagrangian method ([3]), described in Fig. 2.1, was used for such task.

Repeat until convergence:
Maximize $\mathcal{L} \leftarrow \frac{\mu_k}{2}\theta_{reg}^2 + \lambda_k \theta_{reg}$.
Set $\lambda_{k+1} \leftarrow \lambda_k - \mu_k \theta_{reg}$,
If θ_{reg} has not decreased, $\mu_{k+1} \leftarrow 10\mu_k$. Otherwise, $\mu_{k+1} \leftarrow \mu_k$.

Figure 2.1: Augmented Lagrangian method

2.2 Trajectory Optimization

2.2.1 Trajectory Optimizer

We optimize the trajectory with respect to the reward function learned in IRL. I used a tuned version of the iterative Linear Quadratic Gaussian (iLQG), introduced in [9]. The idea is the same as in the log likelihood estimation algorithm of IRL phase in that the dynamics is linearized and the reward is approximated quadratically.

In [9], V is defined as the maximum reward that can be gained by taking the optimal actions starting from $\mathbf{x}$ at timestep t. Let $V(\mathbf{x}, t) \equiv \max_{\mathbf{u}=\mathbf{u}_t,\ldots,\mathbf{u}_T} \mathbf{r}(\mathbf{x}_t, \mathbf{u})$. We can use dynamic programming in order to find $V(\mathbf{x}_0, 0)$ by proceeding backwards in time

from the last timestep, with the help of the following relationship.

$$V(\mathbf{x}, t) = \max_{\mathbf{u}_t} [r(\mathbf{x}, \mathbf{u}_t) + V(\mathcal{F}(\mathbf{x}, \mathbf{u}_t), t + 1)]. \tag{2.15}$$

We use the second order Taylor expansion of the argument $r(\mathbf{x}, \mathbf{u}_t) + V(\mathcal{F}(\mathbf{x}, \mathbf{u}_t), t + 1)$ that locally approximates the landscape of the value function around the i-th $(\mathbf{x}, \mathbf{u}_t)$. To simplify calculation, let's introduce a perturbation term $Q(\delta\mathbf{x}, \delta\mathbf{u})$ such that

$$r(\mathbf{x}+\delta\mathbf{x}, \mathbf{u}_t+\delta\mathbf{u})+V(\mathcal{F}(\mathbf{x}+\delta\mathbf{x}, \mathbf{u}_t+\delta\mathbf{u}), t+1) = r(\mathbf{x}, \mathbf{u}_t)+V(\mathcal{F}(\mathbf{x}, \mathbf{u}_t), t+1)+Q(\delta\mathbf{x}, \delta\mathbf{u}), \tag{2.16}$$

and expand $Q(\delta\mathbf{x}, \delta\mathbf{u})$ to second order around $\mathbf{0}$.

$$Q(\delta\mathbf{x}, \delta\mathbf{u}) \approx Q_{\mathbf{x}}^T\delta\mathbf{x} + Q_{\mathbf{u}}^T\delta\mathbf{u} + \delta\mathbf{x}^T Q_{\mathbf{xu}}\delta\mathbf{u} + \frac{1}{2}\delta\mathbf{x}^T Q_{\mathbf{xx}}\delta\mathbf{x} + \frac{1}{2}\delta\mathbf{u}^T Q_{\mathbf{uu}}\delta\mathbf{u}.$$

Trajectory optimization is closely related to the process of computing the value $V(\mathbf{x}, t)$ at $t = 1$ as it involves finding the optimal value $\mathbf{u}$ that maximizes the total reward. We can analytically compute $V(\mathbf{x}, t)$ by directly maximizing $Q(\delta\mathbf{x}, \delta\mathbf{u})$ with respect to $\delta\mathbf{u}$.

$$\delta\mathbf{u}^* = \arg\max_{\delta\mathbf{u}} Q(\delta\mathbf{x}, \delta\mathbf{u}) = -Q_{\mathbf{uu}}^{-1}(Q_{\mathbf{u}} + Q_{\mathbf{ux}}\delta\mathbf{x}). \tag{2.17}$$

Plugging in $\delta\mathbf{u}^*$ to Equation 2.15, we get

$$\begin{aligned} V(\mathbf{x}, t) &= r(\mathbf{x}, \mathbf{u}_t) + V(\mathcal{F}(\mathbf{x}, \mathbf{u}_t), t + 1) + Q(\delta\mathbf{x}, \delta\mathbf{u}^*) \\ &\approx r(\mathbf{x}, \mathbf{u}_t) + V(\mathcal{F}(\mathbf{x}, \mathbf{u}_t), t + 1) + Q_{\mathbf{x}}^T\delta\mathbf{x} + Q_{\mathbf{u}}^T\delta\mathbf{u}^* + \delta\mathbf{x}^T Q_{\mathbf{xu}}\delta\mathbf{u}^* \\ &+ \frac{1}{2}\delta\mathbf{x}^T Q_{\mathbf{xx}}\delta\mathbf{x} + \frac{1}{2}\delta\mathbf{u}^{*T} Q_{\mathbf{uu}}\delta\mathbf{u}^*. \end{aligned} \tag{2.18}$$

Observe that the quadratic approximation of Q resembles the likelihood estimation in IRL. Therefore, we can reuse the algorithm that estimates the value of a trajectory in IRL. To reiterate, at each timestep t,

$$Q_{\mathbf{x}} = r_{\mathbf{x}} + \mathcal{F}_{\mathbf{x}}^T V'_{\mathbf{x}} \tag{2.19}$$

$$Q_{\mathbf{u}} = r_{\mathbf{u}} + \mathcal{F}_{\mathbf{u}}^T V'_{\mathbf{x}} \tag{2.20}$$

$$Q_{\mathbf{xx}} = r_{\mathbf{xx}} + \mathcal{F}_{\mathbf{x}}^T V'_{\mathbf{xx}} \mathcal{F}_{\mathbf{x}} + V'_{\mathbf{x}} \cdot \mathcal{F}_{\mathbf{xx}} \tag{2.21}$$

$$Q_{\mathbf{uu}} = r_{\mathbf{uu}} + \mathcal{F}_{\mathbf{u}}^T V'_{\mathbf{xx}} \mathcal{F}_{\mathbf{u}} + V'_{\mathbf{x}} \cdot \mathcal{F}_{\mathbf{uu}} \tag{2.22}$$

$$Q_{\mathbf{ux}} = r_{\mathbf{ux}} + \mathcal{F}_{\mathbf{u}}^T V'_{\mathbf{xx}} \mathcal{F}_{\mathbf{x}} + V'_{\mathbf{x}} \cdot \mathcal{F}_{\mathbf{ux}} \tag{2.23}$$

$$\Delta V(i) = -\frac{1}{2} Q_{\mathbf{u}} Q_{\mathbf{uu}}^{-1} Q_{\mathbf{u}} \tag{2.24}$$

$$V_{\mathbf{x}}(i) = Q_{\mathbf{x}} - Q_{\mathbf{u}} Q_{\mathbf{uu}}^{-1} Q_{\mathbf{ux}} \tag{2.25}$$

$$V_{\mathbf{xx}}(i) = Q_{\mathbf{xx}} - Q_{\mathbf{xu}} Q_{\mathbf{uu}}^{-1} Q_{\mathbf{ux}}, \tag{2.26}$$

and we proceed backward in time. Once the quantities $Q_{\mathbf{uu}}$, $Q_{\mathbf{u}}$, and $Q_{\mathbf{ux}}$ are computed for all timesteps, using (2.17), we can proceed forward in time to compute the new trajectory $\hat{\mathbf{x}}$.

$$\hat{\mathbf{x}}(1) = \mathbf{x}(1) \tag{2.27}$$

$$\hat{\mathbf{u}}(t) = \mathbf{u}(t) - Q_{\mathbf{uu}}^{-1} Q_{\mathbf{u}} - Q_{\mathbf{uu}}^{-1} Q_{\mathbf{ux}} (\hat{\mathbf{x}}(t) - \mathbf{x}(t)) \tag{2.28}$$

$$\hat{\mathbf{x}}(t+1) = \mathcal{F}(\hat{\mathbf{x}}(t), \hat{\mathbf{u}}(t)) \tag{2.29}$$

As can be seen, the forward trajectory optimization is very similar to sec. 2.1.1. This is because both IRL and the forward trajectory optimization assume linear dynamics and quadratic reward landscape. The difference is that IRL maximizes the log likelihood with respect to the reward function parameters, but the trajectory optimizer does it with respect to the controls.

2.2.2 Regularization

In [9] by Tassa, Erez, and Todorov, they "introduced a scheme that penalizes deviations from the states rather than controls". On the other hand, our algorithm used in IRL actually penalizes deviations from the controls. Both approaches symantically make sense, and they could be used interchangeably without clear distinction. For the sake of consistency, state regularization was used throughout this report.

2.2.3 Improved Line Search

Although finding local optimum could be advantageous in IRL because it removes the assumption that the expert demonstration is globally optimal, it does have some disadvantage in the forward trajectory optimization problem. As we iterate through (2.28) and (2.29) to generate a new trajectory, $\hat{\mathbf{x}}$ and $\hat{\mathbf{u}}$ may deviate too far from the original region of quadratic approximation. This means the reward of the new trajectory may even be smaller than the original trajectory. In order to address this problem, Tassa et al. in their paper in 2012 ([9]) introduced a simple solution that pulls back the new trajectory to the original one if divergence occurs. Let's add a new parameter $0 < \alpha \leq 1$ to (2.28).

$$\hat{\mathbf{u}}(t) = \mathbf{u}(t) - \alpha Q_{\mathbf{uu}}^{-1} Q_{\mathbf{u}} - Q_{\mathbf{uu}}^{-1} Q_{\mathbf{ux}} (\hat{\mathbf{x}}(t) - \mathbf{x}(t))$$

$\alpha = 1$ means we fully accept the new trajectory, while $\alpha = 0$ means we use the original trajectory unchanged. We can calculate the expected reward change using (2.24). Starting with $\alpha = 1$, we cut down α by half until the actual reward increase comparably matches the estimated reward increase.

3 Evaluation

This report evaluates the suggested IRL and the forward trajectory optimization mainly on two models, the simple 3-link snake model and the 2D walker.

3.1 Model Control Simulator

Controlling a character model in virtual environment requires simulating the kinematics and the dynamics of the model as well as its interaction with the environment based on physics. As this report focuses on optimal character control and not on accurate simulation, the simulator developed by [10] was used. The simulator can use several contact models, and the one used in this report is the SpringTau model that appears in Section III B of [9]. Timestep of size $10ms$ was used.

3.2 Simple 3-Link Snake Model

The first one is a simple 3-link snake. This snake model consists of three links and two joints as seen in Fig. 3.1. The snake model is surrounded by a medium with some viscosity but no gravity, so it can swim through the medium. In fact, the two joints are revolute joints with one degree of freedom, so the motion of the snake is limited in a horizontal plane. The goal is to have the snake model learn how to swim forward. Since this model is very simple, the desired motion could be learned even

without the help of IRL, which is how I actually acquired the expert demonstration. Still, this simple snake model can be used for checking the validity of IRL algorithm and also as a stepping stone for more complicated models.

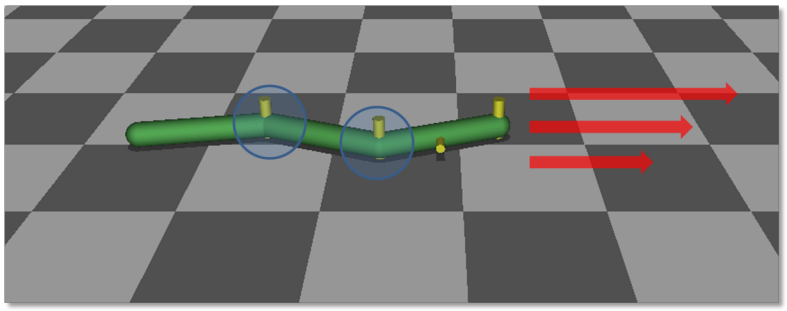

Figure 3.1: Simple 3D snake model

3.2.1 Feature Construction

3.2.1.1 Quadratic velocity deviation from the desired velocity.

This feature expresses the velocity of the root position of the snake model.

$$f_{velocity,quadratic}(\mathbf{x}_t, \mathbf{u}_t) = -\frac{1}{2}||v_t - v^*||^2,$$

where v_t is the velocity of the snake model and $v^*$the desired velocity.

3.2.1.2 Radial basis function (RBF) on the desired velocity

Another method of expressing the velocity is using radial basis functions.

$$f_{velocity,rbf}(\mathbf{x}_t, \mathbf{u}_t) = \exp(-\frac{1}{2}w||v_t - v^*||^2),$$

where v_t is the velocity of the snake model , $v^*$the desired velocity, and w the width of the basis function.

3.2.1.3 Penalty on torques

For realistic motion of the swimming snake, it is crucial that the snake does not take unnecessary movements that waste energy. This can be represented as minimizing the joint torques, which roughly correspond to muscle usage.

$$f_{torque}(\mathbf{x}_t, \mathbf{u}_t) = -\frac{1}{2}\sum_i ||u_{t,i}||^2,$$

where $u_{t,i}$ is the torque on the ith joint at time t. I created a single feature that accounts for all torques exerted by the snake model.

3.2.2 Expert Demonstration

The example motion of the snake model was created directly from the forward trajectory optimization. Two features were used: quadratic deviation from the desired velocity of the model, and quaratic function on the magnitude of the torque. Because the model needs to be as close to the desired velocity as possible while minizing the torques, the weights on the features were chosen to be positive and negative, respectively. I could handpick the set of weights that created the desired motion within a couple of trials.

3.2.3 Inverse Reinforcement Learning and Forward Trajectory Optimization

To show that IRL successfully recovers the feature weights that will generate the same kind of forward motion that the example trajectory has demonstrated, IRL was run multiple times with different set of features, and verified that every single time IRL prefers higher velocity than zero velocity. To do this, I created for each experiment multiple RBF features of sec. 3.2.1.2 with $v^* = startpos + k \cdot gap$, where $k = 0, \ldots, num - 1$. Different values of $startpos$, gap, and num were tried as shown

in Fig. 3.2. Observe that the curves are generally ascending functions of velocity, which means that the reward function generally prefers higher velocities. Also note that the curves sometimes peak around velocity 0.15 and become decreasing past that, because the velocity of the example trajectory was 0.76, so even faster motion should be less desirable.

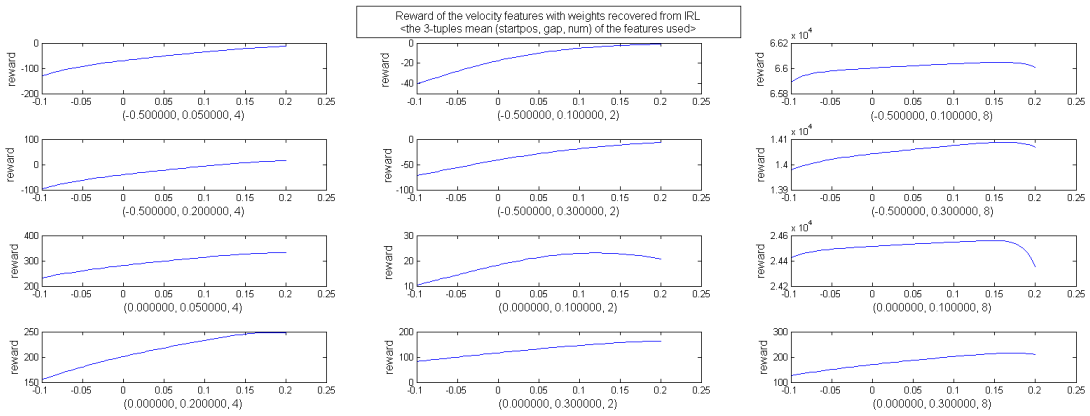

Figure 3.2: Reward - velocity curves of the velocity features with weights recovered by IRL.

The forward trajectory optimization was run with each of the reward functions recovered by IRL using different $(startpos, gap, num)$. Random initial poses were used. All optimizations successfully generated motions that look very similar to the example trajectory. Note that the example and the optimized trajectory are not completely identical because different random initializations were used, as shown in the video uploaded online(http://youtu.be/IV0dsQxiro4).

3.3 2D Walker

The next model is what I call a 2D walker. This model consists of the torso, two thighs, two calves, and two feet, as shown in Fig. 3.3. Since all the joints are revolute joints along the same axis, the motion of this model is limited to 2D. Although this model is a significantly simplified model of a real human, it still conveys roughly the same mechanism as human walking. Therefore, this model can be deemed as a intermediate step before working with the real human motion.

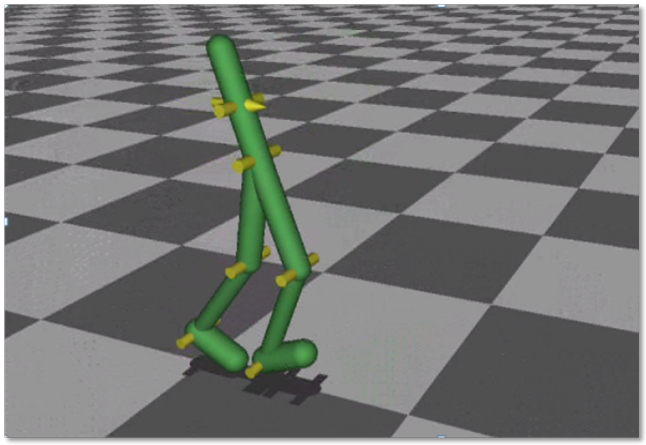

Figure 3.3: 2D Walker

3.3.1 Expert Demonstration

The expert demonstration of this model would be a realistic walking motion. I created a walking motion based on [11], which used a balance heuristic and PD control to create a set of joint torques that generates a reasonable walking motion in real time.

3.3.2 Trajectory Initialization

The forward trajectory optimizer generates the desired motion by optimizing an arbitrary trajectory into the one that maximizes the reward function. Ideally, the forward optimizer should be able to create the same motion no matter how we take the initial trajectory. Unfortunately, a truly arbitrary initial trajectory tends to fall into a local optimum in the course of optimization. At high level, this means the optimization produces the final motion that is far from the user intention. Indeed, in this 2D walker example, random initialization of the action could not be used to produce a walking motion because the optimizer had to start with a trajectory that falls on the ground. Then the trajectory optimizer would have to first make the walker stand up on the ground and then generate walking motion. This combination of standing up and walking forward could not be accomplished with the set of features that were designed to create a forward walking motion, and the trajectory optimization ends up finding a local optimum that is far from the natural walking motion. Therefore, the initial trajectory had to be designed to be the one that does not fall to the ground. A reasonable initial trajectory would then be the one in which the walker stands still at a fixed position.

Nevertheless, creating such static trajectory is a nontrivial problem. Due to the geometry of the walker, applying zero torque to the character model makes the model fall to the ground. This problem was overcome by running another pass of forward trajectory optimization with the reward function

$$r(\mathbf{x}_t, \mathbf{u}_t) = -\frac{1}{2}||\mathbf{q} - \mathbf{q}^*||^{\mathbf{2}},$$

where $\mathbf{q}^*$ is the desired initial static pose of the 2D walker. That is, the reward function penalizes the quadratic deviation from the desired pose. Since creating the vector $\mathbf{q}^*$ is simple and the reward function is very specific, running trajectory

optimization with this reward function converged fast. The initial trajectory of this optimization was taken to be the expert demonstration in which the model walks forward.

3.3.3 Feature Construction

Both Inverse Reinforcement Learning and forward trajectory optimization need features whose weighted sum will produce the reward function. The features must be differentiable, so indicator functions that return either 0 or 1 cannot be used as they are not continuous. Moreover, these features should not be too specific to the motion so that they can be portable accross different environments, but also not too general in order to distinguish the desired motion from the crowd of other motions that achieve similar goals. One example of a feature that is too specific is the one used in the trajectory initialization that measures quadratic deviation from the example trajectory, since the example trajectory would not be appropriate in other environments. At the other end, using only one feature that measures the speed of the character is too general to create the desired walking motion, because there are many other ways to create the same speed, such as hopping or crawling. Therefore, a combination of such general features should be used in order to find a good compromise between the two ends. The list of features used to generate the walking gait of the 2D walker is introduced.

3.3.3.1 Torso height

This feature expresses that a walking motion keeps the torso high from the ground, and the height stays relatively constant.

$$f_{torsoheight}(\mathbf{x}_t, \mathbf{u}_t) = -\frac{1}{2}||h_t - h^*||^2,$$

where h_t is the torso height of the walker at time t and $h^*$is the desired height of the 2D walker. A nice choice of h^* would be the height of the torso at the standing pose or the average height of the example trajectory. This report took $h^*$to be the avergae height of the example trajectory. However, even if the user is not sure on what value of h^* to use, one can rely on IRL to have it find the suitable h^*.

$$\begin{aligned} f_{torsoheight}(\mathbf{x}_t, \mathbf{u}_t) &= -\frac{1}{2}||h_t - h^*||^2 \\ &= -\frac{1}{2}(h_t^2 - 2h_t h^* + h^{*2}). \end{aligned} \tag{3.1}$$

Note that h^* is a constant. Therefore, instead of using $f_{torsoheight}(\mathbf{x}_t, \mathbf{u}_t)$, we can construct two features f_1 and f_2.

$$f_1(\mathbf{x}_t, \mathbf{u}_t) = {h_t}^2$$

$$f_2(\mathbf{x}_t, \mathbf{u}_t) = h_t.$$

Once IRL successfully assigns the weights w_1 and w_2 on $f_1$and f_2, we can find the equivalent h^*.

$$w_1 f_1 + w_2 f_2 = w_1 {h_t}^2 + w_2 h_t \tag{3.2}$$

$$= w_1({h_t}^2 + \frac{w_2}{w_1}h_t) \tag{3.3}$$

$$= w_1(h_t - \frac{-w_2}{2w_1})^2 - w_1(\frac{-w_2}{2w_1})^2. \tag{3.4}$$

Since the last term $-w_1(\frac{-w_2}{2w_1})^2$ is a constant, the equivalent $h^*$is $\frac{-w_2}{2w_1}$. Thus, I did not have to guess the exact value of h^* by using the two features instead of one.

3.3.3.2 Velocity along the desired direction

This feature expresses the velocity of the root position of the model.

$$f_{velocity}(\mathbf{x}_t, \mathbf{u}_t) = -\frac{1}{2}||v_t - v^*||^2,$$

where v_t is the velocity of the walker at time t and v^* is the desired velocity of the walker. Here the same technique as in sec. 3.3.3.1 can be used to have IRL find out the value of v^*. That is, we can construct two features that measure v_t^2 and v_t instead of a quadratic feature centered at an arbitrary velocity.

3.3.3.3 Foot movement

A walking motion can be characterized by periodic swing movement of the left and the right foot. One way to capture this is to measure the velocity of the swing foot with respect to the support foot. To do this, we need to identify which foot is planted on the ground. Moreover, the identification should be a differentiable function over states and actions. This consraint was met by introducing a Gaussian weighting on the height of each foot. That is, consider the function H defined as

$$H(\mathbf{x}_t) = e^{-\frac{1}{2}wh_t},$$

where h_t is the height of the foot at time t. Let H_{lf} and H_{rf} be the two H functions applied on the left foot and the right foot, respectively. The following $f_{swingfoot}$ captures the velocity of the swing foot with respect to the support foot.

$$f_{swingfoot}(\mathbf{x}_t, \mathbf{u}_t) = H_{lf}(\mathbf{x}_t, \mathbf{u}_t) \cdot (v_{rf}) + H_{rf}(\mathbf{x}_t, \mathbf{u}_t) \cdot (v_{lf}),$$

where v_{lf} and v_{rf} are the velocities of the left foot and the right foot, respectively.

When the left foot is the support foot, H_{rf} becomes very small, so $f_{swingfoot}$ captures the velocity of the right foot which is the swing foot. On the other hand, it represents the velocity of the left foot when the right foot is the support foot. Moreover, this feature $f_{swingfoot}$ is a differentiable function because H_{lf}, H_{rf}, v_{lf} and v_{rf} are all differentiable.

While $f_{swingfoot}$ well expresses the walking motion by measuring the swing foot velocity, it is also helpful to measure the velocity of the support foot. The support foot velocity feature $f_{supportfoot}$ is just

$$f_{supportfoot}(\mathbf{x}_t, \mathbf{u}_t) = H_{lf}(\mathbf{x}_t, \mathbf{u}_t) \cdot (v_{lf}) + H_{rf}(\mathbf{x}_t, \mathbf{u}_t) \cdot (v_{rf}).$$

3.3.3.4 Penalty on torques

Penalizing the torques on the joints will produce more efficient motion in terms of muscle usage.

$$f_{torque,i}(\mathbf{x}_t, \mathbf{u}_t) = -\frac{1}{2}||u_{t,i}||^2,$$

where $u_{t,i}$ is the torque on the ith joint at time t. Each $f_{torque,i}$ can be created for all joints, but that will increase the number of features and thus the running time. This problem can be avoided by applying a uniform weight on all joints. That is,

$$f_{torque}(\mathbf{x}_t, \mathbf{u}_t) = -\frac{1}{2}\sum_i ||u_{t,i}||^2.$$

This approach, however, will not be able to finetune the subtle differences among the joints. In case of the 2D walker, we can exploit the symmetry by creating the torque features for the hip, knees, and the ankles, and apply the same weight for

the same body parts, ignoring whether the joint is on the left side or the right.

3.3.4 Inverse Reinforcement Learning

All the features introduced in sec. 3.3.3 were used. That is, the reward function f is defined to be

$$f(\mathbf{x}_t, \mathbf{u}_t) = w_1 f_{torsoheight} + w_2 f_{velocity} + w_3 f_{swingfoot} + \\ w_4 f_{supportfoot} + w_5 f_{torque,hip} + w_6 f_{torque,knee} + w_7 f_{torque,ankle},$$

and the job of IRL is to assign the feature weights w_i's so that f is optimal around the example trajectory. Indeed, this could be verified by querying the reward of some perturbation around the example trajectory. Fig. 3.4 shows that IRL has successfully optimized the reward function. This graph was generated by applying a random perturbation to the states and plotting its reward versus the 2-norm of the perturbation. The left graph shows the perturbation around the example trajectory and the right graph shows it around a random state sequence. As can be seen from the figure, IRL successfully optimized the reward function so that the expert trajectory has the highest reward in its local neighborhood.

Note that the reward of the expert trajectory is much higher than that of a random state sequence. This might seem, but the local optimization algorithm used in this report does not guarantee this. In fact, there were some cases in which the initial trajectory of the standing character had a higher reward than the example trajectory when fewer features were used. For exanple, only using $f_{torsoheight}$ and f_{torque} will always favor the standing pose to the walking motion for all positive weights, because the standing pose has the heightest possible torso height and also uses smaller torques than the walking motion. This problem occurs because the

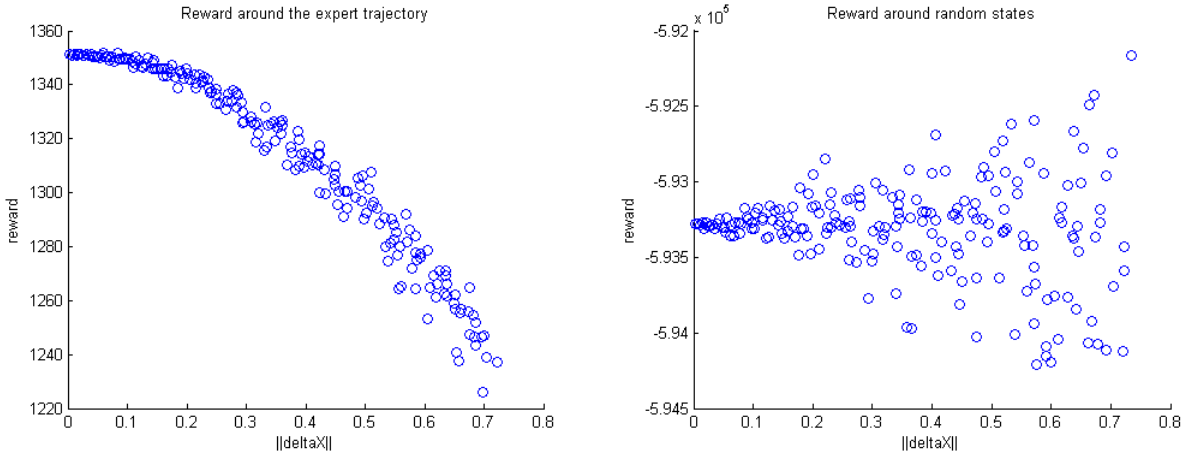

Figure 3.4: Reward of the 2D Walker computed by IRL

set of features used cannot sufficiently characterize the walking motion, and can be overcome by using a larger set of expressive features. Moreover, the existence of a trajectory that has a higher reward than the expert trajectory is not a drawback but a strength of this algorithm. In many cases, the expert does not demonstrate globally optimal motions, so we should not assume that the expert trajectory is a global optimum. In short, IRL in this paper can be seen as learning the essense of the expert demonstration, and still leave the possiblity of an even better motion.

The computed values of the feature weights are in the following table. For both the velocity and the torso height features, I used both the quadratic and linear features to have IRL determine the optimal value of the desired velocity and torso height, as in sec. 3.3.3.1. Using $\frac{-w_2}{2w_1}$, we can see that the aimed torso height was $\frac{-617}{2\cdot(-321)} = 0.9611$. This is a little different from the actual average height of the walker in the example trajectory, which was 1.1724. The aimed velocity should be interpreted differently. $f_{velocity,quadratic}$ and $f_{velocity,linear}$ together contribute to the total reward by a quadratic function that is convex downward, centered at

$\frac{-11.7}{2 \cdot 8.37} = -0.6989$. This means that any velocity greater than -0.6989 is preferred by $f_{velocity,quadratic}$ and $f_{velocity,linear}$. However, walking at very high velocity will cost more torques that penalize the reward function.

$f_{torque,hip}$	$f_{torque,knee}$	$f_{torque,ankle}$	$f_{swingfoot}$	$f_{supportfoot}$
2.61	19.1	80.8	-0.00176	-0.103

$f_{velocity,quadratic}$	$f_{velocity,linear}$	$f_{torsoheight,quadratic}$	$f_{torsoheight,linear}$
8.37	11.7	-321	617

Table 3.1: The weights on the features of the walker model computed by IRL

3.3.5 Forward Trajectory Optimization

The forward trajectory optimization was run with the reward function recovered by IRL algorithm. While both the state and the action regularization work well, the state regularization was used for the results shown in this report. The resulting optimized trajectory produced a reasonable walking motion. The video that compares the walking motion of the example trajectory and the result of the trajectory optimization was uploaded online, and can be viewed at (http://youtu.be/i-l54bqDhrA).

Fig. 3.5 compares the velocity of the center of mass of the example trajectory and the optimized trajectory. Since the example trajectory of [11] was generated by directly specfying the joint torques, the example trajectory is guaranteed to show regular motion. On the other hand, the optimized trajectory shows a little more irregularity, and it tends to slow down more at the local minima of the velocity than the example trajectory. The author thinks that the irregularity and sudden slowdowns were caused by the bouncing of the ground. The ground should not be bouncy in truly accurate simulation of the real world, but the contact between the feet and the ground was simulated in MuJoCo([10]) using a spring model. Even though an extremely stiff contact model will prevent the bouncing, such stiffness was

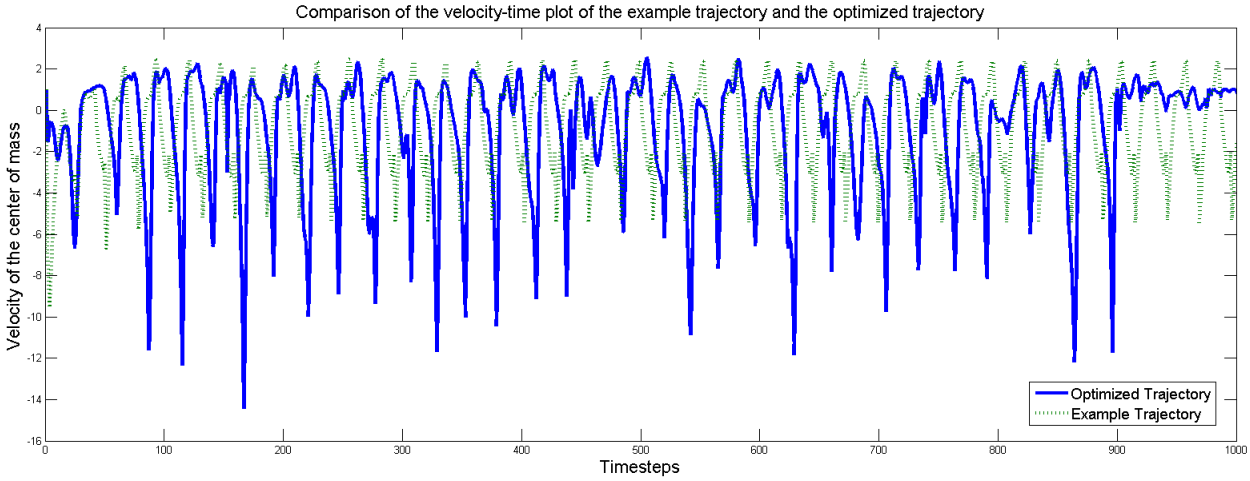

Figure 3.5: Comparison of the velocity-time plot of the example trajectory and the optimized trajectory of the 2D Walker

limited by numerical stability. In addition, note that the fluctuation of the velocity decreases near the end of the timestep. This is because the walker model just falls forward instead of stepping forward at the last moment. As the optimization was performed with finite horizon, this was clearly a more efficient way of generating a forward motion than walking. An analogous of this behavior would be the situation where the runner dives toward the base in baseball.

3.3.6 Portability of the Reward Function

The whole point of animating characters using trajectory optimization with IRL is to adapt to new environment. To show this, a new environment with gravity 1/6 of the original gravity was created. This new environment resembles the surface of the moon with smaller gravity. Fig. 3.6 shows the X-Y positions of the replay of the action sequence of the original example trajectory and the optimized trajectory.

The video of the two trajectories in the reduced gravity can be viewd online at (http://youtu.be/uzUCXZgdHCA). As can be seen in the figure, the original action sequence of the example trajectory becomes completely irrelevant, and makes the model fall to the ground. The work by [11] does not cause the model to fall to the ground, but becomes unstable and slower. On contrary, trajectory optimization with the same reward function learned under the original gravity successfully produced a very stable walking motion. Furthermore, the trajectory optimization maintained the velocity of the walker, which only changed from 1.231 to 1.232 units per second as the gravity changed.

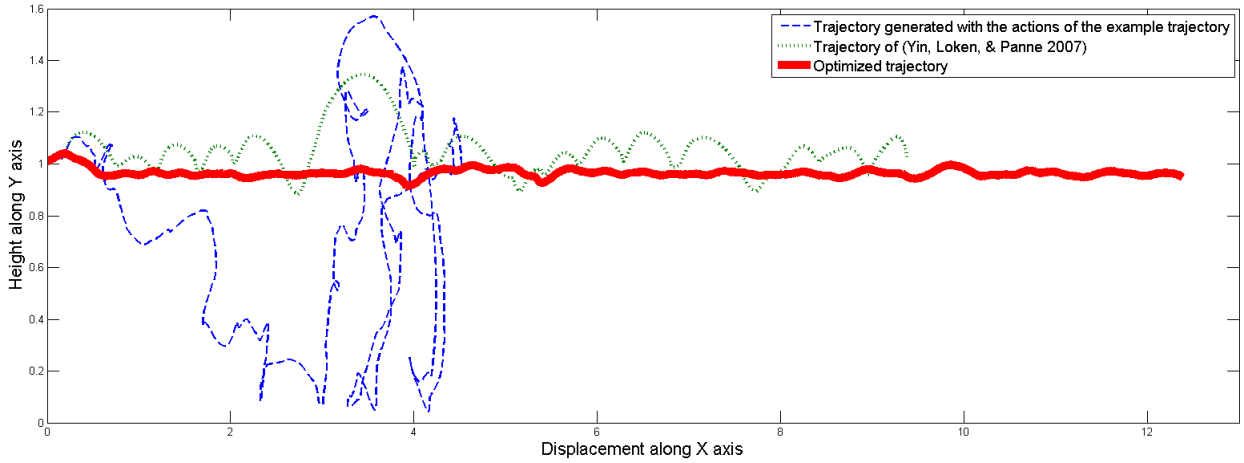

Figure 3.6: Comparison of the X-Y root position trajectory of the example and the optimized trajectory of the 2D Walker

This experiment indeed shows the power of creating character animation using reinforcement learning with IRL. In a virtual environment like above that has artificially lower gravity, it is hard to generate a physically plausible motion that still resembles the original demonstration. It would have been also impossible for the expert

to perform demonstration in reduced gravity, as such environment is very hard to construct.

4 Discussion and Future Work

4.1 Future Work

The 2D walker model used in this report was a stepping stone to move onto the real human model. Representing the actual human can produce more meaningful results in character animation. The complexity of the human model makes it very costly to handcraft animations. In addition, using motion capture each time a different motion is needed is also very demanding. Therefore, creating animation of real human model with reinforcement learning with IRL has a big merit.

Moreover, the power of IRL on the actual human model lies in learning what the humans actually optimize when they perform a certain task. This can be of interest in the robotics field that tries to mimic human behaviors using humanoid robots. Once the reward function is recovered using IRL, it can be directly applied to a humanoid robot to perform the same task. Moreover, since the simulation is physics-based, the reinforcement learning can be tested in the virtual environment. Therefore, controlling a humanoid robot can be achieved by first learning the reward function from real human demonstrations, and then porting the same reward function to reinforcement learning on the humanoid robot in virtual environment.

4.2 Limitations

The local approximation used in both IRL and the forward trajectory optimization made the optimization computationally feasible and compatible with the assumption that even the experts do not know globally optimal behavior. However, it also brought some limitations, especially converging to a local optimum that is not very meaningful. This can be seen in Fig. 3.5, where the optimized trajectory is less periodic than the expert demonstration. Since the environment around the model is static, we expect that one optimal cycle of the walking motion should replace all the other cycles that are suboptimal. However, the local approximation produced different motions at each cycle of strides, even though the forward trajectory optimization had completely converged.

I suggest a few methods to address this issue. One easy solution is to perturb the actions when the optimization converges to an unwanted local optimum. However, in the 2D walker case, even a very small perturbation could make the model fall to the ground, which makes it considerably harder for the optimizer to have the model stand back up and walk. Another approach is sufficiently smoothing the reward landscape around the local optimum. If the peak of the local optimum is not too big, which suggests bad local optimum, just smoothing the reward landscape may get rid of the peak. Smoothing can be done by taking random samples around some perturbation of the trajectory and averaging the corresponding rewards. However, deciding the magnitude of the perturbation remains ambiguous, because it will depend on the size of the peak of the local optimum. Lastly, we can enforce regularity of the motion by taking the approach of [2]. This approach saves the state and action pairs of the optimized trajectory in a data structure and take the action corresponding to the state closest to the queried state. In case of a walking motion, this approach may work well, because if the model is in same configuration, it should

repeat the same action.

4.3 Conclusion

In this report, a character animation was generate on a simple walker model using reinforcement learning with the reward function recovered by IRL. The computational complexity that comes from the high dimensional joint space was overcome by making quadratic local approximation to the trajectory. The strength of this approach is that the reward function learned by the IRL is portable accross different models and environments. This observation was indeed verified by successfully generating the walking motion of the 2D walker under reduced gravity using the reward function constructed under the normal graivity.

Acknowledgments

Above all, I am deeply grateful to Vladlen Koltun and Sergey Levine for their insight, advice, and patience with my research. This research would not have been possible without their guidance over the past year. Their insights and support kept me going through many obstacles, and immersed me in the delight of research. I also would like to thank Oussama Khatib for providing invaluable advice and reviewing this report. His lectures always helped me understand the material in character control.

Bibliography

[1] Abbeel, P., and Ng, A. Y. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning* (2004), ACM, p. 1.

[2] Atkeson, C. G., and Stephens, B. J. Random sampling of states in dynamic programming. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 38*, 4 (2008), 924–929.

[3] Birgin, E. G., and Martinez, J. M. Practical augmented lagrangian methods. *Encyclopedia of Optimization,* (2007), 3013–3023.

[4] Dvijotham, K., and Todorov, E. Inverse optimal control with linearly-solvable mdps. In *Proceedings of the Interntional Conference on Machine Learning. Citeseer* (2010).

[5] Lee, S. J., and Popović, Z. Learning behavior styles with inverse reinforcement learning. *ACM Transactions on Graphics (TOG) 29*, 4 (2010), 122.

[6] Levine, S., and Koltun, V. Continuous inverse optimal control with locally optimal examples. *arXiv preprint arXiv:1206.4617* (2012).

[7] Levine, S., and Popovic, J. Physically plausible simulation for character animation. In *Eurographics/ACM SIGGRAPH Symposium on Computer Animation* (2012), The Eurographics Association, pp. 221–230.

[8] Ratliff, N. D., Bagnell, J. A., and Zinkevich, M. A. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning* (2006), ACM, pp. 729–736.

[9] Tassa, Y., Erez, T., and Todorov, E. Synthesis and stabilization of complex behaviors through online trajectory optimization. IROS.

[10] Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. *Under Review*.

[11] Yin, K., Loken, K., and van de Panne, M. Simbicon: simple biped locomotion control. In *ACM Transactions on Graphics (TOG)* (2007), vol. 26, ACM, p. 105.

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