Imperial College London
Department of Computing

Model-based Apprenticeship Learning for Robotics in High-dimensional Spaces

Yuanruo Liang

Submitted in partial fulfilment of the requirements for the MSc degree in
Computing Science of Imperial College London
September 2014

Abstract

This project shows that model-based, probabilistic inverse reinforcement learning (IRL) is achievable in high-dimension state-action spaces with only a single expert demonstration. By implementing the IRL *max-margin* algorithm with a probabilistic model-based reinforcement learning algorithm named PILCO, we can combine the algorithms to create the *IRL/PILCO* algorithm, which is capable of reproducing expert trajectories by choosing suitable features.

Using IRL/PILCO, we carry out a simulation with a cart-pole system where the goal is to invert a stiff pendulum, and demonstrate that a policy replicating the task can be reproduced without explicitly defining a cost function from features.

We also carry out an experiment with the Baxter robot, using both of its arms (28 DOF) to reproduce a sweeping action by holding a brush and dustpan using velocity control. This task involved applying PILCO to a state-action space with 42 dimensions. The task was replicated nearly exactly with very high data efficiency and a minimal amount of interaction (7 trials, 56s).

Acknowledgements

I would like to express gratitude my gratitude to:

- Dr. Marc Deisenroth, for his many hours of guidance
- Dr. Yiannis Demiris, for kindly allowing use of the Personal Robotics Lab
- Miguel Sarabia del Castillo, for his time and assistance with ROS and Baxter
- My family and friends

To those striving towards the Singularity.

Contents

List of Figures

Chapter 1

Introduction

Figure 1.1: A photo of the Baxter robot. This robot was taught to learn to sweep with a dustpan and brush from a single demonstration, as an example of apprenticeship learning.

1.1 Motivation and Objectives

Programming robots to perform specific tasks can be a tedious and difficult process. Furthermore, it is likely that the result is applicable only to specific situations and circumstances. Even where classical reinforcement learning methods are used, many of these methods require hundreds or thousands of trials to succeed, and not all tasks can be easily specified by a reward function *a priori*. Where versatility and quick skill acquisition is required, new methods must be devised.

We attempt to circumvent some of these problems by doing probabilistic, model-based apprenticeship learning. Apprenticeship learning, or apprenticeship via inverse reinforcement learning (IRL), is a concept in the field of imitation learning. It deals with Markov Decision Processes (MDPs), where instead of a given, well-defined reward function, an expert demonstration doing the task we wish to perform is provided instead. The goal of apprenticeship learning is to recover the behaviour of the expert by learning the reward function.

From examining animal and human behaviour, it is clear that the parameters of the reward function should often be considered "as an unknown to be ascertained through empirical investigation" (Ng and Russell, 2000). For example, when driving on the road one would typically have to weigh speed against other factors such as traffic conditions, pedestrians, weather conditions, and so on. It would be difficult to determine the various trade-offs *a priori*; however, as human beings we would typically learn from an expert agent such as an experienced driver, and additionally through trial and error.

Another main motivation is that classical model-based learning methods in reinforcement learning (RL) often relies on idealised assumptions and expert knowledge, in order to derive realistic mathematical formulations for each system. This frequently leads to model bias (Atkeson and Santamaria, 1997), and it is known that policies learnt from inaccurate models are generally not useful. Additionally, exact state information is usually not known, but only within a range of uncertainty. It is therefore important that rather than an engineering solution, there is a general and principled framework for efficiently learning and modelling the dynamics of physical systems (Deisenroth, 2010).

In this project, we aim to achieve the goal of apprenticeship learning by combining the *max-margin* method of inverse reinforcement learning (IRL) (Ng and Russell, 2004) with PILCO, a highly-efficient, probabilistic policy search algorithm (Deisenroth and Rasmussen, 2011), and show that this will enable apprenticeship learning with high data efficiency.

According to Ng and Russell (2000) and Russell (1998), the IRL problem can be characterised informally as follows:

Given

1. Measurements of an agent's behaviour over time in various circumstances in its environment
2. Measurements of the sensory inputs to the agent
3. Optionally, a model of the environment

Determine

1. The reward function, and/or
2. The agent's policy

We present some of the accomplishments of IRL in recent years.

1.1.1 Accomplishments of IRL

Helicopter Acrobatics

In 2010, Pieter Abbeel, Adam Coates and Andrew Ng successfully presented an apprenticeship learning demonstration of flying helicopter acrobatics (Figure 1.2), despite the challenging nature of autonomous helicopter flight in general.(Abbeel et al., 2010)

Ball-in-a-cup

The ball-in-a-cup is a children's motor game where a ball is hanging from the bottom of the cup via a string. The goal is to toss the ball into the cup by moving only the cup (Figure 1.3). The actions correspond to the Cartesian accelerations of the cup, and both the state and action space are continuous.

Figure 1.2: Screenshot of helicopter acrobatics. Taken from Abbeel et al. (2010)

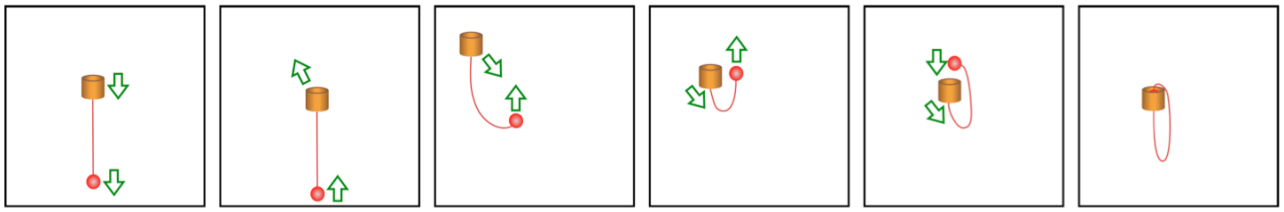

Figure 1.3: The ball-in-cup problem. The green actions indicate direction of action. Taken from Boularias et al. (2011)

Autonomous Navigation and Legged Locomotion

Figure 1.4: From left to right: 1. Autonomous ground vehicle Crusher, built by the National Robotics Engineering Center(NREC). 2. Quadruped robot LittleDog, built by Boston Dynamics. Taken from Ratliff et al. (2009b)

Some tasks, such as autonomous navigation, locomotion and grasping tasks have been successfully demonstrated using a set of algorithms known collectively as LEARCH (Figure 1.4), which is able to more efficiently non-linearise cost functions while satisfying common constraints more naturally (Ratliff et al., 2009b), compared to other methods of non-linearisation (Ratliff et al., 2007).

Table-tennis Ball Hitting

Another approach to IRL is to find policies such that predicted trajectories directly match observed expert trajectories. Using probability distributions of trajectories and matching them with the Kullbeck-Leibler divergence, Englert et al. (2013a) were able to use a BioRob robot to consistently hit a table tennis ball upwards (Figure 1.5).

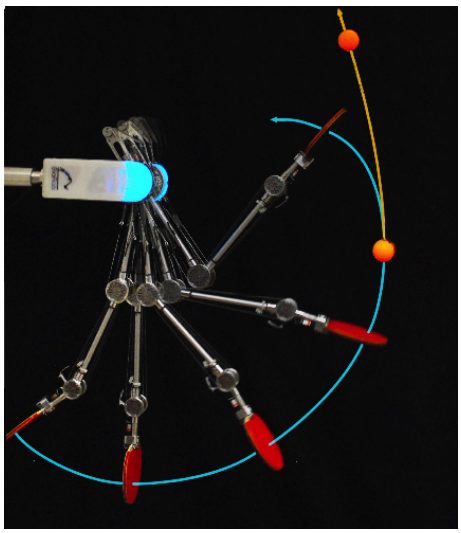

Figure 1.5: The BioRob hitting a table-tennis ball. The blue spline shows the racquet trajectory; the yellow spline shows the ball trajectory. Taken from Englert et al. (2013a)

Turn prediction

It was a neural-network based behavioural cloning approach that first indicated the possibility of predicting driver steering based on road conditions (Pomerleau, 1989), by using camera images coupled with a laser range-finder. Later on, by combining behavioural cloning with long-term inverse reinforcement learning methods, Ratliff et al. (2009a) were able to do turn prediction for taxi drivers.

1.2 Contributions

In this project, we incorporated the *max-margin* inverse reinforcement learning algorithm by Ng and Russell (2004) within the PILCO framework(Deisenroth and Rasmussen, 2011), to produce the *IRL/PILCO* algorithm. We implemented this algorithm using the Baxter robot to demonstrate its viability. By demonstrating the task of sweeping with a brush and dustpan only once, the algorithm was able to almost exactly reproduce the expert demonstrated trajectory with only 7 iterations and 56s of experience time.

1.3 Thesis Layout

The rest of this thesis is organised as follows:

In Chapter 2, we explain the three key concepts that were necessary in this project. Firstly, both the basic theory of Markov Decision Processes (MDPs) and both classical reinforcement learning (RL) are discussed in Section 2.1. Next, we introduce policy search methods as an alternative to classical RL, and explain why they can perform better. Subsequently, we introduce the problem of inverse reinforcement learning (IRL) including the *max-margin* algorithm. Afterwards, we have a brief introduction to Gaussian Processes (GPs) and explain how they allow for probabilistic inference over function spaces, which is useful for model building. Finally, we will explain the policy search method called PILCO and explain how this algorithm is able to use GPs to do policy search and learn control.

In Chapter 3, we specify how the IRL *max-margin* algorithm is incorporated into the PILCO framework to produce a combined IRL/PILCO algorithm that is able to use Bayesian reasoning to learn policies to reproduce expert demonstrated behaviour.

In Chapter 4, we show how the algorithm performs on a simulated cart-pole system specified by a set of ordinary differential equations (ODEs).

In Chapter 5, we examine in detail how the IRL/PILCO algorithm is implemented on the Baxter robot, in order to reproduce a demonstrated task of sweeping up an object with both arms.

In Chapter 6, we discuss how the algorithm performs, comparing between the cart-pole simulation and the Baxter robot, and also comparing between two methods of computing long-term cost, the *features expectation* method and the *feature matching* method.

Finally, Chapter 7 concludes by summarising key results, re-stating the pros and cons of this approach, and any future work that could be done to improve on performance.

Chapter 2

Background Theory

2.1 Markov Decision Processes and Reinforcement Learning

In classical reinforcement learning (RL), learning is accomplished within the framework of Markov Decision Processes (MDPs), which is a model for decision-making where an agent applies controls stipulated by a policy (Bellman, 1957). Outcomes can be partially random and partly under the control of a control agent. The agent is able to interact with its environment, and the goal is to obtain an optimal policy which maps states to actions for maximum reward, which is usually specified in a reward function.

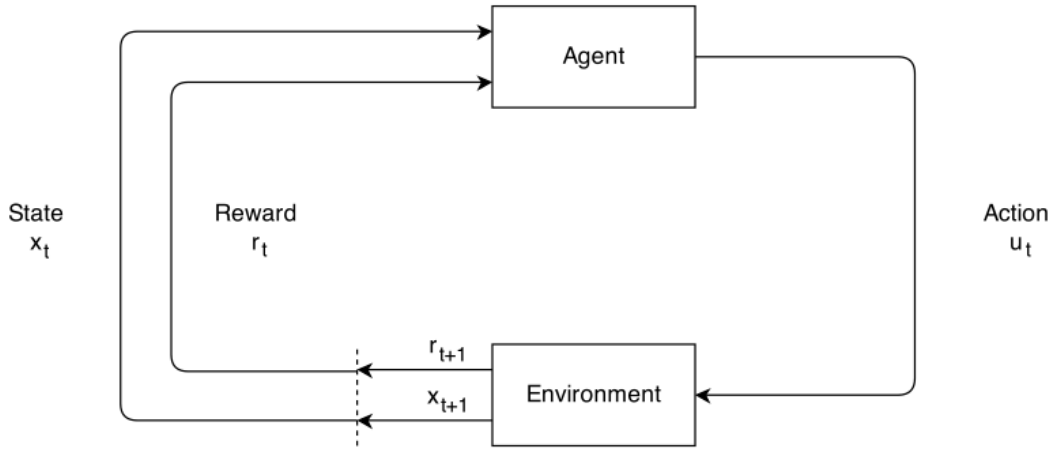

Figure 2.1: The agent-environment interface within an MDP.

The agent-environment interaction of an MDP is shown in Figure 2.1, and is as follows:

1. A reinforcement learning agent interacts with its environment in discrete time steps.
2. At each time step t, the agent receives an observation x_t and the reward r_t.
3. The agent then chooses an action u_t from the set of actions available, which is subsequently undertaken in the environment.
4. The environment (including the agent) then transitions to a new state x_{t+1} and the reward r_{t+1} associated with the transition (x_t, u_t, x_{t+1}) is determined.

The action to take is prescribed by a policy, usually denoted by π. The value V of a policy π is the discounted cumulative reward r at each time step t, and is defined as

$$V^{\pi} = E[\sum_{t} \gamma^{t} r_t | \pi] \tag{2.1}$$

where $\gamma \in [0, 1)$ is the discounted reward.

The goal of a reinforcement learning agent is to maximise the long-term reward V^{π}. The agent can choose any action based on past experience, or even by trial-and-error.

2.1.1 Markov Decision Process

Markov Decision Processes provide a framework for modelling decision-making, even where outcomes are probabilistic. We define mathematically the notation used as follows:

Mathematical Definition

A finite MDP is a tuple $M = (X, U, P, \gamma, R)$ where

1. X is a finite set of N states.
2. $U = u_1, ..., u_k$ is a set of k actions.
3. $P(x, u, x')$ are the state transition probabilities upon taking action u in state x leading to state x',
   i.e. $P(x, u, x') = P(x'|x, u)$.
4. $\gamma \in [0, 1)$ is the discount factor.
5. $R : X \mapsto \mathbb{R}$ is the reward function with absolute value bound R_{max}.

A policy is defined as any map $\pi : X \mapsto U$. For a given policy π, the value function at a state x_1 is given by

$$V^{\pi}(x_1) = E[R(x_1) + \gamma R(x_2) + \gamma^2 R(x_3) + ...|\pi], \tag{2.2}$$

which is simply the cumulative expected reward from state x_1 onwards given a policy π, without having chosen any action.

The Q-function is defined as

$$Q^{\pi}(x, u) = R(x, u, x') + \gamma E[V^{\pi}(x')|P(x, u, x')] \tag{2.3}$$

which is essentially the value at a state x after choosing action u, given policy π.

Basic Properties

Let an MDP $M = \{X, U, P, \gamma, R\}$ and a policy $\pi : X \mapsto U$ be given. Then, two of the classical results concerning finite-state MDPs are Bertsekas and Tsitsiklis (1995) and Sutton and Barto (1998) –

1. Bellman Equations

$$V^{\pi}(x) = \max_{u} Q^{\pi}(x,u) \tag{2.4}$$

$$Q^{\pi}(x,u) = R(x,u,x') + \gamma \sum_{x'} P(x,u,x')[V^{\pi}(x')] \tag{2.5}$$

which are essentially Equations (2.2) and (2.3) for finite-state MDPs.

2. Bellman Optimality

For an MDP, a policy $\pi : X \mapsto U$ is optimal if and only if for all $x \in X$,

$$\pi(x) \in \arg\max_{u \in U} Q^{\pi}(x,u) \tag{2.6}$$

Then, for an optimal policy denoted π^* (where * indicates optimality),

$$V^*(x) = \max_{u \in U} Q^*(x,u) \tag{2.7}$$

$$Q^*(x,u) = R(x) + \gamma \sum_{x'} P(x,u,x')V^*(x') \tag{2.8}$$

Algorithms for Solving Discrete MDPs

Equations (2.7)–(2.8) form the basis for algorithms using the MDP framework. These algorithms include

- Q-learning (Watkins and Dayan, 1992)
- Value Iteration (Sutton and Barto, 1998)
- Temporal Difference Learning (Sutton, 1988)
- SARSA (Rummery and Niranjan, 1994)
- Least-Squares Policy Iteration (Lagoudakis and Parr, 2003)
- Dynamic Programming Methods (Bellman and Kalaba, 1965)

Although important in their own right, these approaches are not directly used in this project because many are useful only in discrete state-action spaces. However, because they constitute the background for MDPs, the first three algorithms are included and explained in **Appendix A**.

Discrete algorithms are generally not useful in cases where the state space is large or continuous. such as in robotics and other physical systems. Modelling a continuous system using discrete spaces will make a problem computationally intractable. Furthermore, model-free methods such as value-iteration would require substantial experimental time and can significantly degrade the physical system at hand. Additionally, the value and Q-functions can be complicated and difficult to solve, even though there may be simple and compactly representable policies that can perform very well(Ng and Jordan, 2000). This has led to interest to *direct policy search* methods.

From Deisenroth et al. (2013), we know that policy search methods can work quite well in a robotics context, compared to classical methods.

2.2 Policy Search

Classical RL suffers from being data-inefficient, where too many trials are needed to learn a task. Direct policy search is faster than RL, and have successfully been applied to many robotics problems such as Ng and Jordan (2000), Peters et al. (2010) and Deisenroth and Rasmussen (2011). Policy search can be classified as *model-free* or *model-based*.

In model-free methods, policies are applied directly to the robot to generate sampling trajectories, which is then directly used to update the policy.

In contrast, model-based methods use sampled trajectories to first learn a model of the physical system, which is then used to generate predictive trajectories for policy improvement.

In this project, Gaussian Processes are used to model the dynamics of the system in order to perform direct policy search.

2.2.1 Model-based Policy Search

Between the choice of model-based and model-free policy search, it has been shown that model-based methods are generally faster and more effective (Bagnell and Schneider, 2001).

However, model-based function approximation methods are not without its drawbacks.

Firstly, it can lead to model bias - imagine attempting linear regression through a data plot that is clearly exponential. Additionally, long-term predictions of any system are essentially arbitrary, and yet function approximators do not give any uncertainty estimates.

Additionally, policy search assumes that the reward function is available. However, in many tasks it is not easy to write down a reward function; instead it is easier to demonstrate desired behaviour.

These drawbacks leads us to two solutions:

- Use Gaussian Processes as probabilistic function approximators to build a model for policy search, which gives uncertainty estimates, from PILCO (Deisenroth and Rasmussen, 2011).
- Use the inverse reinforcement learning *max-margin* algorithm by Ng and Russell (2004) to replace the reward function with expert demonstration instead.

2.3 Inverse Reinforcement Learning

In contrast to reinforcement learning, apprenticeship learning or apprenticeship via inverse reinforcement learning (IRL) is defined by the *missing* reward function. Instead,observed behaviour as demonstrated by an expert, usually assumed to be acting optimally, is given. (Ng and Russell, 2000). Rather than learn the values associated with each state, the goal of IRL is to first learn the parameters of the reward function, before extracting the optimal policy π^* which is able to reproduce the expert trajectory.

Inverse RL suffers from similar problems with classical RL, such as the need for function approximation in continuous reward space. We will first touch on the discrete state-action and reward case, before delving into the continuous case.

2.3.1 IRL being an ill-defined problem

It has been shown that for R to make the policy $\pi(x) \equiv u_1$ optimal in the discrete case, the condition

$$(P_{u_1} - P_u)(I - \gamma P_{u_1})^{-1} R \succeq 0 \tag{2.9}$$

must hold for discrete state x, where $P_u = P(x, x'|u)$ is the state transition probability matrix.

The equivalent condition

$$E_{x' \sim P_{xu_1}}[V^\pi(x')] \geq E_{x' \sim P_{xu}}[V^\pi(x')] \tag{2.10}$$

must hold for continuous states x (Ng and Russell, 2000).

These conditions pose a few problems, which are

1. Trivial solutions

   From Equation (2.9), it is obvious that $R = 0$ (or any other constant vector) is always a solution, because if the reward is the same regardless of action, then any policy, including $\pi(x) \equiv u_1$ will be optimal. Furthermore, there are likely many choices of R that will meet the criteria.

   Without excessive detail, Ng and Russell (2000) showed that the full approach to resolve this is to use a linear programming formulation of the problem as such:

   $$\text{maximize} \sum_{i=1}^{N} \min_{u \in \{u_2, \ldots, u_k\}} \{(P_{u_1}(i) - P_u(i))(I - \gamma P_{u_1})^{-1} R\} - \lambda ||R||_1 \tag{2.11}$$

   such that $(P_{u_1} - P_u)(I - \gamma P_{u+1})^{-1} R \succeq 0$ for all $u \in U$
   u_1, and $|R_i| \leq R_{max}, i = 1, ..., N$

   where $P_u(i)$ denotes the i-th row of P_u and λ is an adjustable penalty coefficient such that R is bounded away from 0 for some constant $\lambda < \lambda_0$.

2. Infinite constraint size

   For large state spaces, there may be infinitely many constraints in the form of Equation (2.9), which makes it impractical to check them all.; however this can be avoided algorithmically by sampling a large but finite subset of states $u_0 \in U$.

2.3.2 Rewards in Continuous State Spaces

In real-world applications of MDPs, states and actions are not discrete, but are continuously-valued. Thus there appears to be an infinite set of states that one may be in, and an infinite set of actions to choose from.

For instance, the state of a simple straight, stiff robotic arm with a hinge may be described by the angle variables θ and $\dot{\theta}$. To ameliorate the problem of intractibility, Ng and Russell (2000) suggest defining the reward function linearly in the manner

$$R(x) = w_1 \phi_1(x) + w_2 \phi_2(x) + ... + w_d \phi_d(x) \tag{2.12}$$

where the w terms are the parameters to "fit" and ϕ_is are fixed, known basis functions mapping $X \mapsto R$.

The value function following Equation (2.12) is then given as

$$V^{\pi} = \sum_{i=0}^{d} w_i V_i^{\pi} \tag{2.13}$$

where V_i^{π} is the expected cumulative reward contributed by $w_i \phi_i(s)$ computed under policy π.

2.3.3 Policy Search In Continuous State Space

In continuous-valued state and action spaces, function approximation is required in order to not run into problems with excessive experimentation and model bias. Methods such as Monte-Carlo value sampling is excessive and could take thousands of robot interactions in order to build up a useful value function of the sort given by Equation 2.2.

Other approaches, such as that proposed by Ziebart et al. (2008) use the principle of maximum entropy to probabilistically define a globally normalised distribution over decision sequences.

Another approach by Boularias et al. (2011) uses a model-free IRL algorithm, where the relative entropy between the empirical distribution of state-action trajectories under a baseline policy and their distribution under a learned policy is minimised by stochastic gradient descent.

In our approach, we will use Gaussian processes via the PILCO framework (Deisenroth and Rasmussen, 2011) in order to effectively solve this problem. Gaussian processes are used as a probabilistic dynamics model in continuous state/action space and discrete time intervals, in order to predict future evolution of the state space. PILCO is able to solve this with minimum experience using indirect policy search.

This will be further elaborated upon in Section 2.5 on PILCO.

2.3.4 Features Expectation

The feature expectations $\mu(\pi)$ is defined as

$$\mu(\pi) = E[\sum_{t=0}^{\infty} \gamma^t \phi(x_t) | \pi] \tag{2.14}$$

and this is essentially the discounted sum of features over trajectory space for a given policy. Then, for a linear reward function, the expected value of a state can be written as

$$E[V^{\pi}(x_0)] = w \cdot \mu(\pi) \tag{2.15}$$

and as the reward function is linear, such as that in Equation 2.12, the features expectation for a given policy π can completely specify the expected sum of discounted rewards for acting on that policy.

2.3.5 The Max-Margin Algorithm

The IRL problem is where we are trying to find the weights w to a reward function $R = (w^{(i)})^\top \phi$ in order to extract a policy π^* similar or close to demonstrations drawn from an expert policy pi_E.

An algorithm published by Ng and Russell (2004), known as the *max-margin* algorithm is shown here in Algorithm 1:

Algorithm 1 Max-Margin IRL

1: **Init:** Randomly pick a policy $\pi^{(0)}$ and compute $\mu^{(0)} = \mu(\pi^{(0)})$, and set $i \leftarrow 1$
2: **while** True **do**
3: **IRL:** Compute $t^{(i)} = \max_w \min_j w^\top(\mu_E - \mu^{(j)})$, where $\|w\|_2 \leq 1$ and $0 \leq j \leq (i-1)$. Let $w^{(i)}$ be the value of w that attains this maximum on the current iteration.
4: **if** $t^{(i)} \leq \epsilon$ **then**
5: **return** $\pi^{(i)}$
6: **RL:** Compute the optimal policy $\pi^{(i)}$ for the MDP with the reward function $R = (w^{(i)})^\top \phi$ using any suitable RL algorithm.
7: Compute $\mu^{(i)} \leftarrow \mu(\pi^{(i)})$
8: Set $i \leftarrow i + 1$

The step in Step 3 can be viewed as an inverse reinforcement learning step where the algorithm is trying to find the weights w to a reward function $R = (w^{(i)})^\top \phi$ such that

$$E[V^{\pi_E}(x_0)] \geq E[V^{\pi^{(i)}}(x_0)] + t \tag{2.16}$$

i.e. a reward on which the expert does better by a margin of t, than any of the previous i policies already found, from some initial state $x_0 \sim X_0$. A full explanation can be found in Ng and Russell (2004, Section 3).

As part of model-based apprenticeship learning, Step 3 will be implemented in the IRL/PILCO algorithm later in Chapter 3.

Model-based policy search with PILCO uses Gaussian processes (GPs) as the dynamics model. We elaborate further on GPs in the next section.

2.4 Introduction to Gaussian Processes

A Gaussian Process (GP) is a stochastic process, with random variables associated with a range of time (or space), such that each random variable is normally distributed. In this manner, it can be thought of as an extension of Gaussian distributions over function space (Rasmussen and Williams, 2006).

Given a data set $\{X, y\}$, consisting of input vectors x_i and corresponding observations y_i, we would like to model the underlying function $h(x)$ such that

$$y_i = h(x_i) + \epsilon \tag{2.17}$$

where ϵ is a noise term such that $\epsilon \sim N(0, \sigma_\epsilon^2)$. The goal is to infer a model of the unknown function $h(x)$ that has generated the data.

Now, let $X = [x_1, x_2, \ldots, x_n]$ be the matrix of training input vectors, and $y = [y_1, \ldots, y_n]$ be the corresponding observations. Then, the inference of $h(x)$ is described by the posterior

$$p(h|X,y) = \frac{p(y|h,X)p(h)}{p(y|X)} \tag{2.18}$$

Just as a (multivariate) Gaussian distribution can be fully described by a mean vector and covariance matrix, a GP can be specified by a mean function $m(x)$ and a covariance function $k(x,x')$. In this way, a GP can be thought of as a distribution over functions. Thus we may write $h \sim GP(m,k)$.

For a GP model of the underlying function h, we are interested in predicting the function values $h(x_*) = h_*$ for any arbitrary input x_*. The predictive marginal distribution of h_* for input x_* is Gaussian distributed with a mean and variance given by

$$E[h_*] = k(x_*,X)(K+\sigma_\epsilon^2 I)^{-1}y \tag{2.19}$$

$$var(h_*) = k(x_*,x_*) - k(x_*,X)(K+\sigma_\epsilon^2 I)^{-1}k(X,x_*) \tag{2.20}$$

where $K \in \mathbb{R}^{n\times n}$ is the kernel matrix, defined by the kernel function k such that every element in the covariance matrix is given by $K_{ij} = k(x_i,x_j)$.

A common covariance function k is the squared exponential (SE) kernel with automatic relevance determination.

$$k(x,x') = \alpha^2 \exp(-\frac{1}{2}(x-x')^\top \Lambda^{-1}(x-x')) \tag{2.21}$$

where α^2 is considered the variance of the latent function f and $\Lambda = diag(l_1^2,\ldots,l_n^2)$ is a diagonal matrix which adjusts for the characteristic length scales l_i for each state feature. The parameters of the covariance function $\{\sigma_\epsilon, \alpha, \Lambda\} \in \theta$ are optimised by evidence maximisation in order to train the GP.

The optimal hyperparameters θ are obtained by maximising the *log marginal likelihood*, given by

$$\log(p(y|X,\theta)) = \log \int p(y|h(X),X,\theta)p(h(X)|X,\theta)dh \tag{2.22}$$

$$= -\frac{1}{2}y^\top (K_\theta + \sigma_\epsilon^2 I)^{-1}y - \frac{1}{2}\log|K_\theta + \sigma_\epsilon^2 I| - \frac{n_x}{2}\log(2\pi) \tag{2.23}$$

2.5 Introduction to PILCO

PILCO (Probabilistic Inference for Learning COntrol) is a practical, data-efficient model-based policy search method. It is both a framework and a software package, and it is able to reduce model bias and conduct policy search using state-of-the-art approximate inference in a principled manner, with very high data efficiency. (Deisenroth and Rasmussen, 2011)

In PILCO, dynamic systems are considered in the form

$$x_t = f(x_{t-1}, u_{t-1}) \tag{2.24}$$

with continuous-valued states $x \in \mathbb{R}^D$ and actions $u \in \mathbb{R}^F$, with unknown transition dynamics (probabilities) f.

The objective of PILCO is to find a deterministic policy $\pi : x \mapsto \pi(x) = u$ such that it minimises the expected cost, which is computed as

$$J^{\pi}(\theta) = \sum_{t=0}^{T} E[c(x_t)], \tag{2.25}$$

$$x_0 \sim N(\mu_0, \Sigma_0) \tag{2.26}$$

and therefore J^{π} is the cumulative cost of following π for T steps, where $c(x_t)$ is the cost (negative reward) of being in state x at time t.

2.5.1 Dynamics Model Learning

In order to predict x_{t+1} from a prior state x_t, the probabilistic dynamics model is implemented as a Gaussian process, where the training inputs are tuples of $(x_{t-1}, u_{t-1}) \in \mathbb{R}^{D+F}$ and the target predictions are state differences $\Delta_t = x_t - x_{t-1} + \epsilon$, such that $\Delta_t \in \mathbb{R}^D$ and $\epsilon \sim N(0, \Sigma_\epsilon), \Sigma_\epsilon = diag([\sigma^2_{\epsilon_1}, ..., \sigma^2_{\epsilon_D}])$.

The Gaussian Process generates predictions for every time step along the trajectory horizon

$$p(x_t|x_{t-1}, u_{t-1}) = N(x_t|\mu_t.\Sigma_t), \tag{2.27}$$

$$\mu_t = x_{t-1} + E_f[\Delta_t], \tag{2.28}$$

$$\Sigma_t = var_f[\Delta_t] \tag{2.29}$$

The prior mean function is set at $m \equiv 0$ and the kernel is the squared exponential (SE) kernel with automatic relevance determination as per Equation (2.21). Input training points are defined as $\tilde{x} = [x^\top u^\top]^\top$, such that

$$k(\tilde{x}, \tilde{x}') = \alpha^2 \exp(-\frac{1}{2}(\tilde{x} - \tilde{x}')^\top \Lambda^{-1}(\tilde{x} - \tilde{x}')) \tag{2.30}$$

and the posterior GP hyper-parameters are learned by evidence maximisation, where the log marginal likelihood from Equation (2.22) is maximised:

$$\theta^* \leftarrow \arg\max_{\theta} \log p(\Delta|\tilde{x}, \theta) \tag{2.31}$$

2.5.2 Policy Evaluation

The optimal policy π^* is found by minimising J^{π} in Equation (2.25), which requires long-term predictions of the evolution of the state distributions $p(x_1), ..., p(x_T)$. This is accomplished in PILCO by cascading uncertain test inputs through the Gaussian process dynamics model (Deisenroth and Rasmussen, 2011). The test inputs are assumed to be Gaussian distributed; the test outputs are assumed independent and do not covary with each other.

The mean prediction $\mu_\Delta$for target dimensions $a = 1, \ldots, D$ can be computed following the *law of iterated expectations*

$$\mu_\Delta^a = E_{\tilde{x}_{t-1}}[E_f[f(\tilde{x}_{t-1})|\tilde{x}_{t-1}]] \tag{2.32}$$
$$= E_{\tilde{x}_{t-1}}[m_f(\tilde{x}_{t-1})] \tag{2.33}$$
$$= \int m_f(\tilde{(x)}_{t-1}) N(\tilde{x}_{t-1}|\tilde{\mu}_{t-1}, \tilde{\Sigma}_{t-1}) d\tilde{x}_{t-1} \tag{2.34}$$
$$\tag{2.35}$$

which has a closed form solution. A similar method is employed for the predicted covariance matrix $\Sigma_\Delta \in \mathbb{R}^{D\times D}$ (Deisenroth and Rasmussen, 2011, Section 2.2).

2.5.3 Gradient-based Policy Improvement

Policy search is accomplished by computing the gradient $dJ/d\theta$ analytically and using gradient descent methods. This is firstly done by differentiating J with respect to the policy parameters θ from Equation (2.25), such that

$$\frac{dJ}{d\theta} = \frac{d}{d\theta} \sum_t E_{x_t}[c(x_t)] \tag{2.36}$$
$$= \sum_t \frac{d}{d\theta} E_{x_t}[c(x_t)] \tag{2.37}$$

Using the shorthand $\mathcal{E}_t = E_{x_t}[c(x_t)]$, and without excessive detail, the derivative at every time step is computed with the chain rule

$$\frac{d\mathcal{E}_t}{d\theta} = \frac{d\mathcal{E}_t}{dp(x_t)} \frac{dp(x_t)}{d\theta} \tag{2.38}$$
$$= \frac{\partial \mathcal{E}_t}{\partial \mu_t} \frac{d\mu_t}{d\theta} + \frac{\partial \mathcal{E}_t}{\partial \Sigma_t} \frac{d\Sigma_t}{d\theta} \tag{2.39}$$

where $p(x_t) = N(\mu_t, \Sigma_t)$. The full derivation for computing $d\mathcal{E}/d\theta$ can be found in Deisenroth and Rasmussen (2011, Section 2.2).

These gradients are then used within a standard optimiser (e.g., conjugate gradients or BFGS (Broyden, 1965), (Fletcher and Powell, 1963), (Shanno, 1970)) to improve the policy.

2.5.4 The PILCO Algorithm

The high-level algorithm is as shown in Algorithm 2.

In Line 6, the long-term cost J^π is computed with Equation (2.25). We aim to reduce this by gradient-descent methods, so we calculate the derivative $\frac{dJ}{d\theta}$ in Line 7 using Equations (**??**)–(**??**). The minimised policy parameters θ^* are then used to update and improve the policy controller in line 10.

Algorithm 2 PILCO

init: Apply random policy $\theta \sim N(0, I)$ and record a single state trajectory
repeat
    Learn GP dynamics model using all state trajectories
    Model-based policy search:
    repeat
        Approximate inference for policy evaluation, compute $J^{\pi}(\theta)$
        Gradient-based policy improvement, compute $dJ^{\pi}/d\theta$
        Optimise policy parameters θ(with CG/L-BFGS)
    until convergence, **return** θ^*
    Set $\pi^* \leftarrow \pi(\theta^*)$
    Apply π^* to system and record single trajectory
until task learned

2.6 Implementing a combined IRL/PILCO algorithm

By combining PILCO with the *max-margin* IRL algorithm, we aim to create a high-efficiency apprenticeship learning method within the PILCO framework.

In the next chaper, we explain how the IRL *max-margin* algorithm is incorporated into the PILCO algorithm.

Chapter 3

The IRL/PILCO Algorithm

3.1 Generic Algorithm

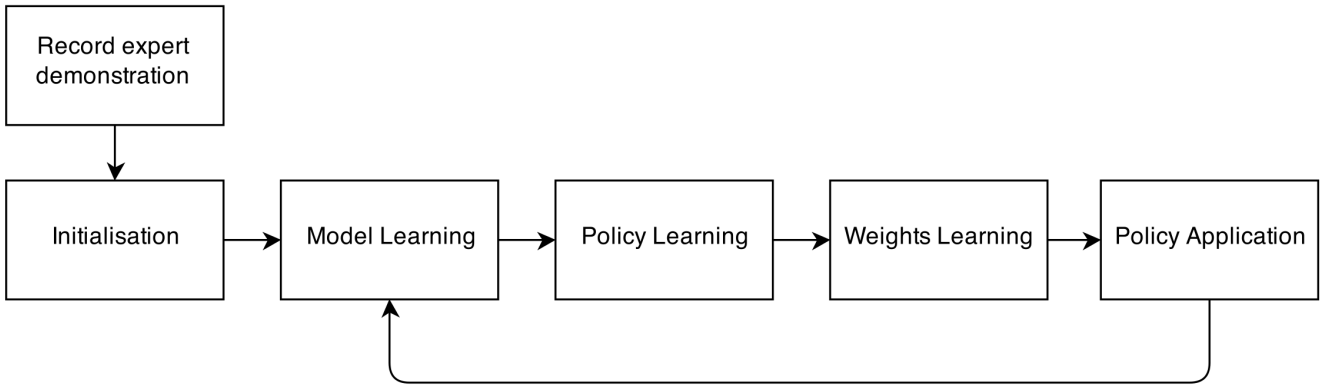

Figure 3.1: The IRL/PILCO computational flowchart.

The simple computational steps involved in the IRL/PILCO algorithm is shown in Figure 3.1. Firstly, the demonstration by an expert is recorded. Next, initialisation takes place where a random policy is applied. The resulting trajectory is used to train the GP during the model learning stage, which in turn is the basis for predicting trajectories in the policy learning step. The optimal policy is then computed by using the GP model to minimise the long-term trajectory cost. Once the weights of the cost function are minimised, the policy is applied to the system. The algorithm returns to model learning, until the task is learnt.

3.2 The IRL/PILCO Pseudocode

The high-level algorithm for IRL/PILCO is as shown in Algorithm 3. The new IRL additions to PILCO are highlighted in yellow.

3.3 Complexity

Training a GP using gradient-based evidence maximisation is an $O(n^3)$ operation due to matrix inversion of the kernel matrix K in Equations (2.19)–(2.20), for an n-sized data set. For E states, this gives a complexity of $O(En^3)$.

Algorithm 3 IRL/PILCO

expert: Record expert demonstration(s) x_E. Set $\mu_E \leftarrow \mu(x_E)$
init: Create random policy $\theta \sim N(0, I)$ and record a single trajectory
Model-based policy search:
repeat
    Apply policy and record a single state trajectory
    Learn GP dynamics model using all state trajectories.
    repeat
        Approximate inference for policy evaluation, compute $J^\pi(w, \theta)$.
        Gradient-based policy improvement, compute $dJ^\pi/d\theta$
        Optimise policy parameters θ(using CG/L-BFGS).
    until convergence, **return** θ^*
    Set $\pi^* \leftarrow \pi(\theta^*)$
    max-margin: Find $w \leftarrow \arg\min_w w^\top(\mu_E - \mu_x)$ s.t. $\|w\|_2 = 1$ (using non-linear/QP solver)
    Apply π^* to system and record single trajectory x.
    Set $\mu_x \leftarrow \mu(x)$
until task learned

The IRL/PILCO algorithm bottlenecks on PILCO policy search, which has a complexity of $O(E^2n^2D)$, where D is the dimensionality of the training inputs. The full derivation is convoluted; interested readers are invited to see Deisenroth (2010).

3.4 Computing Long-Term Cost

We implemented two ways of computing long-term cost, termed the *feature expectations* method from Ng and Russell (2004) and also the *feature matching* method.

3.4.1 Feature Expectations Method

In the feature expectations method, we simply sum the discounted features in the manner

$$\mu = E[\sum_t \gamma^t \phi^{(t)}] \tag{3.1}$$

where $\phi^{(t)}$ is the feature vector at time step t and γ is the discount factor. The cost for the trajectory is given by

$$J = \alpha w^\top \|\mu - \mu_E\|_2^2 \tag{3.2}$$

where α is a constant for numerical reasons and w is the weight vector between the different feature expectations. The subscript on μ_E denotes the expert. In this method, for a linear cost function, the cost over the trajectory will be the same as the expert if, on average, the same features are visited (even if they are visited in a different order).

3.4.2 Feature Matching Method

The feature matching method uses the trajectory feature vector on every time step to penalise for deviation from the expert trajectory.

The features expectation μ at each time step t of the trajectory is simply

$$\mu^{(t)} = E[\phi^{(t)}] \tag{3.3}$$

where $\phi^{(t)}$ is the feature vector at time step t. The cost function is then

$$J = \sum_t \gamma^t c(t) \tag{3.4}$$

$$c(t) = \alpha w^\top \|\mu^{(t)} - \mu_E^{(t)}\|_2^2 \tag{3.5}$$

$$\tag{3.6}$$

where $\alpha = 1000$ is a constant set for numerical reasons, and γ is the discount factor. The subscript on μ_E denotes the expert. This cost function allows us to penalise for every feature that deviates from the expert trajectory.

Chapter 4

Simulations with the Cart Pole

4.1 Cart-pole Swing Up

The cart-pole swing up problem is a physical system where a cart, moving in one dimension, has a stiff pendulum pole attached. The goal is to balance the pendulum above the cart by applying a force to the cart. The system can be specified by four parameters $\{x_1, \dot{x}_1, \theta_2, \dot{\theta}_2\}$ which denote the positions and velocities of the cart and the pendulum, respectively.

Figure 4.1: A physical pole-cart (inverted pendulum) being balanced. Taken from (Deisenroth and Rasmussen, 2011)

Classical reinforcement learning with PILCO was able to balance the cart-pole using only 7 trials with a total interaction time of 17.5s. (Deisenroth and Rasmussen, 2011)

4.2 Simulation Dynamics

The cart-pole system is a simulated system in which a cart of mass m_1 starting at the $x = 0$ position, has an attached pendulum of mass m_2 which is able to swing freely in the plane with angular position θ_2. The cart is able to move horizontally in the x-direction with an external applied force u.

Solving the Lagrangian for the system gives the equations of motion

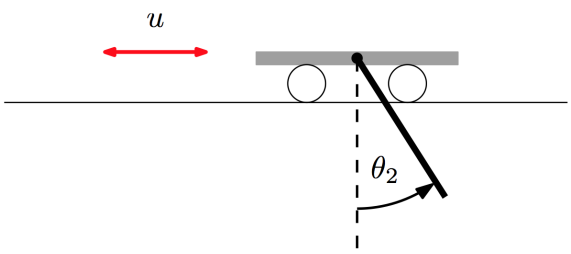

Figure 4.2: The idealised cart-pole dynamics system. Taken from Deisenroth (2010)

$$(m_1 + m_2)\ddot{x}_1 + \frac{1}{2}m_2 l\ddot{\theta}_2 \cos\theta_2 - \frac{1}{2}m_2 l\dot{\theta}_2^2 \sin\theta_2 = u - b\dot{x}_1 \tag{4.1}$$

$$2l\ddot{\theta}_2 + 3\ddot{x}_1 \cos\theta_2 + 3g \sin\theta_2 = 0 \tag{4.2}$$

which is solved numerically and simulated as coupled differential equations in MATLAB.

4.3 Implementation Details

4.3.1 Obtaining the Cart-Pole State

The state of the Cart-Pole system at each time step t is represented with the position variables

$$x^{(t)} = E[x \ \sin\theta \ \cos\theta]^\top \tag{4.3}$$

and the control signal (action) is simply the force u which is able to pull/push the cart on every time step. $\sin\theta$ and $\cos\theta$ are used instead of the angular position θ so that we can take advantage of angular wrap-around over the period 2π. The state-action tuple is then defined as

$$\tilde{x}^{(t)} = [x^\top u^\top]^\top \tag{4.4}$$

which gives $\tilde{x} \in \mathbb{R}^5$. $\tilde{x}$ is the predictive input for the GP dynamics model for the output $x^{(t+1)}$.

4.3.2 Optimising Weights

The weights w were obtained by using the MATLAB nonlinear optimiser *fmincon* on the equation

$$w = \arg\min_w t^{(i)}(w) \tag{4.5}$$

where

$$t^{(i)}(w) = w^\top(\mu_E - \mu^{(i)}), \tag{4.6}$$

on the constraints $\|w\|_2 = 1$, $t^{(i)} \leq t^{(i-1)}$, where i is the current iteration of the IRL/PILCO algorithm.

4.3.3 Controller Application

The cart-pole controller is a Radial Basis Function (RBF) network and is given by

$$u = \sum_{i=1}^{n} w_i \exp(-(x - m_i)^\top W (x - m_i)/2) \tag{4.7}$$

where m_i and w_i are the centres and weights of each radial basis function respectively, W is a diagonal matrix which controls the spread of the basis functions along the axes, n is the total number of RBFs.

The controller output u is subsequently mapped through to a squashing function

$$u_{control} \leftarrow u_{max} \cdot (9\sin(u) + \sin(3u))/8 \tag{4.8}$$

which limits the control signal such that the applied signal $|u_{control}| \leq u_{max}$.

4.4 Results from Feature Matching on Every Time Step

The features expectation μ at each time step t of the state trajectory is simply

$$\mu^{(t)} = x^{(t)} \tag{4.9}$$

and the cost function is

$$J = \sum_t \gamma^t c(t) \tag{4.10}$$

$$c(t) = \alpha \|\mu^{(t)} - \mu_E^{(t)}\|_2^2 \tag{4.11}$$

$$\tag{4.12}$$

where $\alpha = 1000$ is a constant set for numerical reasons, and $\gamma = 1$ is the discount factor. This cost function allows us to penalise for every feature that deviates from the expert trajectory. The expert feature expectations μ_E was obtained by using PILCO to do classical RL.

Using this cost function, a policy that could balance the cart-pole was learnt on only the 4th iteration of IRL/PILCO (Figures 4.3–4.4). The best results were from the 15th iteration where a policy that could almost exactly recover the expert trajectory was learnt (Figures 4.5–4.6).

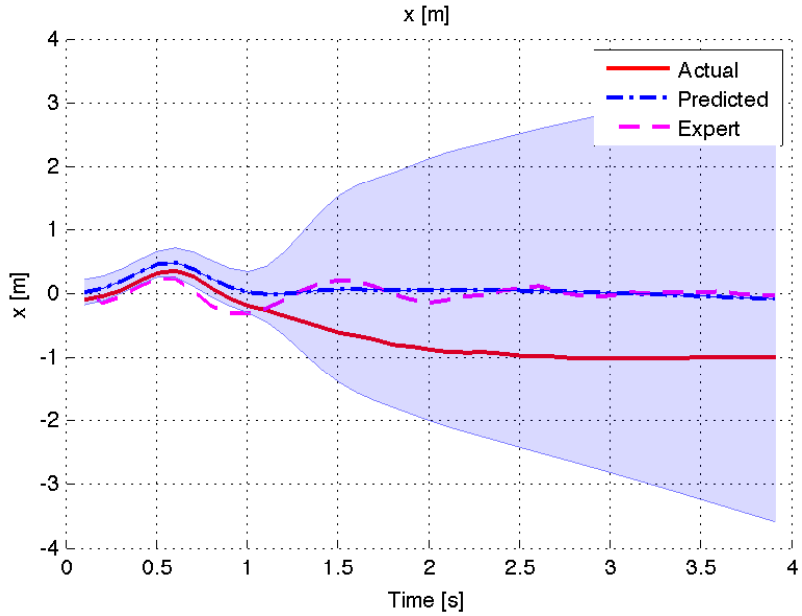

Figure 4.3: x-position of the cart-pole, from the 4th iteration, using the feature matching method. Although the controller balanced the cart-pole, it did so sub-optimally because the x-position reaches a steady state of about $0.2m$ away from the $x = 0$ position. The predicted trajectory by the GP is well-trained and the policy remains somewhat useful as it is still able to balance the cart-pole. The shaded area is the confidence interval of the dynamics model.

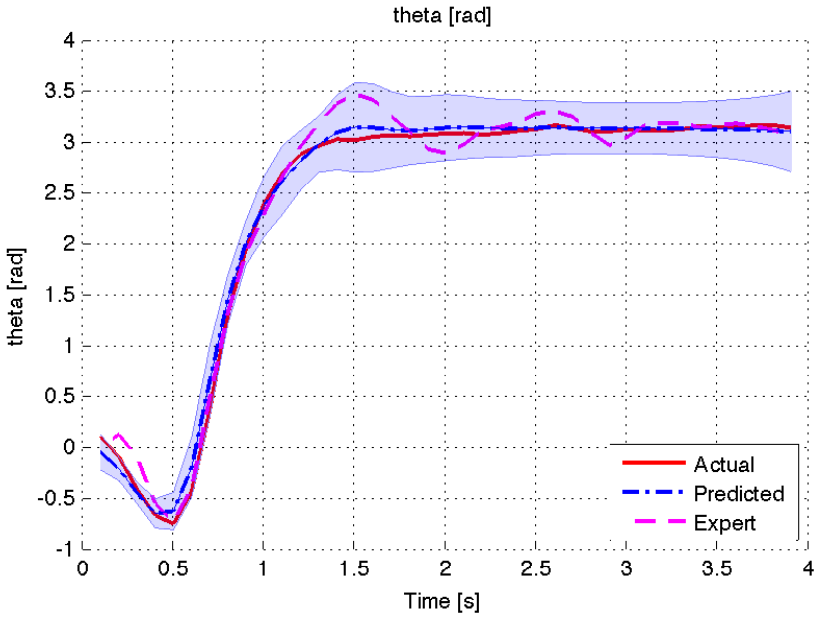

Figure 4.4: Graph of θ, on the 4th iteration, using the feature matching method. As can be observed, the cart-pole system is able to converge quickly to the $\theta = 0$ position without overshooting; however this comes at the drawback of not balancing at the cart-pole at the $x = 0$ position.

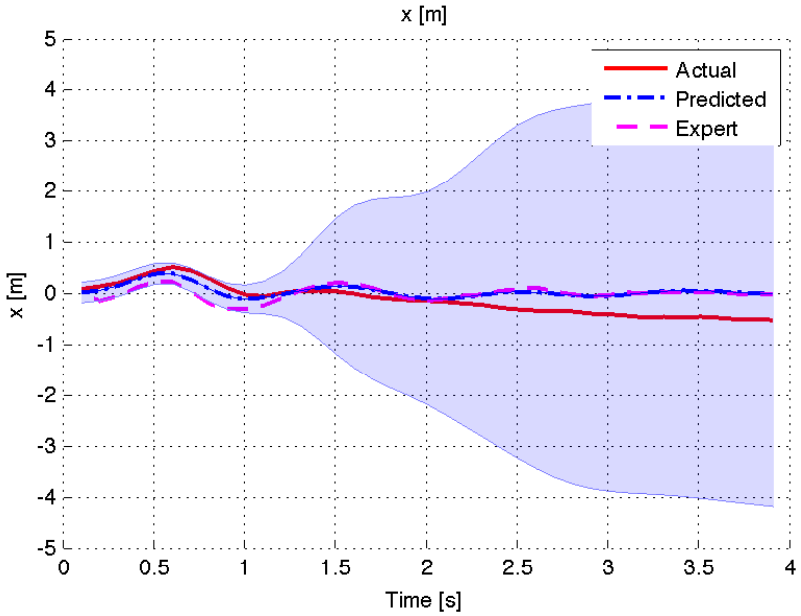

Figure 4.5: x-position of the cart-pole, from the 15th iteration, using the feature matching method. Although the controller balanced the cart-pole, it did so sub-optimally because the x-position reaches a steady state, or about $0.2m$ away from the $x = 0$ position. The shaded area is the confidence interval of the predictive dynamics model.

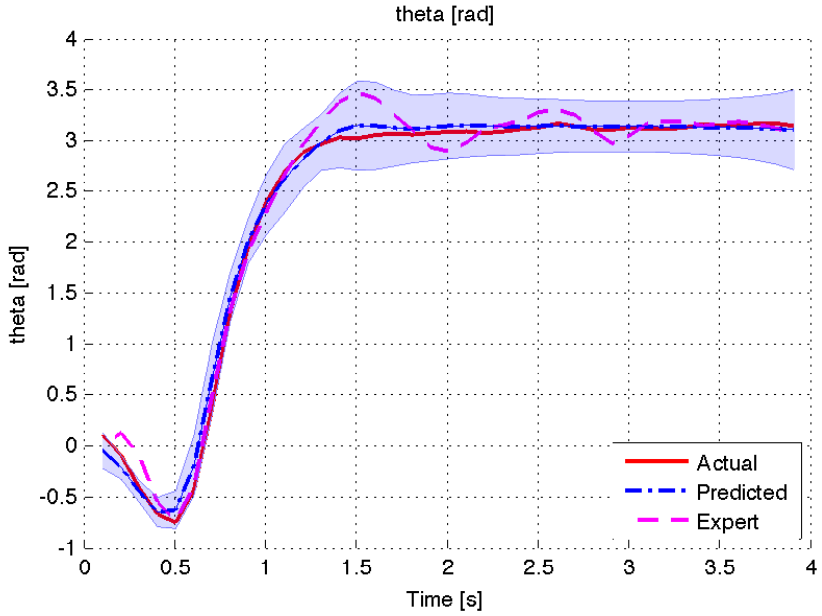

Figure 4.6: Graph of θ, on the 15th iteration, using the feature matching method. As can be observed, the actual trajectory matches up nearly exactly with the expert's trajectory.

4.5 Results from Feature Expectations

In feature expectations matching for the cart-pole system, we simply sum the discounted features in the manner

$$\mu = \sum_t \gamma^t x^{(t)} \tag{4.13}$$

and the cost for the trajectory is given by

$$J = \alpha \|\mu - \mu_E\|_2^2 \tag{4.14}$$

Using this approach, we were able to obtain a policy that could somewhat balance the inverted pendulum on the 15th iteration(Figures 4.7–4.8).

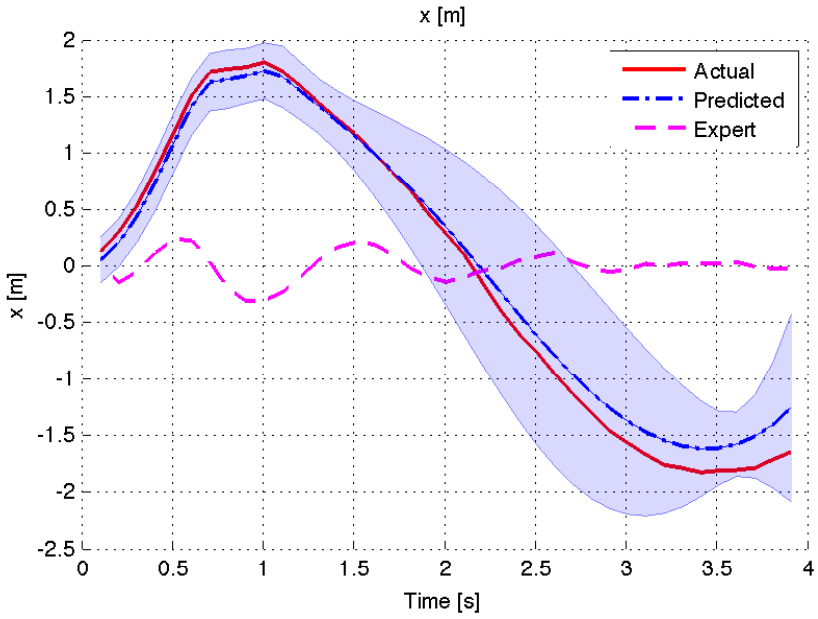

Figure 4.7: x-position of the cart-pole, from the 15th iteration, using the features expectation method. The controller discovers a trajectory that gives the same feature expectation as the expert trajectory. The shaded area is the confidence interval of the dynamics model. With the features expectation method, the cost of following this trajectory is the same for the expert and the actual trajectories, even though the physical state trajectory differs wildly.

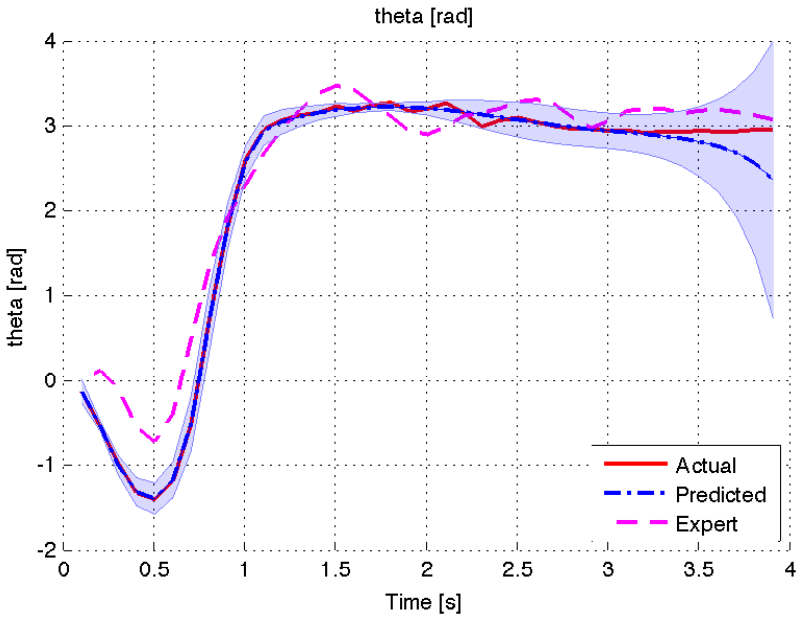

Figure 4.8: Graph of θ, on the 15th iteration, using the features expectation method. It is probable that because the end-goal of inverting the cart-pole is only implied – that is, the steady-state of the inverted cart-pole is not obvious with a shorter trajectory – that the features expectation approach is only able to balance the cart-pole near $\theta \approx \pi$ and not precisely on $\theta = \pi$.

Chapter 5

Experiments with Baxter Robot

In this project, I taught the Baxter robot (Figure 1) to sweep a surface using a one-handed dustpan and brush, using solely the Baxter motor encoders and velocity control of the joints. This is a non-trivial task due to the large state-action space from using 2 arms and 14 joints.

5.1 Design and Implementation

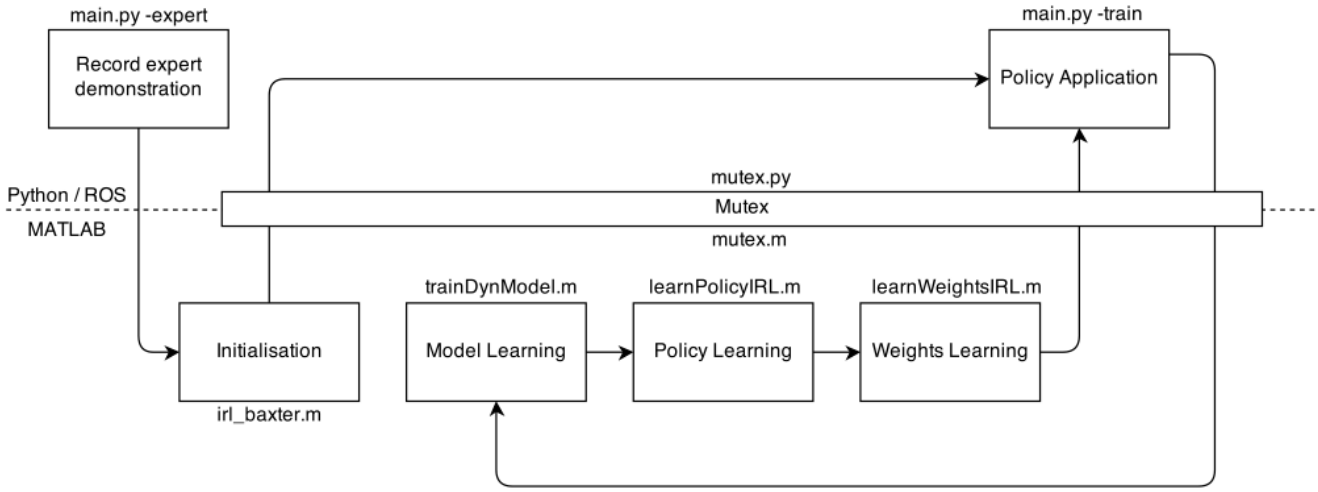

Figure 5.1: The Baxter IRL/PILCO setup.

The specific implementation of the IRL/PILCO algorithm within the Baxter setup is shown in Figure 5.1. This is simply a more detailed and implementation-specific flowchart of Figure 3.1.

The Baxter program for implementing the policy controller and recording the expert trajectory were written in `Python` on `ROS Hydro`(Quigley et al., 2009) on a virtual Linux Precise platform, using the `baxter_interface` library from RethinkRobotics. The MATLAB scripts were written and run natively on OSX.

A mutex was also written using a shared file to allow control execution to pass between the MATLAB learning script and Python Baxter controller. This is simply to bridge the gap between MATLAB and the robot, which were executing on different platforms, and is strictly an implementation-specific step. Alternatives include using TCP/IP to communicate between the Linux-based Python controller and the rest of the MATLAB learning algorithm.

5.2 Implementation Details

5.2.1 Obtaining the Robot State

The state of the Baxter robot at each time step t is represented with the vector

$$x^{(t)} = E[\theta_l^\top \dot{\theta}_l^\top \theta_r^\top \dot{\theta}_r^\top]^\top \tag{5.1}$$

and the control signals (action) u of the robot is the vector

$$u^{(t)} = E[u_l^\top u_r^\top]^\top \tag{5.2}$$

where θ is the angular position of the arm joint in radians, $\dot{\theta} = d\theta/dt$ is the angular velocity, and the subscripts l and r indicate the left and right arms of the Baxter respectively at a specific time step.

Each arm of the robot has 7 joints, whose state can be fully specified by the angular position and velocity of these joints. This gives a dimensionality of $x \in \mathbb{R}^{28}$. The control actions are the setting of joint velocities; with 14 joints in total giving $u \in \mathbb{R}^{14}$. The state-action tuple is then defined as

$$\tilde{x}^{(t)} = [x^\top u^\top]^\top \tag{5.3}$$

which gives $\tilde{x} \in \mathbb{R}^{42}$. $\tilde{x}$ is the predictive input for the GP dynamics model for the output $x^{(t+1)}$. A sampling frequency of 2 Hz was used.

5.2.2 Computing Features Expectation

The features expectation μ at each time step t of the state trajectory is simply the angular position vectors

$$\mu^{(t)} = [\theta_l^\top \theta_r^\top]^\top \tag{5.4}$$

and using Equation (5.4) we can compute the features expectation of the expert $\mu_E^{(t)}$ at every time step for $\mu \in \mathbb{R}^1 4$. The angle θ is directly used rather than $\sin(\theta), \cos(\theta)$ because the robot joints cannot rotate more than 2π and the angles do not wrap around.

5.2.3 Policy Learning

To do policy learning, the trajectory cost J^π and its derivative w.r.t. the policy parameters $J^\pi/d\theta$ must be computed. These are used to obtain the optimal policy π^* via gradient descent methods.

The cost at every time step in calculated as

$$c^{(t)}(\mu) = \gamma\alpha\|\mu^{(t)} - \mu_E^{(t)}\|_2^2 \tag{5.5}$$

where γ is the discount factor and α is a scaling factor constant (usually set to 10^3) for numerical purposes. This formulation with the 2-norm ensures that the cost is always positive. The derivative with respect to the feature expectation is then

$$\frac{dc^{(t)}}{d\mu} = 2\gamma\alpha(\mu^{(t)} - \mu_E^{(t)}) \tag{5.6}$$

The long-term cost for the entire trajectory is $J^\pi = \sum_t c^{(t)}$ as per Equation (2.25) and the derivative $dJ^\pi/d\theta$ is computed from Equations (2.36)–(2.38), after which gradient descent methods are used by the PILCO algorithm to conduct policy search.

5.2.4 Optimising Weights and Controller Application

As is the case with the cart-pole simulation, the weight vector of the cost function is given by Equations (4.5)–(4.6)

5.2.5 Controller Application

The controller implemented is linear, of the form

$$u = wx + b \tag{5.7}$$

where w is a weight matrix, b is the bias vector, x is the state and $\{w, b\} \in \theta$.

This is then mapped through to the squashing function as shown in Equation (4.8) in order to obtain the applied control signal $u_{control}$.

5.3 Results

5.3.1 Results from Feature Expectations

The results from feature expectations were unsuccessful, which was not unexpected. A sample outcome is shown here in Figure 5.2.

As can be observed, one of the hallmarks of the feature expectations approach is that the time-order of features need not match – because in the case of a linear cost function, the total cost is the same when the discounted sum of features is the same. Ultimately, it does not matter to the algorithm which order the features are visited in, as long as the feature expectations matches that of the expert trajectory.

The result is that the algorithm produces a policy in which the same costs are occurred as the expert demonstration, but the resulting state trajectory can be much more varied than the expert's state trajectory.

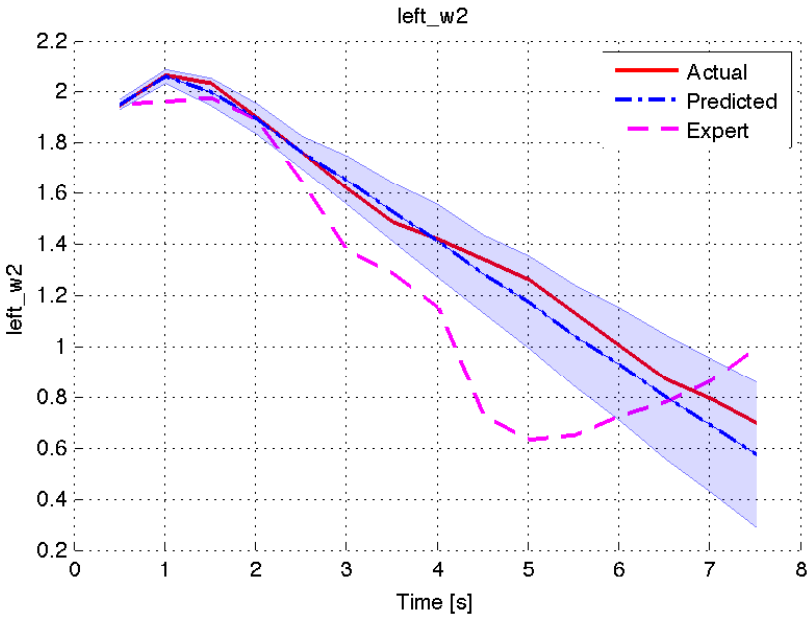

Figure 5.2: Graph of angular position of $left\_w2$, on the 13th iteration using the feature expectations method. This plot of $left\_w2$ is typical of the other feature variables, where the policy matches feature expectations by simply producing an "average" trajectory of the expert demonstration. Compare with Figure 5.3, where the trajectories were much closer with the feature matching method.

5.3.2 Results from Feature Matching on Every Time Step

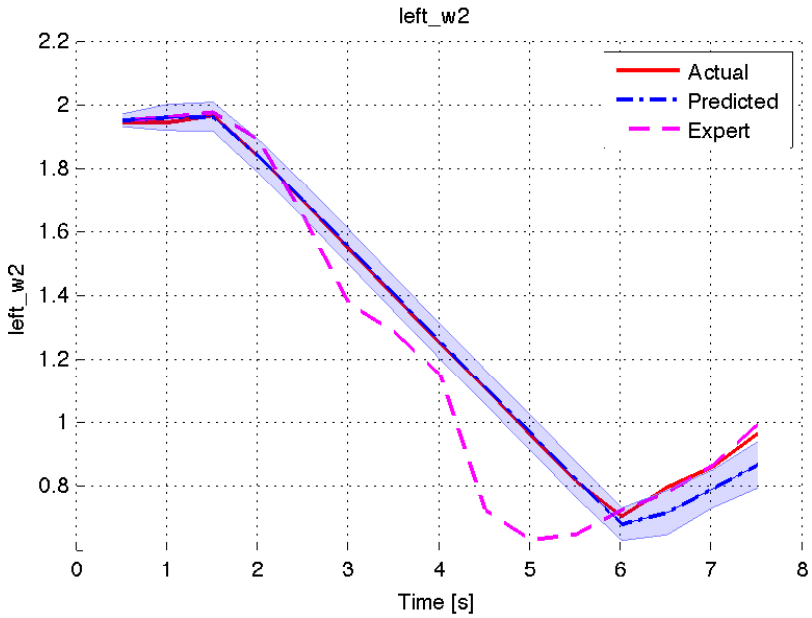

Figure 5.3: Graph of angular position $left\_w2$, on the 15th iteration using the feature matching method. The linear controller matches the robot trajectory to the expert trajectory much more closely at the same joint, although the trajectory is not particularly smooth. This is due to the nature of linear controllers, which although faster to train, are less flexible than radial basis function networks. Compare with Figure 5.2, which uses the feature expectations method.

Video Footage of Results

Figure 5.4: The Baxter robot sweeping using a brush and dustpan.

Screen captures of the Baxter sweeping can be seen at Figure 5.4. A video demonstration of the Baxter experiment is available at `http://www.doc.ic.ac.uk/~yl7813/irlpilco_baxter`.

Chapter 6

Discussion

Even though there is a lack of standard performance metrics (Argall et al., 2009), we can compare the performance of our algorithm across different implementations by using a mean-squared difference method inspired by Wu and Demiris (2010). The mean-squared error ϵ is given by

$$\epsilon = \frac{1}{DT} \sum_{t=0}^{T} \|\phi_E^{(t)} - \phi^{(t)}\|_2^2 \tag{6.1}$$

where D is the number of features, $\phi^{(t)}$ is the vector of features at time step t. By normalising over the number of features and the number of time steps, we can compare the results more objectively between the cart-pole simulation and the Baxter experiment. Note that in the case of feature matching, $\phi^{(t)} = \mu^{(t)}$. We compare results from the feature expectations method (Section 3.4.1) against the feature matching method (Section 3.4.2) in Figure 6.1.

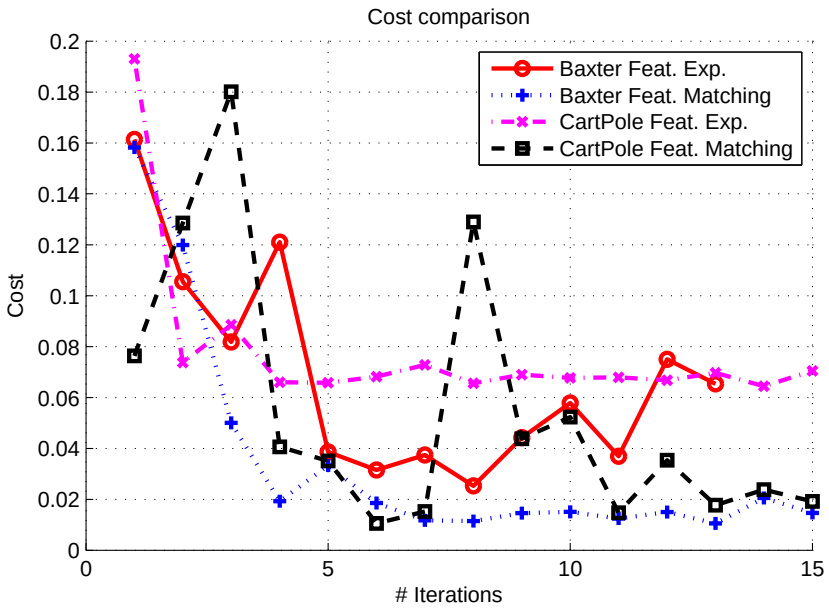

Figure 6.1: Comparison between feature expectations (Feat. Exp.) and feature matching on every time step (Feat. Matching) for both the cart-pole simulation and the Baxter experiment.

From Figure 6.1, it is clear that the feature matching approach results in a closer matching trajectory to the expert trajectory than the feature expectations approach. This is not unexpected, because the feature expectations method does not penalise when features are visited outside of

their time order, unlike feature matching, which penalises any difference in features on every time step.

In addition, the Baxter feature expectations approach appears to be diverging, which may possibly be due to noise - the algorithm may converge if run for further iterations, but it is expensive to do so due to the large state-action space of $\mathbb{R}^{42}$ as used with the Baxter.

Our approach in using the angular state positions of the Baxter joints in this project has a few drawbacks. Because of the task at hand (sweeping), the position and orientation of the Baxter's end-effector is more useful than the joint angles alone.

Two main problems arise, the first being that joints higher up in the robot arm (e.g. shoulder joints) can cause error amplification: a small angular movement at the shoulder joint would cause the end-effector to move a much larger distance than an equivalent angular movement at an elbow joint. The second problem is that the choice of features ought to capture the task at hand. In the case of balancing the cart-pole, the task can be fully specified using only position variables x and the angle θ. However, in the case of the Baxter's sweeping action, the task of sweeping would be much harder without some form of external input measurement on the state of the environment, such as camera feedback on the position of objects within its vicinity.

Chapter 7

Conclusion

7.1 Key Results and Contributions

In this project, we extended a model-based policy search reinforcement learning method to apprenticeship learning, where the underlying reward function is unknown. However, expert demonstrations are given from which the reward function can be inferred.

We considered a real-robot application of apprenticeship learning where a Baxter robot with 14 degrees of freedom (2 arms with 7 joints each) was supposed to imitate a demonstrated sweeping movement. In the context of robot learning, it is essential to keep the number of interactions with the robot low, since interactions are time-consuming and wear the hardware out. Therefore, we generalised a state-of-the-art reinforcement learning algorithm (Deisenroth and Rasmussen, 2011) to the apprenticeship learning set-up.

We have demonstrated the success of our proposed IRL/PILCO algorithm in the context of apprenticeship learning. Additionally, due to its exceptional data efficiency, motor babbling for model learning is unnecessary, reducing interaction time to only 7 trials and 56 seconds with the Baxter robot.

We have successfully applied our proposed algorithm to learning a sweeping task with a Baxter robot, where we learned low-level controllers for both arms, such that the demonstrated behaviour was closely imitated.

With the IRL/PILCO algorithm, an excessive number of robot interactions is unnecessary, and only a single expert demonstration is required for successful learning. Our method is thus potentially useful for *one-shot learning*.

Finally, to the best of our knowledge we were the first to apply PILCO (or any other RL/IRL method) to such a high-dimensional state-action space ($\tilde{x} \in \mathbb{R}^{42}$), thus demonstrating that PILCO is a viable approach for learning low-level robot controllers in high-dimensional spaces.

7.2 Future Work

One could considering using non-linear reward functions, which can better capture task requirements (Levine et al., 2011). Additionally, if the form of the reward function is difficult or complicated to specify, feature extraction methods such as those by Guyon et al. (2006) can be used, although more expert demonstrations may be needed.

Next, one could incorporate a variance penalty to the model to discourage uncertainty. For instance, Englert et al. (2013b) used Kullback-Leibler (KL) divergence between the predicted trajectory distribution and the distribution over expert trajectories for imitation learning.

Additionally, one could improve on the dynamics model to predict features from state variables. In doing the sweeping motion with the Baxter, it is fairly obvious that the position and orientation of the arm end-effector is a more succinct representation of features for the task at hand, rather than the angular positions of each joint. However, this would require implementing a principled framework to convert the probability distributions of state information μ_x, Σ_x to features μ_ϕ, Σ_ϕ which will require substantive work.

The time complexity of PILCO could also be improved on. The current implementation of IRL/PILCO on the Baxter robot with 28 degrees of freedom requires $\sim$15 hours of runtime with the features expectation method, and $\sim$30 hours with the feature matching method on a Haswell i7 processor. Improvements could be made by implementing key parts such as matrix inversion and kernel computations using GPUs or multi-threading in order to boost performance.

Finally, one could leverage on the Baxter cameras, by using image recognition to allow the Baxter to properly interact fully with its environment.

Bibliography

Abbeel, P., A. Coates, and A. Y. Ng
2010. Autonomous helicopter aerobatics through apprenticeship learning. *International Journal of Robotics Research*.

Argall, B. D., S. Chernova, M. Veloso, and B. Browning
2009. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483.

Atkeson, C. G. and J. C. Santamaria
1997. A comparison of direct and model-based reinforcement learning. In *International Conference on Robotics and Automation*, Pp. 3557–3564. IEEE Press.

Bagnell, J. A. and J. G. Schneider
2001. Autonomous helicopter control using reinforcement learning policy search methods. In *International Conference on Robotics and Automation*, volume 2, Pp. 1615–1620. IEEE.

Bellman, R.
1957. A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684.

Bellman, R. and R. E. Kalaba
1965. *Dynamic programming and modern control theory*. Academic Press New York.

Bertsekas, D. P. and J. Tsitsiklis
1995. Neuro-dynamic programming: an overview. In *Proceedings of the 34th IEEE Conference on Decision & Control*, volume 1, Pp. 560–564. IEEE.

Boularias, A., J. Kober, and J. Peters
2011. Relative entropy inverse reinforcement learning. *International Conference on Artificial Intelligence and Statistics*, Pp. 182–189.

Broyden, C. G.
1965. A class of methods for solving nonlinear simultaneous equations. *Mathematics of computation*, Pp. 577–593.

Deisenroth, M. P.
2010. *Efficient Reinforcement Learning using Gaussian Processes*, volume 9. KIT Scientific Publishing.

Deisenroth, M. P., G. Neumann, and J. Peters
2013. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142.

Deisenroth, M. P. and C. E. Rasmussen
2011. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on Machine Learning*, Pp. 465–472.

Englert, P., A. Paraschos, J. Peters, and M. P. Deisenroth
2013a. Model-based imitation learning by probabilistic trajectory matching. In *International Conference on Robotics and Automation*, Pp. 1922–1927. IEEE.

Englert, P., A. Paraschos, J. Peters, and M. P. Deisenroth
2013b. Probabilistic model-based imitation learning. *Adaptive Behavior*, 21(5):388–403.

Fletcher, R. and M. J. Powell
1963. A rapidly convergent descent method for minimization. *The Computer Journal*, 6(2):163–168.

Guyon, I., S. Gunn, M. Nikravesh, and L. Zadeh
2006. Feature extraction. *Foundations and applications*.

Lagoudakis, M. G. and R. Parr
2003. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149.

Levine, S., Z. Popovic, and V. Koltun
2011. Nonlinear inverse reinforcement learning with gaussian processes. *Advances in Neural Information Processing Systems*, Pp. 19–27.

Ng, A. Y. and M. Jordan
2000. Pegasus: A policy search method for large mdps and pomdps. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, Pp. 406–415. Morgan Kaufmann Publishers Inc.

Ng, A. Y. and S. Russell
2000. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, Pp. 663–670.

Ng, A. Y. and S. Russell
2004. Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine Learning*, Pp. 1–8.

Peters, J., K. Mülling, and Y. Altun
2010. Relative entropy policy search. In *AAAI*.

Pomerleau, D. A.
1989. Alvinn: An autonomous land vehicle in a neural network. Technical report, DTIC Document.

Quigley, M., K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng
2009. ROS: An open-source robot operating system. In *ICRA workshop on open source software*, volume 3, P. 5.

Rasmussen, C. E. and C. K. I. Williams
2006. *Gaussian Processes for Machine Learning*. MIT Press.

Ratliff, N., D. Bradley, J. A. Bagnell, and J. Chestnutt
2007. Boosting structured prediction for imitation learning. *Robotics Institute*, P. 54.

Ratliff, N., B. Ziebart, K. Peterson, J. A. Bagnell, M. Hebert, A. K. Dey, and S. Srinivasa
2009a. Inverse optimal heuristic control for imitation learning. In *International Conference on Artificial Intelligence and Statistics*.

Ratliff, N. D., D. Silver, and J. A. Bagnell
2009b. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 27(1):25–53.

Rummery, G. A. and M. Niranjan
1994. On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Department of Engineering, University of Cambridge, Trumpington Street, Cambridge CB2 1PZ, UK.

Russell, S.
1998. Learning agents for uncertain environments. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, Pp. 101–103. ACM.

Shanno, D. F.
1970. Conditioning of quasi-newton methods for function minimization. *Mathematics of computation*, 24(111):647–656.

Sutton, R. S.
1988. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44.

Sutton, R. S. and A. G. Barto
1998. *Introduction to Reinforcement Learning*. MIT Press.

Watkins, C. J. and P. Dayan
1992. Q-learning. *Machine learning*, 8(3-4):279–292.

Wu, Y. and Y. Demiris
2010. Towards one shot learning by imitation for humanoid robots. In *International Conference on Robotics and Automation (ICRA)*, Pp. 2889–2894. IEEE.

Ziebart, B. D., A. Maas, J. A. Bagnell, and A. K. Dey
2008. Maximum entropy inverse reinforcement learning. *AAAI*, Pp. 1433–1438.

Appendix A

Algorithms for MDPs

A.0.1 Algorithms for MDPs

Value Iteration

Here, we assume that an an MDP $M = \{X, U, P(x, u, x')\}, \gamma, R$ be given. The goal is to find the optimal values at every state.

The algorithm used is value iteration:

1. Let $V_0(x) = 0$ for all $x \in X$
2. Update via dynamic programming:

$$V_{k+1} \leftarrow \max_u E[R(x, u, x') + \gamma V_k(x')] \tag{A.1}$$

The value iteration algorithm always converges from terminal nodes.

Policy Extraction

If, given an MDP, then the best action at every state may be computed using

$$\pi(x) \leftarrow \arg\max_u E[R(x, u, x') + \gamma V^*(x')] \tag{A.2}$$

if the optimal values V* are known, or

$$\pi(x) \leftarrow \arg\max_u Q^*(x, u) \tag{A.3}$$

if the optimal Q-values Q^* are known.

Policy Evaluation

Under a fixed policy, the actions taken are dictated by the policy π, so the value of states can be computed as

$$V^{\pi}(x) = E[R(x, \pi(x), x') + \gamma V^{\pi}(x')] \tag{A.4}$$

Using Equation (A.4), we can write the iterative algorithm to evaluate policies for each state $x \in X$:

1. Let $V_0(x) = 0$ for all $x \in X$
2. Update via dynamic programming:

$$V^{\pi}_{k+1}(x) \leftarrow \max_{\pi(x)} E[R(x, \pi(x), x') + \gamma V^{\pi}_{k}(x')] \tag{A.5}$$

Policy Iteration

For policy iteration, we assume we have on hand a policy π that is non-optimal, and we wish to improve it. We can combine policy evaluation and policy extraction to improve our policy. This is known as policy iteration:

1. Calculate (non-optimal) utilities for some fixed policy

$$V^{\pi_k}(x) = E[R(x, \pi_k(x), x') + \gamma V^{\pi_k}(x')] \tag{A.6}$$

2. Update policy using policy extraction with converged utility values

$$\pi_{k+1}(x) \leftarrow \arg\max_{u} E[R(x, u, x') + \gamma V^{\pi_k}(x')] \tag{A.7}$$

Incorporating GPs into IRL via PILCO

Using Gaussian process regression, it is possible to learn the behavior of a probabilistic model given sufficient measurements of state trajectories and actions.

We will use GP prediction as a method of function approximation, in order to determine the transitive probabilities, which will then be incorporated into IRL and solved for the reward function.

The implementation for this can be easily provided for using PILCO, a practical, data-efficient model-based policy search method. PILCO is able to cope with very little data and can learn with very few trials via approximate inference. Deisenroth and Rasmussen (2011)

A.0.2 Standard Algorithms and Techniques in Classical RL

In reinforcement learning, the learning is still done using an MDP, except that the transition probabilities $P_{xu}(x')$ and the reward/reinforcement function $R(x, u, x')$ unknown. Therefore, in order to discover the transition probabilities and the reward function, it is necessary to interact with the system.

Temporal-Difference Learning

Temporal-difference learning is a model-free method, because R and P are not considered. In temporal-difference learning, a fixed policy π is decided upon. Then, the algorithm is to update a vector of values $V(x)$ each time we experience a sample represented by a tuple (x, u, x', r):

1. $sample = R(x, \pi(x), x') + \gamma V^{\pi}(x')$
2. $V^{\pi}(x) \leftarrow (1 - \alpha)V^{\pi}(x) + \alpha \cdot sample$

where α is a learning rate constant. A decreasing learning rate over iterations can give converging averages.

Q-Value Iteration

In temporal-difference learning, the values of the states could be learnt, but it would not be helpful in extracting a new policy without the Q-values. Hence, Q-value iteration:

1. Set $Q_0(x, u) = 0$ for all $x \in X$, $u \in U$
2. Then, do the iterative update

$$Q_{i+1}(x, u) \leftarrow \sum_{x'} P_{xu}(x')[R(x, u, x') + \gamma \max_{u'} Q_i(x', u')] \tag{A.8}$$

Q-Learning

In Q-learning, the q-values are directly learnt from the samples by iterative updates, similar to temporal-difference learning:

1. $sample = R(x, u, x') + \gamma \max_{\alpha'} Q(x', u')$
2. $Q(x, u) \leftarrow (1 - \alpha)Q(x, u) + \alpha[sample]$

where as before, α is a learning rate constant. A decreasing learning rate over iterations can give converging averages.

Feature-based Representations

In realistic situations, it is often impractical to visit every state, because

1. There are too many states to visit them all in training, or
2. Too many states to hold the q-tables in memory, or
3. The state space is continuous

Thus, we would prefer to generalise and learn about only a small number of training states. This is accomplished by using a vector of features to describe the state, rather than the state variables itself. Hence, values and q-values can be represented simply by a weighted set of features:

$$V(x) = w_1 f_1(x) + w_2 f_2(x) + ... + w_d f_d(s) \tag{A.9}$$

$$Q(x,u) = w_1 f_1(x,u) + w_2 f_2(x,u) + ... + w_d f_d(x,u) \tag{A.10}$$

With this approach, the experience can be summed up by only a few numbers, which will significantly reduce the state space. However, the disadvantage is that states may share features, but actually have very different values.

Appendix B

Results of the Baxter Experiment

B.1 Cost

These results were obtained by feature matching at every time step.

B.2 Plots

B.2.1 Left Arm

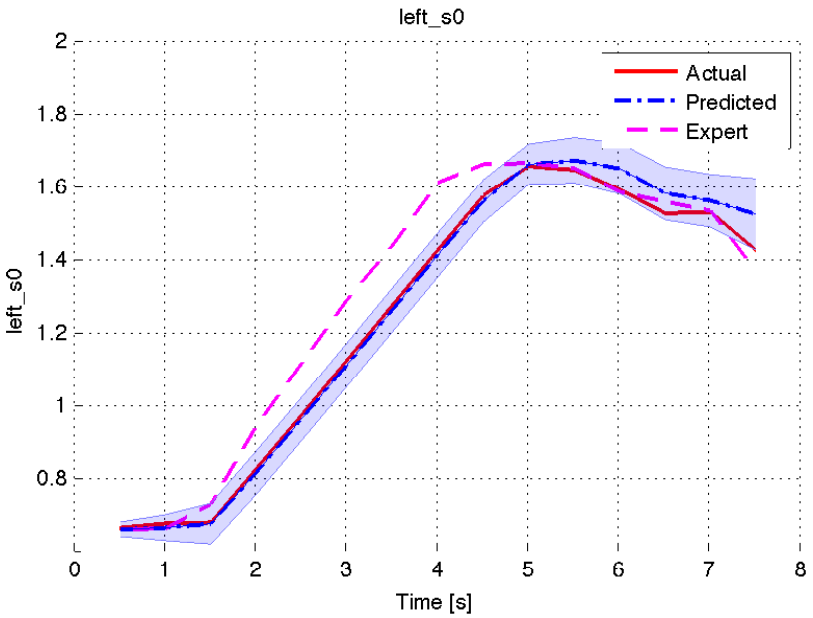

Figure B.1: Graph of angular position of $left\_s0$, on the 15th iteration.

B.2.2 Right Arm

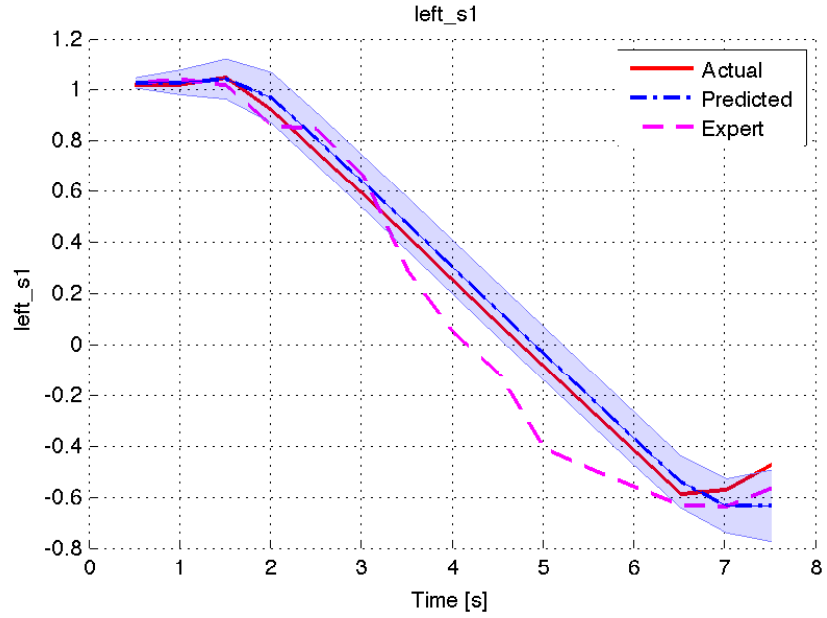

Figure B.2: Graph of angular position of $left\_s1$, on the 15th iteration.

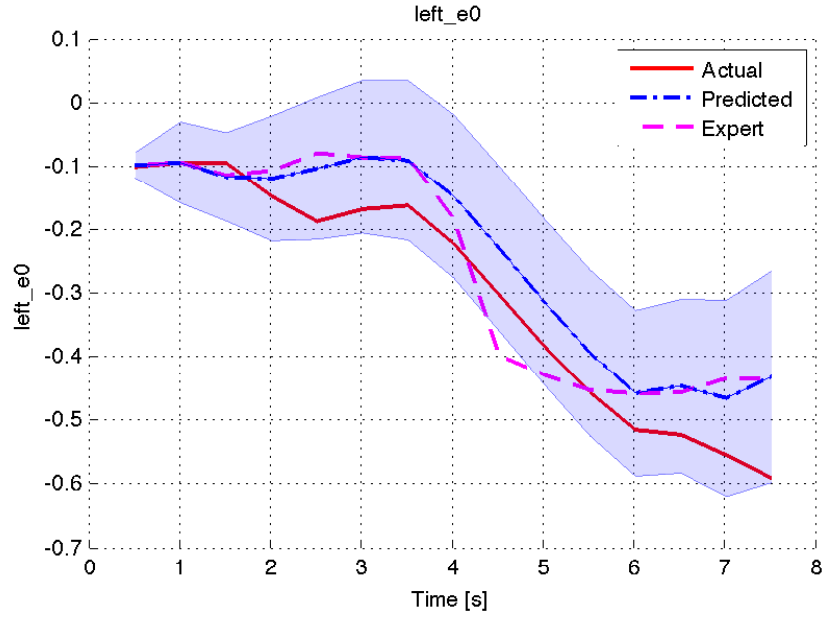

Figure B.3: Graph of angular position of $left\_e0$, on the 15th iteration.

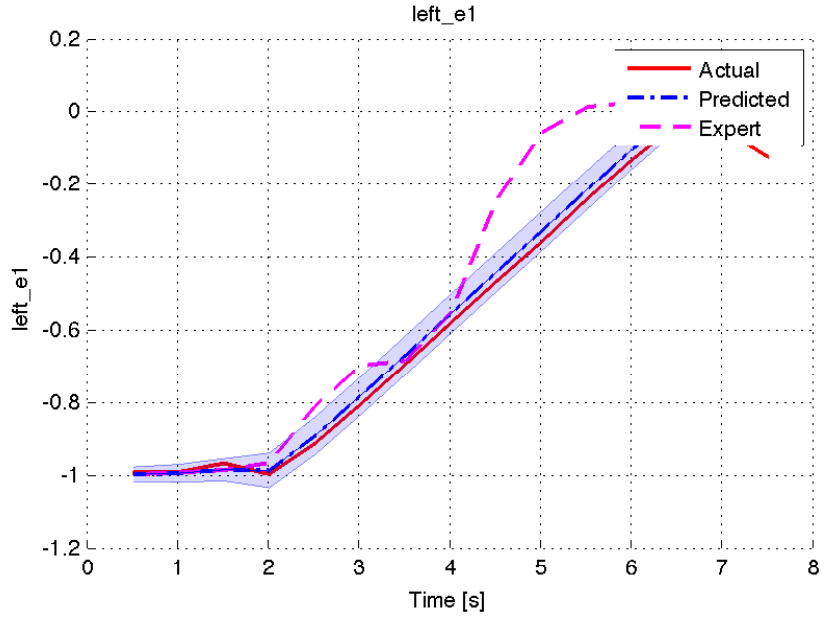

Figure B.4: Graph of angular position of $left\_e1$, on the 15th iteration.

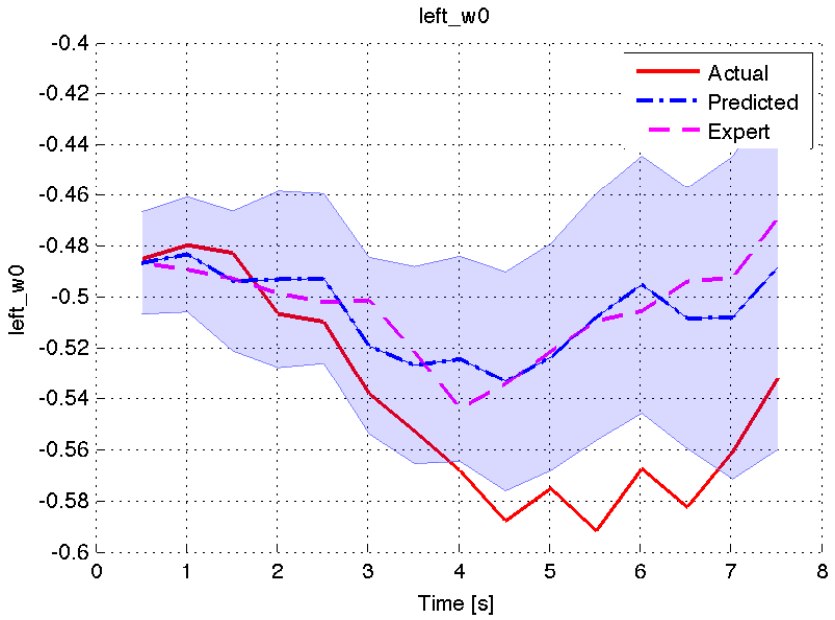

Figure B.5: Graph of angular position of $left\_w0$, on the 15th iteration.

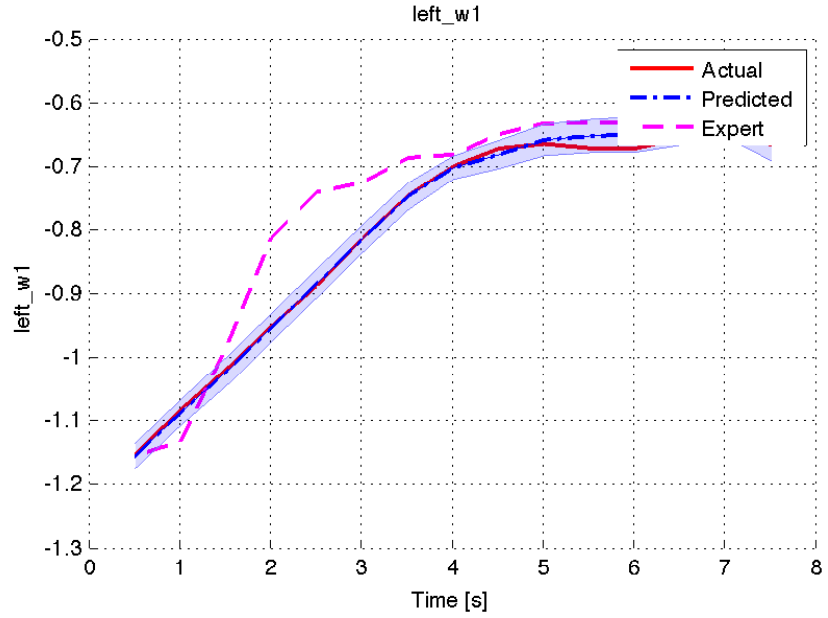

Figure B.6: Graph of angular position of $left\_w1$, on the 15th iteration.

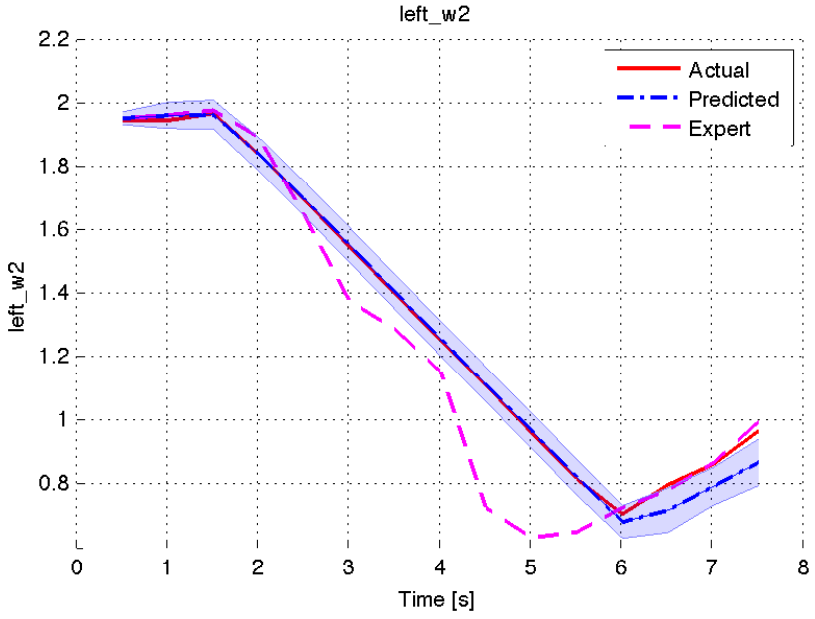

Figure B.7: Graph of angular position of $left\_w2$, on the 15th iteration.

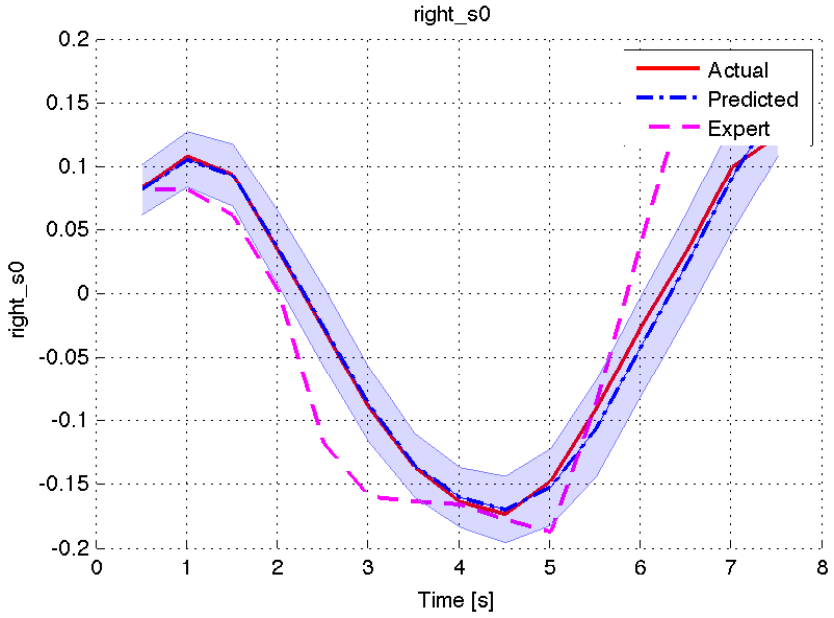

Figure B.8: Graph of angular position of *right_s*0, on the 15th iteration.

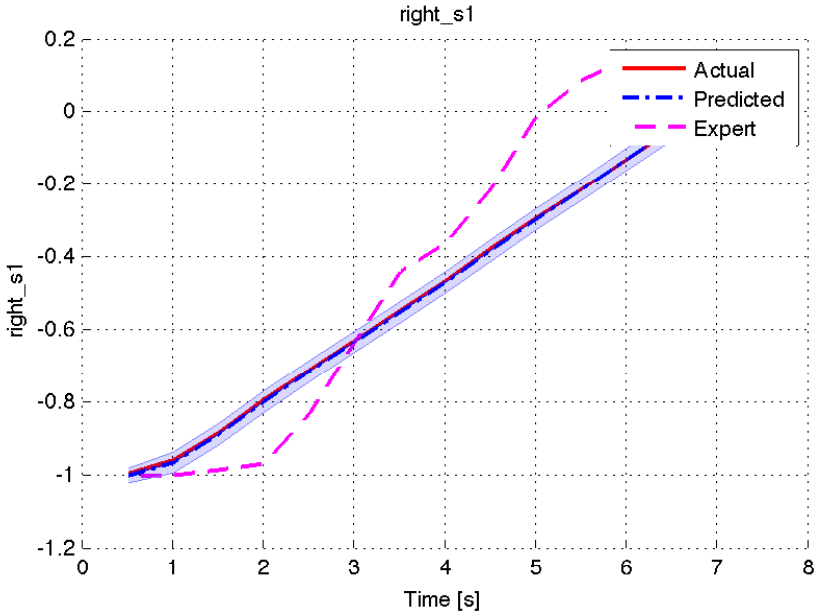

Figure B.9: Graph of angular position of *right_s*1, on the 15th iteration.

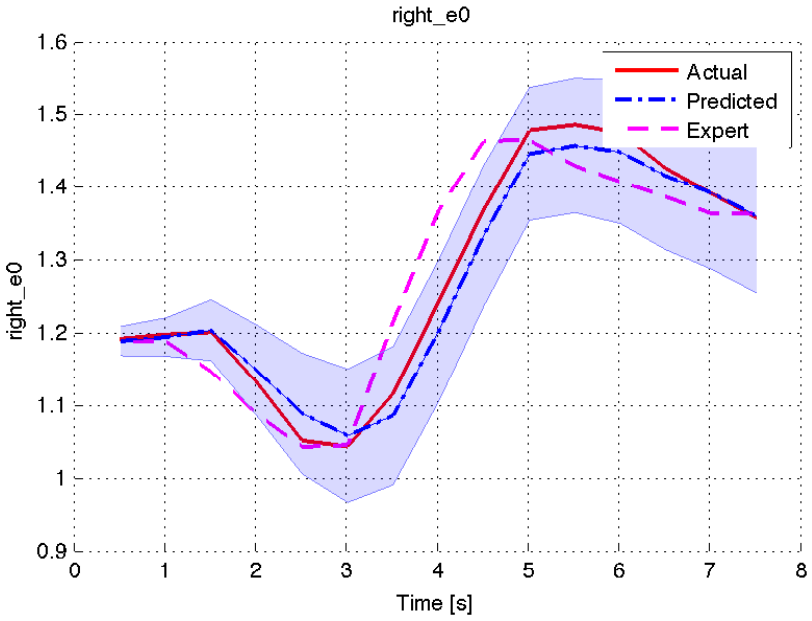

Figure B.10: Graph of angular position of *right_e*0, on the 15th iteration.

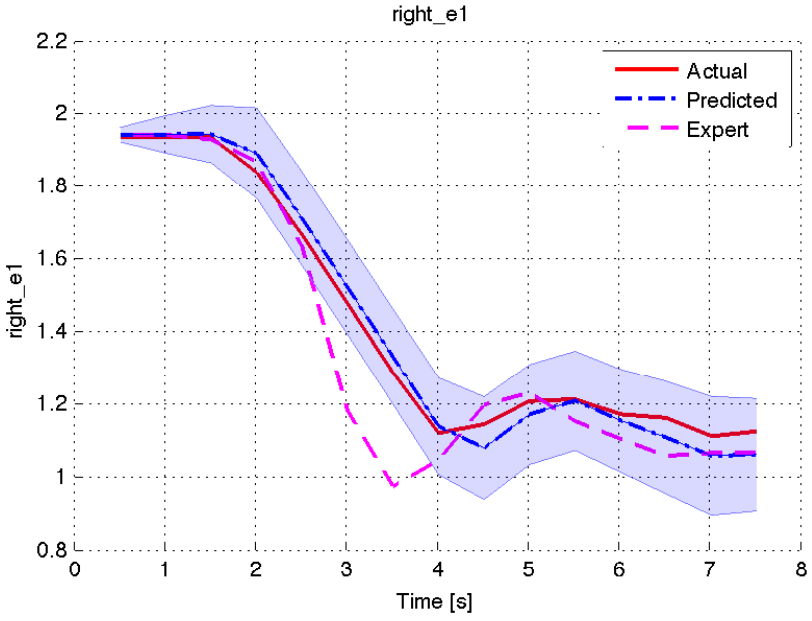

Figure B.11: Graph of angular position of *right_e*1, on the 15th iteration.

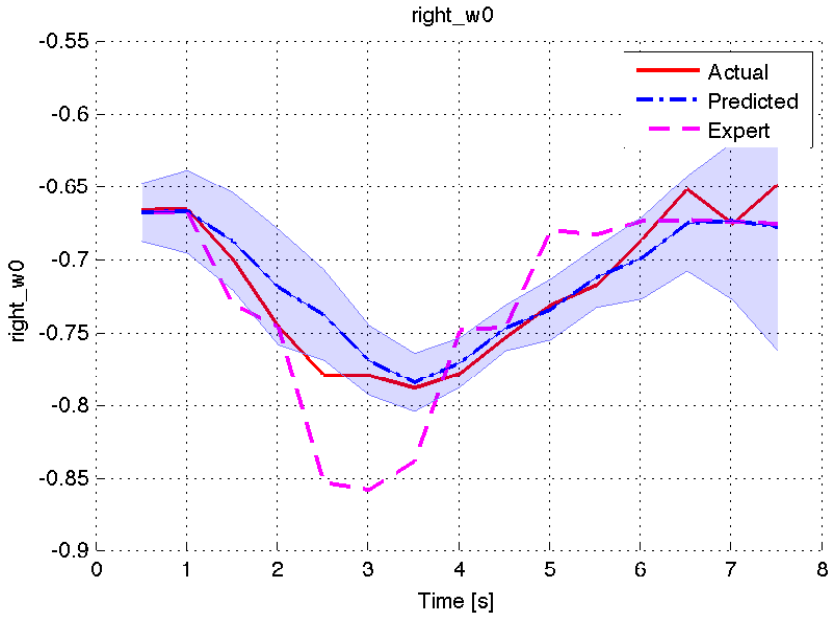

Figure B.12: Graph of angular position of *right_w*0, on the 15th iteration.

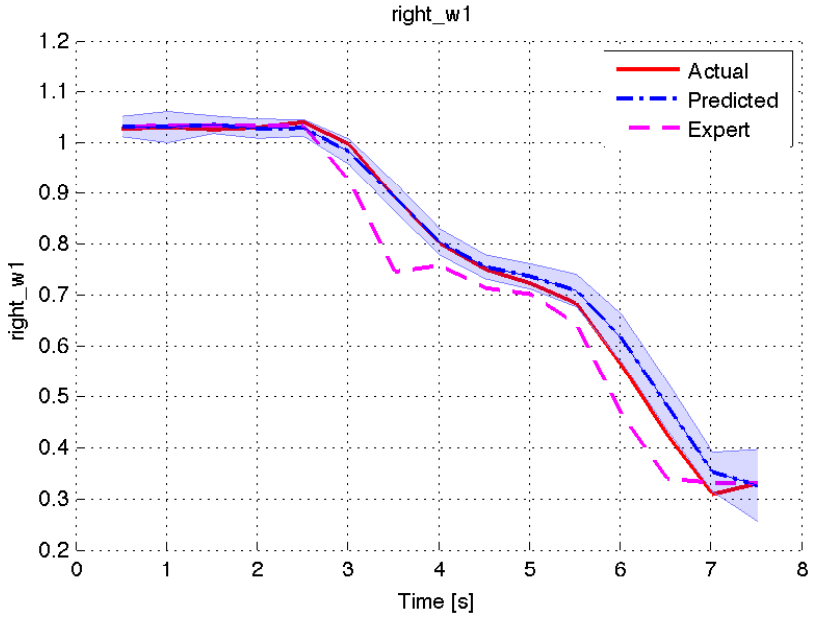

Figure B.13: Graph of angular position of *right_w*1, on the 15th iteration.

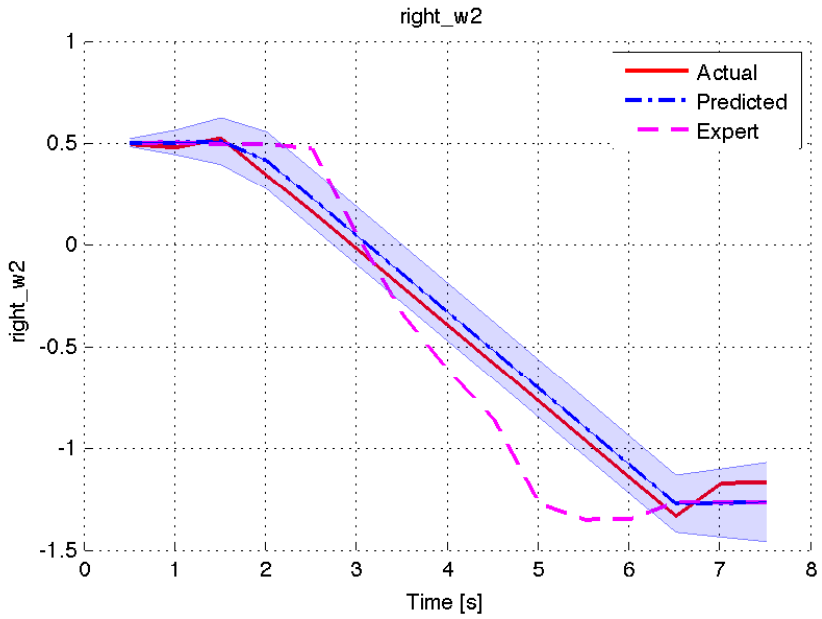

Figure B.14: Graph of angular position of *right_w2*, on the 15th iteration.