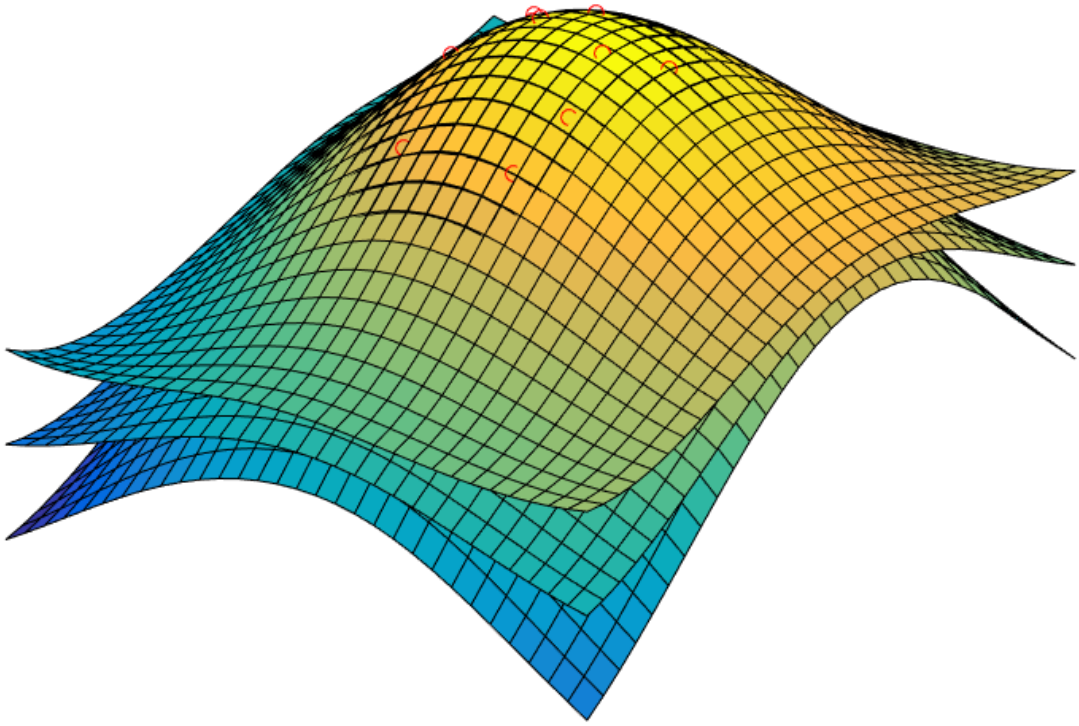
Segmented active reward learning

R. M. Olsthoorn

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

Segmented active reward learning

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

R. M. Olsthoorn

April 15, 2017

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

The use of robotic systems outside the branch of tasks currently common in industry requires the development of novel intelligent control methods.

In this thesis we will aim to improve on a recent machine learning method known as active reward learning. This method is able to teach a robotic system a task using human expert ratings on demonstrated robotic trajectories. Current implementations of this method use information collected from complete trajectories without regard for time specific features. This work will incorporate time segmentation as a new feature in two extensions of the active reward learning framework. In one extension, demonstrations are still rated over entire trajectories, leaving extraction of the important time segments to the learning algorithm. In the second extension we allow the expert to rate trajectory segments directly.

The two constructed algorithms are tested using a robot simulator. It is shown that these new methods are able to learn simple end effector tasks using reasonable numbers of queries and rollouts.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robotic systems are widely used in industry nowadays. The use of robots in various applications instead of humans can lead to improvements both in terms of costs and task performance. Robotic systems show their use mostly in applications that are so-called "4D tasks", tasks that are dangerous, dull, dirty or dumb [2]. A common property of the 4D tasks is that the robots are used in a repetitive setting: the same exact task is performed repeatedly. Examples of common robotic tasks can be found in material handling, such as welding or 3D printing. The human interaction with such a robot is often not allowed during operation. Usually, a cage is present to guarantee the safety of employees that are present. When using robots in this environment, it makes sense to program each robotic task manually. It allows the programmer to optimize the robot for each task. For example, a robot could be programmed to perform a task as fast a possible, for maximum production speed. Therefore, manual implementation of robotic tasks is still the most used practice in the industrial application of robots [3].

There are two aspects of the manual programming approach prone for improvement. First off, the process of programming a robotic task is expensive and time consuming. The cost of robotic experts makes the use of robots economically unfeasible for small series of tasks, especially tasks outside 4D area. Furthermore, the controllers that result from this process generalize very poorly [3]. In a different environment the robot will not perform as well, it has to be tuned again. Also, if the application of the robot operates in a dynamic environment, such as a human-robot interaction task, a more adaptive control strategy is needed.

Several control strategies have been developed that focus on the control systems operating in dynamic environments. One of these methods called Reinforcement Learning (RL), has interesting properties for robotic applications. Reinforcement learning methods can operate dynamic and noisy environments without the need of a model [4]. The RL method also applies optimization to achieve its pre-defined purpose [5]. An important element of the RL algorithm, the reward function, defines the assignment that the robot has to execute. In many applications, this reward function is defined by the programmer and tuned manually to produce the best result. This form of manual programming is still expensive and time consuming. Also, for many tasks, a measure of success is hard to define explicitly [6].

A simple example in Figure 1-1 illustrates how a RL algorithm converges to a different outcome than originally intended. The task, which is intended by the programmer, is for the end effector of a robot arm to move from an initial position towards a goal. In between it should pass through a point in space. As can be seen from the final trajectory, the end effector passes the viapoint with precision. However, the path towards the viapoint can be described as an inefficient detour. The problem in this example is that the reward function, which is the RL equivalent of the cost function used in optimal control, is defined too simple. A more elaborate reward function could also take into account the traveled distance of the end effector as a penalty function. However, if we would include this penalty function in the reward function we must also determine the relative importance between tracking accuracy and traveled distance by assigning weights to the two objectives. In order to tune the importance weights, we need to obtain multiple solutions of the RL problem, which is a time consuming procedure in practice.

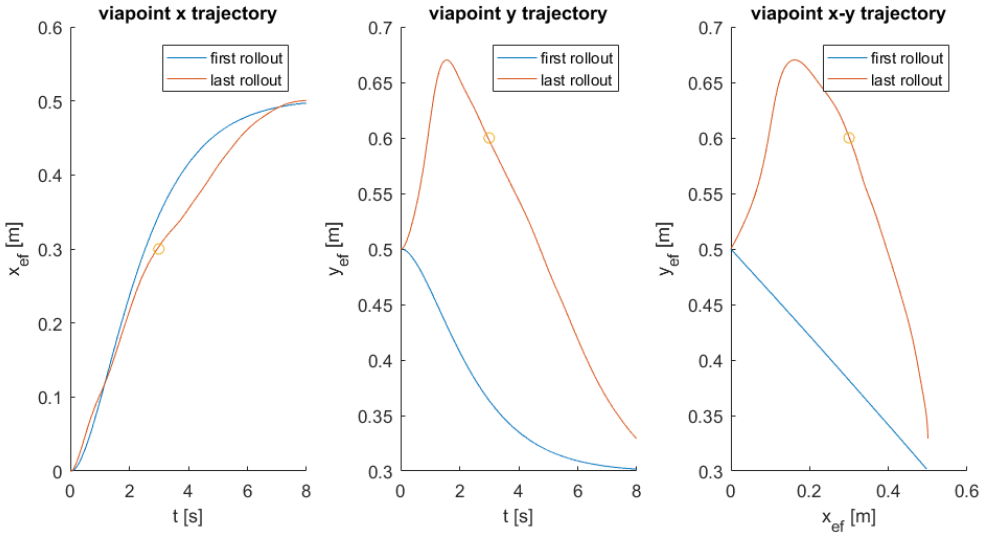


Figure 1-1: Resulting trajectories of a simple via point task.

A better alternative would be to teach the robot a task from another entity, called the expert, that knows exactly how the task needs to be performed. There are several approaches that accomplish machine learning from an expert. In the field of reinforcement learning, two approaches have been studied extensively: Apprenticeship learning and Inverse Reinforcement Learning (IRL). Both methods rely extensively on the availability of example (sub-) optimal trajectories, supplied by the expert. Inverse reinforcement learning differs from apprenticeship learning by the notion that IRL methods try to obtain the underlying reward function where apprenticeship learners try to obtain the optimal policy. The former class of methods is more interesting as an approach for adaptive control, because the obtained reward function can be used in different environments.

A more recent method is the Active Reward Learning (ARL) method [7]. The idea of active reward learning is similar to inverse reinforcement learning as this method also constructs a reward function, using the expert as source of information. In this setting, the expert is used to give feedback on robot trajectories. Active reward learning has been applied to a small number of robotic problems. The experiments in [7] show how ARL can be applied

to an inverse learning problem. In this experiment, the robot learns how to cross a viapoint in Cartesian space by controlling motor commands in joint space. As input for the reward function the robot is given the difference between the position of the end effector and the viapoint. As a result, a reward model containing a squared relation with respect to the viapoint error is returned. In a more advanced experiment, a robot hand is taught how to grab an unknown object, based on the calculated forces at the fingertips.

1-1 Task description

In this thesis we adopt the task description of the first experiment in [7], which is a viapoint task, as we would like to improve on this work. The viapoint task is described as follows: Given an initial position $\boldsymbol{x}_{\text{init}}$, goal position $\boldsymbol{x}_{\text{goal}}$ and a viapoint $\boldsymbol{x}_{\text{viapoint}}$ in Cartesian space, construct a path in Cartesian space such that the end effector of a robot arm travels from $\boldsymbol{x}_{\text{init}}$ to $\boldsymbol{x}_{\text{goal}}$, crossing $\boldsymbol{x}_{\text{viapoint}}$ at a specific point in time t_{viapoint}. An example of such a task is shown in Figure 1-2. In all experiments conducted, we assume the trajectories to be of fixed length t_N in discrete time.

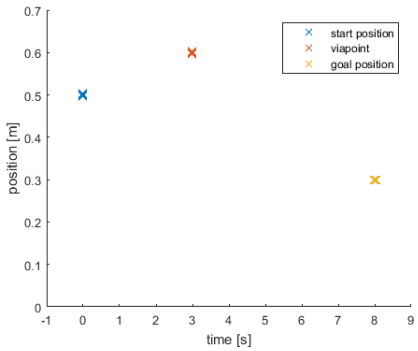


Figure 1-2: Viapoint task in one dimension.

A second task that we use in the experiments, which is very similar to the viapoint task, is a viaplane task. Similar to the viapoint task, during the execution of a viaplane task, the end effector of a robotic arm needs to travel from $\boldsymbol{x}_{\text{init}}$ to $\boldsymbol{x}_{\text{goal}}$ in Cartesian space. However, instead of crossing a single point in between $\boldsymbol{x}_{\text{init}}$ and $\boldsymbol{x}_{\text{goal}}$, the end effector has to track a pre-defined plane in a specific region in the time line. Such a task can be combined with a viapoint task to generate a more challenging goal for the robot arm to achieve, as is shown in Figure 1-3.

The combination of viapoints and viaplanes can be used to define various end effector movements. For example, we can teach a robot arm to move towards a known goal and avoid an area with obstacles. Also, we can teach the robot to approach the end goal from a certain angle. This technique can be used for example to teach the robot an insertion task.

In this thesis, a simulated 2 degrees-of-freedom robotic arm is used for conducting experiments. We use high level commands to interact with the robotic system as is illustrated in Figure 1-4. As can be seen in this graphic overview, the algorithm is used to define a path in Cartesian end effector space, so the reinforcement learning agent will define a path in x

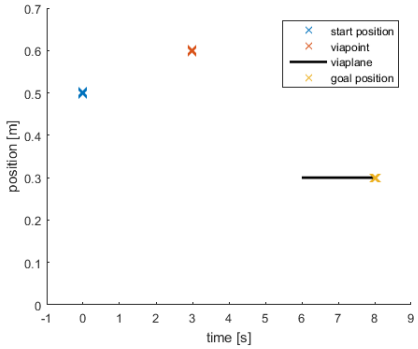


Figure 1-3: Viaplane in combination with viapoint task in one dimension.

and y coordinates. This trajectory is subsequently run on a simulated entity referred to as the robotic system. This robotic system contains a simulator of a robot arm as well as a low level joint controller. There is an inverse kinematics and a forward kinematics solver present in this entity as well in order to convert the input trajectory from end effector space to joint space and to convert the output trajectory back from joint space to end effector space. The output of this robotic system will be a trajectory in end effector space which will be evaluated by the reward model. There will be tracking errors between the input and output trajectory due to imperfect inverse and forward kinematics conversion and imperfect tracking of the joint controller. The algorithms created in this thesis regard the complete robotic system as a black box system, they can only observe the input and output trajectories.

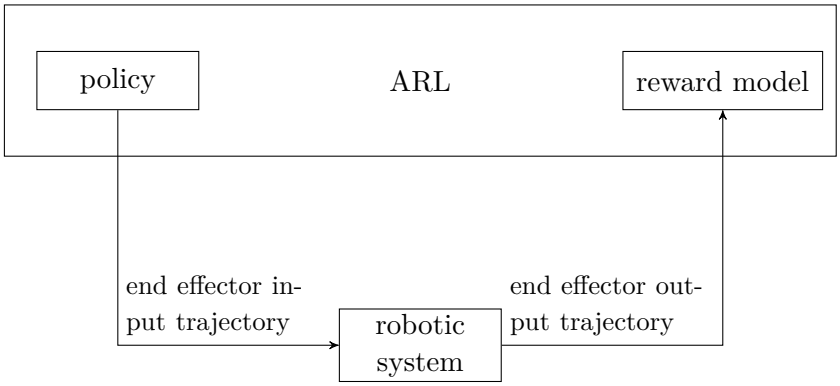


Figure 1-4: Overview of the interaction between the robot and the ARL algorithm.

1-2 Research goals

Although the ARL method has been proven to work in a practical example, there is room for more research. To be more specific, the input of the reward model constructed in [7] contains information about the complete trajectory of a rollout. In case of the viapoint experiment conducted in [7] the exact position of the viapoints and time slot at which the end effector should cross it is known by the reward model but this is usually absent in practical cases. In this thesis we assume that task specific information is unknown to the robot and should

be learned instead. In order to compensate for the lack of task specific information, time segmentation is implemented in the ARL algorithm. In this work we propose two segmented active reward learning methods, applied to learn end effector tasks purely on expert feedback.

The first proposed Segmented Active Reward Learning (SARL) algorithm achieves time segmentation by creating segment-specific groups of reward model inputs. Since the reward model proposed in [7] is able to distinguish relevant inputs from non-relevant inputs, this new method should be able to determine relevant time segments as well.

In the first proposed SARL method, we assume that the human expert is only able to give a rating over full trajectories. The amount of information that the expert gives is limited to one grade per complete trajectory. We could extend this framework such that the expert can give feedback on time segments of the trajectory. A more sophisticated adaptation on the ARL algorithm, which basically constructs a different reward model for each time segment, is needed to achieve this kind of time segmented learning. The hypothesis is that with this new source of expert information, the accuracy of the end effector trajectories will improve.

The purpose of this thesis is to show the results of two time segmented ARL algorithms. In this experiment we try to answer the following research questions:

- Can we use time segmented active reward learning to teach a robotic arm a simple end effector tasks described in Section 1-1?
- How many expert queries are necessary to achieve convergence for teaching a robot end effector tasks, described in Section 1-1, using time segmented active reward learning? How many expert queries are necessary if segment specific demonstrations are used compared to full trajectory demonstrations?
- How many roll outs are needed to teach a robot arm simple end effector tasks? Is the number of rollouts of the proposed SARL methods within practical limits?

1-3 Roadmap

In order to give an overview of the contents of this work, a roadmap is provided in Figure 1-5. The order of subjects is best explained when we take a brief look at the general formulation of reward learning. As shown in semi-mathematical Equation (1-1), active reward learning is essentially solving a double optimization problem. The main goal is to maximize the expert ratings through querying demonstrated rollouts. We can adjust the reward function in order to accomplish better results from the reinforcement learning agent. These results flow from the second optimizer, which is a straight-forward RL problem.

$$\underbrace{\max_{\text{reward function}} \text{expert} \left(\overbrace{\max_{\text{policy}} [\text{reward function(policy)}]}^{\text{Reinforcement learning (Chapter 2)}} \right)}_{\text{Reward learning (Chapters 3-5)}} \tag{1-1}$$

In Chapter 2, we will focus on the second optimizer, the forward RL method. We will review the RL methods that are applied to continuous systems as this is the application that we

are focusing on. One specific reinforcement learning method will be described in more detail, as this method will be used in Chapters 4-5 as well. After that, two different approaches to reward learning are explained in Chapter 3, namely inverse reinforcement learning and active reward learning. We will explain the difference between the two methods and discuss the advantages and disadvantages of both methods in detail. In Chapter 4, the active reward learning method is described in more detail. Important concepts of the active reward learning method such as the acquisition function and the reward model will be explained. After that, in Chapter 5, time segmented active reward learning is introduced. We will describe two novel variants of ARL in which we will use the concept of time segments. After a theoretical introduction to these new variants of ARL, we will show the results of this new method in Chapter 6 and compare them to the original ARL method. At last we will review the results of the research and show some recommendations for further research.

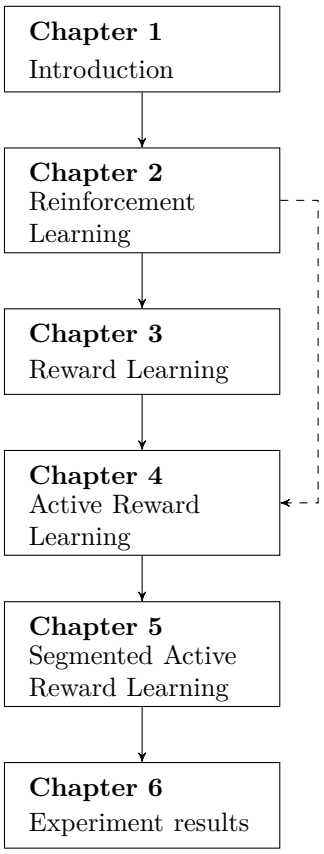


Figure 1-5: Roadmap of the report. The dashed line represents an a shortcut through the material.

Chapter 2

Reinforcement Learning

Reinforcement Learning (RL) is a machine learning method, developed in the 1980's [4]. The concept of RL is a combination of different methods in the field of artificial intelligence, adaptive- and optimal control. Reinforcement learning has successfully been applied in the field of artificial intelligence to teach computers to play chess and backgammon. In the field of control, reinforcement learning is a particular interesting method, but hard to realize in the field of robotics. The reason for this is that the state-space of robotic systems is continuous in multiple dimensions, where as the state-space of a game of chess is finite. The search space for optimal solutions is therefore of a completely different order. Since the RL framework assumes no model of the system (some methods incorporate model learning), applying RL on a robotic system can be described as model-free, adaptive, optimal control. However, the representative power of the RL framework comes at a cost in lack of tractability properties [8].

In this chapter we will outline the method of reinforcement learning as well as describe several reinforcement learning methods that are suitable for robotic applications.

In Section 2-1, a mathematical framework called the Markov Decision Process (MDP), which we will be using throughout is introduced. Several definitions used to describe the RL method, applied to robotic systems, are given as well. After that, in Section 2-2, several main approaches of reinforcement learning are elaborated upon. We will finish the Chapter with a specific reinforcement learning method called black-box policy improvement (PI^{BB}), in Section 2-3.

2-1 Markov Decision Process

The RL framework considers an intelligent robot, called the agent, that has to perform a task. In order to achieve the task, the agent can manipulate the environment by performing actions which will cause a transition to another state. Mathematically, the environment of the agent described above is called a Markov decision process. This process can be described as the following tuple: (X, U, T) [4]:

- X: A set of possible states that represent the dynamics of the system. It is assumed that the agent can sense the complete state.
- U: A set of possible actions that the agent can select at each time step.
- T: The state transition probability function $T : X \times U \times X \to \mathbb{R}$. For each state $\boldsymbol{x} \in X$ the probability of transforming to the next state $\boldsymbol{x}' \in X$ by inferring action $\boldsymbol{u} \in U$ is given by $T(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}')$.

Note that the transition of the agent from state $\boldsymbol{x}$ to $\boldsymbol{x}'$ is not influenced by states prior to state $\boldsymbol{x}$. This is called the Markov property.

In this work we assume that all trajectories to have the same length in time. As such, we can define a trajectory $\boldsymbol{\tau}$ as the ordered set of state-action pairs from the initial state to the final state: $\boldsymbol{\tau} = \{\{\boldsymbol{x}_0, \boldsymbol{u}_0, t_0\}, ..., \{\boldsymbol{x}_N, \boldsymbol{u}_N, t_N\}\}$.

2-1-1 Policy

The policy, in some areas referred to as the control law, is the function $\pi : X \to U$ that makes the decision for control action. In discrete-state problems, the policy chooses an action $\boldsymbol{u}$ from a fixed set of actions. If the MDP is continuous, the control action is calculated. A policy can be a deterministic function but it can be a stochastic as well.

In continuous-time reinforcement learning problems, a common practice is to use parameterized policies. Let us denote vector $\boldsymbol{\theta}$ as the policy parameter. The function that maps such a policy from action $\boldsymbol{x}$ to the control action $\boldsymbol{u}$ would now also be dependent on the parameter vector $\boldsymbol{\theta}$ as such: $\boldsymbol{u} = \pi(\boldsymbol{x}, \boldsymbol{\theta})$. The policy parameters allows us to directly change the policy.

2-1-2 Reward

In the RL framework, the reward function is used to reinforce desired behaviour of the agent. A large variety of reward functions is present in literature since each application has a different definition of desired behaviour. Some applications require a reward function that maps each state-action pair of trajectory $\boldsymbol{\tau}$ to a reward value $r(\boldsymbol{x}, \boldsymbol{u})$. The total amount of reward obtained, also called the *return* of a trajectory can be defined as:

$$R(\boldsymbol{\tau}) = \sum_{\{\boldsymbol{x},\boldsymbol{u}\} \in \boldsymbol{\tau}} r(\boldsymbol{x}, \boldsymbol{u}). \tag{2-1}$$

The return of a trajectory can also be calculated using the complete trajectory of trajectory segments, as will be shown in Section 4-2 and Section 5-2-1. Reinforcement learning algorithms are designed to maximize the expected return per trajectory. The expression for accumulated reward is very similar to the cost function used in optimal control algorithms, only difference being that the expression for accumulated reward should be maximized where as the cost function should be minimized.

In common feature in RL algorithms, is the use of a discount factor $\gamma \in (0, 1]$. This variable is used as such

$$R(\boldsymbol{\tau}) = \sum_{k=1}^{\infty} \gamma^k r(\boldsymbol{x}_k, \boldsymbol{u}_k),$$

where k denotes the discrete time index. Incorporating a discount factor is particularly useful to bound the possibly unbounded return of variable length trajectories but it also regulates the importance of future rewards. In this work we assume that all trajectories have the same length in time and that the reward at each time segment is equally important, so we use a $\gamma = 1$ for all experiments.

2-1-3 Value function

We continue by defining a useful definition for the accumulated reward that the agent obtains in time. The so-called value function or state-value function defines the expected return of an agent in state $\boldsymbol{x}$, when the optimal policy is used.

$$V(\boldsymbol{x}) = \max_{\boldsymbol{u}_i:\boldsymbol{u}_N} \mathbb{E}\left[\sum_{\boldsymbol{\tau}_{t_i}:\boldsymbol{\tau}_{t_N}} r(\boldsymbol{x}, \boldsymbol{u})\right]$$

2-2 RL learning algorithms

The Markov decision process is a very powerful tool to create adaptive learning methods. Several algorithms are developed for MDPs over the years. In general, we can establish a common optimization problem that all RL methods solve in a different manner:

$$\pi = \arg\max_{\pi} \mathbb{E}\left[R(\boldsymbol{\tau})\right]$$

The group of reinforcement learning algorithms can roughly be divided into three main groups [9]: actor-only, actor-critic and critic-only methods, where the actor denotes the policy and the critic denotes the value function.

2-2-1 Critic-only methods

In critic-only methods, the value function plays an important role. A critic-only algorithm consists of a method to learn the value function and a policy that chooses the control action in order to maximize this function. If the state-space and action-space of the MDP are finite, an approximation of the value for each state-action pair can be stored in computer memory. The values of each state-action pair can be updated using the temporal difference error. In that case, dynamic programming methods can be used as a policy. In continuous space the value function can be approximated. Also, discretizing a continuous state and action space is often done in critic-only algorithms. In critic-only methods, the value function can be updated either each time step or after each episode. Critic-only methods use an optimization algorithm in its policy. Each time step, the next control action is determined by optimizing

the estimate of the value function. This is the basis for the Q-learning [10] and SARSA [11] algorithms. Critic-only methods tend to work very well in finite-space MDP problem such as playing back-gammon [12]. In continuous state-space problems, such as robotics, critic-only methods are not often used. One of the main disadvantages of critic-only methods are their slow convergence speed.

2-2-2 Actor-critic methods

If both the value function and the policy function are learned by the agent we call the algorithm an actor-critic method. In these methods the critic is used to update the current policy, prescribed by the actor, by evaluating the performance of the agent.

In each iteration, two updates occur: a value function update and a the policy update. In the first update step, the value function of the previous state is updated using the reward that followed. After that, the policy parameters are updated using the value function. In finite space MDPs, the two updates can be performed after each time step, as shown in [4]. A more practical approach for robotic applications is episodic natural actor critic method [13]. In this particular method, the update is calculated after sampling a batch of rollouts, a common feature in many robotic RL methods.

2-2-3 Actor-only methods

Algorithms in the actor-only class of reinforcement learning are characterized by a policy learning algorithm. Typically, actor-only methods define the policy of the agent as a function with variable parameters, which can be tuned to optimize the expected return.

Actor-only methods do not use the definition of value function such as critic-only methods, to learn which actions result in high returns. Instead, direct optimization over the definition of the expected return is common practice. A group of actor-only methods called gradient methods relies on optimizing the accumulated reward, by approximating the gradient of the expected return with respect to the policy parameters. An example of a known policy gradient method is REINFORCE [14]. One major drawback of this approach is that the policy gradient tends to have a high noise variance.

A second class of actor-only methods, called reward weighted average methods, circumvents the problem of gradient variance. This class of methods update the policy after performing multiple different trajectories, also called rollouts, on the robot. After each batch of rollouts, the algorithm will update the policy based on which rollouts turned out to have the highest return. This particular class of methods have shown to deal reasonable well in the high dimensional continuous space of robotic systems. Several methods of this type have been developed both originating from reinforcement learning, as well as the field of stochastic optimal control (see Figure 2-1). Two main flavors of reward weighted average algorithms are relative entropy policy search (REPS [15]) and path integral approaches (PI^2[16]). Section 2-3 will describe a reward weighted, actor-only algorithm called black-box policy improvement (PI^{BB}) which will be used in the active reward learning algorithm.

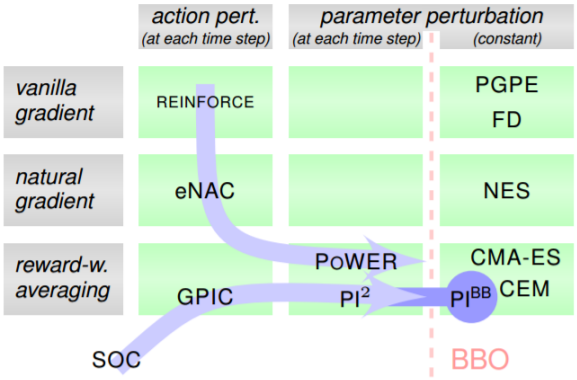


Figure 2-1: Overview of RL methods commonly used in robotics [1]. The blue arrows indicates progress over time.

2-3 Black box PI algorithm

The specific algorithm that we will use in the active reward learning framework is called black box policy improvement (PI^{BB}) [1]. This specific algorithm is a black box optimizer that uses reward weighted averaging for improving its policy. This particular algorithm is well suited to work with the active reward learning method for the following reason: The algorithm does not require a reward specified in each time step. Instead, the PI^{BB} algorithm only requires the return over complete trajectories for each sampled rollout. Learning a reward function for each part in state-space is much more difficult than learning the return over complete trajectories, since the expert feedback will be given over complete or segmented trajectories, not for each state.

2-3-1 Episode based learning

The PI^{BB} algorithm is an episode based method. A common feature among episode based methods is that the agents' policy does not change during an episode. In each iteration of the PI^{BB} algorithm a fixed number of episodes, also called rollouts, will be run or simulated, resulting in multiple trajectories $\boldsymbol{\tau}_k$, where $k = 1, ..., K$. We define $\mathcal{T}$ as the set of sampled rollouts $\{\boldsymbol{\tau}_1, ..., \boldsymbol{\tau}_K\}$. After sampling, the return of each rollout $\boldsymbol{\tau}_k$ is calculated and the new policy parameters $\boldsymbol{\theta}$ are calculated. This approach has proven to be more effective in robotics compared to traditional RL methods, in which an update is performed every time step [1].

2-3-2 Parameterized policy

Since the PI^{BB} method is an actor-only algorithm, we need to use an adaptive policy. The policy used in PI^{BB} is parameterized as $\pi(\boldsymbol{\theta}+\boldsymbol{\epsilon}_k)$. In this definition, $\boldsymbol{\theta}$ is the policy parameter vector and $\boldsymbol{\epsilon}_k$ is the exploration noise.

As mentioned in Section 2-3-1, the policy will be updated after sampling a batch of K rollouts, which means that the policy vector $\boldsymbol{\theta}$ will stay the same during sampling.

In order to explore the impact of changing policy parameters, exploration noise vector $\boldsymbol{\epsilon_k}$ is introduced as a direct addition to the policy parameters. The exploration noise $\boldsymbol{\epsilon}_k$ is different

for each rollout $\boldsymbol{\tau}_k$. Furthermore, $\boldsymbol{\epsilon}_k$ is a zero mean white noise vector with variance matrix $\boldsymbol{\Sigma}_\epsilon$. In order to regulate the exploration-exploitation trade-off, the variance of the exploration noise is decreased each iteration in a simulated annealing fashion using an annealing factor λ_ϵ.

In the experiments we will use a policy called Dynamic Movement Primitive (DMP) [17], a policy suited for learning trajectories with a pre-specified start- and end position. This policy will construct end effector trajectories as described in Section 1-1.

2-3-3 Reward weighting

After sampling, the return of each trajectory will be calculated using the return function $R(\boldsymbol{\tau})$. Secondly, the probability for each rollout is approximated using the return of each rollout. We assume that rollouts resulting in a high return value should be more probable than rollouts resulting in a low return value. In oder to approximate this probability, we first scale the return of each trajectory to obtain a measure of return relative with respect to the other rollouts in the batch of samples

$$\tilde{S}(\boldsymbol{\tau}_k) = \frac{-h(R(\boldsymbol{\tau}_k) - \min_{\boldsymbol{\tau}_i \in \mathcal{T}}(R(\boldsymbol{\tau}_i)))}{\max_{\boldsymbol{\tau}_i \in \mathcal{T}}(R(\boldsymbol{\tau}_i)) - \min_{\boldsymbol{\tau}_i \in \mathcal{T}}(R(\boldsymbol{\tau}_i)))},$$

where h is a tuning factor which determines how dense the probabilities of the different trajectories are distributed. The obtained value for relative return $\tilde{S}(\boldsymbol{\tau}_k)$ is then used to calculate the reward based probability of each rollout using the softmax operator as such:

$$P(\boldsymbol{\tau}_k) = \frac{e^{-\tilde{S}(\boldsymbol{\tau}_k)}}{\sum_{\boldsymbol{\tau}_i \in \mathcal{T}} e^{-\tilde{S}(\boldsymbol{\tau}_i)}}.$$

We will use the probability of each rollout to formulate the policy update. We can observe that trajectories resulting in a high return value used a highly successful exploration noise vector $\boldsymbol{\epsilon}_k$. To obtain a reward weighted average update we simply need to multiply $P(\boldsymbol{\tau}_k)$ with the exploration noise $\boldsymbol{\epsilon}_k$ for each sample in the batch of samples $\mathcal{T}$:

$$\delta\boldsymbol{\theta} = \sum_{\boldsymbol{\tau}_k \in \mathcal{T}} P(\boldsymbol{\tau}_k)\boldsymbol{\epsilon}_k.$$

In Algorithm 1, the pseudo code of the PI^{BB} method is given.

Algorithm 1 PI^{BB} pseudo code

initialize policy $\pi(\boldsymbol{x}, \boldsymbol{\theta}_0)$
initialize exploration noise and annealer $\Sigma_{\boldsymbol{\epsilon}}$, λ_ϵ
while not converged **do**
    for k = 1,...,K **do**
        Generate exploration noise
        $\boldsymbol{\epsilon}_k \sim \mathcal{N}(0, \Sigma_{\boldsymbol{\epsilon}})$
        Collect samples from the sample policy
        $\boldsymbol{\tau}_k = \{\boldsymbol{x}_i, \boldsymbol{u}_i = \pi(\boldsymbol{\theta}, \boldsymbol{\epsilon}_k)\}, i \in 1, ..., N$
        $\mathcal{T} = \{\boldsymbol{\tau}_1, ..., \boldsymbol{\tau}_K\}$
        Calculate return
        $R(\boldsymbol{\tau}_k) = \phi_{t_N} + \sum_{j=0}^{N-1} r_(\boldsymbol{x}_{t_j}, \boldsymbol{u}_{t_j}, t_j)$
    Calculate relative return
    $\tilde{S}(\boldsymbol{\tau}_k) = \frac{-h(R(\boldsymbol{\tau}_k) - \min_{\boldsymbol{\tau}_i \in \mathcal{T}}(R(\boldsymbol{\tau}_i)))}{\max_{\boldsymbol{\tau}_i \in \mathcal{T}}(R(\boldsymbol{\tau}_i)) - \min_{\boldsymbol{\tau}_i \in \mathcal{T}}(R(\boldsymbol{\tau}_i)))}$
    Calculate reward weighted probability
    $P(\boldsymbol{\tau}_k) = \frac{e^{-\tilde{S}(\boldsymbol{\tau}_k)}}{\sum_{\boldsymbol{\tau}_i \in \mathcal{T}} e^{-\tilde{S}(\boldsymbol{\tau}_i)}}$
    Calculate policy update
    $\delta\boldsymbol{\theta} = \sum_{\boldsymbol{\tau}_k \in \mathcal{T}} P(\boldsymbol{\tau}_k)\boldsymbol{\epsilon}_k$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \delta\boldsymbol{\theta}$
    Apply noise annealing
    $\boldsymbol{\Sigma}_\epsilon = \lambda_\epsilon \boldsymbol{\Sigma}_\epsilon$

Chapter 3

Reward learning

The reward function plays an important role in any Reinforcement Learning (RL) method. In many practical applications, programming and tuning the reward function is a critical design step and this is mostly done by an engineer or programmer. Over the years, methods have been developed that try to construct a reward function, using information from an expert. A well studied method called Inverse Reinforcement Learning (IRL), reconstructs a reward function based on expert example trajectories [18]. More recent work has been published in the field of active reward learning, which is based on human expert feedback [7].

In this chapter we will review reward learning methods that are studied over the years. There are important similarities in a number of reward learning methods, which will be highlighted.

In the first section, different methods to interact with an expert are explained. Two different methods of using the expert as a source of information are described. After that, the reward function will be revisited in Section 3-2. We continue by explaining the inverse reinforcement learning approach briefly in Section 3-3. In the last section we will outline the basic algorithm of Active Reward Learning (ARL).

3-1 Expert reward learning

In order to obtain a suitable reward function without manual programming, an expert can be used. This expert does not have to be a programmer, it can be a human or robot that knows how to execute a specific task very well. In literature, two methods to obtain a reward function are studied: learning by expert example and learning by expert feedback.

3-1-1 Example trajectories

Since the resulting policy of any RL agent is based on optimizing the reward function, examples of well executed trajectories contain information about the underlying reward function.

We can use an expert to provide us with (sub-) optimal examples trajectories to teach a movement to an RL agent. Such an example trajectory can be written as a sequence of state-action pairs. Usually, a batch of these trajectories is gathered: $\mathcal{T}_E = \{\boldsymbol{\tau}_1, ..., \boldsymbol{\tau}_m\}$.

There are several ways expert trajectories can be obtained in practice as described in [19]. Four categories of gathering expert trajectories can be distinguished:

Teleoperation An expert operates the robot via teleoperation. The measurements from the robot joints or end effector can directly be used as examples trajectories. No mapping is needed from the state-space of the teacher to the robot, which is the reason this method is often used. However, teleoperation requires that operating the robot be manageable, which can be problematic for some systems. For example, if we want to record joint motion on a robotic arm which happened to have more joints than a human arm.

Shadowing While an expert demonstrates a certain behavior, the robot tries to mimic the movement. As with teleoperation, the measurements from the robot can be used as example trajectories. In this approach, the quality of the expert examples is limited to the accuracy of the mimicing algorithm used.

Sensors on teacher The human expert demonstrates a specific movement with sensors attached to its body. This data must be mapped to the joint or end effector space of the robot, before it is used as example trajectory.

External observation An expert demonstrates a certain behavior without any sensors attached. The robot tries to extract the state/actions from external observations, such as a camera. In this case, a mapping must be made from the external sensors to the state-action space of the expert as well as a mapping from the state-action space of the teacher to the robot.

The obtained batch of expert trajectories $\mathcal{T}_E$ can be used for reward function extraction. This approach, known as inverse reinforcement learning, is described in more detail in Section 3-3.

3-1-2 Expert queries

As explained in the previous section, obtaining high quality example trajectories can be problematic in practice. However, there are more ways we can extract information from the expert as human experts are able to distinguish desired and undesired behavior. According to Miller [20], several studies have shown that the capacity for a human to judge different stimuli, the so called channel capacity, is between 1.6 and 3.9 bits for a two dimensional representation.

A safe option would be let the expert distinguish trajectories in a binary fashion, only giving the expert the options “better” or “worse” on a pair of rollouts. Since this setting operates below the channel capacity, the expert would always succeed. However, such a setup would only give one bit of information for two demonstrations, which is extremely low. In this work, demonstrations are rated in a scale of the expert’s choice, displaying the previous ratings in the background to maximize the expert’s channel capacity (see Section 6-1-2).

Instead of using a set of example trajectories, the active reward learning algorithm uses a dynamic set of demonstrated trajectories and their respective expert rating:

$$\mathcal{D} = \{\{\boldsymbol{\tau}_1, R_{E_1}\}, ..., \{\boldsymbol{\tau}_m, R_{E_m}\}\}$$

This approach has been studied in order to teach a grasping task to a KUKA lightweight arm [7]. Chapter 4 will describe this method in more detail.

3-2 Reward function structure

Before we describe the various methods that can achieve reward learning, we review the approximation techniques that are used to construct the reward function. Since we cannot consider an infinite space of possible reward functions, simplifications must be made regarding the structure of the reward function.

3-2-1 Discrete reward table

In discrete MDPs, there is the option of defining a reward value for each possible state of the MDP $r(\boldsymbol{x}_i) = r_i, \forall \boldsymbol{x}_i \in X$. These reward functions are agnostic about the underlying dynamics of the reward function [18] [21]. Usually, the resulting reward functions assign a high value for a small set of highly desirable state, whilst assigning zero or near zero reward for the rest of the state space.

3-2-2 Linear combination of features

When a reward value must be assigned for each state in an MDP, the number of optimization variables for a reward learning algorithm can grow out of proportion. For continuous state-space MDPs this approach is impossible. Besides that, the underlying dynamics for the reward function remain unknown after learning the reward function. Both disadvantages can be alleviated by the introduction of candidate reward functions, also referred to as *feature functions*. Inspired by the task at hand, several feature functions can be constructed as $\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{u})$, where each element of vector $\boldsymbol{\phi}$ is a different candidate reward function. In order to obtain an approximation of the true reward function, we define a weight vector $\boldsymbol{w}$ and construct the reward function as a linear combination of feature functions $r(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{u})$. Using a linear combination of feature functions and weights reduces the reward learning problem to finding the optimal weight vector $\boldsymbol{w}$. This approach is applied in most IRL algorithms, both in discrete and continuous state-space [22].

3-2-3 Non-linear function of features

Besides using a linear combination of features as a reward function, some IRL methods use non-linear function approximators. For example, Dvijotham uses adaptive radial basis functions in state-space to construct a value function. From this value function a reward function is derived using the Hamilton-Jacobi-Bellman equations [23].

Several works use Gaussian Processes (GPs) to model a non-linear relation between features and reward in different ways. For instance, Levine et al [24] use this model to learn a function between the features at a certain state and the reward at that state $r(\boldsymbol{x}, \boldsymbol{u}) = \mathcal{GP}(\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{u}))$. In discrete MDPs, it is possible to model a different reward function for each possible action in the action space. Following this approach, a GP for each action $\boldsymbol{u} \in U$ can be trained that maps states to rewards $r_{\boldsymbol{u}_i}(\boldsymbol{x}) = \mathcal{GP}(\boldsymbol{x})_{\boldsymbol{u}_i} \forall \boldsymbol{u}_i \in U$ [25].

Instead of modeling a reward for each state, it is also possible to model the a non-linear relation between the features and the return of complete trajectories. The active reward learning technique described in [7] uses a GP to learn such a relation $R(\boldsymbol{\tau}) = \mathcal{GP}(\boldsymbol{\phi}(\boldsymbol{\tau}))$, since in active reward learning the return of demonstrated trajectories is given by the expert.

3-3 Inverse reinforcement learning

In literature, learning from expert examples is shown in several different methods. Some methods that try to recover the relation between states and actions, based on expert examples, are known as apprenticeship learning methods. Applications used to demonstrate apprenticeship learning include a pendulum swing-up task [26], segway control [27], grasping tasks [28] and an egg flipping task [29]. One disadvantage of this approach is that the resulting policies do not generalize over changing environments.

If we want to teach a RL agent a specific task, a well defined reward function should be sufficient, regardless of the environment. Inverse reinforcement learning methods use the concept of apprenticeship learning to achieve reward learning by using expert demonstrations to extract a reward function.

3-3-1 IRL problem statement

The goal of inverse reinforcement learning is to create a reward function based on expert examples. This process can be divided into two parts: example acquisition (see Section 3-1-1) and reward function deduction. In general, the approach of the second part is to come up with a reward function such that the example trajectories are the optimal solution of a standard RL problem.

It can easily be observed that any reward learning problem, including inverse reinforcement learning and active reward learning, is ill-posed. Multiple reward functions exist that return an optimal reward for the set of examples $\mathcal{T}_E$. For example if we consider a reward function for which $r(\boldsymbol{x}, \boldsymbol{u}) = 0$ for each state and action, any trajectory in state-action space is an optimal trajectory and therefore this reward function is always a valid solution of the inverse reinforcement learning problem. It can also be observed that the solution of a RL problem is invariant under a reward function scaling. So for each reward function r there exists an infinite set of reward functions $r^* = c \cdot r$ which will share the same optimal policy.

3-3-2 Maximum margin planning

Several IRL algorithms have been developed over the years, all of which try to enforce a unique reward function as result. There is no standard formulation to describe the optimization

problem of IRL. Each method defines what it considers to be a "good" resulting reward function differently.

For instance the max-margin planning method tries to maximize the gap between the reward of the expert examples (the optimal solution) and all other possible policies [30]. In a discrete state-space MDP, this optimization problem can be solved using a gradient descent method. A more advanced method called LEARCH [31], by the same authors extends this method to incorporate non-linear reward functions as well. Applications using this approach include route planning for an autonomous vehicle, quadrupedal locomotion planning and a grasping task. All of these problems are solved as a planning problem in discrete state-space.

3-3-3 Bayesian IRL

Other IRL methods take a stochastic approach for defining a good reward function. In this group of methods, the expert examples are seen as evidence that the reward of these trajectories should be high. Using this line of thought we can construct probability distributions as such: $P(r|\mathcal{T}_E)$. This is the probability of reward function r being the "true" reward function given the set of expert examples $\mathcal{T}_E$. If we can calculate or approximate this probability, we can define the IRL problem as to maximize this probability with respect to the reward function $\max_r [P(r|\mathcal{T}_E)]$. In this class of methods, the reward function is modeled as a vector, containing a reward value for each state in discrete state. We can use Bayes' rule to expand the objective function, transforming it into a maximum a posteriori. This is the reasoning behind the Bayesian IRL method [21] and Preference Elicitation IRL [32]. The latter method is extended to a multi-task IRL algorithm [33].

3-3-4 Entropy based IRL methods

In contrast to Bayesian probability algorithms, entropy based IRL methods reverse the reasoning of the stochastic IRL problem. Instead of maximizing the probability of a reward function under condition of the examples, entropy based methods maximize the probability of the examples under condition of a reward function $P(\mathcal{T}_E|r)$. More specifically, the entropy of this distribution is maximized under certain constraints: $\max \log(P(\mathcal{T}_E|r))$ [34]. The reward model in this case is modeled as a linear combination of feature functions, $r = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x})$, so we are looking for the optimal $\boldsymbol{w}$. This optimization problem can be solved using the gradient of the cost function, which can be obtained using the visiting frequencies of each state in discrete state space per trajectory. An interesting extension to this method is relative entropy IRL [35]. This method minimizes the relative entropy between two probability distributions of $P(\boldsymbol{\tau})$ and $Q(\boldsymbol{\tau})$, where $P(\boldsymbol{\tau})$ is the new distribution under reward r and $Q(\boldsymbol{\tau})$ is the distribution of a baseline policy. There are some reasons to prefer the relative entropy method over the maximum entropy method, the most important being that relative entropy IRL is a model-free method.

3-3-5 Non-linear IRL

Besides Bayesian and entropy based methods, several other works have been published about IRL. These methods often use non-linear models to construct the resulting reward function.

For linearly solvable MDPs, an IRL method is developed that derives the reward function from the value function [23]. It is possible to learn a value function with adaptive radial basis functions and using the Hamilton-Jacobi-Bellman equations we can calculate the reward function accordingly. The inference method of this technique is very similar to maximum entropy IRL, since the probabilities of expert trajectories being taken by a RL agent is maximized: $\max_{\boldsymbol{w}} \log P(\mathcal{T}_E|\boldsymbol{w})$, where $\boldsymbol{w}$ are the parameters of the radial basis functions.

Other non-linear IRL methods primarily use Gaussian processes to model the reward function. In discrete MDPs, this is done in a Bayesian setting [25]. In this particular method, there exists a GP for each action in U. Each GP gives the reward of taking their corresponding action $\boldsymbol{u}$, given a state $\boldsymbol{x}$. The training points for the GP are constructed using a preference graph.

In continuous MDPs, no finite set of GPs can be constructed to represent the reward function. Instead, we can use feature functions and train a GP to map the features to rewards $r(\boldsymbol{x}, \boldsymbol{u}) = \mathcal{GP}(\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{u}))$ [24]. In order to construct a GP that is capable of doing this, we need a set of training points $\{\{\boldsymbol{\phi}(\boldsymbol{x}_1, \boldsymbol{u}_1), r_1\}, ..., \{\boldsymbol{\phi}(\boldsymbol{x}_m, \boldsymbol{u}_m), r_m\}\}$. As inputs for our training set we can take points along the example trajectories plus a few random points in state-action space. The corresponding rewards for these points $\{r_1, ..., r_m\}$ along with all the hyper parameters of the GP itself are the parameters that we optimize for. The inference distribution is similar to the maximum entropy method. We can use the LFBG method to solve this optimization problem.

3-3-6 IRL in robotics

Not many IRL methods have been implemented in robotic applications yet. The vast majority of publications use discrete grid worlds to compare their respective results with other methods. However, there are interesting applications outside the field of robotics where inverse reinforcement learning showed nice results. A comparative study over IRL methods, showed that IRL can recover the reward of decision making problems very well by learning the strategies of table tennis players [36].

There are multiple reasons for the lack of applicability in continuous applications. First off, the vast majority of IRL methods require solving the forward RL problem for every iteration, which is either an intense computational burden in simulators. If there is no model or simulator available, such an approach is unfeasible because real-life rollouts are very costly to perform, both in terms of hardware (which can break) as in computation time (rollouts are very slow). One IRL method is an exception to this rule: the PI^2 variant of maximum entropy IRL [37].

The second disadvantage, which is apparent in most IRL methods, is the need of a model. Usually, modeling the environment of a robot is costly, moreover the environment can change over time. Also, by using a model for learning the reward, one of the biggest advantages of RL is canceled out.

The last disadvantage of the IRL approach is inherent to its use of expert examples. In real life applications, expert examples are often imperfect or incomplete, only a sub optimal policy is available. For most IRL methods, the example trajectories are considered as the correct behaviors. However, only a notion of what is the correct behavior limits the information for

the agent. In many tasks, examples of failed attempts can be at least as valuable if the RL agent learns how to actively avoid this behavior.

3-4 Active reward learning

As mentioned before in Section 3-1, the active reward learning method is based on learning a return function from expert ratings. In this method, a non-parametric model is used to map the feature functions $\boldsymbol{\phi}$ to reward values, using the expert ratings as training samples. Another important aspect of active reward learning with respect to inverse reinforcement learning is that in active reward learning, both the return function as well as a (sub) -optimal policy is learned in hybrid fashion.

3-4-1 Reward model

The return function used in active reward learning is modeled as a Gaussian process, a non-parametric statistical model [38]. Gaussian processes can be used to learn non-linear relations, in case of active reward learning, to learn a relation between feature functions and return: $R(\boldsymbol{\tau}) = \mathcal{GP}(\boldsymbol{\phi}(\boldsymbol{\tau}))$. We will use the expert ratings obtained in demonstrations as learning samples for the reward model. The GP reward model can give an estimate for the return of non-demonstrated trajectories, which we need for learning the policy, as well as an indication of the variance for this estimate. The latter is particularly handy for the ARL algorithm to determine which rollouts are interesting for expert querying.

3-4-2 Forward learning method

Since the reward model only gives a return according to a complete trajectory, this has implications for the forward RL method that we can use in combination with active reward learning. In fact, only black-box policy learning techniques can be used within active reward learning [7].

Chapter 4

Active reward learning

The active reward learning method [7] is a relatively recent machine learning algorithm designed to learn a return function using expert feedback. In this chapter, the active reward learning algorithm will be explained in detail.

In Section 4-1, we will first present the basic ARL algorithm, to give a good overview of this method. After that, we will specify how the different parts of this algorithm work in detail. In Section 4-2 we explain the reward model of the ARL method in more detail. The statistical inference model that we use, the Gaussian Process (GP), will be introduced after that in Section 4-3. We will elaborate upon the details of using this inference method to our purpose as well. We will finish this chapter in Section 4-4 by explaining how the query acquisition method works.

4-1 Basic ARL algorithm

In Algorithm 2, the basic algorithm of active reward learning is illustrated. As can be seen in the pseudo-code, the algorithm starts by initializing a basic policy and an initial reward model. Several methods exist to initialize the initial policy, such as random initialization, zero initialization or using prior knowledge. In this work we choose to use zero initialization, which means that all policy parameters in $\boldsymbol{\theta}_0$ are zero. In order to initialize the reward model we sample a number of rollouts using the initial policy and demonstrate result to the expert. We collect the set of rated rollouts in the initial demonstration set $\mathcal{D}_0$ which will be the basis for the initial reward model.

After that, an iterative procedure begins which include three steps: sampling, reward update and policy update. In the first step, a number of rollouts are sampled (line 6). This procedure is similar to lines 6-9 in Algorithm 1. Several different trajectories are produced, based on a policy parameter $\boldsymbol{\theta}$ using a different exploration noise vector $\boldsymbol{\epsilon}_k$ for each rollout. The resulting trajectories $\{\boldsymbol{\tau}_1, .., \boldsymbol{\tau}_K\}$ are collected in a set of samples $\mathcal{T}$.

In the second part of the iteration, lines 8-17, we investigate if any of the sampled rollouts in $\mathcal{T}$ are worth demonstrating. As will be described in Section 4-4, we will use an acquisition

function to determine which rollouts are interesting for demonstration. This acquisition function will return a high value for interesting trajectories. If the highest obtained acquisition value is above a certain threshold λ_a we will ask the expert to rate this rollout. Subsequently we will update the reward model and repeat the procedure until there are no more rollouts interesting enough to demonstrate.

In the last step of the iteration, lines 19-21, the return of each trajectory in $\mathcal{T}$ is calculated according to the reward model. After that a policy update is executed according to our forward reinforcement learning algorithm (see Algorithm 1 lines 13-18).

Algorithm 2 Basic ARL pseudo code

```
1: initialize policy π(θ_0)
2: initialize reward model D = D_0
3:
4: while not converged do
5:     Sample K rollouts
6:     T = {τ_1, ..., τ_K}
7:
8:     FindNominee = true
9:     while FindNominee do
10:        Nominated outcome:
11:        τ_+ = arg max_{τ∈T} u(φ(τ)) (See Section 4-4)
12:        if τ_+ ∉ D ∧ u(φ(τ_+)) > λ_a then
13:            Demonstrate τ_+ to expert. Retrieve R_{E_+}
14:            D = D ∪ {φ(τ_+), R_{E_+}}
15:            Optimize GP hyper parameters (See Section 4-3-3)
16:        else
17:            FindNominee = false
18:
19:    Calculate return trajectories T
20:    R(τ) = GP(φ(τ)|D), ∀τ ∈ T
21:    Update policy π(θ) (See Section 2-3)
```

4-2 Reward model

The reward model is a very important entity in the active reward learning method. It is used to describe a mapping between a trajectory $\boldsymbol{\tau}$ and its return $R(\boldsymbol{\tau})$. We would like this mapping to approximate the expert ratings over a trajectory as close as possible $R(\boldsymbol{\tau}) \approx R_E(\boldsymbol{\tau})$.

In Figure 4-1, the layout of the reward model according to [7] is illustrated. As can be seen, the reward model will not use the information present in trajectory $\boldsymbol{\tau}$ the directly. Instead, we will define a number of reward features $\phi(\boldsymbol{\tau})$, which can be seen as candidate reward functions. The outcomes of all reward features are collected in vector $\boldsymbol{\phi}$ which is subsequently used as an input for a Gaussian process (see Section 4-3). The Gaussian process approximates the return $R(\boldsymbol{\tau})$ of trajectory $\boldsymbol{\tau}$ based on the outcomes $\boldsymbol{\phi}$ of the feature functions. Using feature functions reduces the dimensionality of the inputs of the GP, which is necessary to make the

reward learning problem feasible. In Chapter 5 we will extend the overall layout of the reward model to include time segmentation into the reward model.

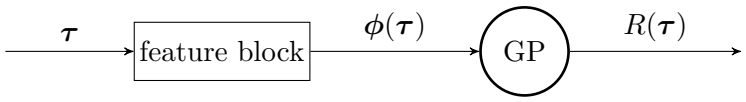


Figure 4-1: Reward model layout viewed schematically.

4-3 Gaussian process

We will use a Gaussian process [38] to identify the mapping between reward features $\boldsymbol{\phi}$ and the return over a trajectory. We assume that there is a relation f between the feature outcomes $\boldsymbol{\phi}$ and the return over the trajectory $R(\boldsymbol{\tau})$. We can only observe this mapping by demonstrating a certain trajectory $\boldsymbol{\tau}$ and obtaining a rating for this rollout by the expert:

$$R_E(\boldsymbol{\phi}(\boldsymbol{\tau})) = f(\boldsymbol{\phi}(\boldsymbol{\tau})) + \epsilon_E,$$

where R_E is the experts' return and f is the unknown return function. This mapping is disturbed by expert rating noise ϵ_E, which is considered to be a zero mean Gaussian white noise with standard deviation σ_E.

For notation simplicity, we will refer to $\boldsymbol{\phi}(\boldsymbol{\tau})$ as $\boldsymbol{\phi}$. In the demonstration set $\mathcal{D}$, we have a number of demonstrated trajectories with their respective feature outcomes and ratings. We can collect the feature outcomes of all trajectories in $\mathcal{D}$ to form a matrix $\boldsymbol{\Phi}_m = \begin{bmatrix}\boldsymbol{\phi}_{m_1} \cdots \boldsymbol{\phi}_{m_n}\end{bmatrix}$. Also, there is a set of input inference points $\boldsymbol{\Phi}_*$ at which we would like to know the return. Similarly, we will refer to an expert rating $R_E(\boldsymbol{\phi})$ simply as R_m. The set of measured ratings present in $\mathcal{D}$ can be collected into a vector $\boldsymbol{R}_m = \begin{bmatrix}R_{m_1} \cdots R_{m_n}\end{bmatrix}^T$.

In order to identify the mapping described above, a Gaussian process is used, which is a statistical non-parametric model. The mathematically correct description of a GP is as follows:

Definition 1. *A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution [38].*

However a more intuitive definition is to describe it as a *distribution over functions.* From this distribution, the most likely function, called the posterior mean function, can be evaluated at inference points which gives us an approximation of the mapping at those points. Besides providing the most likely value at some inference points, GPs can also provide the approximated variance at these inference points, expressed in the posterior covariance function.

In order to construct this distribution, a number of assumptions have to be made. First off, we have to design a prior covariance function in which we will specify what class of functions we consider and which functions to rule out. Optionally, we could define a prior mean function which tells the GP which function within the considered set is the most likely.

Finally we need a set of training points, in our case the set of demonstrations $\mathcal{D}$. Based on the prior assumptions and the set of training points, the posterior distribution can be calculated (see Section 4-3-2).

4-3-1 Prior functions

The design of a GP is determined for the most part by the choice of prior functions. In the construction of the prior mean and prior covariance function we incorporate all prior knowledge about the mapping that we would like to identify with the GP.

Prior covariance function The covariance function, also known as the *kernel*, that we choose determines what class of functions we consider likely by the mapping that we are looking for and eliminates all other possible functions. Since we do not have any prior knowledge of the return function, we cannot be too restrictive in our choice. If we choose the squared exponential covariance function, we still have a broad class of function under consideration, namely all differentiable functions. The squared exponential covariance function is defined as follows:

$$k(\boldsymbol{\phi}, \boldsymbol{\phi}') = \lambda_f^2 \exp\left(-\frac{1}{2}\left(\boldsymbol{\phi} - \boldsymbol{\phi}'\right)^T \boldsymbol{\Lambda}_\phi \left(\boldsymbol{\phi} - \boldsymbol{\phi}'\right)\right) \tag{4-1}$$

The covariance function relates two input vectors $\boldsymbol{\phi}$ and $\boldsymbol{\phi}'$ to a measure of correlation. In case of the squared exponential function, we can easily observe that input vectors that are close to each other obtain a high value.

Equation 4-1 also contains a few constants, namely λ_f^2 and matrix $\boldsymbol{\Lambda}_\phi$. These constants are called hyper parameters. The so called signal variance, λ_f^2, reflects how much variance is present in the mapping itself. Setting a high value for λ_f^2 indicates that we expect a high level of variation in the function.

Matrix $\boldsymbol{\Lambda}_\phi$ is a diagonal matrix, whose elements represent length scales.

$$\boldsymbol{\Lambda}_\phi = \begin{pmatrix} \lambda_{\phi_1}^{-2} & 0 & \cdots & 0 \\ 0 & \lambda_{\phi_2}^{-2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_{\phi_n}^{-2} \end{pmatrix}$$

There is a length scale λ_{ϕ_i} present in $\boldsymbol{\Lambda}_\phi$ for every input dimension. The value of each λ_{ϕ_i} indicates how much we expect that particular input dimension to have influence on the mapping.

Let us assume that we have two sets of input vectors stored in a matrices $\boldsymbol{\Phi}_m$ and $\boldsymbol{\Phi}_*$. The covariance matrix of according to $\boldsymbol{\Phi}$ would look as such:

$$\mathbf{K}(\boldsymbol{\Phi}_m, \boldsymbol{\Phi}_*) = \begin{bmatrix} k(\boldsymbol{\phi}_{m_1}, \boldsymbol{\phi}_{*_1}) & \cdots & k(\boldsymbol{\phi}_{m_1}, \boldsymbol{\phi}_{*_{n^*}}) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{\phi}_{m_n}, \boldsymbol{\phi}_{*_1}) & \cdots & k(\boldsymbol{\phi}_{m_n}, \boldsymbol{\phi}_{*_{n^*}}) \end{bmatrix} \tag{4-2}$$

In order to simplify notation we will refer to $\mathbf{K}(\boldsymbol{\Phi}_m, \boldsymbol{\Phi}_*)$ as $\mathbf{K}_{m*}$. Furthermore we will refer to its transpose variant as $\mathbf{K}(\boldsymbol{\Phi}_*, \boldsymbol{\Phi}_m) = \mathbf{K}_{*m} = \mathbf{K}_{m*}^T$. We also need a covariance matrix which correlates the individual measurement points with each other: $\mathbf{K}(\boldsymbol{\Phi}_m, \boldsymbol{\Phi}_m) = \mathbf{K}_{mm}$. The last covariance matrix we need is one that correlates the inference points with each other: $\mathbf{K}(\boldsymbol{\Phi}_*, \boldsymbol{\Phi}_*) = \mathbf{K}_{**}$.

Prior mean function Besides choosing a prior covariance function, a prior mean function also has to be chosen. The prior mean function indicates what function we find to be the most likely. As we do not have any knowledge of the relation between the features and the return, we cannot specify any function to be the most likely a priori. This can easily be specified by setting the prior mean function as a zero function: $\mu(\boldsymbol{\phi}) = 0$.

Prior measurement noise variance The last prior element we need to determine is the measurement noise variance. In case our application, this hyper parameter directly represents the expert rating noise σ_E. As before, we do not take into account any prior knowledge about this value, since we do not know anything about the expert beforehand. However, as we will see in Section 4-3-3, we can approximate this hyper parameter on the fly, using the set of demonstration data we collect during learning.

4-3-2 Posterior functions

Now that we have determined our prior mean and covariance function, we can construct the posterior function distribution. We assume for now that we have the correct hyper parameters. In order to compute a posterior distribution, we must apply Bayes' rule:

$$P(\text{Hypothesis}|\text{Evidence}) = \frac{P(\text{Evidence}|\text{Hypothesis}) \cdot P(\text{Hypothesis})}{P(\text{Evidence})}$$

Where the left side term, also called the posterior, is the distribution we want to obtain. In the nominator, two terms are present: the likelihood and the prior. The denominator term is called the marginal likelihood.

Let us assume that we have a set of demonstrations $\mathcal{D} = \{\boldsymbol{R}_m, \boldsymbol{\Phi}_m\}$ at our disposal and we would like to know the posterior distribution of some set of inference points $\{\boldsymbol{R}_*, \boldsymbol{\Phi}_*\}$. According to Bayes' rule, we should solve the following:

$$P(\boldsymbol{R}_*|\boldsymbol{\Phi}_*, \boldsymbol{R}_m, \boldsymbol{\Phi}_m) = \frac{P(\boldsymbol{R}_m, \boldsymbol{\Phi}_m|\boldsymbol{R}_*, \boldsymbol{\Phi}_*) \cdot P(\boldsymbol{R}_*, \boldsymbol{\Phi}_*)}{P(\boldsymbol{R}_m, \boldsymbol{\Phi}_m|\boldsymbol{\Phi}_*)}$$

According to Bayes rule we can thus merge the prior assumptions with our measurement data to obtain a posterior distribution. If we assume that our measurement data is recorded with a zero mean Gaussian measurement noise, the posterior is again a Gaussian distribution

$$P\left(\begin{bmatrix}\boldsymbol{R}_m \\ \boldsymbol{R}_*\end{bmatrix}|\boldsymbol{\Phi}_*, \boldsymbol{R}_m, \boldsymbol{\Phi}_m\right) \sim \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_m \\ \boldsymbol{\mu}_*\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_{mm} & \boldsymbol{\Sigma}_{m*} \\ \boldsymbol{\Sigma}_{*m} & \boldsymbol{\Sigma}_{**}\end{bmatrix}\right).$$

As can be seen, the posterior of a Gaussian process provides an estimate of the inference points, expressed in $\boldsymbol{\mu}_*$ as well as an estimate of the accuracy, expressed in covariance matrix $\boldsymbol{\Sigma}_{**}$. The values for $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ can be expressed in terms of the prior functions and the measurement values

$$\begin{bmatrix} \boldsymbol{\mu}_m \\ \boldsymbol{\mu}_* \end{bmatrix} = \begin{bmatrix} \mathbf{K}_{mm}(\mathbf{K}_{mm} + \boldsymbol{\Sigma}_E)^{-1}\boldsymbol{R}_m \\ \mathbf{K}_{*m}(\mathbf{K}_{mm} + \boldsymbol{\Sigma}_E)^{-1}\boldsymbol{R}_m \end{bmatrix} \tag{4-3}$$

$$\begin{bmatrix} \boldsymbol{\Sigma}_{mm} & \boldsymbol{\Sigma}_{m*} \\ \boldsymbol{\Sigma}_{*m} & \boldsymbol{\Sigma}_{**} \end{bmatrix} = \begin{bmatrix} \mathbf{K}_{mm}(\mathbf{K}_{mm} + \boldsymbol{\Sigma}_E)^{-1}\boldsymbol{\Sigma}_E & \boldsymbol{\Sigma}_E(\mathbf{K}_{mm} + \boldsymbol{\Sigma}_E)^{-1}\mathbf{K}_{m*} \\ \mathbf{K}_{*m}(\mathbf{K}_{mm} + \boldsymbol{\Sigma}_E)^{-1}\boldsymbol{\Sigma}_E & \mathbf{K}_{**} - \mathbf{K}_{*m}(K_{mm} + \boldsymbol{\Sigma}_E)^{-1}\mathbf{K}_{m*}, \end{bmatrix} \tag{4-4}$$

where $\boldsymbol{\Sigma}_E$ is a diagonal matrix expressing the expert noise variance $\boldsymbol{\Sigma}_E = \sigma_E^2 I$.

4-3-3 Hyper parameter optimization

So far, we assumed that we knew the true values of each hyper parameter we use. In general, this is not the case. The choice of hyper parameter value is important when using GPs for inference. If the values of the hyper parameters are not chosen carefully, the data fit can turn out be really poor. For some choices of hyper parameters, the data fit can also be too strict. In that case, the prediction on inference points have poor results as well because the GP is overtrained.

There are several hyper parameters we need to determine: λ_f, $\boldsymbol{\Lambda}_\phi$ and σ_E. For simplicity, we will write these hyper parameters as $\boldsymbol{\xi}$.

Since we cannot calculate the "true" hyper parameters, we look at which combination of hyper parameters is the most likely, based on the set of training points in $\mathcal{D}$. We can express the likelihood of $\boldsymbol{\xi}$, as an optimization problem

$$\max_{\boldsymbol{\xi}} P(\boldsymbol{\xi}|\boldsymbol{R}_m, \boldsymbol{\Phi}_m).$$

Using Bayes' rule we can write this as probability as follows:

$$P(\boldsymbol{\xi}|\boldsymbol{R}_m, \boldsymbol{\Phi}_m) = \frac{P(\boldsymbol{R}_m|\boldsymbol{\xi}, \boldsymbol{\Phi}_m) \cdot P(\boldsymbol{\xi}|\boldsymbol{\Phi}_m)}{P(\boldsymbol{R}_m|\boldsymbol{\Phi}_m)}. \tag{4-5}$$

Since the marginalization term in the denominator does not depend on $\boldsymbol{\xi}$, we can simply ignore it. In the nominator we find a likelihood term $P(\boldsymbol{R}_m|\boldsymbol{\xi}, \boldsymbol{\Phi}_m)$ and a prior probability $P(\boldsymbol{\xi}|\boldsymbol{\Phi}_m)$. If we examine the likelihood term, we find that it equals the posterior distribution of the GP using $\boldsymbol{\xi}$ as hyper parameter. Finally, if we look at the prior probability $P(\boldsymbol{\xi}|\boldsymbol{\Phi}_m)$, we first observe that we can ignore $\boldsymbol{\Phi}_m$, since the probability of our hyper parameters being likely does not depend on the measurement inputs. If we assume the probability for each set of hyper parameters $P(\boldsymbol{\xi})$ is equal, this term can be treated as a constant in our optimization problem. We can thus express our posterior distribution as follows:

$$P(\boldsymbol{\xi}|\boldsymbol{R}_m, \boldsymbol{\Phi}_m) \propto \frac{1}{\sqrt{|2\pi(\mathbf{K}_{mm} + \boldsymbol{\Sigma}_E)|}} \exp\left(-\frac{1}{2}\boldsymbol{R}_m^T(\mathbf{K}_{mm} + \boldsymbol{\Sigma}_E)^{-1}\boldsymbol{R}_m\right)$$

Maximizing over this expression directly is difficult, due to the exponential term. However, taking the logarithm of this expression does not change the optimization problem. Doing so results in the following:

$$\log(P) \propto -\frac{n_m}{2}\log(2\pi) - \frac{1}{2}\log|\mathbf{K}_{mm} + \mathbf{\Sigma}_E| - \frac{1}{2}\boldsymbol{R}_m^T(\mathbf{K}_{mm} + \mathbf{\Sigma}_E)^{-1}\boldsymbol{R}_m \tag{4-6}$$

The first term in (4-6) is a normalization constant. Because this term does not depend on $\boldsymbol{\xi}$, we can ignore it during optimization. The last term is called the data fit, which denotes how well our hyper parameters can explain the observed data. We could easily maximize this term by giving $\mathbf{K}_{mm} + \mathbf{\Sigma}_E$ extremely high values. This comes down to assuming so much noise that every kind of measurement fits within the model. To prevent this from happening during optimization, the second term penalizes high values for $\mathbf{K}_{mm} + \mathbf{\Sigma}_E$. This term is called the complexity term. If we combine the data fit term and the complexity term into our optimization function, we are able to obtain $\boldsymbol{\xi}$ for which the data fit is as high as possible with a low risk of overfitting.

We can maximize the above mentioned optimization function using a gradient descent approach. Since we chose a prior covariance function which is differential with respect to its hyper parameters, we can take the gradient of our cost function analytically. To be more specific, we use the Polak-Ribiere flavour of conjugate gradient descent [39] in combination with a line search method to optimize for $\boldsymbol{\xi}$.

4-4 Demonstration acquisition

An important aspect of the ARL method is the decision whether or not the reward model needs to be improved by querying the expert. In order to sufficiently learn the return function, enough expert queries are necessary, but we would like to keep the number of expert queries to a minimum, as it would be too costly to learn movements. In each iteration a batch of rollouts is sampled (see Algorithm 2). If there are rollouts present in this new batch that are interesting for the reward model, we should demonstrate this rollout for the expert. The ultimate goal of active reward learning is to reach the optimum of an unknown return function by strategically selecting points of function evaluation (expert queries). This problem is also known as Bayesian optimization.

The decision whether or not to query the expert is captured in the acquisition function $u(\boldsymbol{\phi}(\boldsymbol{\tau}))$. The acquisition function maps the features of a rollout to a number which indicates how interesting this particular rollout is for querying: $u : \boldsymbol{\tau} \rightarrow \mathbb{R}$. According to this value we can decide whether or not to demonstrate a particular rollout. In case of active reward learning, we will only demonstrate rollouts whose acquisition value is higher than a certain threshold, provided that we did not already query that particular rollout.

The GP provides us with some useful information regarding the return of new rollouts. First off, the GP provides a "best estimate" of the return of a rollout in the form of the mean function: $\mu(\boldsymbol{\phi}(\boldsymbol{\tau}))$. Besides that, the GP also provides an estimate of its own variance: $\sigma(\boldsymbol{\phi}(\boldsymbol{\tau}))$. We can interpret the variance as a confidence level of the accuracy of the model.

A number of different acquisition functions have been proposed in literature in the field of Bayesian optimization [40].

Probability of Improvement (PI) This acquisition function chooses the rollout that has the highest probability of improvement, according to a normal probability distribution [41]. If we denote $\mu^+ = \max_{\boldsymbol{\tau} \in \mathcal{T}} \mu(\boldsymbol{\phi}(\boldsymbol{\tau}))$, the probability of improvement can be defined using the normal distrubtion: $P(\mu(\boldsymbol{\phi}(\boldsymbol{\tau})) \geq \mu^+) = \Phi\left(\frac{\mu(\boldsymbol{\phi}(\boldsymbol{\tau})) - \mu^+ - \xi}{\sigma(\boldsymbol{\phi}(\boldsymbol{\tau}))}\right)$. Where the variable ξ denotes an exploration parameter and Φ denotes the normal cumulative distribution function.

Expected Improvement (EI) Instead of maximizing the probability that the cost function will improve, the expected improvement focuses how much the cost function will improve. This subtle difference results in an acquisition function that is a little less greedy in its search towards optima than PI. If we define $d = \mu(\boldsymbol{\phi}(\boldsymbol{\tau})) - \mu^+ - \xi$, we can work out an expression for expected improvement as such:

$$u(\boldsymbol{\phi}(\boldsymbol{\tau})) = \begin{cases} d\Phi(d/\sigma(\boldsymbol{\phi}(\boldsymbol{\tau}))) + \sigma(\boldsymbol{\phi}(\boldsymbol{\tau}))\phi(d/\sigma(\boldsymbol{\phi}(\boldsymbol{\tau}))) & \text{if } \sigma(\boldsymbol{\phi}(\boldsymbol{\tau})) > 0 \\ 0 & \text{if } \sigma(\boldsymbol{\phi}(\boldsymbol{\tau})) = 0 \end{cases} \tag{4-7}$$

Where the function $\Phi(\cdot)$ and $\phi(\cdot)$ denote the CDF and the PDF of the normal distribution.

Upper confidence bound (UCB) The upper confidence bound acquisition function uses the mean and standard deviation directly to define the acquisition value: $u(\boldsymbol{\tau}) = \mu(\boldsymbol{\phi}(\boldsymbol{\tau})) + \sqrt{v\beta}\sigma(\boldsymbol{\phi}(\boldsymbol{\tau}))$. Where v denotes a hyper parameter and β is used as a learning rate variable.

GP Hedge The GP Hedge acquisition function is built upon a portfolio of different acquisition functions. It basically combines the results of PI, EI and UCB in order to give the best estimate of $u(\boldsymbol{\tau})$. This method, along with the above mentioned methds work well in the field of Bayesian optimization. However, in the next section a specific acquisition function, designed for ARL, will be explained.

4-4-1 Expected policy divergence

The above mentioned acquisition functions work well for Bayesian optimization problems. However, the active reward learning differs slightly from Bayesian optimization problems. This difference has been exploited to create a novel acquisition function designed for active reward learning: Expected Policy Divergence (EPD) [7]. If we look at the Bayesian optimization acquisition functions, one assumptions holds for all: the main purpose is to optimize the objective function (return function in our case). In case of active reward learning, we are not primarily interested in the return function but also in the resulting policy. Taking the policy into account makes sense when designing an acquisition function. We should not demonstrate any rollouts if the expected effect on the policy is very small.

In Figure 4-2 a visual representation of the steps taken in expected policy divergence is shown. We start with a set of trajectories $\mathcal{T}$, one of which, $\boldsymbol{\tau}$, we would like to evaluate in the acquisition function. Furthermore, we have a set of demonstrated trajectories $\mathcal{D}$ on which the reward model is based and a policy π.

First off we calculate the return of each trajectory in $\mathcal{T}$ according to the current reward model. We can collect the feature outcomes of all trajectories in $\mathcal{T}$ in matrix $\boldsymbol{\Phi}(\mathcal{T})$ and

use these outcomes as inference points for the GP. Furthermore, we will use the following notation $\mathcal{GP}(\mathbf{\Phi}(\mathcal{T})|\mathcal{D})$ to describe the calculation of the return, using the features of set $\mathcal{T}$ as inference points and the demonstrations in set $\mathcal{D}$ as training points. We will denote the vector of resulting returns from the reward model $\mathcal{GP}(\mathbf{\Phi}(\mathcal{T})|\mathcal{D})$ as $\tilde{\boldsymbol{R}}$. After obtaining the return for each trajectory, we can use the update rule of the reinforcement learning agent to calculate the resulting policy, which we will call $\tilde{\pi}$.

After evaluating the unmodified policy update, we investigate at what happens if trajectory $\boldsymbol{\tau}$ is queried. If we evaluate the GP for this particular rollout $\mathcal{GP}(\boldsymbol{\phi}(\boldsymbol{\tau})|\mathcal{D})$ we receive $\mu(\boldsymbol{\tau})$ and $\sigma(\boldsymbol{\tau})$, the posterior mean and variance. The posterior mean $\mu(\boldsymbol{\tau})$ can be interpreted as a "best estimate" of the return while the variance can be interpreted as the "confidence" that we have that this is the correct return. It also tells us what rating we expect the expert to give trajectory $\boldsymbol{\tau}$. From this information we determine two sigma points: $s_+ = \mu + \sigma$, $s_- = \mu - \sigma$. We can extend the current set of demonstrations with one of these sigma points to obtain a new, extended reward model. After extending the reward model we evaluate the extended GP for each trajectory in $\mathcal{T}$ to obtain the return vector $\boldsymbol{R}^* = \mathcal{GP}(\mathbf{\Phi}(\mathcal{T})|\mathcal{D}^*)$. We can also calculate the resulting policy according to the new return values, denoted by π^*. Now that we have the two different policies we want to compare, we calculate Kullback-Leibler divergence (see Section 4-4-1) over the resulting policies π^* and $\tilde{\pi}$. Note that we repeat this procedure twice, since we have two sigma points to evaluate and take the average result as final value for $u(\boldsymbol{\tau})$.

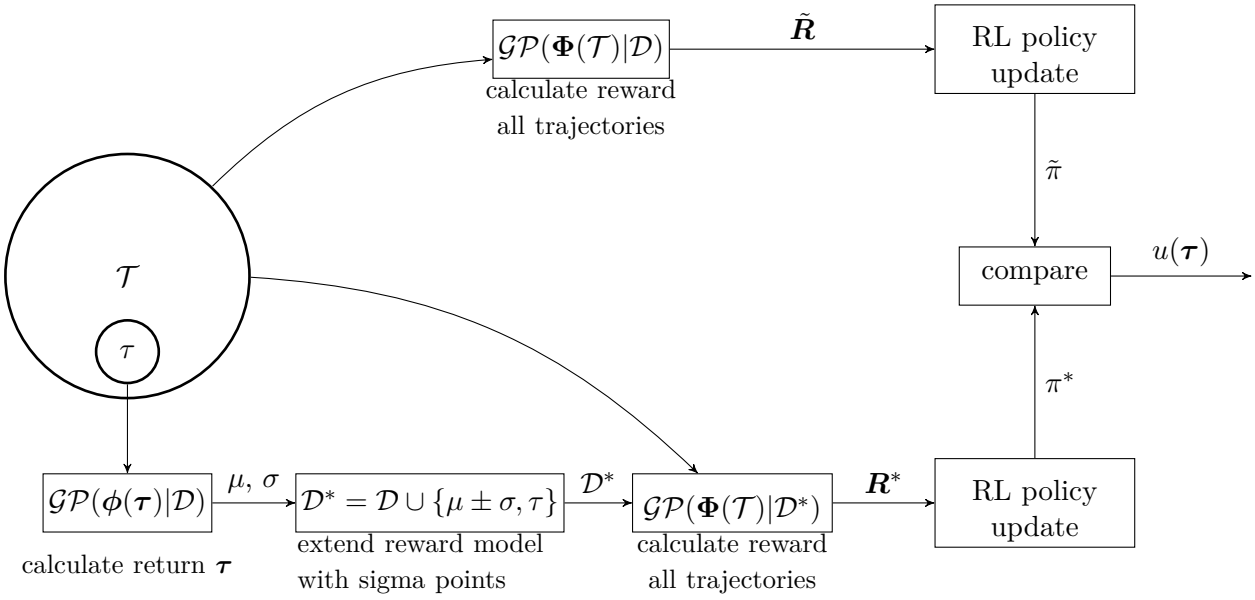


Figure 4-2: Expected policy divergence viewed schematically

Policy comparison Since we use a stochastic policy, it makes sense to compare the two resulting policies π^* and $\tilde{\pi}$ using a stochastic measure. A commonly used method for comparing two stochastic variables is the Kullback-Leibler divergence [42]. This measure represents the relative entropy between two probability distributions. We would like to maximize the expected value of the KL divergence between π^* and $\tilde{\pi}$:

$$\mathop{\mathbb{E}}_{P(R|\boldsymbol{\tau},\mathcal{D})} D_{\mathrm{KL}}(\pi(\boldsymbol{\theta})^*||\tilde{\pi}(\boldsymbol{\theta})) = \iint \pi^*(\boldsymbol{\theta}) \log\left(\frac{\pi^*(\boldsymbol{\theta})}{\tilde{\pi}(\boldsymbol{\theta})}\right) d\boldsymbol{\theta} dR.$$

Unfortunately, we cannot solve this integral in closed form. Instead we have to resort to sample based methods. However, we should avoid Monte-Carlo sampling to solve this particular problem as sampling is a very expensive operation in our application. Instead we will use the limited set of samples we already obtained. We could for example use the policy samples $\boldsymbol{\theta}_i$ that were used in sampling the rollouts in $\mathcal{T}$, to compute a sample-based approximation:

$$\mathrm{KL}(\pi^*(\boldsymbol{\theta})||\tilde{\pi}(\boldsymbol{\theta})) \approx \frac{1}{N}\sum_{i=1}^{N} \frac{\pi^*(\boldsymbol{\theta}_i)}{\pi(\boldsymbol{\theta}_i)} \log \frac{\pi^*(\boldsymbol{\theta}_i)}{\tilde{\pi}(\boldsymbol{\theta}_i)}.$$

In this approximation, we denote π as the original policy, where the samples were drawn from. We use importance sampling to correct for the sampling bias. This approximation can be numerically unstable, especially when the number of parameters in $\boldsymbol{\theta}$ is high (e.g. greater than 20).

Another alternative is to assign a weight γ_i each sample in $\mathcal{T}$ and calculate the Kullback-Leibler divergence directly over the weights. We denote $\tilde{\gamma}_i$ as the weight used for trajectory $\boldsymbol{\tau}_i$ in the policy update. Similarly, we denote γ_i^* as the weight for trajectory $\boldsymbol{\tau}_i$ in the alternative policy update. The PI^{BB} algorithm already provides such a weight for us, which is given in Equation (2-3-3).

Combining the weights of γ^* and $\tilde{\gamma}$ leads to the following approximation of the Kullback-Leibler divergence:

$$\mathrm{KL}(\pi^*(\boldsymbol{\theta})||\tilde{\pi}(\boldsymbol{\theta})) \approx \sum_{i=1}^{N} \gamma_i^* \log \frac{\gamma_i^*}{\tilde{\gamma}_i}.$$

4-5 Summary

In this chapter we have described the active reward learning method [7] in detail. The ARL algorithm achieves reward learning and policy in a hybrid setting by updating both the reward model and the policy each iteration.

In order to achieve reward learning we use a Gaussian process, a regression technique that is both able to estimate a function value as well as the variance of its estimate. A set of feature functions must be created to reduce the dimensionality of the GP input.

Every iteration, the acquisition function is evaluated for each sampled rollout, in order to determine which samples are interesting for demonstration. In each evaluation of the acquisition function we obtain the variance of the reward estimate from the GP. This variance is then used to calculate whether we expect the policy of the reinforcement learning agent to change if we would demonstrate the respective sample. Only samples that obtain a high acquisition value are demonstrated.

After updating the reward model, the Gaussian process is evaluated for every sampled rollout to calculate the return. After that, the policy of the reinforcement learning agent is updated according to the PI^{BB} update rules.

In Chapter 5, we will adapt the algorithm described in this chapter by including trajectory segmentation. Two new algorithms will be derived, both able to distinguish relevant segments of the trajectory. In the first adaptation of active reward learning, the segmentation will be accomplished by assigning a group of feature functions for each segment. In the second algorithm, a separate GP will be constructed for each different time segment that we define. Using different GPs for each time segment allows for segment specific expert ratings.

Chapter 5

Segmented active reward learning

In this chapter we will extend the active reward learning algorithm by including trajectory segmentation. Furthermore, we will design two novel algorithms able to learn tasks defined in end effector space purely based on expert feedback. The first Segmented Active Reward Learning (SARL) algorithm implements trajectory segmentation by creating a group of features for each defined segment. The resulting groups of features are used as input for a single Gaussian process, which should be able to distinguish important segments from unimportant ones. In the second algorithm we extend this algorithm even further by constructing a different GP for each defined segment. By using multiple GPs in a reward model we can implement segment specific demonstrations.

This chapter is divided into two main parts. In Section 5-1, a novel adaptation of active reward learning will be described. This section explains how we can implement trajectory segmentation by creating separate groups of feature functions in order teach simple end effector tasks introduced in Section 1-1. In the second part of this chapter (Section 5-2), we built upon the algorithm described in Section 5-1 to create another novel active reward learning algorithm. We will use the same feature segmentation presented in Section 5-1 and extend the reward model to include multiple GPs. Furthermore, we describe an acquisition function that selects interesting trajectory segments for demonstration.

5-1 Trajectory segmentation using feature grouping

In this section we will introduce an adaptation to the ARL method, designed to learn end effector tasks described in Section 1-1. We will introduce trajectory segmentation based on time segments in Section 5-1-1. After that, we introduce the feature functions designed for our application in Section 5-1-2. In this section a new layout of the reward model will be shown as well, introducing groups of features for each specific time segment. Apart from the feature functions and the reward model layout, the basic algorithm of active reward learning does not have to be adapted for this new setup to work, so we can reuse Algorithm 2 (Section 4-1) as well as the EPD acquisition function (Section 4-4-1).

5-1-1 Trajectory segmentation

One property of the end effector tasks described in Section 1-1 is that not all parts of the trajectory are important. In order to exploit this feature, we introduce time segmented trajectories. We assume that we cannot use any prior knowledge indicating which points in time are relevant. Instead we will create a specific number of segments n_s, and divide the trajectories into equidistant parts in time (see Figure 5-1).

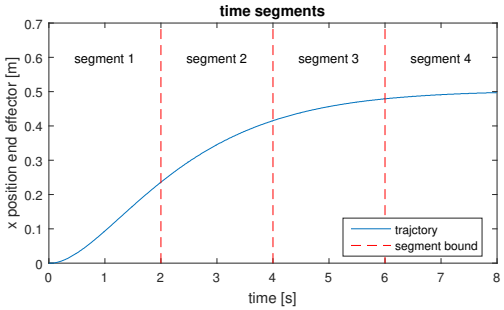


Figure 5-1: Example of equidistant time segmentation.

We can distinguish n_s different trajectory segments denoted as $\boldsymbol{\tau}^{(1)}, ..., \boldsymbol{\tau}^{(n_s)}$.

5-1-2 Segment specific feature functions

Feature functions are candidate reward functions that we use to simplify the reward learning problem, as discussed in Section 3-2. In order teach the robot end effector tasks, we need to design a number of feature functions that give relevant input information to the GP. We assume that we cannot program task specific parameters into our feature functions. For example if the task of a robot arm is to cross a specific point, we cannot program a feature that would measure the squared error with respect of that point. Instead, we use feature functions that give a more global representation of the end-effector trajectory in Cartesian space. For each trajectory segment, the following feature functions are considered:

- Mean x position of the end effector
- Mean y position of the end effector
- (Optionally) Variance x position of the end effector
- (Optionally) Variance y position of the end effector

We will conduct part of our experiments using only the mean positions of the end effector as feature functions. In some experiments we will include the variance of the end effector position as well. With this extended set of features we should be able to reward viaplane tasks with more accuracy.

In Figure 5-2, the layout of the adapted reward model is illustrated. As can be seen in this block diagram, we create separate groups of feature functions, each group linked to a specific

time segment. In each group, features describing the end effector position are present. The result is a set of functions that describes the global position of the end effector in each time segment. We will use only one GP to calculate the return of a trajectory which takes all groups of feature functions as its input. However, this GP should be able to distinguish relevant inputs from non relevant inputs via hyper parameter optimization (see Section 4-3-3), so it should be able to distinguish relevant time segments from non-relevant time segments as well.

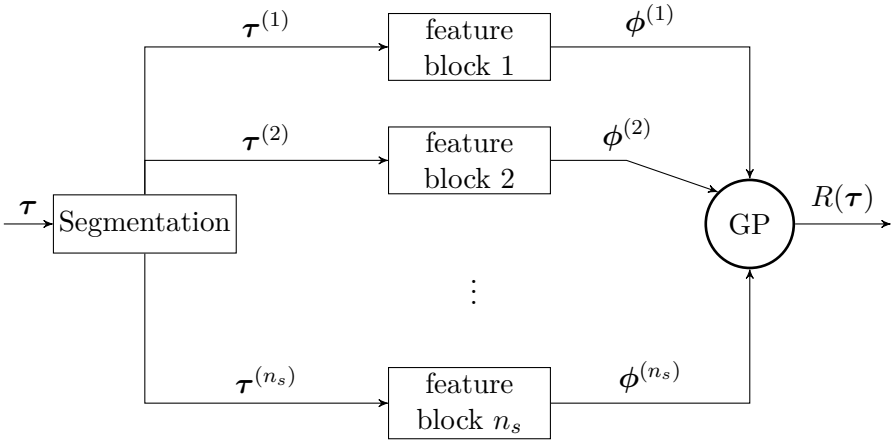


Figure 5-2: Segmented reward model

5-2 Trajectory segmentation using multiple Gaussian processes

The previous section described how we can implement trajectory segmentation using a different reward model layout. In this section we will extend the reward model presented in Section 5-1-2 to implement segment specific expert ratings.

There are two main reasons why it is interesting to investigate segment based expert queries. First off, we are able to acquire more specific information about the reward function. In most robotic tasks there are parts that are critical for the performance and parts that are not that important. By querying only the parts of the task that are critical for performance, we effectively apply a filter for non-relevant information for the reward model. In other words, we can make our reward model more focused. Another benefit of segment specific demonstrations can be found when we consider the expert. Human experts only have a limited capacity (also called bandwidth [20]) for judging performances. In our case, the human expert will make judgement errors if the number of demonstrations becomes too big. Segment specific demonstrations can be an important factor for increasing the bandwidth of the expert. This is especially interesting when we consider a multi-task objective, such as a double viapoint or a viapoint/viaplane objective. In this scenario, the expert can focus on increasing the performance for one task per demonstration, provided that the different tasks are separated into different time segments.

In Section 5-2-1 a new reward model layout that is needed to implement segment specific expert ratings is described. After that, we will revisit the acquisition function described in Section 4-4-1. In order to implement segment specific expert ratings we need to change the

acquisition function from a function over complete trajectories to a function over trajectory segments. Finally, the pseudo code of this new variant of ARL is elaborated upon.

5-2-1 Multi GP reward model layout

In order to implement segment specific expert ratings, we need a reward model that is adaptive for each defined time segment. Figure 5-3 illustrates a reward model layout that is adaptive for each time segment such that segment specific expert ratings can be achieved. As can be seen in this overview, we create groups of feature functions for each defined time segment similar to the reward model shown in Section 5-1-2. However, instead of using only one GP to learn the return function, we use several. For each time segment we will construct a different GP, taking only the feature functions for that specific time segment as input argument. Each segment specific GP also holds a segment specific set of demonstrations $\mathcal{D}^{(i)} = \{\{\boldsymbol{\tau}_1^{(i)}, R_{E_1}^{(i)}\}, ..., \{\boldsymbol{\tau}_n^{(i)}, R_{E_n}^{(i)}\}\}$. Note that not every segment needs to have the same number of demonstrations. Typically, the time segments which are critical for the objective hold a much larger set of demonstration compared to other time segments. Each time-specific GP will produce an estimate of the reward for its time segment upon evaluation. In order to obtain the total return for the trajectory, we simply add up the rewards of all the time segments.

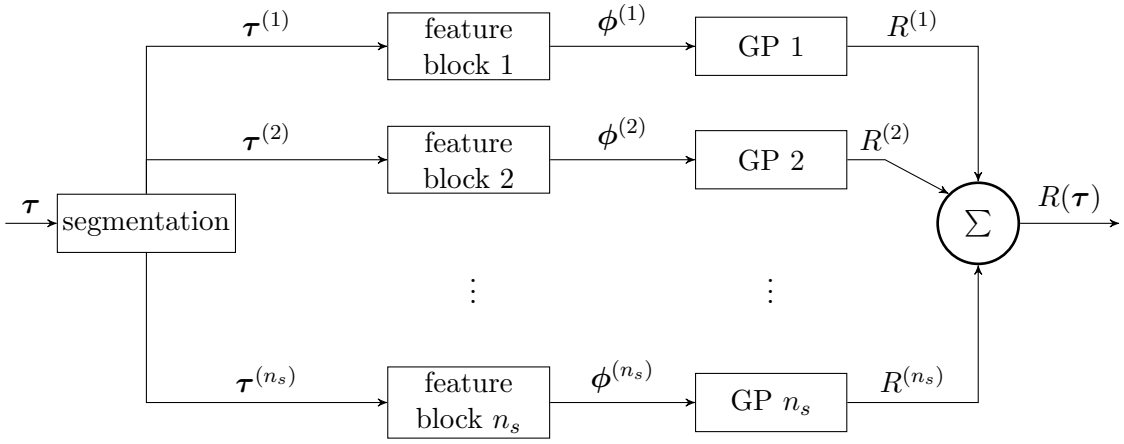


Figure 5-3: Segmented reward model

5-3 Demonstration acquisition over multiple segments

Now that we have constructed a reward model that can be adjusted in each time segment, we focus on the acquisition. In Section 4-4-1 an acquisition function is described that is able to select interesting trajectories for expert demonstration. In this section we will modify this acquisition function such that it evaluates trajectory segments instead of full trajectories. However, we will reuse the basic idea of expected policy divergence as our measure of acquisition.

To give an overview, the flow of procedures of the designed acquisition function is illustrated in Figure 5-4. We start by considering a set of sampled trajectories $\mathcal{T}$ as before. Furthermore,

we have a reward model consisting of four GPs, each of which has a different set $\mathcal{D}^{(i)}$ of demonstrations. We would like to know which trajectory segment is most important for acquisition. To accomplish this, we have to calculate the expected policy divergence of each trajectory segment $\boldsymbol{\tau}_j^{(i)} \in \boldsymbol{\tau}_j$ for all trajectories $\boldsymbol{\tau}_j \in \mathcal{T}$.

Let us consider a single trajectory segment $\boldsymbol{\tau}^{(i)}$. In order to obtain the expected policy divergence of this segment, we need to calculate the return of each trajectory in $\mathcal{T}$ using the original reward model. Let us denote $\mathcal{T}^{(i)}$ as the set of all i-th trajectory segments apparent in the trajectories of set $\mathcal{T}$. We can calculate the return of each trajectory in $\mathcal{T}$ by summing over the return of all defined segments

$$\tilde{\boldsymbol{R}} = \sum_{k=1}^{n_s} \mathcal{GP}^{(k)}(\boldsymbol{\Phi}(\mathcal{T}^{(k)})|\mathcal{D}^{(k)}),$$

which results in vector $\tilde{\boldsymbol{R}}$ containing the return of each trajectory $\boldsymbol{\tau} \in \mathcal{T}$.

Secondly, we need to calculate expected change in return for demonstrating trajectory segment $\boldsymbol{\tau}^{(i)}$. We start by evaluating $\boldsymbol{\tau}^{(i)}$ using its segment specific GP

$$\mu, \sigma = \mathcal{GP}^{(i)}(\boldsymbol{\phi}(\boldsymbol{\tau}_j^{(i)})|\mathcal{D}^{(i)}),$$

which results in a mean value μ and a variance σ. In order to obtain an approximation of the expected expert rating, we compute two sigma points: $s_+ = \mu + \sigma$, $s_- = \mu - \sigma$ and look at the consequence of adding one of these sigma points to the reward model. So let us assume one sigma point s is added to the set of demonstrations of its particular segment:

$$\mathcal{D}^{(i)} \leftarrow \{\mathcal{D}^{(i)} \cup s\}.$$

The reward model now consists of $n_s - 1$ unaltered GPs and one altered GP which contains the sigma point. We continue by calculating the total return of each trajectory according to this altered reward model,

$$\boldsymbol{R}^* = \sum_{k=1}^{n_s} \mathcal{GP}^{(k)}(\boldsymbol{\Phi}(\mathcal{T}^{(k)})|\mathcal{D}^{(k)})$$

which results in a vector $\boldsymbol{R}^*$ containing the returns of each trajectory $\boldsymbol{\tau} \in \mathcal{T}$.

After obtaining the return values for all according to the unaltered and altered reward model, we proceed as described in Section 4-4-1 by evaluating the policy update functions according to $\boldsymbol{R}^*$ and $\tilde{\boldsymbol{R}}$ and compare the results with the Kullback-Leibler divergence measure. We repeat this procedure twice, since we have two sigma points for each trajectory segment at our disposal. The acquisition value for a trajectory segment is the average of the two resulting EPD values.

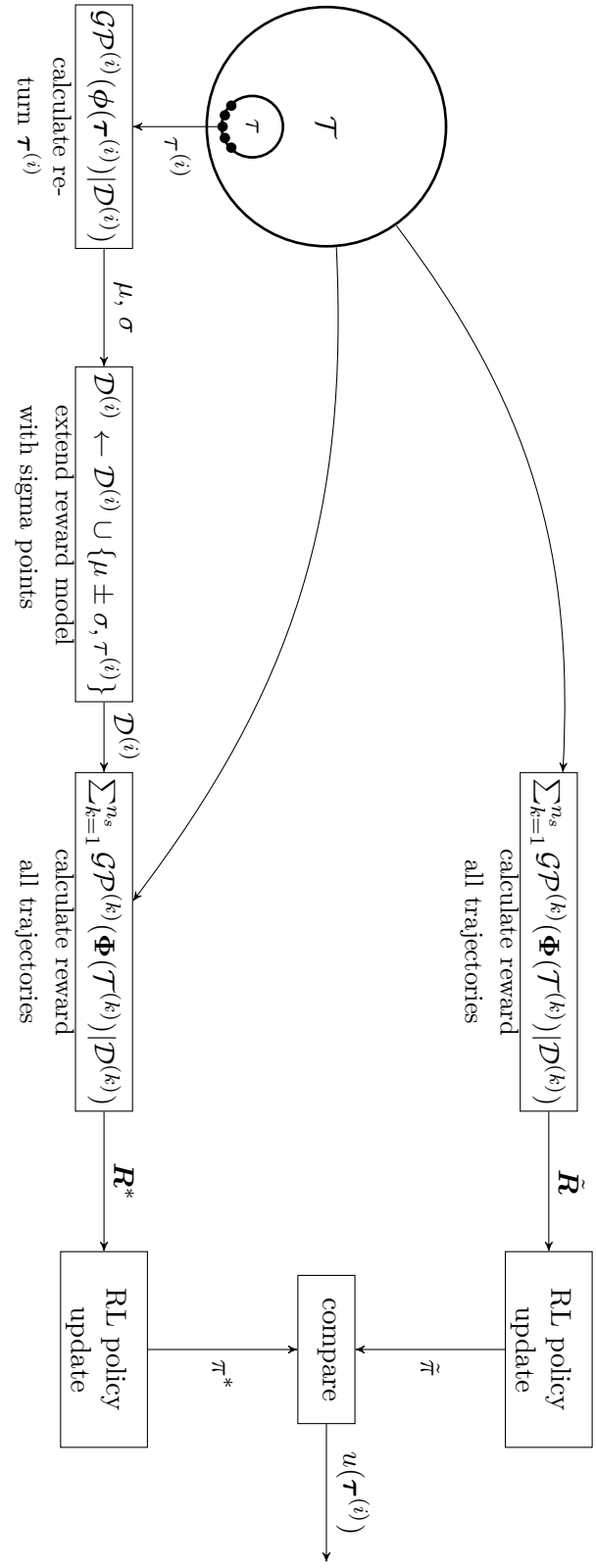


Figure 5-4: Expected policy divergence viewed schematically

5-3-1 Resulting algorithm

Even though significant changes have been made to the reward model and the acquisition function, the global structure of the ARL algorithm remains the same. As can be seen in Algorithm 3, most of the changes are incorporated in the reward model update (lines 8-17). By implementing segment specific demonstrations means we need to evaluate the acquisition function for every trajectory segment instead of every full trajectory. This extension results in a higher computational load depending on the number of defined segments. Empirically, this increase in computational load is negligible compared to the computational load of simulating robot trajetories.

Algorithm 3 Mutlit GP segmented active reward learning pseudo code

1: initialize policy $\pi(\boldsymbol{\theta}_0)$
2: initialize reward model $\mathcal{D}^{(1)}, ..., \mathcal{D}^{(n_s)}$
3:
4: **while** not converged **do**
5: Sample K rollouts
6: $\mathcal{T} = \{\boldsymbol{\tau}_1, ..., \boldsymbol{\tau}_K\}$
7:
8: FindNominee = true
9: **while** FindNominee **do**
10: Nominated outcome:
11: $\boldsymbol{\tau}_+^{(i)} = \arg\max_{\boldsymbol{\tau}^{(i)} \in \boldsymbol{\tau}, \forall \boldsymbol{\tau} \in \mathcal{T}} u(\boldsymbol{\phi}(\boldsymbol{\tau}^{(i)}))$ (See Section 5-3)
12: **if** $\boldsymbol{\tau}_+^{(i)} \notin \mathcal{D}^{(i)} \wedge u(\boldsymbol{\phi}(\boldsymbol{\tau}_+^{(i)}) > \lambda_a$ **then**
13: Demonstrate segment $\boldsymbol{\tau}_s^+$ to expert. Retrieve R_E^+
14: $\mathcal{D}^{(i)} = \mathcal{D}^{(i)} \cup \left\{\boldsymbol{\phi}(\boldsymbol{\tau}_+^{(i)}), R_{E_+}\right\}$
15: Optimize $\mathcal{GP}^{(i)}$ hyper parameters (See Section 4-3-3)
16: **else**
17: FindNominee = false
18:
19: Calculate return trajectories $\mathcal{T}$
20: $R(\boldsymbol{\tau}) = \sum_{i=1}^{n_s} \mathcal{GP}^{(i)}(\boldsymbol{\tau}|\mathcal{D}^{(i)}), \forall \boldsymbol{\tau} \in \mathcal{T}$
21: Update policy $\pi(\boldsymbol{\theta})$ (See Section 2-3)

5-4 Summary

This chapter introduced two new active reward learning algorithms incorporating time segmentation.

In Section 5-1 we described an algorithm that implements time segmentation by creating groups of feature functions assigned to specific time segments. With the specified end effector tasks in mind we constructed features functions for each time segment that represent the position of the end effector in that particular segment. The resulting reward model should be able to distinguish important time segments from unimportant time segments by recognizing relevant inputs. Incorporating the new reward model in the ARL framework does not have

an effect on the structure of the algorithm. The resulting method is able to learn end effector trajectories by expert feedback over full trajectories.

In Section 5-2 another extension to the ARL method is shown which incorporates segment specific demonstrations. In order to support segment specific demonstrations we have created a reward model which is based on multiple Gaussian processes, each one dedicated to a specific trajectory segment. The groups of feature functions designed in Section 5-1 are used as inputs for the GPs. To calculate the return over complete trajectories, a simple summation over the result of the different GPs suffices. The acquisition function of the standard active reward learning algorithm is adapted, such that it returns the acquisition value of a trajectory segment instead of a full trajectory. This new acquisition function is still based on the expected policy divergence principle but allows the algorithm to choose the most interesting trajectory segment. In order to implement the segmented reward model and acquisition function, the standard ARL has to be adjusted (See Algorithm 3). Using expert feedback over trajectory segments, this algorithm is able to teach end effector tasks to robotic systems.

Chapter 6

Experimental results

In this chapter, the results of multiple simulation experiments will be presented. More specifically, we will present the outcome of the algorithms shown in Chapter 5 applied to different robotic end effector tasks. First off, we will describe two different rating methods in Section 6-1. In Section 6-2, we will describe a number of important details regarding the policy learner (PI^{BB}) used in the experiments. After that, we will discuss the results of a simple viapoint task in Section 6-3. Both the results of a hard coded expert and a computer expert are shown. We continue in Section 6-4 by switching to a more advanced task which includes both a viapoint and a viaplane objective. In the last section we will try to improve the results of this advanced task by including variance of the different segments as a feature function in our reward model.

The implementation of all experiments conducted in this chapter can be found on github [1].

6-1 Rating methods

An important aspect of the algorithms we would like to test is the resulting reward model. We consider the result of our algorithms successful when, besides delivering a good policy, the resulting reward model captures the intention of the expert. This is hard to test when using a human as the expert since the true intentions of human ratings cannot be measured. So in order to test the convergence of the reward model with respect to the "true" return function, a computer expert is created. Using a computer expert allows us to evaluate the algorithm in a statistical manner, summarizing the result of different trials.

Besides performing experiments using a computer expert, we also test the SARL methods using a human expert in order to gain insight in the practical aspects of the algorithms rating procedure.

[1] https://github.com/RonaldOlsthoorn/RewardLearning

6-1-1 Computer expert

Fortunately, the performance measure of the tasks described in Section 1-1 can be expressed accurately in terms of mathematical functions. In the first task, the end effector of the robot arm needs to pass a viapoint $\boldsymbol{x}_{\text{viapoint}}$ at a specific time t_{viapoint} in its trajectory. We can use the Euclidean distance between the end effector at time t_{viapoint} and the viapoint $\boldsymbol{x}_{\text{viapoint}}$ and describe this measure as a penalty in the reward function.

$$R_{\text{true}} = -\|\boldsymbol{x}_{t_{\text{viapoint}}} - \boldsymbol{x}_{\text{viapoint}}\|$$

A suitable reward function for the viaplane objective can be constructed similarly. Instead of a single point in time we need to formulate a measure for penalizing the difference between the end effector and the viaplane in a time segment. If we denote the specific time segment as τ_{plane} we can use the Sum of Squard Errors (SSE) as our true reward.

$$R_{\text{true}} = -\sum_{t \in \tau_{\text{plane}}} (x_t - x_{\text{plane}})^2$$

If both a viaplane and a viapoint objective are used in the same experiment, a linear combination of the above mentioned expressions is used as a true return function.

In order to simulate human rating errors, the computer expert is extended with a zero mean Gaussian white noise. This computer expert is used when the algorithm asks for a rating on a given rollout. We will also evaluate a validation rollout each iteration using a computer expert without rating noise. This validation rollout will be generated by simulating the outcome of the policy without any exploration noise. Not only will the validation rollout be used to generate the policy learning curve but it is also useful to track the convergence of reward learning.

6-1-2 Manual ratings

In order to test the performance of the algorithms in practice, each task is also performed using a human expert. Every time a new demonstration is required, a graphical representation of the specific rollout is shown to help the expert in its decision making. Two different windows are designed for presenting rollouts, one for complete trajectories and one for trajectory segments. Examples of the two different interfaces are shown in Figure 6-1 and Figure 6-2. As can be seen, the window shows the requested trajectory or trajectory segment in each end effector dimension as well as the viapoint that we try to teach the robot. In order to enhance the accuracy of the expert we add annotations of previously rated rollouts in the background. Furthermore, the mean position of the different segments is also visible, since this is an important feature function for the reward model.

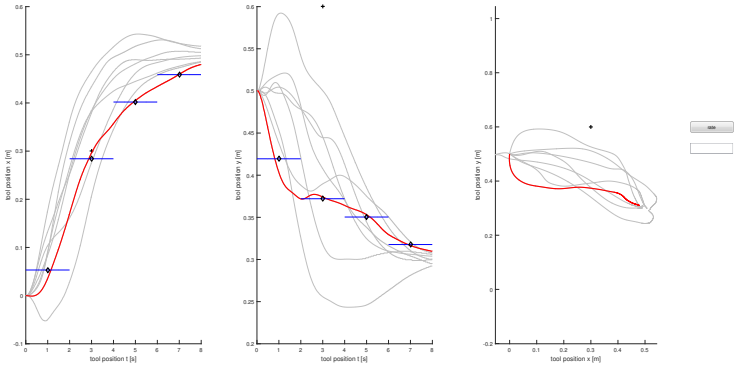


Figure 6-1: Rating window used to rate demonstrations of full trajectories.

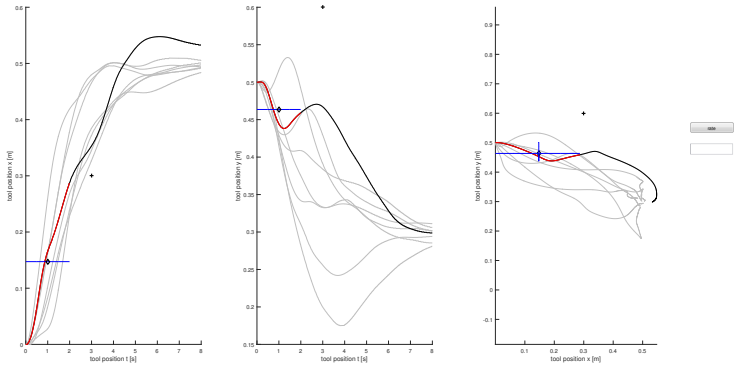


Figure 6-2: Rating window used to rate demonstrations of trajectory segments.

6-2 Design RL method

As discussed in Section 2-3, the PI^{BB} method is used for policy learning. Before generating the results of the two SARL methods, we need to design and tune the policy learning algorithm using a programmed return function. The results of PI^{BB} will give valuable data for comparison as will be shown in Section 6-3, Section 6-4 and Section 6-5.

Table 6-1 gives an overview of the design parameters used in the PI^{BB}. As can be seen in the table, we sample a small number of rollouts each iteration. However, a number of these samples are recycled to be used in the next policy update. We select the reusable rollouts based on the return. This procedure improves the number of rollouts needed for convergence which results in a total number of 250 rollouts or less for all conducted experiments.

The exploration noise variance is tuned to make sure a large part of the state-space is explored. In order to preserve enough exploration after converging to a local or global minimum, a minimum noise annealing factor is maintained. This is particularly useful considering a reward model that will change over time.

Table 6-1: Design parameters PI^{BB}.

Measure	value	Unit
Samples per iteration	5	-
Number of reused samples	5	-
Exploration noise variance Σ_ϵ	$\begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}$	-
Noise annealing factor λ_ϵ	0.95	-
Minimum noise annealing	0.1	-
Relative return scaling h	10	-

6-3 Single viapoint task

In this section we will apply the two active reward learning algorithms we constructed to teach a robot arm a simple viapoint task. We will investigate if it is possible to teach a robot such a viapoint task purely based on expert feedback. As input for the reward model we will only use the average x and y position of the end effector, the variance of the end effector positions will not be used. First we will show the result of the algorithm using a computer expert. After that, the results of the algorithm with a human expert are shown.

6-3-1 Computer expert result

The resulting trajectories of a viapoint task using a computer generated expert are shown in Figure 6-3 and Figure 6-4, where a single GP and a multi GP reward model is used respectively. In these figures, the average trajectory of twenty trials is shown as well as the standard deviation of the resulting trajectories. We also compare the results of the active reward learning algorithms with the results of the PI^{BB} RL method, using the computer expert as a return function.

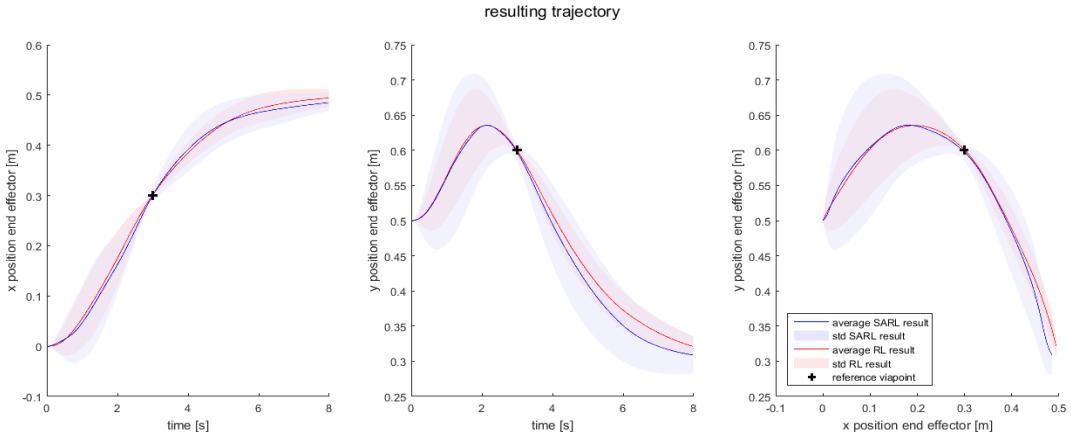


Figure 6-3: Resulting trajectory of a simple viapoint task using a single GP reward model.

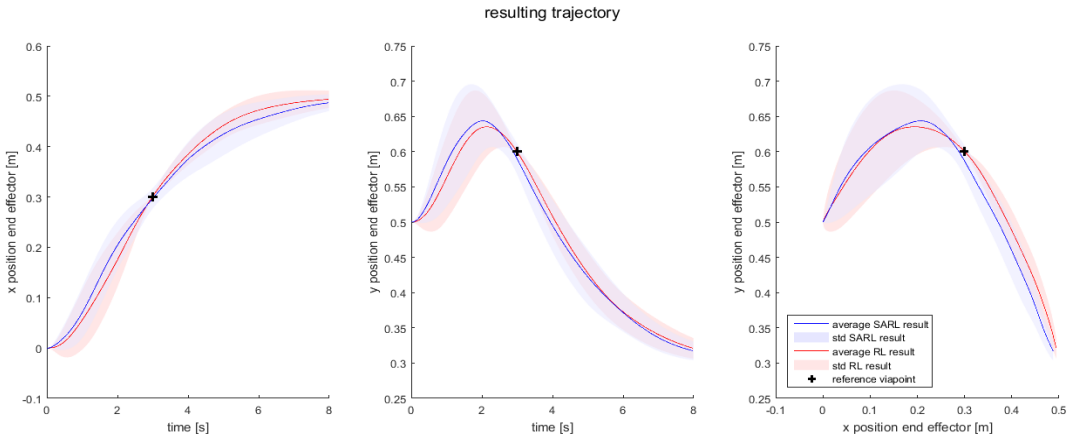


Figure 6-4: Resulting trajectory of a simple viapoint task using a multi GP reward model.

It is clear from the resulting trajectories of the active reward learning method as well as the reinforcement learning algorithm that the preferred trajectory that crosses the viapoint is one in which the end effector approaches the viapoint from above. This is explained by the policy of the reinforcement learning agent. We use a dynamic movement primitive as a parametrization for our policy, which causes the trajectories to be drawn to a certain end goal point at the end of the trajectory. In case of our specific viapoint task, this end goal is placed below the viapoint. Since the reward model is only based on the average position of the end effector in a specific time segment, the agent will learn to approach the viapoint from above.

Besides learning a decent policy that results in the end effector passing the viapoint, the reward model is also obtained in both algorithms. Since a computer expert is used, we are able to compare the obtained reward model with the "true" return function. In Figure 6-5 and Figure 6-6, the reward learning process is visualized. It is clear from the figures that both single GP and the multi GP reward model are able to express the true reward with reasonable accuracy. However in both graphs it is clear that the difference between the reward model and the true reward does not converge to zero. Important to note here is that an absolute difference between the reward model and the true return function does not necessarily have

to be a problem. The reinforcement learning method used in the experiments uses a softmax scaling method on the return of each trajectory in order to calculate the policy update (see Algorithm 1). This means that the reinforcement learning process is invariant under a scaling of the reward function. The only thing that matters is the relative difference in reward between different rollouts.

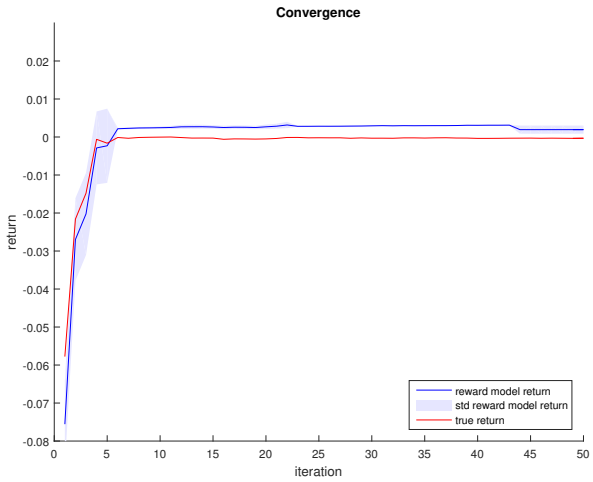


Figure 6-5: Convergence plot showing the return of the single GP reward model compared to the "true" return.

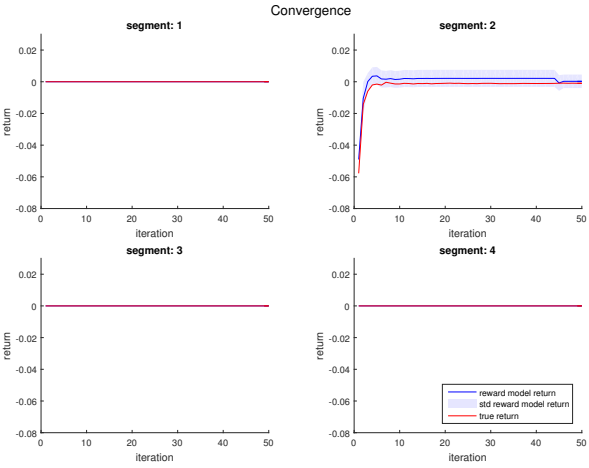


Figure 6-6: Convergence plot showing the return of the multi GP reward model compared to the "true" return.

If we use a multi GP reward model, the only feature functions are used as input for each GP are the average x and y position of the end effector. This means that we are able to visualize the reward model, as can be seen in Figure 6-7. The return function of the first, third and last segment are all zero functions, since no tasks are placed in these time segments. In the second time segment we can clearly observe an optimum, which coincides with the viapoint. Important to note is that an initial number of eight rollouts are used for initializing the reward model. During learning, the only extra demonstrations are performed in the second segment of the trajectory.

Now that the qualitative properties of the results are discussed we can focus on the quantitative measures of the algorithms. In this task there are three distinct measures we can use to judge the performance on. Foremost, we use the distance of the end effector to the viapoint $\boldsymbol{x}_{\text{viapoint}}$ at time t_{viapoint}. Secondly, we would like to keep the amount of expert queries to a minimum. In Table 6-2, the resulting measures of twenty attempts are summarized for each algorithm. We compare the SARL algorithms with a PI^{BB} RL method that uses the computer expert as its return function.

It is clear that for this simple viapoint task, the single GP algorithm is a better choice in terms of performance. First off, the average distance to the viapoint is lower compared to the multi GP method. Also, the total amount of expert queries is significantly lower. Important to note is that the multi GP method needs more expert queries to initialize its reward model, since each of the individual segments needs to be initialized with segment specific ratings. The number of queries needed during learning is actually lower when using a multi GP reward model.

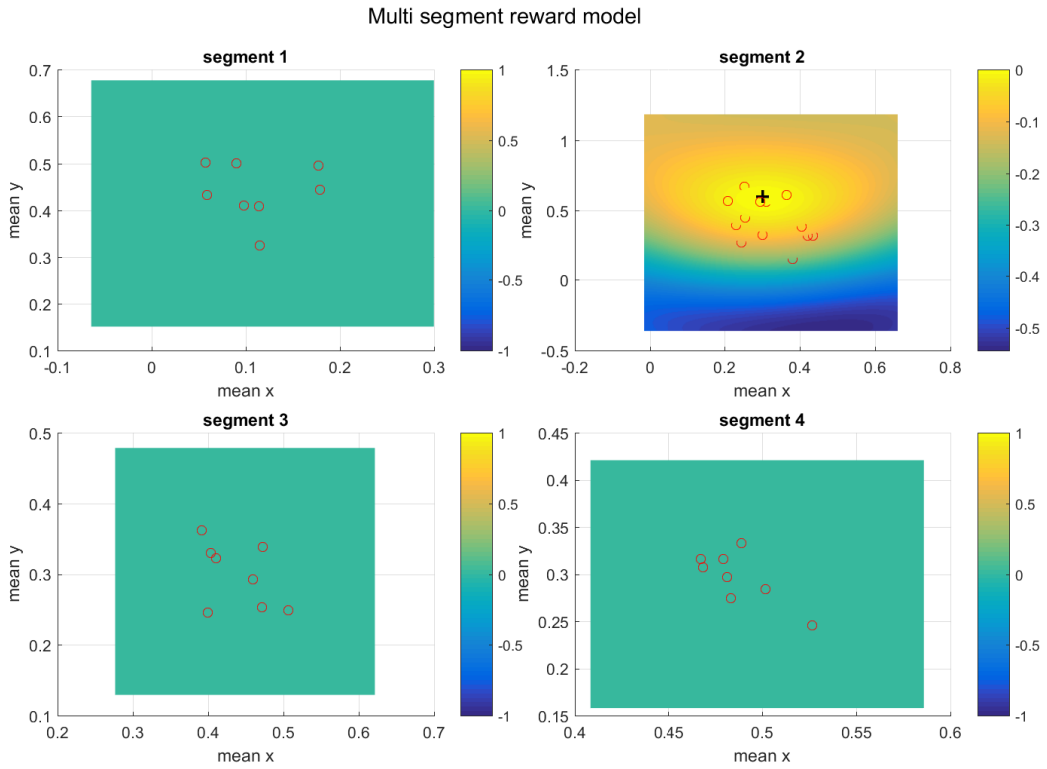


Figure 6-7: Graphical representation of the resulting multi GP reward model. The red dots represent queried rollouts. The black mark denotes the viapoint.

If we compare the SARL algorithms results with the results of PI^{BB} we can conclude that using a programmed return function is still a significantly better choice for this simple task.

Table 6-2: Results of segmented active reward learning and PI^{BB} applied to a viapoint task using a computer expert.

Measure	single GP	multi GP	PI^{BB} RL	Unit
Average distance viapoint	$1.5 \cdot 10^{-2}$	$2.3 \cdot 10^{-2}$	$7.5 \cdot 10^{-4}$	m
Standard deviation distance viapoint	$1.21 \cdot 10^{-2}$	$2.08 \cdot 10^{-2}$	$4.9 \cdot 10^{-4}$	m
Initial number expert queries	8	32	0	-
Average number expert queries	20.5	40.35	0	-
Standard deviation number expert queries	3.8	3.5	0	-

6-3-2 Manual expert result

Now that we have shown the results of segmented active reward learning using a computer expert, we will show the results of the same two algorithms using a human expert (the author).

The computer rated algorithm is not completely identical to the human rated algorithm. Besides the fact that the source of demonstration ratings is different, the tolerance parameter for expected policy divergence λ_a is also increased. This is due to a higher rating noise that the human expert is expected to produce. Keeping this tolerance parameter unchanged would significantly increase the number of demonstrations.

The resulting trajectories are shown in Figure 6-8 and Figure 6-9. It can easily be observed that the viapoints are tracked very accurately compared to the resulting trajectories of the computer expert experiments.

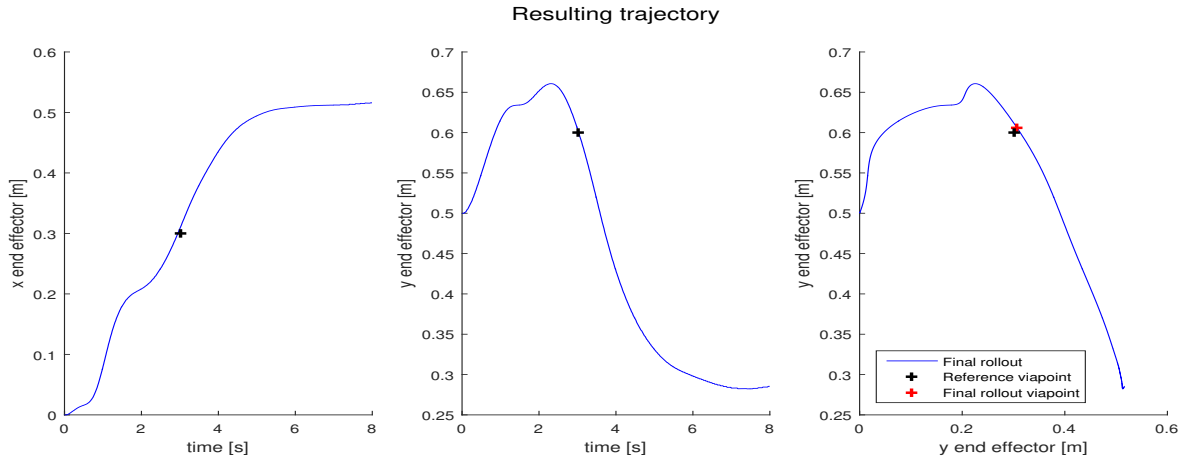


Figure 6-8: Resulting trajectory of a simple via point task using a single GP reward model.

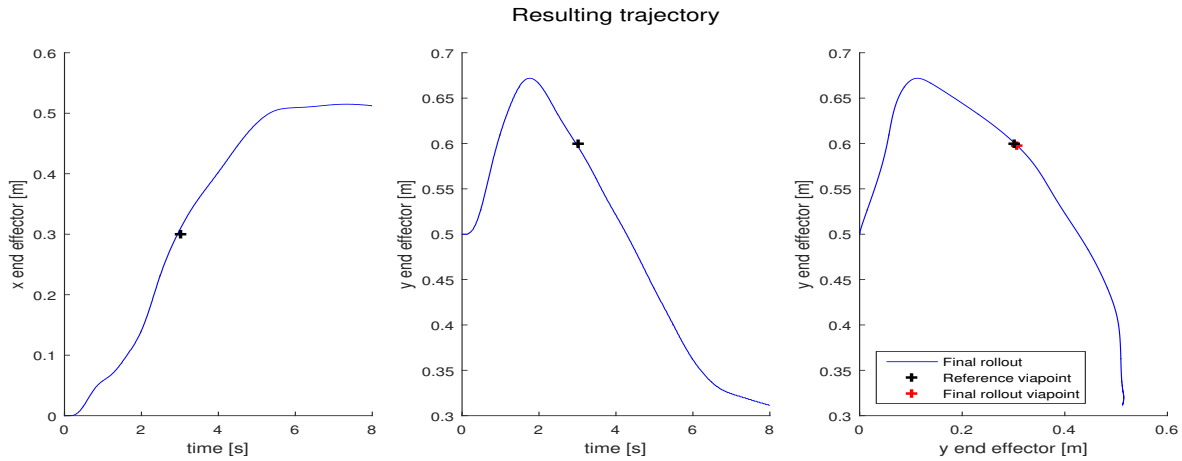


Figure 6-9: Resulting trajectory of a simple via point task using a multi GP reward model.

The resulting trajectories show a preference for approaching the viapoint from above as was observed in the previous section. It is difficult to create a reward model that prefers more elegant solutions, such as an approach from below, since the inputs for the reward model only contain information about the average position of the end effector. The expert cannot resolve this problem.

Since we do not have a "true" return function at our disposal, we cannot use this for comparing the reward learning process. Instead we will compare the convergence of the reward model

with the ratings that are given by the human expert, as is visible in Figure 6-10 and Figure 6-11.

A few observations can be made from these convergence plots. First off, the multi GP reward model requires more expert queries than the single GP reward model as was also observed in Section 6-3-1. The total number of expert queries matches the number of queries to the computer expert very accurately. In the convergence plots we observe that the reward model return is typically higher than the return of the demonstrated rollouts. Important to note is that the reward model return in Figure 6-10 and Figure 6-11 is calculated based on a rollout without exploration noise. All the demonstrated rollouts are perturbed by exploration noise, and these rollouts are usually found in unexplored areas of the state-space. Only a small number of these demonstrated rollouts obtain a high rating from the expert, which means that a noiseless trajectory is usually better.

A typical aspect of the human rated algorithm is that the expert is allowed to choose his or her own rating scale. So the fact that the single GP reward model shows a higher expert return in the later stages compared to the multi GP reward model does not mean anything, as we observed in the trajectory plots that the resulting trajectory of the multi GP model is actually better. The only task of the human expert is to teach the robot which rollouts are better or worse, relative to what has been rated before. Using a Gaussian process as a reward model in combination with the PI^{BB} reinforcement learning algorithm allows the human expert to use whatever rating scale he or she prefers.

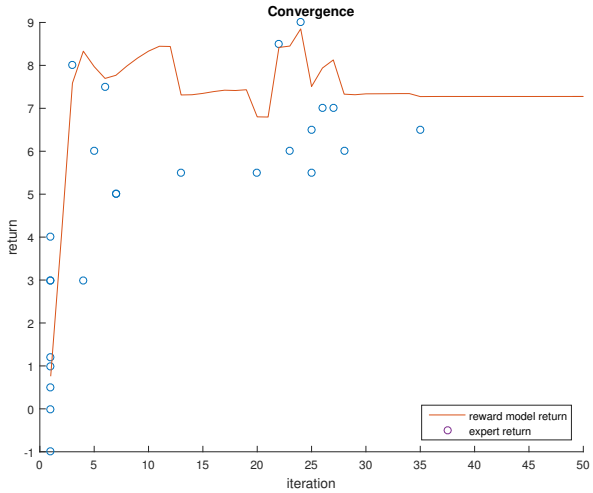


Figure 6-10: Convergence plot showing the return of the single GP reward model compared to the expert return.

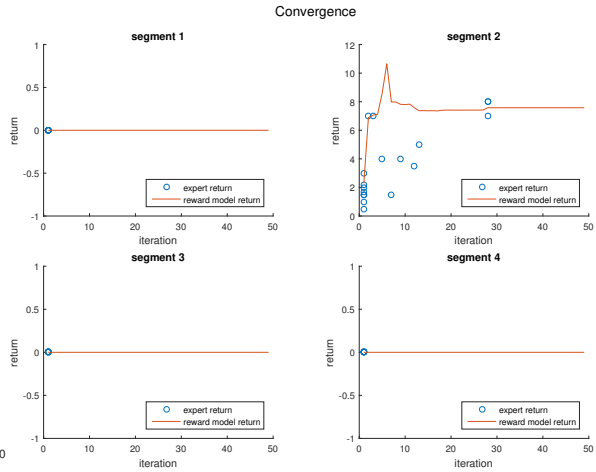


Figure 6-11: Convergence plot showing the return of the multi GP reward model compared to the expert return.

The reward model of the multi GP reward model can be visualized as well. In Figure 6-12 the return functions of the different time segments are shown. As expected, the GPs of the non-relevant segments all return zero. We can also see that the return function of the second segment shows a clear optimum at the viapoint position, but the gradient around this optimum is much higher than observed with the computer rated algorithm. This applies even you ignore the fact that the scaling of the expert ratings is different.

In Table 6-3 the results of the two manual trials are compared with the PI^{BB} RL method. It is

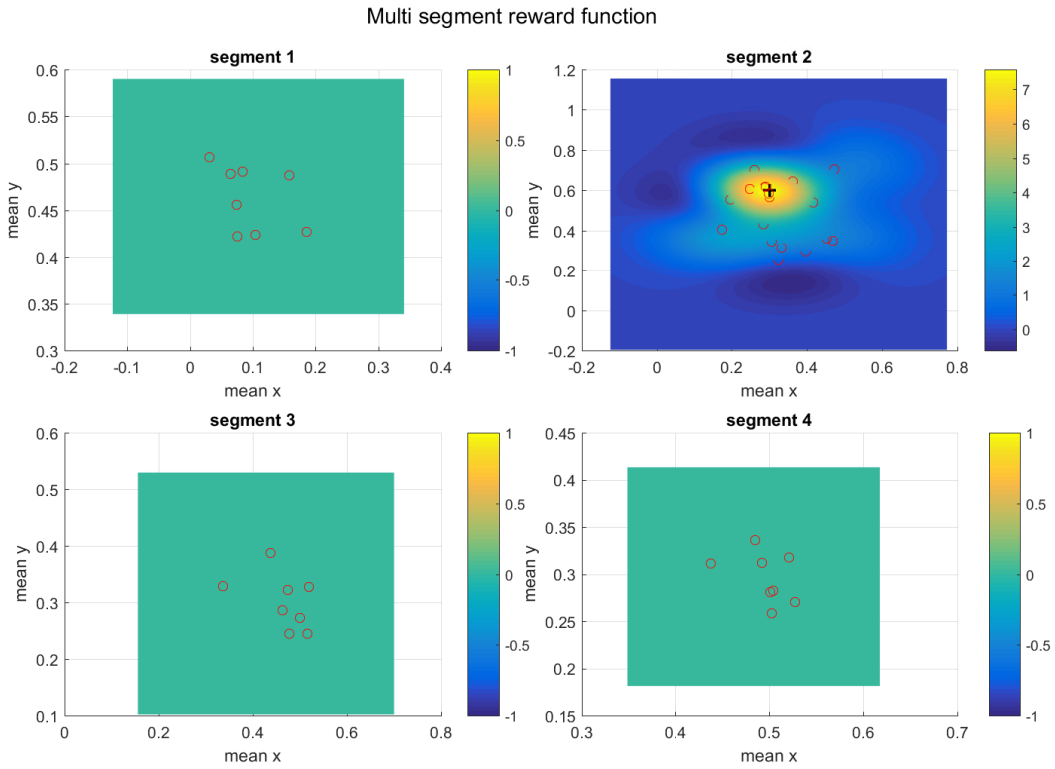


Figure 6-12: Graphical representation of the multi GP reward model. The red dots represent queried rollouts. The black mark denotes the viapoint.

clear from the table that the PI^{BB} method still outperforms both reward learning methods, since the PI^{BB} viapoint tracking measure an order of magnitude better than both reward learning methods.

If we compare the results of the manual trials in Table 6-3 with the results of the computer expert in Table 6-2, we see that it is actually possible to achieve a more accurate tracking result using a human expert. We can conclude that for this task, the human expert is able to create a better return function than the quadratic return function we used as computer expert.

From the figures in the results table, it is also easy to note that the single GP reward model is more efficient in terms of expert queries compared to the multi GP reward model. This difference is mainly caused by the number of initial demonstrations needed for the multi GP model. In terms of viapoint tracking accuracy, the multi GP method has a small advantage.

6-4 Multi objective task

In Section 6-3 we described how we can teach a viapoint task to a robotic arm. We will investigate the performance for teaching the robot a more difficult task in this section. This new task will contain two objectives. First off, we will recycle the viapoint objective that we

Table 6-3: Results of segmented active reward learning and PI^{BB} applied to a viapoint task using a human expert.

Measure	single GP	multi GP	PI^{BB} RL	Unit
Distance viapoint	$9 \cdot 10^{-3}$	$6.8 \cdot 10^{-3}$	$7.5 \cdot 10^{-4}$	m
Initial number expert queries	8	32	0	-
Number expert queries	25	42	0	-

used in Section 6-3. Besides that, the end effector of the robot arm must track a flat plane in space at the end of the trajectory. This second objective is harder to achieve for the robot arm. Also the combination of two objectives makes this task more challenging to solve.

6-4-1 Computer expert result

The computer expert described in Section 6-3 needs to be extended in order to be used for the new task description. We add a term which penalizes deviation from the viaplane in the last time segment. Also we need to multiply the result of the individual objectives (viapoint and viaplane) with a weight parameter, such that the importance of the two objectives is scaled properly.

In Figure 6-13 the average resulting trajectories of multiple succeeded runs using the single GP model are shown. A plot showing the resulting trajectories of the algorithm using a multi GP reward model is shown in Figure 6-14. Both trajectories clearly show an attraction towards the viapoint and the viaplane. We can already observe that the single GP reward model does not track the viapoint as accurate as is the case with a single objective task. In the trajectory plot of the single GP reward model we see that the standard deviation around the viapoint is significantly higher. Apparently, the single GP reward model does not map the multi objective return function as well as the multi GP reward model.

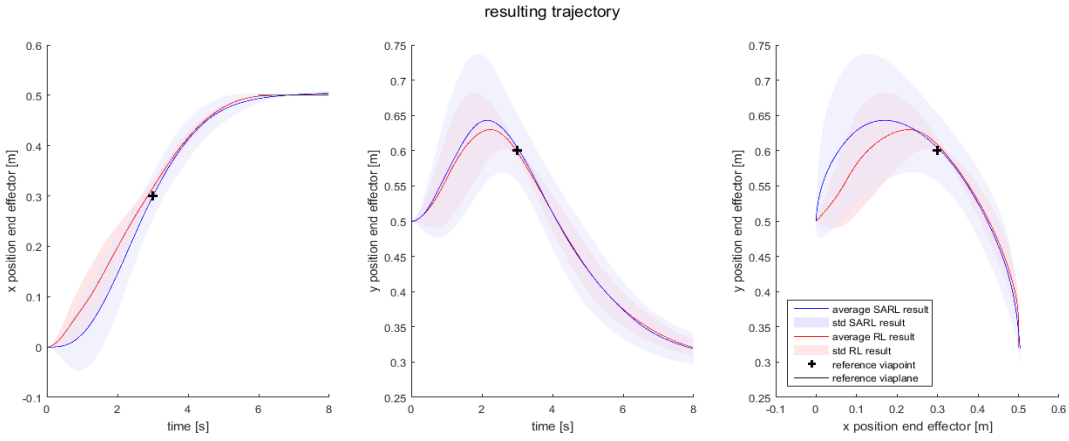


Figure 6-13: Resulting trajectory of a multi objective task using a single GP reward model.

If we look at the convergence plots (Figures 6-15 and 6-16) we can make some remarks regarding the stability issues. As can be seen in the convergence of a successful run using a single GP as reward model, it is clear that the reward model does not converge as fast as

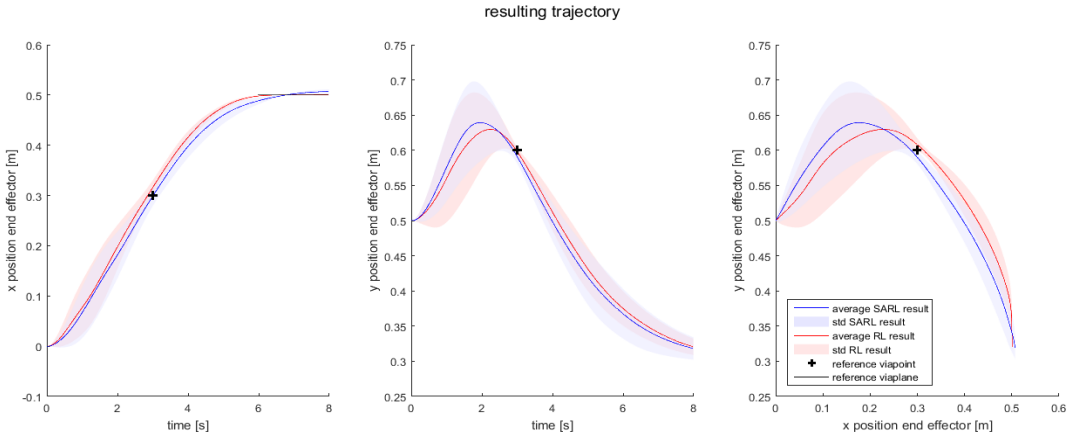


Figure 6-14: Resulting trajectory of a multi objective task using a multi GP reward model.

before. The reward model has a tendency to overestimate the return of trajectories and needs to be severely corrected during learning. If we look at the convergence plot of the multi GP reward model (Figure 6-16), we observe a much more stable learning algorithm.

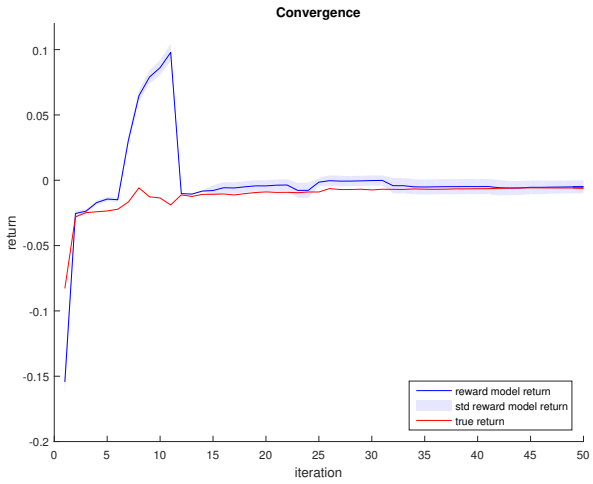


Figure 6-15: Convergence plot showing the return of the single GP reward model compared to the "true" return.

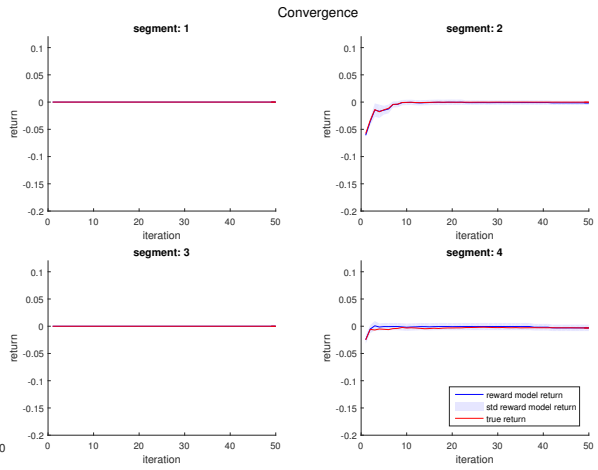


Figure 6-16: Convergence plot showing the return of the multi GP reward model compared to the "true" return.

Since we still use only two feature functions for each segment, we can plot the reward function of the multi GP reward model. As can be seen in Figure 6-17 there are now two relevant time segments. The second time segment contains the viapoint and, as expected, there is an optimum clearly visible in the area of the viapoint. The second objective, is also clearly visible as an optimal line.

If we look at the measurable results of the reward learning algorithms shown in table 6-4, we can make some important remarks. First off, it is clear that the multi GP reward model results in a policy that has better tracking results in both objectives compared to the single GP reward model. This suggests that a multi GP reward model is able to distinguish

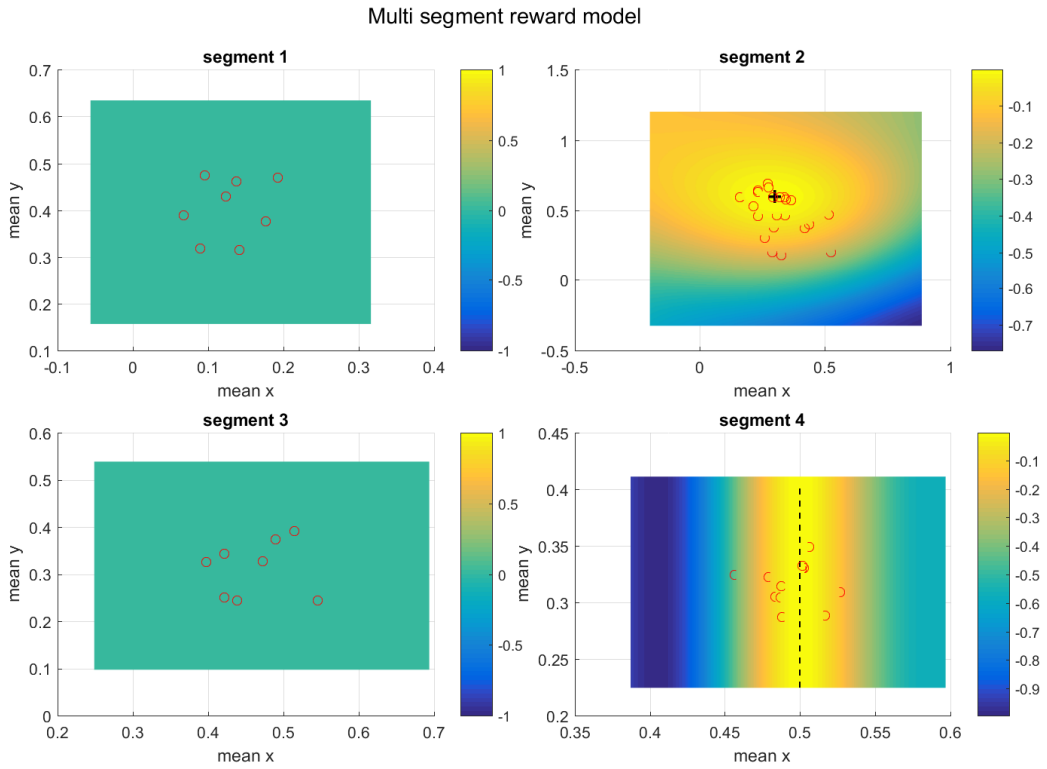


Figure 6-17: Graphical representation of the multi GP reward model. The red dots represent queried rollouts. The black mark denotes the viapoint and the dashed line denotes the viaplane.

tasks in different time segments with more accuracy than a single GP reward model. If we look at the number of expert queries we see that the single GP method still requires less demonstrations compared to the multi GP method. However, it is clear that this difference is due to the number of demonstrations required during the initialization phase of the reward model. During learning, the multi GP reward model requires less expert ratings than the single GP reward model.

If we compare the results of the reward learning methods with the results of the PI^{BB} method, it is clear that a hard coded return function still gives a better performance in terms of tracking. Especially if we look at the viaplan tracking results, we see an order of magnitude difference in performance.

6-4-2 Manual expert result

As can be seen in Figure 6-18 and Figure 6-19, it is possible to teach a multi objective task using a human expert as well. It is clear from the figures that both objectives are tracked with a reasonable accuracy. Also, the single GP method shows a better trajectory then the multi GP method. This is in contrast to the results we saw earlier in Section 6-4-1.

The reward learning results are shown in Figure 6-20 and Figure 6-21. In Section 6-3, we showed the convergence results of the multi GP reward model in the same format as we did

Table 6-4: Results of segmented active reward learning and PI^{BB} applied to a viapoint/viaplane task using a computer expert.

Measure	single GP	multi GP	PI^{BB}	Unit
Average distance viapoint	$4.18 \cdot 10^{-2}$	$2.00 \cdot 10^{-2}$	$1.68 \cdot 10^{-2}$	m
Standard deviation distance viapoint	$5.09 \cdot 10^{-2}$	$1.14 \cdot 10^{-2}$	$1.34 \cdot 10^{-2}$	m
Average SSE viaplane	$4.9 \cdot 10^{-3}$	$8.3 \cdot 10^{-3}$	$1.70 \cdot 10^{-4}$	m
Standard deviation SSE viaplane	$5.7 \cdot 10^{-3}$	$5.4 \cdot 10^{-3}$	$1.9 \cdot 10^{-4}$	m
Initial number expert queries	8	32	0	-
Average number expert queries	45.05	52.2	0	-
Standard deviation number expert queries	14.92	9.35	0	-

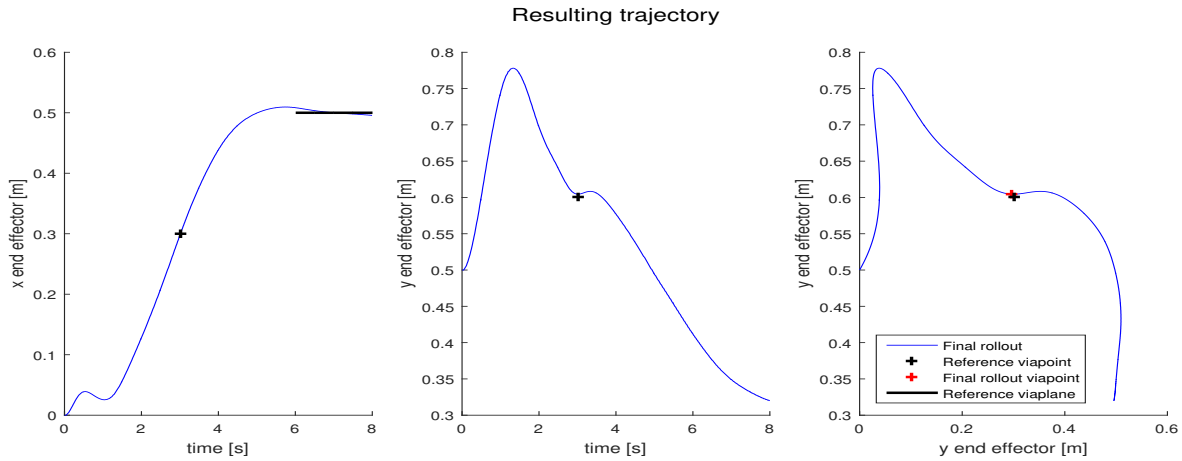


Figure 6-18: Resulting trajectory of a multi objective task using a single GP reward model.

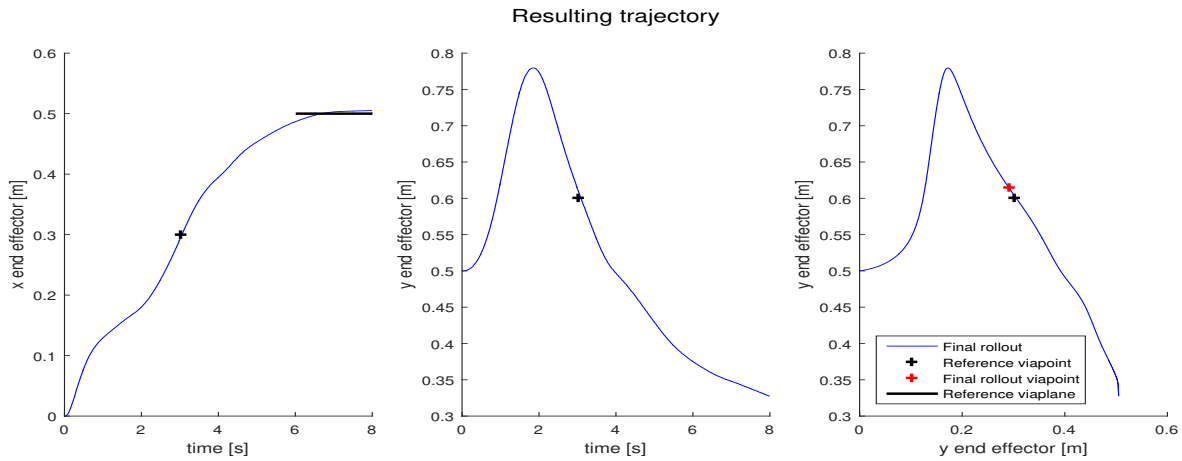


Figure 6-19: Resulting trajectory of a multi objective task using a multi GP reward model.

with the single GP reward model. Now that we have more than one relevant time segment we will present the results of the multi GP reward model per segment as is shown in Figure

6-21. We can make a few remarks regarding the convergence of the multi objective task. If we look at the multi GP reward model we can also conclude that the focus of the algorithm in terms of segments occurs in batches. In the first half of the learning phase, the algorithm only provides demonstrations in the last segment. Only when this objective seems to be learned, the focus shifts to the second segment in order to improve the viapoint objective.

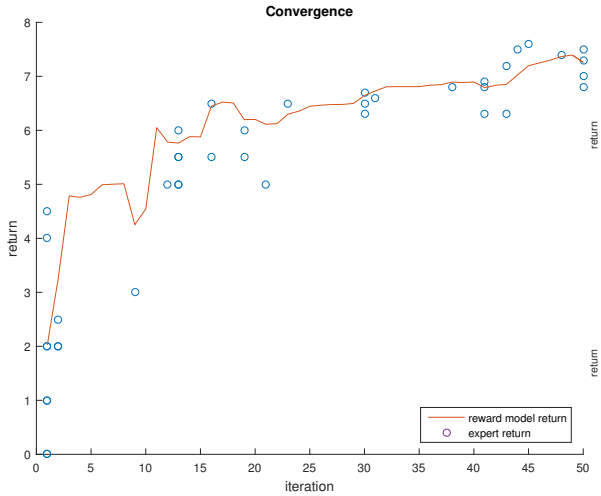


Figure 6-20: Convergence plot showing the return of the single GP reward model compared to the expert return.

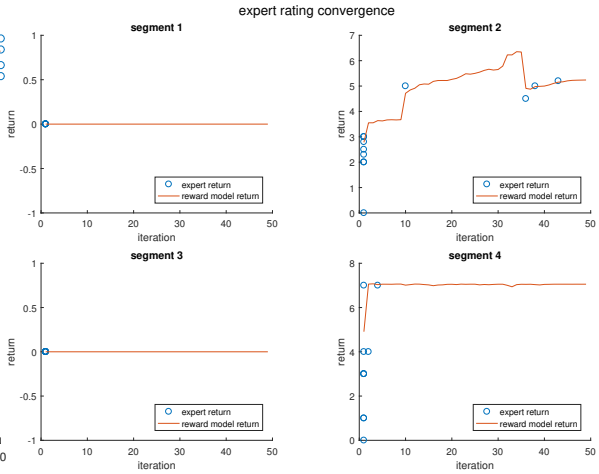


Figure 6-21: Convergence plot showing the return of the multi GP reward model compared to the expert return.

If we look at the resulting reward model in Figure 6-22 we clearly observe that the multi-objective task is extracted properly. In the second segment we observe a single optimal point right at the coordinate of the viapoint. It is clear that the gradient in the neighbourhood of the objective is much higher compared to the computer generated reward model (Figure 6-17). In the last time segment, the viaplane is clearly visible as an optimal line in space. Similar to the reward function of the second segment, the gradient around this line is much higher, compared to the gradient of the computer generated reward model.

A summary of the results of the multi objective task using a human expert is shown in Table 6-5. We cannot objectively state which reward model is the better choice for this task since both methods are able to perform quite well. In terms of viapoint and viaplane tracking, the single GP reward model gives a better result over a multi GP reward model but in terms of expert queries we should chose a multi GP reward model.

The two trials that are shown in the table are performing exceptionally well. This does not hold for all trials performed. There is an element of chance involved in teaching the robot a certain task, which is related to the acquisition function. In general, the acquisition function chooses which rollouts are interesting for demonstration. In the design of the acquisition function, we chose take the expected policy divergence as a measure of acquisition, which means that the acquisition function does not take into account the expected return. Poorly performing rollouts can be just as interesting as far as the acquisition function is concernded. However, it can happen that one of the demonstrated queries is tracking the viapoint of viaplane with a high accuracy. The expert can exploit this demonstration by giving such a demonstration a rating relatively higher then any other previous demonstration, hereby

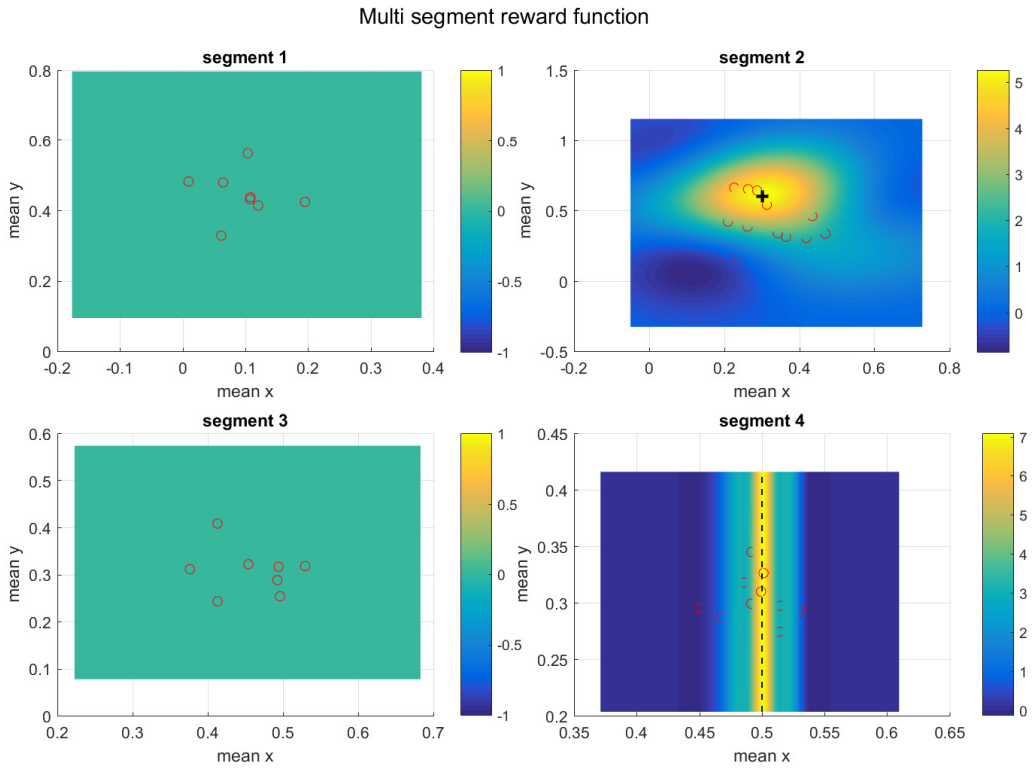


Figure 6-22: Graphical representation of the multi GP reward model. The red dots represent queried rollouts. The black mark denotes the viapoint and the dashed line denotes the viaplane.

creating a "spike" in the reward model. This spike will cause the policy learner to converge to a very accurate result.

Extending the task description from a single viapoint to a multi objective task has implications for the human expert as well. When a single GP reward model is used, an obvious disadvantage is that the complexity for rating a rollout increases. The human expert has to keep track of the relative performance of different rollouts for two objectives and summarize this difference in one single grade. The most obvious advantage of using a multi GP reward model is the benefit in rating complexity. The expert only has to focus on one objective per demonstrated trajectory segment (provided that the objectives do not share a time segment). But there are disadvantages as well. In the previous section, the expert was free to choose what rating scale he or she liked. This does not hold for a multi objective task, due to the fact that the difference in priority between the two objectives is dependent on the reward. If one objective is scaled from one to hundred whilst the second objective is scaled from one to five, the reinforcement learning agent as well as the acquisition function will only focus on the former objective. During the experiment it is observed that the time segments which need most improvement according to the expert are not always the time segments that are queried. This problem is that the active reward learning algorithm assumes that the acquisition function provides useful demonstrations for the reward model.

Table 6-5: Results of segmented active reward learning and PI^{BB} applied to a viapoint/viaplane task using a human expert.

Measure	single GP	multi GP	RL	Unit
Distance viapoint	$6.2 \cdot 10^{-3}$	$1.7 \cdot 10^{-2}$	$1.68 \cdot 10^{-2}$	m
SSE viaplane	$2.8 \cdot 10^{-3}$	$5.6 \cdot 10^{-3}$	$1.70 \cdot 10^{-4}$	m
Initial number expert queries	8	32	0	-
Number expert queries	41	38	0	-

6-5 Multi objective task with tracking improvement

In Section 6-4 the results of a multi objective task are shown. Reviewing the results we can conclude the second objective, the viaplane tracking task, is prone for improvement. The reward model used in the previous section only uses the average position of the end effector as an input for the Gaussian process, and this information is not enough for the reinforcement learning agent to learn a viaplane task.

In this section we will extend the reward model to include the variance of the end effector x and y position. This extension should give the reinforcement learning agent an incentive to keep the last segment of the trajectory as flat as possible. The cost for this extension is an increase in complexity for the reward model. Both the single GP and the multi GP reward model will have twice the number inputs as before.

6-5-1 Computer expert result

Besides an extension on the reward model, the rest of the algorithm does not have to be altered in order to work. For example the computer expert used in Section 6-4 can be reused. If we look at the resulting trajectories (Figure 6-23 and Figure 6-23) we can see an improvement in tracking performance. As can be seen in the plots, the standard deviation of the resulting trajectories around the viapoint and the viaplane is smaller compared to the results obtained in Section 6-4. It can also be concluded that the multi GP reward model has a smaller standard deviation around the objective and therefore has a better tracking performance.

The reward learning results, are also different depending on what reward model we use. It is clear in the convergence plots (Figure 6-25 and Figure 6-26) that the multi GP reward model is a much more accurate reward learner for this task. A reason for the decrease in reward learning for the single GP reward model can be found in the increased complexity of the Gaussian process. The GPs used for in the previous sections had a total number of eight inputs. The reward model used in this section has sixteen inputs. An important aspect of reward learning is the ability for the reward model to determine which inputs are relevant for performance and which inputs can be safely ignored. Since we doubled the number of inputs for the single GP reward model, we increased the difficulty for the reward model to make this generalization as well. We also increased the number of inputs for the multi GP reward model but since we use a different GP for each segment, the number of inputs per GP stays relatively low.

If we look at the measurable criteria in Table 6-6, it is clear that the variance extension of the reward model does have a positive effect on the results. First off, it can be noted that the

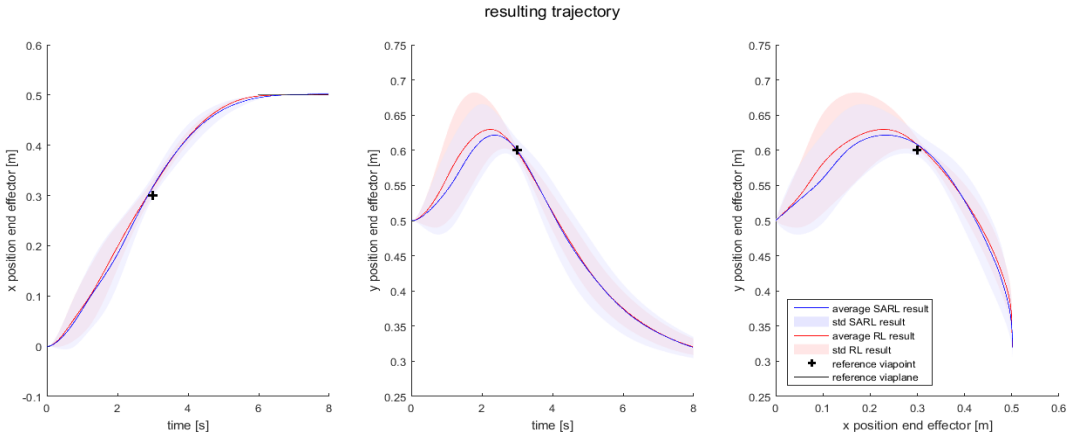


Figure 6-23: Resulting trajectories of a multi objective task using a single GP reward model.

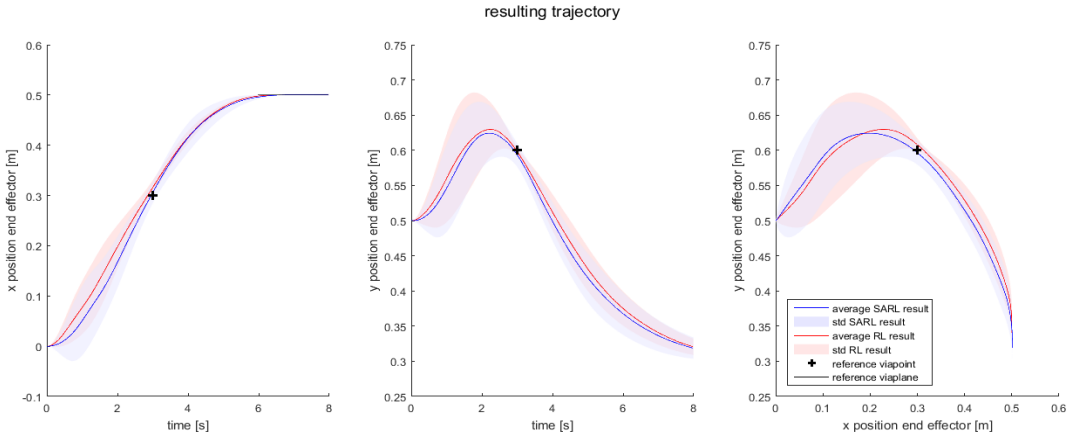


Figure 6-24: Resulting trajectories of a multi objective task using a multi GP reward model.

SSE of the viaplane has dropped significantly compared to the results obtained in Section 6-4. Another positive result is that the average distance to the viapoint also dropped, especially in case of the single GP method. The number of expert queries has increased slightly for both types of reward model. As before, the multi GP reward model needs some more expert queries than the single GP reward model, mostly due to the initialization phase of the algorithm.

6-5-2 Manual expert result

In contrast the computer expert results, there is no performance increase when rating is done by a human expert. Although the reward models are able to take the variance of a trajectory segment into account, it is hard for the expert to do the rating. There is a clear increase rating complexity if the human expert does not only have reward staying close to a plane but also staying flat. Some rollouts for example have a very flat last trajectory segment, but on a different level than is aimed for. These rollouts should be rewarded as well for staying flat, but to the human expert this is counter intuitive.

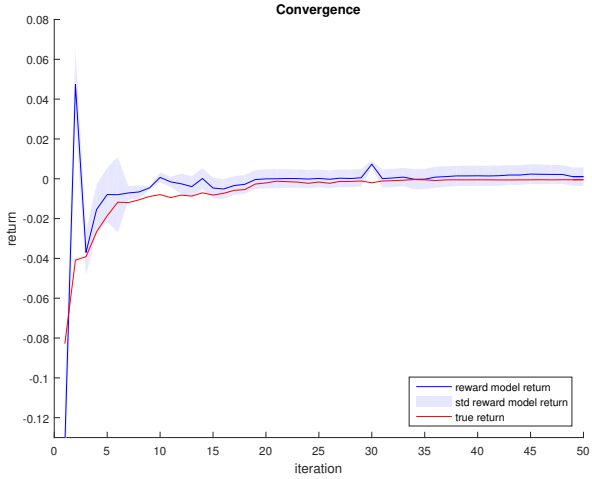


Figure 6-25: Convergence plot showing the return of the single GP reward model compared to the expert return.

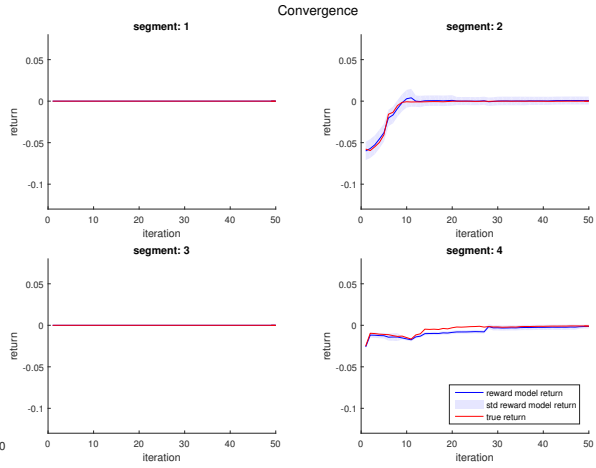


Figure 6-26: Convergence plot showing the return of the multi GP reward model compared to the expert return.

Table 6-6: Results of segmented active reward learning and PI^{BB} applied to a viapoint/viaplane task using a computer expert.

Measure	single GP	multi GP	RL	Unit
Average distance viapoint	$2.79 \cdot 10^{-2}$	$2.35 \cdot 10^{-2}$	$1.68 \cdot 10^{-2}$	m
Standard deviation distance viapoint	$1.35 \cdot 10^{-2}$	$1.45 \cdot 10^{-2}$	$1.34 \cdot 10^{-2}$	m
Average SSE viaplane	$1.7 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$	$1.70 \cdot 10^{-4}$	m
Standard deviation SSE viaplane	$1.6 \cdot 10^{-3}$	$1.5 \cdot 10^{-3}$	$1.90 \cdot 10^{-4}$	m
Initial number expert queries	8	32	0	-
Average number expert queries	49.5	54.1	0	-
Standard deviation number expert queries	15.25	7.65	0	-

In the resulting trajectories, shown in Figure 6-27 and Figure 6-28, it can clearly be observed that the extended reward model does not have a positive effect on the results. In the last segment of the trajectory, the end effector should track the viaplane nicely as was observed in the results of the computer expert. It is clear both reward models did not learn to reward the desired variance.

If we examine the convergence plot, displayed in Figure 6-29 and Figure 6-30, it is also clear that the multi GP reward model does not give the second objective enough priority. As can be seen in the convergence plot of Figure 6-30, it is evident that the majority of demonstrations is performed in the second segment. This is probably one of the reasons why the multi GP reward model fails to include the variance input in its result.

If we look at the objective measures in Table 6-7, we can see a decrease in performance. Both in terms of tracking performance and number of expert queries, the extended reward model cannot match the results in Section 6-4-2. We can conclude that, in contrast to the computer expert results, the extended reward model does not tend to increase the result. The

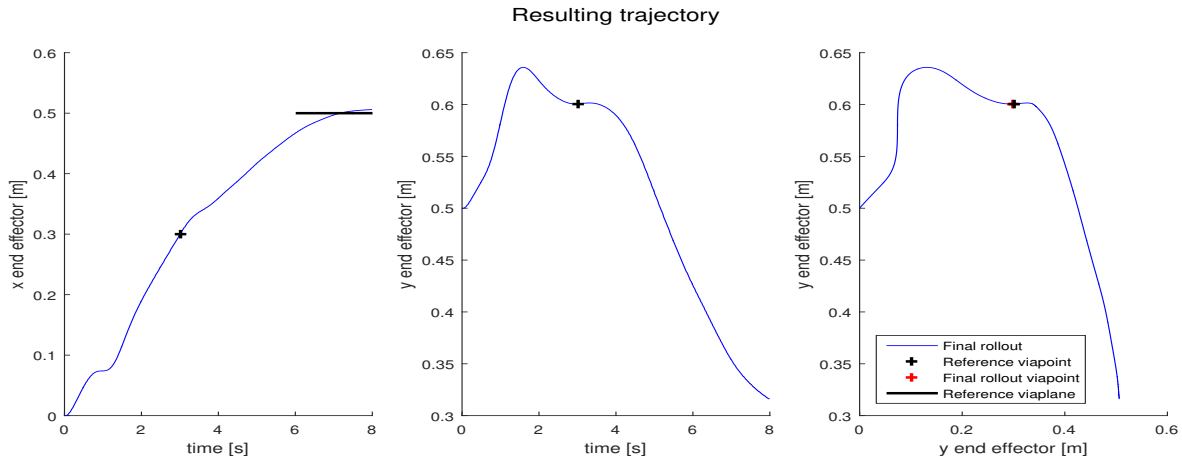


Figure 6-27: Resulting trajectories of a multi objective point task using a single GP reward model.

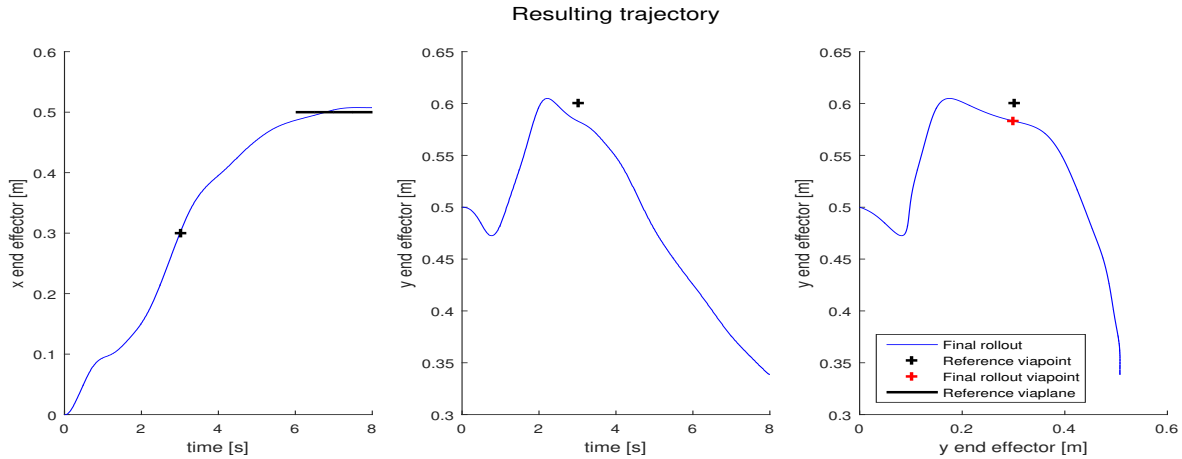


Figure 6-28: Resulting trajectories of a multi objective task using a multi GP reward model.

disadvantages of increasing the complexity for the expert outweigh the advantages gained in learning the reward model with more information.

Table 6-7: Results of segmented active reward learning and PI^{BB} applied to a viapoint/viaplane task using a human expert.

Measure	single GP	multi GP	RL	Unit
Distance viapoint	$1.3 \cdot 10^{-3}$	$1.67 \cdot 10^{-2}$	$1.68 \cdot 10^{-2}$	m
SSE viaplane	$3.69 \cdot 10^{-2}$	$1.00 \cdot 10^{-2}$	$1.70 \cdot 10^{-4}$	m
Initial number expert queries	8	32	0	-
Total number expert queries	37	59	0	-

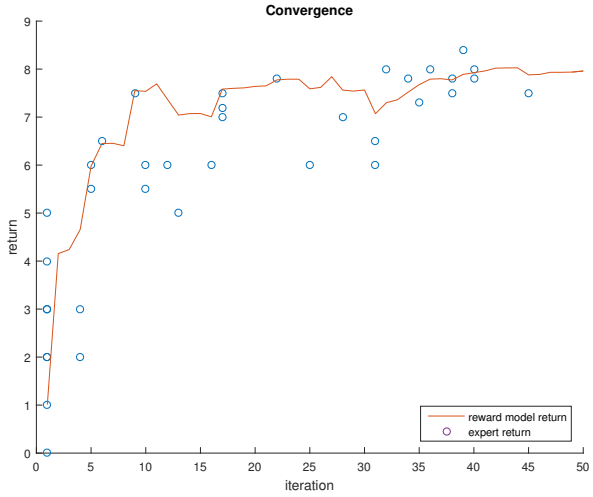


Figure 6-29: Convergence plot showing the return of the single GP reward model compared to the expert return.

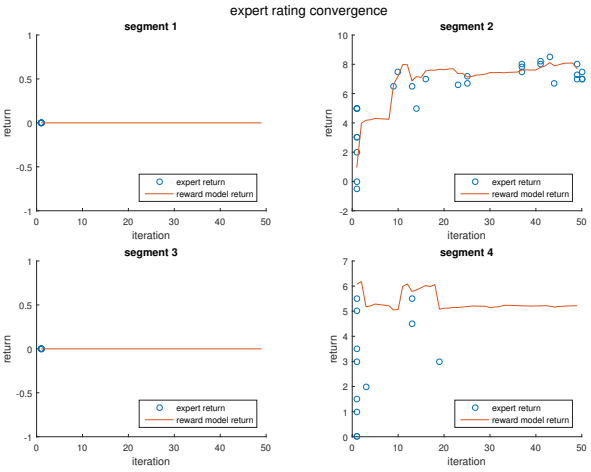


Figure 6-30: Convergence plot showing the return of the multi GP reward model compared to the expert return.

Chapter 7

Conclusions and recommendations

In this thesis, a time segmented variant of active reward learning is developed and applied as a machine learning algorithm to teach simple end effector tasks to robotic arms. In this chapter we will give an overview of what has been constructed in this work as well as conclusions on the results based on the research questions formulated in Section 1-2. Section 7-1 will give a summary of the thesis per chapter and Section 7-2 will give the conclusions of the final results. After that, Section 7-3 will give directions for further research based on the results and conclusions of this work.

7-1 Summary

In Chapter 1 the we observed the necessity of machine learning methods for robotics as an alternative for hard coded robotic programs. We proposed to extend the method of active reward learning, an algorithm able to learn robotic movements by using human expert ratings. This method combines the concept of reinforcement learning with reward learning in a hybrid fashion. Two proposals are proposed in which this framework is extended to use information of trajectory segments, instead of complete trajectories.

Chapter 2 continues by describing the reinforcement learning framework. We reviewed how reinforcement learning algorithms can be used as adaptive controllers that maximize a specific reward function. It is observed that using reinforcement learning is challenging in the field of robotics and that most successful examples of reinforcement learning controllers use policy learning algorithms. A black box policy learner called PI^{BB} is described in detail for use in Chapter 4 and 5.

In Chapter 3 the concept of reward learning is described as the process of obtaining the reward function from an expert. Several methods for using experts as a source of information are reviewed. The majority of these methods are not suited to be applied to robotics as they require to many rollouts. The active reward learning method is different from other reward learning methods as it uses the expert to give feedback on demonstrated rollouts, rather than

generating (sub) optimal rollouts itself. Due to the hybrid method of learning both the reward function as the resulting policy, this methods can be used in the field of robotics.

Chapter 4 describes the method of active reward learning in more detail. In order to learn a reward model, we use a stochastic inference method called Gaussian process. This inference methods provides an estimate of the return of a rollout as well as a measure of confidence for this estimate which we can use for demonstration acquisition. The acquisition function determines which rollouts are interesting for demonstration based on the expected policy divergence of a rollout.

In Chapter 5 we introduce two segmented active reward learning algorithms. In the first algorithm we show how we can implement segmentation by creating segment specific feature functions. In the second algorithm we show how we can implement segment specific expert querying into the active reward learning framework by constructing a different GP for each time segment that we use. We show how the concept of expected policy divergence is used to calculate an acquisition value for each trajectory segment, such that only interesting segments are demonstrated to the expert.

Finally in Chapter 6 we showed the performance of the algorithms constructed in Chapter 5. Using a computer expert, we can simulate the performance of both the algorithms and extract measurable performance figures. We applied the algorithms to a simple viapoint task and a multi objective task containing a viapoint and a viaplane. The results showed that teaching these tasks should be feasible and within practical limits in terms of expert queries and number of rollouts. Overall, the single GP method described in Section 5-1 was less accurate in tracking the objectives but it used less expert queries than the algorithm described in Section 5-2.

When a human expert was used to rate the rollouts, the overall performance was dependent on the complexity level of the task. For a single viapoint task, the expert was able to outperform the computer expert by creating a steeper reward gradient around the objective positions. In case of the multi objective task, it was clear that giving expert ratings on two different objectives introduces an importance scaling problem. However, we showed that it is possible to achieve the same level of performance with the human expert compared to the computer expert. It was also clear that the constructed acquisition function in Section 5-2 does not achieve its intended result which is to point out interesting trajectory segments for demonstration.

7-2 Conclusions

In Chapter 1 we introduced three research questions. From the results obtained in Chapter 6 we can conclude that it is possible to teach a robot end effector tasks using segmented active reward learning with a reasonable accuracy. Furthermore, it is clear from the results that the number of expert queries (rarely exceeding 50 queries) stays within the limits of practical application. We can also conclude that there is no benefit of using segment specific demonstrations if the end effector task is simple. In case of a single viapoint task, rating over complete trajectories gives similar results in terms of accuracy but using significantly less expert queries. When more complex tasks are applied, segment specific demonstrations performs similar to rating over complete trajectories. The number of rollouts needed for

teaching the robot end effector trajectories stayed within practical limits (<250 rollouts for all experiments).

One of the motivations for researching the topic of reward learning is to make reinforcement learning less dependent on reward function programming and tuning. From the manual expert results in Section 6-3-2 we can conclude that human experts are able to outperform computer experts on simple tasks, by increasing the gradient around an objective position. Similar reward functions with can be achieved by programming as well, but would require a lot of tuning.

Another observation can be made by the results of the multi objective task experiments. From these experiments it was clear that the importance of each objective has to be given by rating of the expert as well. So by switching to a reward learning method, we trade the problem of tuning importance weights in a program for a problem in assigning ratings. It can be concluded that human experts can be really effective in creating complex reward functions as long as the complexity of rating demonstrations is not too high.

7-3 Recommendations for further research

In this work we showed how we can learn a robotic arm some simple movements using human expert queries. This approach is relatively unexplored. All though it has been applied to a real robotic application namely a grasping task [7], this took the expert roughly 1000 evaluations, which suggests that real world application of active reward learning is still far away.

In this thesis, we successfully applied the active reward learning method to teach end a robotic arm end effector tasks. This thesis has shown that we can teach a robot arm viapoints and viaplane with a reasonable accuracy and stay within practical limits in terms of rollouts and expert queries. But we also found caveats in our approach that might be worth investigating further.

The acquisition function shown in Section 5-3 did not function as expected. The goal of this function is to point out interesting trajectory segments from a set of samples trajectories. It turned out that the acquisition function often just focuses on one particular segment and completely ignores other segments that might be interesting. The segments that are ignored are always the segments with low rating values, which does not necessarily mean that these segments are unimportant.

There are a number of different potential solutions that can solve this problem. A simple solution for example could be to arrange small number of demonstrations on random samples, similar to policy exploration noise. Another option which is more difficult to implement more expert involvement into the algorithm. For instance the expert could point out which segments are important using weights on each different segment. Another option that could be tried is to bin the acquisition function and make the expert interrupt the learning algorithm whenever he or she likes to query a specific trajectory segment. This option could lead to useful expert feedback since the expert can comment on rollouts that the acquisition function would ignore. But another outcome could be that a lot of expert ratings turn out to be useless because the expert intervened too early and did not let the reinforcement learning agent learn to optimize the reward model.

Another aspect of active reward learning that is prone to improvement is the presentation of demonstrations to the human expert. In the experiments conducted in this thesis, the demonstrated rollouts are presented as a set of graphs. This presentation worked sufficient until we tried to incorporate both average position and position variance into the reward model. The number of different features to consider is just too much if we use this presentation. Presenting extra plots to express variance could help the human expert. We know from previous works [20] that the rating bandwidth of human experts is limited but various widely depending on data presentation.

Bibliography

[1] F. Stulp and O. Sigaud, "Policy Improvement Methods: Between Black-Box Optimization and Episodic Reinforcement Learning." 34 pages, Oct. 2012.

[2] G. Bekey, R. Ambrose, V. Kumar, A. Sanderson, B. Wilcox, and Y. Zheng, "Assessment of international research and development in robotics," tech. rep., World Technology Evaluation Center, 2006.

[3] G. Biggs and B. MacDonald, "A survey of robot programming systems," in *Proceedings of the Australasian conference on robotics and automation*, pp. 1–10, 2003.

[4] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*. Cambridge, Massachusetts 02142: MIT Press, 1998.

[5] F. L. Lewis and D. Vrabie, "Reinforcement learning and adaptive dynamic programming for feedback control," *Circuits and Systems Magazine IEEE*, vol. 9, pp. 32–50, 2009.

[6] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *ICML '04 Proceedings of the twenty-first international conference on Machine learning*, ACM New York, 2004.

[7] C. Daniel, O. Kroemer, M. Viering, J. Metz, and J. Peters, "Active reward learning with a novel acquisition function," *Autonomous Robots*, vol. 39, no. 3, pp. 389–405, 2015.

[8] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, pp. 1238–1274, 2013.

[9] I. Grondman, L. Buşoniu, G. A. D. Lopes, and R. Babuška, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," in *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, pp. 1291–1307, 2012.

[10] C. J. C. H. Watkins, *Learning from delayed rewards*. PhD thesis, King's College, University of Cambridge, 1989.

[11] G. A. Rummery and M. Niranjan, *On-line q-learning using connectionist systems*. PhD thesis, Cambridge University, Tech. Rep., 1994.

[12] G. Tesauro, "Td learning and td-gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[13] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, no. 7, pp. 1180–1190, 2008.

[14] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[15] J. Peters, K. Mülling, and Y. Altun, "Relative entropy policy search.," in *AAAI*, pp. 1607–1612, Atlanta, 2010.

[16] E. Theodorou, J. Buchli, and S. Schaal, "A generalized path integral control approach to reinforcement learning," *The Journal of Machine Learning Research*, vol. 11, pp. 3137–3181, 2010.

[17] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Learning attractor landscapes for learning motor primitives," in *Advances in Neural Information Processing Systems 15 (NIPS2002)*, pp. 1523–1530, MIT Press, 2002.

[18] A. Y. Ng and S. J. Russel, "Algorithms for inverse reinforcement learning," in *ICML '00 Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 663–670, Morgan Kaufmann Publishers Inc., 2000.

[19] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, pp. 469–483, 2009.

[20] G. A. Miller, "The magical number seven, plus or minus two: some limits on our capacity for processing information.," *Psychological review*, vol. 63, no. 2, p. 81, 1956.

[21] D. Ramachandran and E. Amir, "Bayesian inverse reinforcement learning," in *IJCAI'07 Proceedings of the 20th international joint conference on Artifical intelligence*, pp. 2586–2591, Morgan Kaufmann Publishers Inc., 2007.

[22] S. Zhifei and E. M. Joo, "A review of inverse reinforcement learning theory and recent advances," in *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pp. 1–8, IEEE, 2012.

[23] K. Dvijotham and E. Todorov, "Inverse optimal control with linearly-solvable mdps," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 335–342, 2010.

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

[25] Q. Qiao and P. A. Beling, "Inverse reinforcement learning with gaussian process," in *American Control Conference (ACC), 2011*, pp. 113–118, IEEE, 2011.

[26] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," in *ICML*, vol. 97, pp. 12–20, 1997.

[27] B. Browning, L. Xu, and M. Veloso, "Skill acquisition and use for a dynamically-balancing soccer robot," in *AAAI*, pp. 599–604, 2004.

[28] J. D. Sweeney and R. Grupen, "A model of shared grasp affordances from demonstration," in *Humanoid Robots, 2007 7th IEEE-RAS International Conference on*, pp. 27–35, IEEE, 2007.

[29] P. K. Pook and D. H. Ballard, "Recognizing teleoperated manipulations," in *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*, pp. 578–585, IEEE, 1993.

[30] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, "Maximum margin planning," in *ICML '06 Proceedings of the 23rd international conference on Machine learning*, pp. 729–736, ACM New York, 2006.

[31] N. D. Ratliff, D. Silver, and J. A. Bagnell, "Learning to search: functional gradient techniques for imitation learning," *Autonomous Robots*, vol. 27, pp. 25–53, 2009.

[32] C. A. Rothkopf and C. Dimitrakakis, "Preference elicitation and inverse reinforcement learning," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 34–48, Springer, 2011.

[33] C. Dimitrakakis and C. A. Rothkopf, "Bayesian multitask inverse reinforcement learning," in *European Workshop on Reinforcement Learning*, pp. 273–284, Springer, 2011.

[34] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *AAAI Conference on Artificial Intelligence*, pp. 1433–1438, 2008.

[35] A. Boularias, J. Kober, and J. R. Peters, "Relative entropy inverse reinforcement learning," in *International Conference on Artificial Intelligence and Statistics*, pp. 182–189, 2011.

[36] K. Muelling, A. Boularias, B. Mohler, B. Schölkopf, and J. Peters, "Learning strategies in table tennis using inverse reinforcement learning," *Biological cybernetics*, vol. 108, no. 5, pp. 603–619, 2014.

[37] M. Kalakrishnan, P. Pastor, L. Righetti, and S. Schaal, "Learning objective functions for manipulation," in *IEEE International Conference on Robotics and Automation*, pp. 1331–1336, 2013.

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

[39] E. Polak and G. Ribiere, "Note sur la convergence de méthodes de directions conjuguées," *Revue française d'informatique et de recherche opérationnelle, série rouge*, vol. 3, no. 1, pp. 35–43, 1969.

[40] M. D. Hoffman, E. Brochu, and N. de Freitas, "Portfolio allocation for bayesian optimization.," in *UAI*, pp. 327–336, 2011.

[41] H. J. Kushner, "A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise," *Journal of Basic Engineering*, vol. 86, no. 1, pp. 97–106, 1964.

[42] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

Glossary

List of Acronyms

ARL	Active Reward Learning
DMP	Dynamic Movement Primitive
EPD	Expected Policy Divergence
GP	Gaussian Process
IRL	Inverse Reinforcement Learning
MDP	Markov Decision Process
RL	Reinforcement Learning
SARL	Segmented Active Reward Learning
SSE	Sum of Squard Errors

List of Symbols

Table 1: List of Latin symbols.

Symbol	description
$\mathcal{D}$	Set of demonstrated trajectories.
$\mathbb{E}$	Expected value operator.
h	Tuning parameter relative return scaling.
k	Covariance function.
$\mathbf{K}$	Covariance matrix.
n_s	Number of segments.
P	Probability operator.
r	Reward function.
R	Return function.
$\boldsymbol{R}$	Vector containing trajectory returns.
$\mathbb{R}$	Set of all real numbers
s	Sigma point.
$\tilde{S}$	Relative return.
t	Continuous time variable.
t_{viapoint}	Time slot viapoint.
T	Transition function.
$\mathcal{T}$	Set of sampled trajectories.
u	Acquisition function.
$\boldsymbol{u}$	Control action.
U	Set of all possible control actions.
V	State-value function.
x	x position end effector.
x_{viaplane}	x coordinate viaplane.
x_{viapoint}	x position viapoint.
$\boldsymbol{x}$	State of a system.
X	Set of all possible states.
y	y position end effector.
y_{viapoint}	y position viapoint.

Table 2: List of Greek symbols.

Symbol	description
γ	Reward discount factor.
$\gamma^*, \tilde{\gamma}$	Trajectory weights for KL approximation.
ϵ	Exploration noise.
ϵ_E	Expert rating noise.
$\boldsymbol{\theta}$	Policy parameter vector.
λ_ϵ	Exploration noise annealing factor.
λ_a	Acquisition threshold.
λ_f	Output length scale hyper parameter.
λ_ϕ	Input length scale hyper parameter.
$\boldsymbol{\Lambda}_\phi$	Matrix of squared length scales.
μ	Posterior mean GP.
$\boldsymbol{\mu}_m$	Posterior mean training points GP.
$\boldsymbol{\mu}_*$	Posterior mean inference points GP.
ξ	Hyper parameters.
π	Policy, also known as control law.
σ	Posterior standard deviation GP.
σ_E	Expert rating noise standard deviation.
$\boldsymbol{\Sigma}_\epsilon$	Exploration noise covariance matrix.
$\boldsymbol{\Sigma}_E$	Expert rating noise covariance matrix.
$\boldsymbol{\Sigma}_{mm}$	Posterior covariance matrix training points GP.
$\boldsymbol{\Sigma}_{**}$	Posterior covariance matrix inference points GP.
τ	Trajectory in state-space.
$\boldsymbol{\phi}$	Feature function vector.
$\boldsymbol{\Phi}$	Matrix of stacked feature function outcomes.