ÉCOLE CENTRALE DE NANTES POLITECHNIKA WARSZAWSKA

MASTER ERASMUS MUNDUS
EMARO+ "European Master in Advanced Robotics"

2017 / 2018

Bibliography Report

Presented by

Debaleena Misra

On 22 Jan 2018

Learning to grasp with the 'jamming gripper'

Jury

President:	Olivier Kermorgant	Assistant Professor (LS2N, ECN)
Evaluators:	Ina Taralova	Maître de conférences (ECN)
	Jean-Baptiste Mouret	Senior Researcher (INRIA, Nancy)
Supervisor(s):	Olivier Kermorgant	Assistant Professor (LS2N, ECN)
	Jean-Baptiste Mouret	Senior Researcher (INRIA, Nancy)

Laboratory: Laboratoire des Sciences du Numérique de Nantes LS2N

Abstract

Autonomous grasping is a challenging problem in robotics. For efficient grasping and manipulation, the robot should learn from interaction with its surrounding, instead of being preprogrammed to perform specific tasks, usually with objects having known geometric properties. Applying a visual object recognition scheme generalizes the problem to a great extent, as it does not rely on availability of 3d models of the environment. Using a reinforcement learning (RL) approach is very suitable for hand-eye coordination of grasping, as it reduces the complexity of visual feedback. It also enables the robot to autonomously handle changing manipulation dynamics, as well as efficiently respond to external disturbances in dynamic uncontrolled environments. However, the classical RL algorithms are time-consuming and require several interactions with the environment before the system can find optimal behaviour, which is often impractical in robotic applications, due to the hardware constraints and real-time challenges.

This bibliography report defines the research outline for the master thesis on vision-guided grasping learning by a jamming gripper. The research will be based on a variant of state-of-art reinforcement algorithm called Black-Box Data-Efficient Policy Search for Robotics. The algorithm will be implemented on the robot to gather information from multiple modalities of touch and vision and perform the grasping task through trial-and-error-learning. Further improvement would include extending the work to allow learning cycles with fewer resets. Such data-efficient learning methods can help make the robot truly autonomous, making it possible to handle unforeseen damage and quickly recover within few interactions.

The report presents the fundamentals of reinforcement learning, gives an overview of data-efficient RL approaches suitable in robotics and outlines the research plan for the thesis.

Acknowledgements

I would like to express my heartfelt gratitude to my supervisors Prof. Olivier Kermorgant and Dr. Jean-Baptiste Mouret for their continuous guidance, support and motivation in helping me prepare this bibliography study. I am thankful to the EMARO program for providing me a great opportunity to learn from distinguished professors and students from diverse backgrounds, during the entire course of my masters program. Finally, I am grateful to my family members, whose love and guidance are with me in all my pursuits.

Nomenclature

<u>**ABBREVIATIONS**</u>

RL	Reinforcement Learning
MDP	Markov Decision Process
GP	Gaussian Processes
DOF	Degrees of Freedom
MC	Monte Carlo Methods
DP	Dynamic Programming
TD	Temporal Difference
PS	Policy Search
SVM	Support Vector Machines
RBF	Radial Basis Function
DMP	Dynamic Movement Primitives
EM	Expectation-Maximization
DBN	Dynamic Bayesian Networks
MLE	Maximum Likelihood Estimation
BO	Bayesian Optimization
GP-MI	Gaussian Process Model Identification
LWBR	Locally Weighted Bayesian Regression
LWBR	Locally Weighted Linear Regression
RFWR	Receptive Field Weighted Regression
PILCO	Probabilistic Inference for Learning Control
Black-DROPS	Black-Box Data-Efficient Policy Search for Robotics

<u>**SYMBOLS**</u>

α	Learning rate
α_y	Spring constant
β	Basis function parameter
β_y	Damping constant
γ	Discounting rate
μ	Mean
ϕ	Basis function vector
ϕ_K	Kernel Hyper-parameters
π	Policy
σ	Standard Deviation
τ	Trajectory of states and actions
θ	Policy parameter
$\tilde{X}$	Training inputs
a_t	Actions
$D_{1:t}$	GP simulator function

f	Unknown transition dynamics in GP
J_θ	Expected return
K	Kernel matrix
n	Number of training inputs
q_π	Action-value function
R_t	Reward function
s_t	States
T	Transition probabilities
t	Time
u	Controls to the GP states
v_π	Value function
w	Shaping parameter
x	Continuous-valued states in GP
y	Training targets
z_t	Phase variable

List of Figures

1.1 Agent-environment interaction in a Markov decision process [3] 12
1.2 Convergence of value function and policy to optimality [3] 15

2.1 State-space modelling for the ball-paddling reinforcement learning task [6] . . . 17
2.2 Action space (left) vs Parameter space(right) [47] 20
2.3 Model-based RL approaches[4] 21
2.4 Overview of transition models literature [27] 22

3.1 Kuka Youbot fitted with a jamming gripper [46] 30
3.2 Grasping modes of the jamming gripper [28] 30
3.3 Work schedule planned for master thesis 31

Contents

Introduction **9**

1 Basics of Reinforcement Learning **11**

1.1 Background . . . 11
1.2 Markov Property . . . 11
1.3 Returns and Episodes . . . 12
1.4 Policies and Value functions . . . 13
1.5 Bellman Equation . . . 14
1.6 Optimal Policies and Optimal Value Functions . . . 14
1.7 Value-function approaches . . . 15
1.7.1 Dynamic Programming-Based Methods . . . 15
1.7.1.1 Policy-iteration . . . 15
1.7.1.2 Value-iteration . . . 16
1.7.2 Monte Carlo Methods . . . 16
1.7.3 Temporal Difference Methods . . . 16

2 Reinforcement Learning in Robotics **17**

2.1 Challenges . . . 17
2.2 Policy Search Approaches . . . 18
2.2.1 Optimization in Policy Space . . . 18
2.2.2 Typical Policy Representations . . . 19
2.2.3 Model-Free Policy Search . . . 20
2.2.4 Model-Based Policy Search . . . 21
2.3 Data-Efficient Reinforcement Learning . . . 23
2.3.1 Bayesian optimization . . . 24
2.3.2 PILCO (Probabilistic Inference for Learning Control) . . . 24
2.3.3 Black-Box Data-efficient Policy Search for Robotics (Black-DROPS) . . . 24
2.3.4 Using Parameterized Black-Box Priors to Scale Up Model-Based Policy Search for Robotics . . . 26

3 Work Outline **29**

3.1 Summary of objectives . . . 29
3.2 Set-up . . . 29
3.3 Work plan . . . 30
3.4 Schedule . . . 31

Bibliography **31**

Introduction

Grasping is a fundamental skill for manipulation of objects. For humans and animals, object manipulation is performed very easily by fast feedback loop between perception and action, and even complex tasks such as extracting items from a clutter are done without prior planning. However, robot grasping is a complex problem that faces several challenges when trying to imitate human manipulation skills. Without additional human intervention and assistance, the robot must learn to grasp through interactions with various objects in its surrounding, and discover new grasps to improve its grasping abilities. Most prior work in robot grasping [39] require complete 3D or 2D models of the object and based on this precise knowledge, rely on highly specialized control rules that involve advance planning. However, in real-world situations, such methods are often impractical since the complete 3D information may not be available. Besides, the robot may have to operate in new and unknown environments, where it will need to adapt to changes in its surrounding, and recover from damages, if any.

The thesis being planned in this bibliography report, presents a learning-based approach to hand-eye coordination for grasping tasks by an universal jamming gripper [28]. Jamming grippers are highly adaptive and excel at fast gripping of complex objects, but it undergoes deformation during grasping activity, which makes it difficult to model it. It consists of a single mass of granular material which when in contact with an object, flows around it and conforms to its shape. Autonomous grasping approaches in previous works [39] commonly rely on physical model of the gripping behaviour, and thus cannot be applied to our work. So a learning-based grasping approach is adopted, along with the use of sensory inputs of visual information to establish the spatial relationship between the jamming gripper and graspable objects available in the surrounding. Using a learning algorithm brings a great degree of generalization in the grasping, and can be easily applied to previously unseen objects and also equips the system to undertake appropriate actions in uncontrolled environments, by learning from its experiences. Such learning from interaction forms the foundation of important theories of intelligent computations and had an influence on engineering for several decades [6]. Reinforcement learning (RL) [4] is one such important computational approach that focuses on goal-directed learning from dynamic interactions and forms the basis of the thesis work planned in this report.

Reinforcement learning is the class of problems in which a robot finds its optimal behavioural strategy through trial-and-error interactions with a dynamic environment. The term *reinforcement* is derived from animal learning studies in experimental psychology [1], where it refers to a response causing the occurrence of an event such that it leads to an increased probability of the response recurring under the same situation. Hence, if an action is followed by a satisfactory or improved state, then the tendency to repeat that action is strengthened, i.e *re-inforced.* This maybe seen as instrumental conditioning which are the experimental situations where the actions taken by the animal is critical is determining the subsequent events, unlike classical conditioning as Pavlov [1], where the actions do not influence events. So, in reinforcement learning, the agent perceives information about its current state in the environment, can take actions that changes its current state and receives a reward (or punishment) signal that indicates how appropriate this behaviour has been [4]. This feedback signal maybe sparse, delayed or noisy. The objective of an RL problem is to find an optimal policy that maps states to actions and maximizes the long-term reward.

Reinforcement Learning has proven to be a powerful technique that can solve complex tasks like helicopter flight [42], Atari games [43], locomotion [32, 33, 34] and robotic manipulation tasks [6, 18, 20], all through training an agent to optimize behaviours over a reward function. However, a major limitation for most RL algorithms is that they are data extensive and require significantly high number of interactions between the agent and the environment [7]. The agent has to explore and repeatedly interact to gain enough information to find a good strategy. So, in real-world applications where agent-system interactions are scarce or expensive, as in robotics or systems control, this is a major challenge to overcome, along-with managing the problems posed by high-dimensional continuous state and action spaces required by the robot [6]. The data inefficiency is even more critical in cases of damages that have not been anticipated and the robot has to adapt and autonomously discover compensatory behaviours within a few trials.

The data-efficiency of an RL problem can be improved if a model representing the world is built, which summarizes the collected experiences and can be utilized for generalizing and drawing hypotheses about effects of undertaking a specific action [5]. The model can be defined with the help of a transition function which describes the system dynamics by mapping state-action pairs to successor states. When available data is less, there can be several likely functions to explain the mapping of samples. Now, if a single most likely function is chosen with certainty to learn strategies everywhere, it can lead to the problem of model bias. Any single model is simply one of the several plausible explanations of understanding the observed data. Without taking into account the whole set of plausible dynamics, model-based RL algorithms will give in to model-bias counteracting the data-efficiency advantages of a modelling approach. Using Gaussian Processes (GP) [29] provide a probabilistic framework to reduce the problem of model bias and is very useful to account for the model uncertainty, since all plausible functions to describe the dynamics can be incorporated in the decision-making by Bayesian averaging according to a GP distribution. Model-based RL approaches also suffer from the problem of dimensionality [11] that rises from handling complex tasks, involving robots with high degrees of freedom (DOF). The transition function between complete state/action spaces (such as joint angles, velocities etc) become very data intensive as the model complexity significantly increases with every additional DOF. For scaling to such high-dimensional space, it is useful to use prior knowledge of the system, that enables the learning process to begin with an initial meaningful guess of the dynamic model [21, 37].

The thesis aims to perform the vision-based grasping of the jamming gripper by using a data-efficient learning method that applies probabilisitic modelling with priors, and uses black-box optimization techniques to quickly learn a task. We build upon a state-of-art model-based reinforcement learning algorithm called the Black-box Data-efficient Robot Policy Search (Black-DROPS) [20] which has been successfully demonstrated with complex robotic tasks operating in high-dimensional spaces and takes minimal interaction time. The objective is to implement Black-DROPS with priors [21] and extend the algorithm to allow learning cycles with minimum resets. This bibliography report presents the basic aspects of reinforcement learning, the state-of-art algorithms that are applied in robot learning tasks and the proposed outline of research for the master thesis.

Chapter 1

Basics of Reinforcement Learning

1.1 Background

Reinforcement Learning (RL) has its origins derived from the progress made in optimal control in the late 1950s, with the advent of dynamic programming [2]. Although different in their assumptions, the goal of both RL and optimal control is to extract the maximum reward from a system, which is usually formulated as a Markov Decision Process [3]. Hence, strong connections can be established between the two fields. In 1980s, the modern concept of RL was developed when dynamic control was applied with trial-and-error learning methods, to learn controllers for systems with little knowledge available [6]. This view of RL, which has now become broadly researched, is the focus of our study.

In a standard reinforcement learning model, the agent is linked to its environment by perception and action. The agent's objective is to discover an optimal policy π that maps states to actions and maximizes the long-term reward of reinforcement. Compared to supervised learning, RL problems are more general and therefore more difficult since learning is based on considerably less knowledge [4]. Unlike supervised learning, there is no external supervisor with a knowledge of the environment, so there are no input-output pairs. Instead, on selecting an action, the agent is given only an immediate reward and the next state, but there is no guidance on which action would be the most ideal for the long-term interest. Hence the agent has to gather useful experiences about possible system behaviour, actions, transitions actively to behave optimally. Such online decision-making involves a fundamental choice between *exploitation* where the agent makes the best decision considering the current information it has about safe action-reward pairs, and *exploration* to discover new information and strategies to look for higher reward. The optimal long-term strategy may require short-term sacrifices and this problem is commonly known as the *exploration-exploitation* trade-off.

This chapter describes the fundamentals of conventional Reinforcement Learning and the elements that describe a RL problem.

1.2 Markov Property

Reinforcement learning allows both qualitative and quantitative foundations for modeling adaptive decision-making, with rewards and punishments. The elements of RL can be formalized using the *Markov decision processes* [3] framework. MDPs are a standard formalism for learning sequential decision-making and are extensively used in decision-theory planning, RL and other stochastic domain learning problems.

MDP model has the following basic components:

1. *States* **S** - The world is modelled as a finite set of states that the agent can be in.
2. *Actions* **A** - Finite set of available actions for every state.
3. *Transition Probabilities* $\mathbf{T}(\mathbf{s_t}, \mathbf{a_t}, \mathbf{s_{t+1}})$ - Gives the probabilistic understanding of how the execution of actions can affect the state of the agent. An action $a \in A$ in a state $s \in S$ results in a *transition* from current state s to new state $s^{'}$. The transition function T defines a proper probability distribution over all possible next states. The *Markov property* ensures that given the current state and action, the next state is independent of all the previous states and actions. So the current state captures all relevant information to predict the next state which does not depend on prior history.

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, ...) = P(s_{t+1}|s_t, a_t) = T(s_t, a_t, s_{t+1}) \tag{1.1}$$

4. *Reward function* $\mathbf{R}(\mathbf{s_t}, \mathbf{a_t})$ - Gives the immediate reward for being in current state, or performing some action in this state.

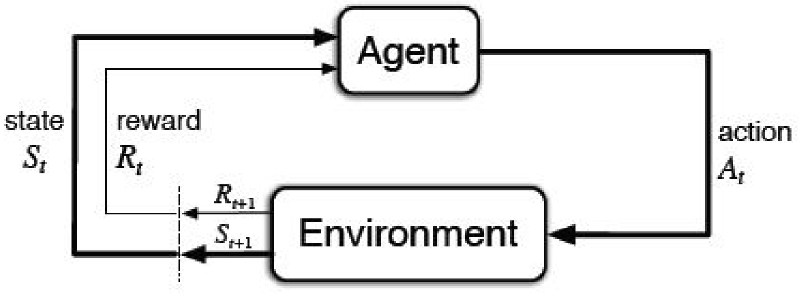


Figure 1.1: Agent-environment interaction in a Markov decision process [3]

These basic elements can then be used to formally describe the MDP framework.

Definition 1.2.1. ***A Markov decision process*** *is a tuple $\langle S, A, T, R \rangle$ in which S is a finite set of states, A a finite set of actions, T a transition function defined as $T : S \times A \times S \rightarrow [0, 1]$ and R a reward function defined as $R : S \times A \times S \rightarrow R$*

Reinforcement Learning assumes that the world behaves like an MDP model but is associated with the more difficult setting in which no (prior) knowledge about the MDP is presented. The agent can act, but the transition model is unknown. The current reward state can be observed but the reward function is unknown. The agent has to interact, or experiment with the environment (i.e. the MDP), in order to gain knowledge about how to optimize its behavior, being guided by the feedback (rewards). The goal is to find an optimal policy, that maps states to actions, to maximize some function (e.g., the mean or expected discounted sum) of the sequence of rewards.

1.3 Returns and Episodes

Reinforcement Learning problems can be divided into two distinct type of interactions - *episodic* and *continuing* tasks.

- *Episodic tasks* - Interactions that can be broken down into sequences, called *episodes* in which the agent interacts with the environment until it reaches a certain *terminal* or *goal* state that initiates a reset to its *initial* state. Each episode begins independently of how the previous one concluded, so all episodes have the same terminating state, but different rewards for different outcomes. Navigating through maze, or games with restarts, are examples of repeated interactions that constitute episodic tasks.
- *Continuing tasks* - Interactions that cannot be broken down into distinguishable sequences, they continue indefinitely. For example, on-going process control task or robots with long life span will perform such tasks.

We seek to maximize the *expected return* which can be defined as some specific function of the reward sequence. Different models of optimal behaviour result in different definitions [8] of this expected return. An usual setting is to select actions such that the sum of rewards it obtains over the future is maximized. A weighting factor called *discounting rate* γ $(0 \leq \gamma \leq 1)$ is applied to determine the current value of the future rewards. The agent chooses A_t to maximize the expected discounted rate:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1.2}$$

Convergence conditions must be met so that the expected return is finite. For episodic tasks, the sum surely converges at some point when the agent reaches the terminal state and the return is sum of finite rewards. In case of continuing tasks, the discounting rate γ must be strictly lesser than 1 to ensure convergence. We must note that the tuning of γ affects the form of optimal solution. Optimizing with very low γ values, can cause the policy to be myopic and greedy resulting in poor performance if long-term rewards are important.

1.4 Policies and Value functions

Given an MDP, $\langle S, A, T, R \rangle$, *policy* is a mapping from states to probabilities of selecting each possible action. So it governs what action a should be selected by the agent when in a given state s and summarizes the decision-making protocol of the agent. The policy $\pi(s, a)$ can be stochastic or deterministic. RL methods describe how the agent's policy is modified due to its experience.

An important concept in RL is the *value function.* It estimates how good it is for the agent to be in a given state, undertaking a given action. It differs from reward function which is immediate in nature, as value functions account for the effect of future rewards, determining the long-term desirability of a given state and is defined in terms of expected return. The *value* of a state s acting under a policy π is the expected return when the agent starts in state s and follows policy π thereafter. In the MDP-framework, *state-value function for policy* π can be formally defined as $v_\pi(s)$:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s\right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right] \tag{1.3}$$

Similarly, a more specific function called *action-value function* $q_\pi(s, a)$ is defined that finds the expected return when the agent starts in state s, performs action a and follows policy π thereafter.

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a\right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right] \tag{1.4}$$

1.5 Bellman Equation

The value functions v_π and q_π can be estimated from experience. Estimation methods such as *Monte Carlo methods* that average over many random samples of actual returns maybe used, or for tasks with many states, v_π and q_π can be kept as parametrized functions (with lesser parameters than states).

$v(s)$ is the expected return when beginning from a state s. Except when in the terminal state, the agent moves from a state s to s' and gains a reward r in the process. Now, $v(s^{'})$ is the expected return for state $s^{'}$). Thus, the value functions satisfy recursive relationships and for any given policy π and a state s, the following consistency holds:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi \left[G_t|S_t = s\right] \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} r_{t+1} + \gamma^k R_{t+k+1}|s_t = s\right] \\ &= \sum_a \pi(s,a) \sum_{s^{'}} P(s,a,s^{'}) \left[R(s,a,s^{'}) + \gamma \mathbb{E}_\pi \left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+2}|s_{t+1} = s^{'}\right\}\right] \end{aligned} \tag{1.5}$$

This formal description of the value function can be expressed as

$$v_\pi(s) = \sum_a \pi(s,a) \sum_{s^{'}} P(s,a,s^{'}) \left[R(s,a,s^{'} + \gamma v^\pi(s^{'})\right] \tag{1.6}$$

which is known as the *Bellman Equation* for $v_\pi(s)$. It expresses a recursive relationship between the value of the current state to the value of its successor states. $v_\pi(s)$ is the unique solution to its Bellman Equation.

1.6 Optimal Policies and Optimal Value Functions

The objective of RL problems is to find a policy that helps to maximize the reward in the long run. A policy π is better than (or equal to) another policy $\pi^{'}$, if its expected return is greater than (or equal to) that of $\pi^{'}$, for all states. That is, $\pi \geq \pi^{'}$ if and only if $v_\pi(s) \geq v_{\pi'}(s^{'})$ for all states $s \in S$.
The *optimal policy* π_* is better than or equal to all other policies. It's state-value function is called the *optimal state-value function* $v_*(s) = \max_\pi v_\pi(s)$ for all $s \in S$.
All optimal policies have the same *optimal action-value function* $q_*(s,a)$ as

$$q_*(s,a) = \max_\pi q_\pi(s,a) = \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a\right] \tag{1.7}$$

The optimal state-value function and optimal action-value function are related as

$$v_*(s) = \max_{a \in A} q_*(s,a) \tag{1.8}$$

The goal of the agent is to maximize its reward by approaching the optimal policy through learning. Now that we have formalised the goal through the Bellman Equation, the next sections will give an overview of algorithms that can be applied to utilise the agent's experience to estimate terms in the Bellman equation and eventually solve it for optimality.

1.7 Value-function approaches

The value function based algorithms attempt to estimate $v^*(s)$ or $q^*(s, a)$ and the following sections discuss the three main classes under this.

1.7.1 Dynamic Programming-Based Methods

These are *model-based* solution methods that assume a standard MDP $\langle S, A, T, R \rangle$ is available with finite and discrete state and action sets. So, it is assumed that a model of the transition probabilities $T(s^{'}, a, s)$ and the reward function $R(s, a)$ is available in order to find the value function. Classical DP algorithms are of limited utility in RL both because of their assumption of a perfect model and because of their great computational expense. However, they are still very relevant as they define fundamental computational mechanisms that maybe used even when no model is available. RL adds to MDPs the concepts of sampling and exploration to handle approximation and incomplete information, as discussed in later sections. Two core DP methods are policy iteration [10] and value iteration [11].

1.7.1.1 Policy-iteration

Policy iteration alternates between two interaction phases -*policy evaluation* and *policy improvement.* The *policy evaluation* estimates the utility of the current policy π (i.e calculates $v_\pi(s)$). Every one-step transition from s is considered to update $v_k(s)$ and new estimator $v_{k+1}(s)$ is produced. Such methods can thus be thought to be *bootstrapping* in the sense that a series of estimators are computed based on previous estimations. Using the information in $v_\pi(s)$, the *policy improvement* step thus seeks to find an improved policy $\pi^{'}$ from the current policy π. So a new policy $\pi^{'}$ is found that is better than the original policy π, by making it greedy with respect to the value function of the original policy.

Policy evaluation (E): $v^\pi_{k+1}(s) = \sum_a \pi(s, a) \sum_{s'} P(s, a, s^{'}) \left[R(s, a, s^{'} + \gamma v^\pi_k(s^{'})\right]$

Policy improvement (I): $v_{\pi'}(s) = \max\limits_a \sum_{s^{'}, r} P(s, a, s^{'}) \left[R(s, a, s^{'} + \gamma v_{\pi'}(s^{'})\right]$

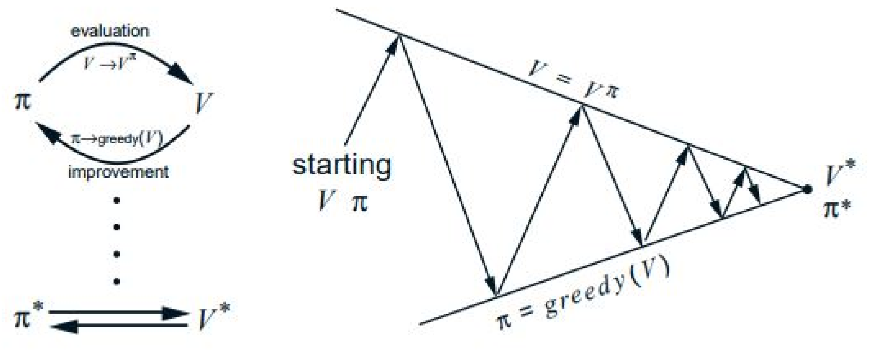


Figure 1.2: Convergence of value function and policy to optimality [3]

So first, a policy determines the value function, and in return, a value function is used by the policy for selection of good actions. A sequence of monotonically improving policies and value functions are obtained as: $\pi_0 \overset{E}{\to} v_{\pi_0} \overset{I}{\to} \pi_1 \overset{E}{\to} v_{\pi_1} \overset{I}{\to} ...\pi_* \overset{E}{\to} v_{\pi_*}$

The two steps of policy evaluation and policy improvement are iterated till convergence.

1.7.1.2 Value-iteration

In this algorithm, a simpler update operation is used that combines policy improvement and truncated policy evaluation steps till it converges to $v_*(s)$

$$v_{k+1}(s) = \max_a \sum_{s'} P(s,a,s') \left[R(s,a,s' + \gamma v_k(s') \right] \tag{1.9}$$

1.7.2 Monte Carlo Methods

The Monte Carlo methods are *model-free* methods that do not require the model of the environment and are learned directly by interaction with the environment in simulation. The value functions and optimal policies are learnt from experience in the form of sample episodes, and hence applied typically to episodic tasks. The MC methods sample and average returns for each state-action pair. For example, in MC policy evaluation, the state-action value of a given state action pair could be estimated by averaging all the returns that were received when starting from them. So the long-term reward is treated as a random variable and the sampled mean is taken as its estimate.

1.7.3 Temporal Difference Methods

These methods combine Monte Carlo (MC) ideas and Dynamic Programming (DP) principles. Temporal Difference (TD) methods can learn directly from interaction with the environment with no prior model, and also uses the bootstrapping features of DP methods to update estimates based on other learned estimates. While MC algorithms increment the value-function $v(s_t)$ at the end of an episode, at the next time step $t+1$ TD methods form an immediate target and make an update with the observed reward R_{t+1} and the current estimate $v(S_{t+1})$. Using a policy π to choose an action a in s_t

$$v(S_t) = v(S_t) + \alpha \left[R_{t+1} + \gamma v(S_{t+1}) - v(S_t) \right] \tag{1.10}$$

where $\alpha > 0$ is a step-size parameter. We can observe that this is a *boot-strapping* method as the updating is done partly using an existing estimate instead of a final reward. TD learning methods can either be *on-policy* or *off-policy.* In on-policy learning, we learn the value of the policy being carried out by the agent including the exploration steps, an example is *Sarsa.* Sarsa is on-policy as it updates the Q-values using the Q-value of the next state s' and the current policy's action a'', so it estimates the return for state-action pairs assuming that the current policy continues to be followed. An off-policy learner learns the value of the optimal policy independently of the agent's actions, an example is *Q-learning.* Here, the Q-values are updated using the Q-value of the next state s' and the greedy action a', it estimates the return for state-action pairs assuming a greedy policy would be followed, and is independent of the policy being currently followed. Hence, off-policy methods are able to update the estimated value functions using hypothetical actions, that have not actually been tried, whereas on-policy methods update value functions based strictly on experience.

Chapter 2

Reinforcement Learning in Robotics

2.1 Challenges

Reinforcement learning has the potential to allow robots to be truly autonomous with minimal human intervention [6]. However, robotic applications of reinforcement learning are very challenging as the learning algorithms have to deal with mechanical hardware constraints, high-dimensionality, real-time requirements, expensive costs of interactions with environment and limited life-span. In robotic systems, the state and action spaces are large, continuous and scale exponentially as the number of degrees of freedom and joints of the robot increases [11]. Continuous state-action space has to be first discretized to consider a resolution for proper representation of the control required over the robot. For instance, if each dimension is discretized into 10 levels, then there will be 10^n possible states for a n-dimensional state space. So the robot usually operates in a very high-dimension space consisting of internal states (such as joint angles, joint velocities, end-effector pose and orientation etc) and external states (such as object locations, other robots etc). An example of a ball-paddling task [6] is shown in Fig 2.1. The given system has 7 degrees of freedom, and the robot's state will be represented by the joint angles and velocities for each of the DOF, along with the position and velocity of the ball. The actions of the robot are the torques and accelerations that are generated as motor commands. A total of 20 state dimensions and 7-dimensional continuous actions can be identified for this task, which could be even more complex if other elaborate tasks are chosen.

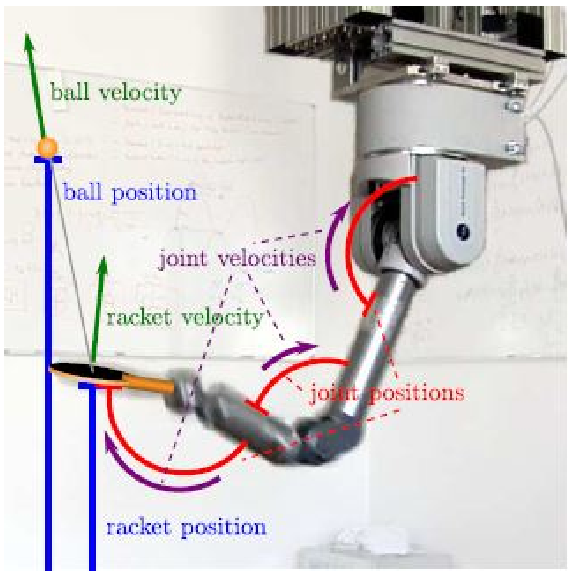


Figure 2.1: State-space modelling for the ball-paddling reinforcement learning task [6]

Exploring the complete state-action space is computationally expensive and data-extensive, as the number of dimensions increase. This leads to the *curse of dimensionality*, a term first used by Bellman [11] in context of optimal control in discrete high-dimensional spaces that exhibited exponential increase of state and action spaces. Learning in such high-dimensional space is necessary and extremely complex in robot reinforcement learning as data collection must be performed throughout the state-space to ensure global optimality.

Classical reinforcement learning framework also depends heavily on the exploration strategies to identify optimal behaviour [6]. However, in the context of robotics, there are heavy constraints on the interaction between the learning algorithm and the robot set-up since arbitrary or extensive exploration maybe both costly as well as damaging for the robot. Real-world risks of wear and tear, careful maintenance and repair that arise from repeated interaction with the environment, makes it necessary to take the hardware constraints into consideration in order to allow safe exploration strategies in the learning process. Repeated interactions between the robot system and the environment during the learning process easily impose practical limitations in robotic systems as many trials can be mechanically infeasible, expensive and also, time consuming. So, in robot RL, it is essential to focus on sample efficient algorithms that can learn from a minimal number of trials [20].

Value-function based approaches discussed in the earlier sections, struggle with the challenges faced in robot RL as they need a complete coverage of the state space and thus cannot be easily applied to high-dimensional robotics. Another significant disadvantage is the error-propagation in value functions [6]. Any error in the value function will swiftly propagate throughout the policy. Minor changes in the estimated value function can have disproportionately large effects on the policy. This brings faster, globally optimal solutions but large policy deviations are potentially hazardous when applied to real systems. In comparison with the value-function based approaches, *Policy Search (PS)* approaches use parametrized policies and focus on the current policy and its neighbourhood to eventually improve performance [12]. PS methods are implemented directly in the parameter space of the parametrized policies and do not involve learning the value function. Parametrized PS methods reduce the search space of possible policies and thus are able to scale the RL problem into high-dimensional continuous space [21, 37]. Besides, policies have fewer parameters than value functions, and often the parameters grow linearly in count, wrt the state dimensions, whereas value functions see an exponential growth of parameters.

This chapter gives an overview of policy-search methods [12, 14, 21, 23, 24] which are more feasible than value-function approaches in solving the challenges of robot RL. The final sections of the chapter focus on data-efficient policy search RL approaches [17, 20, 25, 32] that help reduce number of trials required for policy-searches and hence are more suitable for robotics.

2.2 Policy Search Approaches

2.2.1 Optimization in Policy Space

Policy search methods operate direct searches in the policy space [24]. The general philosophy behind these approaches resonate Vapnik's ideas for support vector machines (SVMs) [45]: *"When solving a problem of interest, do not solve a more general problem as an intermediate step."* In the context of reinforcement learning, it means that there involves no evaluation of the Markov model transition matrices or estimation of states or state-action values. In a very general, black-box sense, policy defines how to choose actions given a complete history.

Policy searches use parametrized policy and obtain locally optimal solutions through local explorations, unlike whole state-space exploring in value-function methods. Hence the cumulative reward is maximized directly by optimization of the policy parameters without estimating and improving the value function [12]

The objective of policy search methods is to optimize the parameters θ of a policy $\pi_\theta(a_t|x_t)$. It defines a probability distribution over the robot actions a_t conditioned on the state s_t at every time step t while executing a task. Let the trajectory of states and actions be $\tau = (x_1, a_1, ..., x_T, a_T)$. With a given task cost function $l(x_t, a_t)$, the total trajectory cost can be expressed as $l(\tau) = \sum_{t=1}^{T} l(x_t, a_t)$. The policy optimization is performed wrt the expected cost of the policy

$$J(\theta) = E_{\pi_\theta}\left[l(\tau)\right] = \int l(\tau)p_{\pi_\theta}(\tau)d\tau \tag{2.1}$$

where $p_{\pi_\theta}(\tau)$ is the policy trajectory distribution given the system dynamics $p(x_{t+1}|x_t, u_t)$:

$$p_{\pi_\theta}(\tau) = p(x_1)\prod_{t=1}^{T} p(x_{t+1}|x_t, a_t)\pi_\theta(a_t|x_t) \tag{2.2}$$

Most standard policy search approaches [12] seek to directly optimize this objective. The next sections describe them in more detail.

2.2.2 Typical Policy Representations

Policy representations [12] for policy search can be either *time-independent* $\pi(x)$ where the same policy is applied for all time-steps, or *time-dependent* $\pi(x, t)$ where individual policies are used for different time-steps. The three main policy representations of robot learning are presented below:

- **Linear Policies**- Simple time-independent controllers where the policy π only linearly depends on the policy parameters.

$$\pi_\theta(x) = \theta^T\phi(x) \tag{2.3}$$

  where ϕ is the basis function vector. Use of linear policies is restricted to problems where the appropriate basis functions are known.

- **Radial Basis Functions Networks**- A non-linear time-independent policy representation is a radial basis function (RBF) network. RBF policy $\pi_\theta(x)$ is given as:

$$\pi_\theta(x) = w^T\phi(x) \tag{2.4}$$

$$\phi_i(x) = exp(-\frac{1}{2}(x-\mu_i)^T D_i(x-\mu_i)) \tag{2.5}$$

  where $D_i = diag(d_i)$ is the diagonal matrix. The policy parameter vector θ is $\theta = \{w, \beta\}$ where $\beta = \{\mu_i, d_i\}_{i=1,...,n}$ express the parameters of the basis function which also need to be learned. RBF is a powerful representation but suffer due to the higher number of non-linear parameters which make them difficult to learn. Also, they work locally and are hard to scale to high-dimensional spaces.

- **Dynamic Movement Primitives**- DMPs are widely used time-dependent policy representations that use non-linear dynamical systems (like a spring–damper) for generating the robot motion. Derivations and full explanations are omitted here for the sake of brevity, only the policy representation by DMP is shown below:

$$\pi_\theta(x_t, t) = \tau^2 \alpha_y(\beta_y(g - y_t) - \dot{y}_t) + \tau^2 f(z_t) \tag{2.6}$$

where w_i are the *shape parameters* that influence the shape of movement by modulating the acceleration profile, g is the goal parameter that denote the final position of movement, τ is the time-scaling coefficient of the DMP, coefficients α_y, β_y define the spring and damping constants respectively and f_t is non-linear forcing function that modulates the damper system. This function depends on the phase variable z_t that is used to scale the execution speed of the movement, i.e $f_t = f(z_t)$ and is constructed as the weighted sum of K basis functions ϕ_i. The policy parameters θ to learn a DMP are generally given by the weight parameters w_i but may also contain the goal g and temporal scaling parameter τ.

The following sections briefly describe the two main approaches in policy search - *model-free* and *model-based* policy searches.

2.2.3 Model-Free Policy Search

In this, no model approximations are necessary and the samples are used to directly update a policy. Stochastic trajectory generation is employed and trajectories are sampled from the real robot, without explicitly requiring any system model. Learning of a policy is simpler than model-learning and thus model-free approaches are widely used. It can be summed in the following three steps:

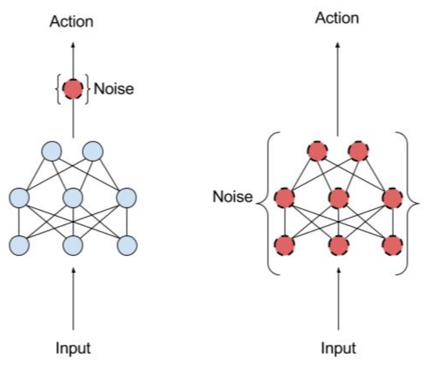


Figure 2.2: Action space (left) vs Parameter space(right) [47]

1. **Explore** - In this phase, trajectories are generated that follow the current policy. To avoid excess exploring, stochastic exploration is used to implement local searches only. Exploration maybe either in *action space*, where an exploration noise is added directly to the executed actions which changes the likelihood linked with each action, that the agent might take, or the exploration can be in *parameter space* where noise is added directly into the parameters of the agent, thus altering the type of decisions it undertakes. Figure 2.2 demonstrates it. Since parameter space exploration noise is added in a deliberate way to the parameters of the policy, it makes an agent's exploration consistent across different time-steps, whereas adding noise to the action space leads to more unpredictable exploration which isn't correlated to anything unique to the agent's parameters. Policy searches may also be differentiated as *step-based exploration* where different noise can be added (in either space) at each time-step, and *episode-based exploration* where the noise is added only at the start of every episode and hence only parameter space exploration is possible.

2. **Evaluate** - The quality of the executed policy is checked in this step. It may either be *episode-based* where the quality of the parameter vector θ that has been used during the episode is assessed, or the evaluation maybe *step-based* which checks how good the state-action pairs are, in terms of assessing the state-action value function (usually Monte-Carlo estimates of Q is used instead of value-function approximations).

3. **Update** - In the update stage, the parametrized policies with parameters θ_i are optimized locally by evaluating the required change $\Delta\theta_i$ that improves the expected long-term return. The iterative update takes the form $\theta_{i+1} = \theta_i + \Delta\theta_i$. Various optimization techniques are used for this update. *Policy gradient* [13] methods are a common approach that use gradient descent to optimize the parametrized policies with respect to the expected return. Such algorithms find a local maximum of the expected return J_θ by ascending the gradient of the policy wrt the parameter θ. So the gradient update is given by $\Delta\theta = \alpha\nabla_\theta J_\theta$ where α is an user-defined learning rate and $\nabla_\theta J_\theta$ is the policy gradient. Estimating the gradient can be achieved using various algorithms such as finite difference policy gradient [14] or the Likelihood-ratio policy gradients [15]. Policy update can also be achieved by using *Natural Gradient*[16] approaches, which often has faster convergence than traditional gradient. If it is desired to avoid specifying the learning rate, which is necessary in the gradient methods, *Expectation–Maximization* approaches can also be applied to infer a new policy where the parameter update is evaluated as a weighted maximum likelihood estimate having a closed form solution for the used policies. A lot of other different techniques can be employed for the policy update and they can be broadly grouped as *black-box* or *white-box* approaches [6]. Black-box methods use stochastic optimization that estimates the expected return by sampling, while the white-box techniques use the internal properties of the RL problem, for example the Markov structure of the problem.

2.2.4 Model-Based Policy Search

Sampling trajectories in model-free policy searches maybe problematic when dealing with mechanical robotic systems as it involves direct interaction with the robot and is time-consuming as well susceptible to damage in the robot. The sample inefficiency is overcome in model-based policy searches by using observed data to formulate forward model of the robot, and then learning the policy. The state evolves using Markovian dynamics and episodic learning is followed to reset the robot to its initial state after every policy execution. The learned policy is derived from the internal simulations of the robot using the learned model, as shown in Figure 2.3.

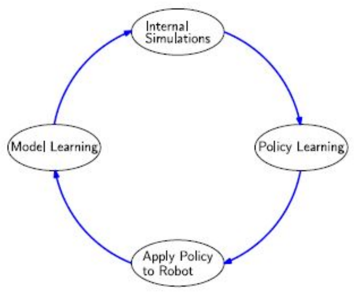
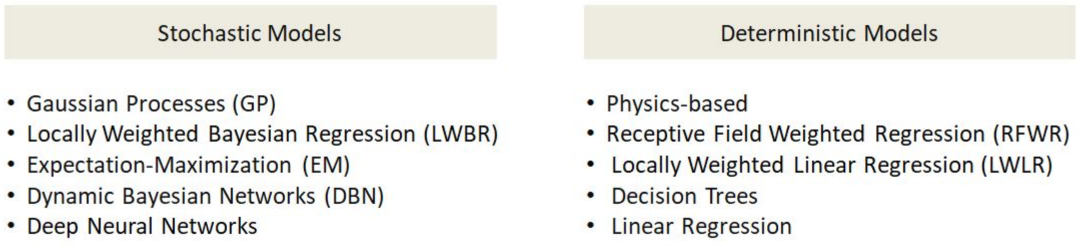


Figure 2.3: Model-based RL approaches[4]

There is a strong dependency on the quality of the learned model and hence, a lot of literature is focused on model building [27]. This is because only the policy execution is implemented on the physical robot, whereas policy learning and simulations use the learned model. So the accuracy of the model directly impacts the learning. The transition model may be built by using deterministic or stochastic approaches. Deterministic models do not depend on a random variable for the prediction of the next state, and the prediction of the model will always be the same for given initial state and action. On the other hand, stochastic models account for uncertainties to get a more or less accurate estimate of the actual robot dynamics by using predictions that are defined by a probability distribution over the future state. Figure 2.4 gives a brief overview of approaches.



Stochastic Models	Deterministic Models
• Gaussian Processes (GP)	• Physics-based
• Locally Weighted Bayesian Regression (LWBR)	• Receptive Field Weighted Regression (RFWR)
• Expectation-Maximization (EM)	• Locally Weighted Linear Regression (LWLR)
• Dynamic Bayesian Networks (DBN)	• Decision Trees
• Deep Neural Networks	• Linear Regression

Table 2.4: Overview of transition models literature [27]

To reduce the impact of model errors, the probabilistic models with uncertainty about the underlying transition dynamics are preferable to deterministic models. The state-of-the-art approach for learning of transition models that this thesis will focus on is *Gaussian Processes* [29] and it is briefly discussed next.

Gaussian Process Regression A Gaussian Process [29] can be defined as "a collection of random variables, any finite number of which have (consistent) joint Gaussian distributions." Here the GP is placed over transition functions. A GP is completely described by its mean and a positive semi-definite covariance function/kernel. The prior mean function of a GP model is assumed to be zero in most cases and a very common choice for kernels are those that belong to the exponential family. Given n training inputs $\tilde{\mathbf{X}} = [\tilde{x_1}, ..., \tilde{x_n}]$ and corresponding training targets $\mathbf{y} = [y_1, .., y_n]^T$, the posterior GP hyper-parameters (length-scales, signal variance and noise variance) are learned using evidence maximization. The posterior GP is a one-step prediction model and the successor state x_{t+1} is Gaussian distributed:

$$p(x_{t+1}|x_t, u_t) = \mathcal{N}(x_{t+1}|\mu^x_{t+1}, \Sigma^x_{t+1}) \tag{2.7}$$

where the mean and variance of the GP prediction are:

$$\mu^x_{t+1} = k_*^T K^{-1} y \tag{2.8}$$

$$\Sigma^x_{t+1} = k_{**} - k_*^T K^{-1} k_* \tag{2.9}$$

with $k_* := k(\tilde{\mathbf{X}}, \tilde{x}_t), k_{**} := k(\tilde{x}_t, \tilde{x}_t)$ where $\mathbf{K}$ is the kernel matrix with entries $K_{ij} = k(\tilde{x}_t, \tilde{x}_t)$

After the transition dynamics have been modelled, long-term predictions using the models can be achieved by using **Sampling-based Trajectory Prediction**- PEGASUS (Policy Evaluation-of-Goodness and Search Using Scenarios) [23] or **deterministic approximate inference methods** such as linearisation, moment matching, or the unscented transformation.

The final policy update step maybe implemented either gradient-free update, such as the Nelder-Mead [22] algorithm related to simulated annealing, or gradient-based update, which could be analytically computed or be a sample-based estimation of the policy gradient. Discussions on model-based policy searches are continued in the next section.

Approach	Advantages	Disadvantages
Model-based	Fewer trials for the robot Faster convergence to optimality	Dependency on dynamic model Model accuracy impacts task learning
Model-free	No prior knowledge required Easy to implement	Slow learning convergence High risk of damage and wear-tear of robot

Table 2.1: Comparison of model-based and model-free RL approaches

2.3 Data-Efficient Reinforcement Learning

Traditional RL algorithms suffer from the limitation of data-inefficiency, since a large number of interactions with the environment and a good understanding of the system are required to solve a given task [6]. Such data-intensive learning methods are a major disadvantage when dealing with mechanical systems and fragile robots as repeated physical interactions with the environment maybe problematic. Robots are easily subject to wear and tear and, hence, millions of interactions may change or damage the system. This data inefficiency makes learning in real control/robotic systems without task-specific priors impractical and prohibits RL approaches in more challenging scenarios, for example in robot damage recovery. Data-efficient learning methods are of great significance in such situations, where the robot has to quickly find a compensatory behaviour when facing an unforeseen environment that was not anticipated earlier. Current damage recovery focus on diagnosis-approaches [44] which use costly self-monitoring sensors and implement pre-designed algorithms to handle anticipated dangerous circumstances. However, it is much more efficient when learning methods are used instead, where the robot creatively learns by trial-and-error learning of the damaged environment. But the situation has to be learnt and overcome in a few trials [32] and it is ineffective to use traditional RL approaches that require many interactions to converge.

Direct policy search, as introduced in the previous section, are the most suitable approach for RL in robotics since it scales well to high-dimensional and continuous state-action spaces [12]. They are also useful in attempting to minimize the number of trials and can be seen as black-box optimization problems that seek to maximize the reward simply by optimizing the policy parameter θ. There exists several direct PS methods that are very data-efficient. Episodic Natural Actor-Critic (eNAC)[40] and Episodic REINFORCE [41] are two important policy gradient-based RL that estimate the policy gradient with respect to the policy parameters, and perform gradient descent in policy space. Compared to direct policy searches, model-based policy searches are even more data-efficient as they aim to reduce the number of interactions required by learning models of the underlying system dynamics [17, 20]. These models can serve as faithful proxy of the real environment, and good policies can be obtained from the model without additional interactions with the real system. Hence, model-based RL methods are more data-efficient as learning the dynamical model allows better extraction of information from available data. However, learning models of the underlying transition dynamics is hard and inevitably leads to model errors. Learning the dynamic model with better accuracy would need higher number of points, and therefore increased interaction time. These challenges from errors and uncertainties can be handled by using probabilistic models [29] that accounts for uncertainty in the model, and is the main focus of data-efficient RL techniques.

A variety of data-efficient RL methods [17, 18, 20] use Gaussian Processes (GPs) [29] to model system dynamics. In this section, we briefly introduce important data-efficient policy searches - Bayesian optimization [30] and PILCO [18]. Next, we focus on the state-of-art data-efficient policy search method, Black-Box Data-efficient Policy Search for Robotics [20] which forms the basis of the thesis work planned in this report.

2.3.1 Bayesian optimization

Bayesian optimization(BO) [30], a model-based approach to black-box optimization under uncertainty has been found to be highly suitable for robotic applications [32, 33, 34] due to its ability to find model parameters in a small number of experiments. It is a state-of-art global optimization method that is used to evaluate the optimal values of functions which are expensive to compute. In context of robot learning, BO is used as a direct policy search approach that can be applied in continuous action and state spaces, and can handle high-dimensions. Instead of modelling the system dynamics, BO directly builts a probabilistic model of the reward function using past interactions. So, without actually evaluating the expensive reward function, this model is used to find the optimal parameters of the policy such that the reward function is maximized. Since a model is used, the number of interactions is also reduced compared to other optimization techniques. The probabilistic framework also permits the integration of prior knowledge directly to the learned model. The most common probabilistic models used in BO are Gaussian Processes [29], which allows prior inclusion easily while others such as random forests may also be used [30]. The prior selection and choice of the acquisition function which can be used to build the utility function are two important features of a BO. In fact, it is termed as *Bayesian* as the next point is selected by computing a posterior distribution of the objective function from the likelihood of the data already obtained and a prior on the choice of function. Bayesian optimization have several successful applications in robot locomotion [31, 32] and gait optimization [33, 34].

2.3.2 PILCO (Probabilistic Inference for Learning Control)

PILCO [18] is a state-of-art data-efficient indirect policy search method that uses Gaussian processes [29] to learn model dynamics in a continuous space and do not need exact value function models. It draws motivation from [19] that considers model uncertainties as uncorrelated noise. PILCO significantly reduces model bias by taking into account the model uncertainty and using probabilistic models. Dynamic systems of the form $x_t = f(x_{t-1}, u_{t-1})$ are used with continuous-valued states $x \in \mathbb{R}^D$, controls $u \in \mathbb{R}^F$ and unknown transition dynamics f. An optimal policy π parametrized by θ, is to be obtained such that the long-term expected cost is minimised. The main components of this algorithm are (i) GP-based dynamic model (ii) analytic approximate policy evaluation, and (iii) gradient-based policy improvement. The probabilistic forward model adopts deterministic approximate inferencing for predicting the sequences of states as a result of following a current policy. The gradient computation is done analytically and a large number of policy parameters can be learned. Although PILCO is highly data-efficient, it is computationally expensive and takes time due to the deterministic approximate inference for policy evaluation. Analytical gradient-based policy update limits its use to differentiable reward functions and thus it cannot handle all kinds of rewards and policies.

2.3.3 Black-Box Data-efficient Policy Search for Robotics (Black-DROPS)

Black-DROPS [20] is a novel model-based policy search that uses purely black box optimization and has the advantages of parallel computations. Black-DROPS is highly data-efficient like PILCO but has the added advantage of not imposing any constraints on the reward or policy function, unlike PILCO. Since it is highly-parallel, it is faster when multi-core systems are available. The dynamic model and reward functions are learnt using Gaussian Processes [29], and Monte Carlo roll-outs are used for policy evaluation. For the optimization step, BIPOP-CMA-ES [26] is used, which is a variant of the successful black-box optimizer for noisy functions-CMA-ES. The best policy is applied to the system and the process continued till the task has been completely learned. The algorithm is presented here, and the complete details can be found in [20]

Algorithm 1 Black-DROPS

```
 1: procedure BLACK-DROPS
 2:     Define policy $\pi : x \times \theta \leftarrow u$
 3:     for $i = 1 \leftarrow N_R$ do
 4:         Set robot to initial state $x_0$
 5:         $D = \emptyset$
 6:         for $j = 0 \leftarrow T - 1$ do
 7:             $u_j =$random_action()
 8:             $x_{j+1}, r(x_{j+1}) =$ execute_on_robot($u_j$)
 9:             $D = D \cup \{\tilde{x}_j \leftarrow \triangle_{x_j}\}$
10:             $R = R \cup \{x_{j+1} \leftarrow r(x_{j+1})\}$
11:         end for
12:     end for
13:     while $task \neq solved$ do
14:         Model learning: train $E$ GPs given data $D$
15:         Reward learning: train 1 GP given data $R$
16:         $\theta^* = argmax_\theta \mathbb{E}\left[G(\theta)\right]$ using BIPOP-CMA-ES
17:         Set robot to initial state $x_0$
18:         for $j = 0 \leftarrow T - 1$ do
19:             $u_j = \pi(x_j|\theta^*)$
20:             $x_{j+1}, r(x_{j+1}) =$ execute_on_robot($u_j$)
21:             $D = D \cup \{\tilde{x}_j \leftarrow \triangle_{x_j}\}$
22:             $R = R \cup \{x_{j+1} \leftarrow r(x_{j+1})\}$
23:         end for
24:     end while
25: end procedure
```

Initially the probabilistic dynamic model and reward functions are learnt during the learning phase in N_R random episodes. Then, the expectation of the rollout, i.e $\mathbb{E}\left[G(\theta)\right]$ is optimized using the BIPOP-CMAES [26]. With this optimized policy, episodes are performed on the robot which provides more data for continuing the loop activity till the task is learned.

Results Black-DROPS can be implemented with any parametrized policy representation, and was successfully demonstrated [20] with Gaussian processes and feed-forward neural network policy, applied to 3 tasks- inverted pendulum, cart-pole swing-up and control of a 4-DOF arm. The first task involves learning a controller to apply the right torque to a freely swinging pendulum such that it swings up and finds balance in an inverted position. Black-DROPS was compared with PILCO for the task, and both approaches solved it within 3 episodes. But when multiple cores are used, Black-DROPS significantly outperformed PILCO in computation speed (1.25 to 3.33 times faster if 12 cores are available). With higher multiple runs, Black-DROPS always succeeded to find a working policy whereas PILCO failed once in 10 runs. In the cart-pole task, the pendulum is placed on a moving cart and the goal is to learn a controller that can apply the right horizontal forces to the cart to balance the pendulum in an inverted position, in the middle of the track. The comparison with PILCO was similar as in the first task, with Black-DROPS outperforming it slightly by finding better performing policies, which could be attributed to search algorithm being more global than gradient-based optimizers of PILCO. Similar to the first task, Black-DROPS is around 1.6 times faster when multiple cores are available. Black-DROPS was also tested with complex robots, and applied to a velocity controlled 4-DOF robotic arm, with the aim to learn a controller that enables the end-effector to reach a desired position quickly. In this task, Black-DROPS demonstrated the importance of taking into account the uncertainty of the model, (i.e variance) in policy optimization. The performance of Black-DROPS is compared with a variant of itself, where deterministic GP models of the dynamics (i.e., using only mean of GPs) is used, and Black-DROPS is computationally faster than this variant.

Black-DROPS removes all constraints to choice of reward functions and policy types, and has successfully demonstrated to be highly data-efficient, as promising as the state-of-art PILCO algorithm, and performs better when multi-core systems are available. Thus, this data-efficient learning algorithm will form the basis of the thesis work being planned. The following section describes an extension of this algorithm, which will be used in the thesis work.

2.3.4 Using Parameterized Black-Box Priors to Scale Up Model-Based Policy Search for Robotics

Black-DROPS is a highly data-efficient policy search with improved computation speed under multiple cores, but like any model-based approach, it too suffers from the curse of dimensionality [11]. Model-based policy searches can usually be effectively used with simple models upto 10-15D of state-action space. The rising complexity in the model with every new degree of freedom, results into a very data intensive process to learn a good model as the state-space dimension increases. The integration of prior knowledge about the system being modelled, helps overcome this problem and enables effective scaling into high-dimensional spaces. In model-based policy searches, the interaction time is reduced by using priors on the model [21], such that the process begins with an initial meaningful guess of the system dynamics and learning is performed on the residual model. This section gives an overview of use of parametrized black-box priors [21] as an extension to the earlier discussed Black-DROPS algorithm [20].

Problem formulation A dynamical system of the form $x_{t+1} = x_t + F(x_t, u_t) + w$ is considered with continuous valued states $x \in \mathbb{R}^E$, controls $u \in \mathbb{R}^U$, i.i.d Gaussian system noise $\mathbf{w}$ and unknown transition dynamics F. A black-box function $M(x_t, u_t)$ having some tunable parameter ϕ_M, is now assumed as an initial dynamics model. A non-parametric model f (with hyper-parameter ϕ_K) is added to capture the inaccuracies of the initial guess M. The overall model can be expressed as $x_{t+1} = x_t + M(x_t, u_t, \phi_M) + f(x_t, u_t, \phi_K) + w$. The goal is to obtain a deterministic policy π parametrized by θ, that maximizes the expected long-term reward when adopting this policy for T time-steps. If immediate reward to be in a state x_t is taken as $r(x_t)$ then the actual function to maximize can be expressed as $J(\theta) = \mathbb{E}\left[\sum_{t=1}^{T} r(x_t)|\theta\right]$

Approaches At the start, the policy is optimized on the prior model, and then applied to real system for data collection. Using this data and prior model, a loop is initiated to learn a new model on which the policy will be optimized. This policy is applied on the real system and process is repeated till task completion. So, the policy initialisation is done from simulations while real data is exploited to explicitly learn the unknown dynamics. As the learned policy is applied to the real robot, the measured states are used either to improve the current policy or to learn an estimate of the unknown dynamics. This greatly reduces the total roll-outs to perform on the real robot and is hence an efficient approach in robotic tasks. Gaussian Processes [29] are used to approximately model the unknown system dynamics F with initial guess M. Two approaches are considered - when the parameter ϕ_M is tunable, and when it is not.

Gaussian Processes with simulator as the mean function The state and control vectors are used to form tuples $\tilde{x}_t = (x_t, u_t) \in \mathbb{R}^{E+U}$ as inputs. Difference of current and next state vectors $\triangle_{x_t} = x_{t+1} - x_t \in \mathbb{R}^E$ form the training targets. So, E independent GPs are used to model each dimension of the difference vector. Assuming $D_{1:t} = F(\tilde{x}_1), ..., F(\tilde{x}_t)$ and $M(\tilde{x})$ as the simulator function, the GP can be queried at a new input point $\tilde{x}_*$ as:

$$p(\hat{F}(\tilde{x}_*)|D_{i:t}, \tilde{x}_*) = \mathcal{N}(\mu(\tilde{x}_*),\, \sigma^2(\tilde{x}_*)) \tag{2.10}$$

The mean and variance predictions of this GP are computed using a kernel vector $\boldsymbol{k}$=$k(D_{1:t}, \tilde{x}_*)$ and a kernel matrix K with elements $K^{ij} = k(\tilde{x}_i, \tilde{x}_j)$ as:

$$\mu(\tilde{x}_*) = M(\tilde{x}_*) + \mathbf{k}^T K^{-1}(D_{1:t} - M(\tilde{x}_*)) \tag{2.11}$$

$$\sigma^2 = k(\tilde{x}_*, \tilde{x}_*) - \mathbf{k}^T K^{-1}\mathbf{k} \tag{2.12}$$

This strategy enables to successfully utilise information from prior knowledge from the simulator along with handling real-world data. Predictions from simulations are corrected whenever real-data is available, and when real-data is absent, predictions rely on the simulator. Using an exponential kernel, the best kernel hyper-parameters ϕ_K can be attained through Maximum Likelihood Estimation (MLE) for a GP (with non-tunable mean function M) by maximizing:

$$p(D_{1:t}|\tilde{x}_{1:t}, \phi_K) = \frac{1}{\sqrt{(2\pi)^t|K|}} e^{-\frac{1}{2}(D_{1:t} - M(\tilde{x}_{1:t}))^T K^{-1}(D_{1:t} - M(\tilde{x}_{1:t}))} \tag{2.13}$$

Any gradient-based optimizer can be used to compute the gradient of the above likelihood function and Rprop [35] was applied.

Algorithm 2 GP-MI Learning Process

```
 1: procedure GP-MI($D_{1:t}$)
 2:     Optimize $\phi_M^*$ by EVALUATEMODEL($\phi_M$, $D_{1:t}$) using a gradient-free local optimizer
 3:     return $\phi_M^*$
 4: end procedure
 5: procedure EVALUATEMODEL($\phi_M$,$D_{1:t}$)
 6:     Initialise E GPs $f_1, ...f_E$ as $f_i(\tilde{x}) \sim \mathcal{N}(M_i(\tilde{x}, \phi_M),\, k_i(\tilde{x}, \tilde{x}))$
 7:     for $i$ to $E$ do
 8:         Optimize the kernel hyper-parameters $\phi_K^i$ of $f_i$ given $D_{1:t}^i$ (assuming $\phi_M$ is fixed)
 9:         $lik_i = p(D_{1:t}|\tilde{x}_{1:t}, \phi_K^i, \phi_M)$
10:         return $\sum_{i=1}^{E} = lik_i$
11:     end for
12: end procedure
```

Mean function with tunable parameters If the mean function $M(\tilde{x}, \phi_M)$ is desired to have tunable parameters ϕ_M then equation 2.13 can be extended to include parametrized mean functions as:

$$p(D_{1:t}|\tilde{x}_{1:t}, \phi_K, \phi_M) = \frac{1}{\sqrt{(2\pi)^t|K|}} e^{-\frac{1}{2}(D_{1:t} - M(\tilde{x}_{1:t}), \phi_M)^T K^{-1}(D_{1:t} - M(\tilde{x}_{1:t}), \phi_M)} \tag{2.14}$$

As the robot model should be consistent in all of the output dimensions, the mean parameter is same for all the E independent GPs whereas the kernel parameter is different for every dimension. Hence, joint optimization of the mean parameters and kernel hyper-parameters for all GPs is performed. Two nested optimization loops are employed to implement gradient-based optimization for ϕ_K and black-box optimization (a variant of Subplex algorithm by NLOpt [36] is used as gradient-free optimizer) for ϕ_M , as shown in the following model-learning approach, also known as *GP-MI* (Gaussian Process Model Identification) [21]

The GP-MI learning approach combines non-parametric model learning and parametric model identification and it's advantages are that it imposes no structure on the prior model and can be used with both inverse and forward dynamics models, and generally with any black-box tunable prior model.

Policy search with Black-DROPS algorithm The Black-DROPS algorithm [20] is used to perform the policy search. There is no prior knowledge about the policy parameters, only a prior dynamic model is used.

Combining Black-DROPS with priors has outperformed similar policy search algorithms with priors to learn residual model, such as PILCO with priors [37] and PI-REM [38], and have demonstrated good efficiency to learn high-dimensional physical robots. Within 1 minute interaction time, it learnt the walking policies from scratch, of a hexapod with 48D state space and 18D action space. Black-DROPS with GP-MI has also successfully integrated diagnosis-based approaches [44] with trail-and-error learning [32] and is therefore very promising for use in diagnosis-based learning for robot damage recovery.

Chapter 3

Work Outline

3.1 Summary of objectives

The thesis aims to apply an extension of the Black-DROPS algorithm in a real-world robotic application of a grasping task. The central idea is to prepare the robot for adapting to damaging environments and recover from unpredictable injuries, without using pre-defined algorithms to overcome an anticipated danger but by learning the new environment and adopt compensatory behaviour to function normally again. It is important to apply an intelligent learning algorithm to allow quick recovery and hence a data-efficient algorithm like Black-DROPS would be highly suitable. However, such model-based policy searches face limitations when scaling up to handle complex problems in high-dimensions, currently they can be efficiently employed only for simple cases upto 10-15D state-action space. The data-efficiency is challenged when more data and computation are needed to cover the complete state-action space of a model with increased dimensionality. In such cases, prior knowledge can significantly improve the learning process. These approaches greatly reduce the search space and, thus, speed up the learning process. This thesis therefore, focuses on implementing the Black-DROPs with priors [21] to a robotic grasping task.

The thesis being planned seeks to apply such a learning approach to allow successful grasps for a universal jamming gripper [28] and its autonomous recovery from unforeseen damaging scenarios. A jamming gripper consists of a flexible membrane containing granular material that can easily harden or soften to grasp objects of varying shapes and sizes by modulating the air pressure within the membrane. Developing an exact physical model of its gripping mechanism is however difficult because it undergoes significant deformation during use. Thus, several grasping approaches based on physical models would be challenging to apply to a jamming gripper. So, we adopt a learning-based approach. Robot grasping in general conditions is a challenging problem due to a multitude of sources of uncertainty. Most robot grasping prior work [39] assumes complete knowledge of the target object geometry as 2D or 3D models. However, in real-world scenarios or unknown environments, perception of object shape is often noisy and partial. In the thesis work, Black-DROPS with priors [21] will be used with a learning-based vision system to identify the objects to grasp that will help generalize the grasping task as it can be easily applicable to previously unseen objects.

3.2 Set-up

The experiments are to be conducted on an omni-directional mobile manipulator manufactured by Kuka, to which a gripper by Empire Robotics [28] has been mounted. Figure 3.1 shows the set-up available in the lab at INRIA Nancy [46].

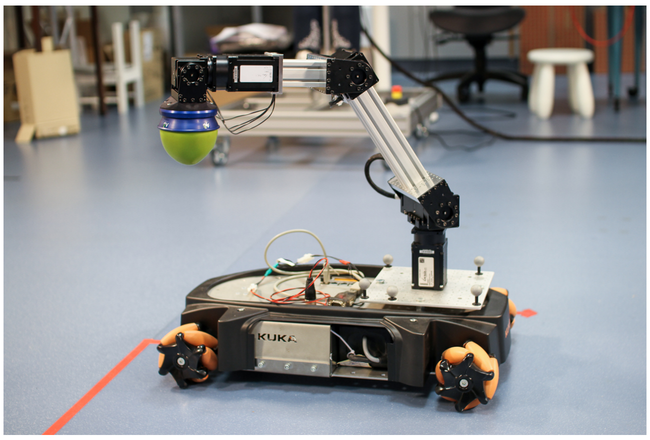


Figure 3.1: Kuka Youbot fitted with a jamming gripper [46]

The gripper [28] has an elastic membrane that contains the granular material, which can be deformed easily through three grasping modes that allows it to passively follow the shape of the object. The grasping maybe achieved by using (i) Static friction from surface contact, (ii) Geometric constraints due to interlocking and (iii) Vacuum suction from an airtight seal. Figure 3.2 demonstrates these possible modes. Positive pressure is used to reverse the transition and return to the deformable state after releasing the object.

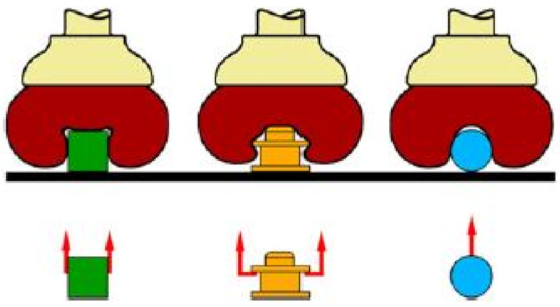


Figure 3.2: Grasping modes of the jamming gripper [28]

3.3 Work plan

The goal is to navigate the mobile manipulator over short distances to grasp small objects and depositing them at the back of the robot. The task can be divided into two stages:

- **Grasping** - A parametrized policy has to be first designed to describe the system and the grasping task. Then the parameters of the designed parametrized policy would be learnt using Black-DROPS with priors [21]. Finally, the algorithm will be extended for use with minimal resets and adopt semi-episodic learning so that the robot is able to learn the behaviour while performing the task without going back to the initial state.
- **Visually locating object using cameras** - A visual object recognition approach will be adopted. A learning-based vision system will be designed such that it is compatible with the model-based policy framework of Black-DROPS. The learning will be guided by

pixels instead of from hand-designed features, and comparisons would be drawn between both methods. Both eye-in-hand as well as eye-to-hand cases will be considered and the object tracking performance compared between them.

3.4 Schedule

	TIMELINE								
TASKS	Dec	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug
State of the art	▨	▨							
Bibliography defence		▨							
Studying available Black-DROPS codes		▨	▨						
Design of parametrized policy			▨	▨					
Learning the policy parameters				▨	▨				
Learning-based vision system				▨	▨				
Simulation & Implementation				▨	▨	▨			
Mid term presentation					▨				
Experimentation					▨	▨	▨		
Analysis of experimental results						▨	▨	▨	
Writing thesis							▨	▨	▨
Master thesis defence									▨

Table 3.3: Work schedule planned for master thesis

Bibliography

[1] Thorndike, Edward Lee. "Edward Lee Thorndike." *Anim. Intell* 1874 (1911): 1949.

[2] Bellman, Richard. *Dynamic programming* . Courier Corporation, 2013.

[3] Puterman, Martin L. *Markov decision processes: discrete stochastic dynamic programming.* John Wiley & Sons, 2014.

[4] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[5] van Otterlo, Martijn, and Marco Wiering. "Reinforcement learning and markov decision processes." *Reinforcement Learning* 12 (2012): 3-42.

[6] Kober, Jens, J. Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey." *The International Journal of Robotics Research* 32.11 (2013): 1238-1274.

[7] Dayan, Peter, and Yael Niv. "Reinforcement learning: the good, the bad and the ugly." *Current opinion in neurobiology* 18.2 (2008): 185-196.

[8] Kaelbling, Leslie Pack, Michael L. Littman, and Andrew W. Moore. "Reinforcement learning: A survey." *Journal of artificial intelligence research* 4 (1996): 237-285.

[9] Nahum Shimkin, Learning in Complex Systems, *Lecture Notes, Spring 2011*

[10] Howard, Ronald A. "Dynamic programming and markov processes." (1960).

[11] Bellman, R. "Dynamic Programming Princeton University Press Princeton." *New Jersey Google Scholar (1957).*

[12] Deisenroth, Marc Peter, Gerhard Neumann, and Jan Peters. "A survey on policy search for robotics." *Foundations and Trends®* in Robotics 2.1–2 (2013): 1-142.

[13] Sutton, Richard S., et al. "Policy gradient methods for reinforcement learning with function approximation." *Advances in neural information processing systems.* 2000.

[14] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robotics Systems*, pp. 2219–2225, Beijing, China, 2006.

[15] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 1992.

[16] S. Amari, "Natural gradient works efficiently in learning," *Neural Computation*, vol. 10, pp. 251–276, February 1998.

[17] Deisenroth, Marc Peter, Dieter Fox, and Carl Edward Rasmussen. "Gaussian processes for data-efficient learning in robotics and control." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.2 (2015): 408-423.

[18] Deisenroth, Marc, and Carl E. Rasmussen. "PILCO: A model-based and data-efficient approach to policy search." *Proceedings of the 28th International Conference on machine learning (ICML-11).* 2011.

[19] Schneider, Jeff G. "Exploiting model uncertainty estimates for safe dynamic control learning." *Advances in neural information processing systems.* 1997.

[20] Chatzilygeroudis, K., Rama, R., Kaushik, R., Goepp, D., Vassiliades, V., & Mouret, J.-B. (2017). "Black-Box Data-efficient Policy Search for Robotics". *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).*

[21] Chatzilygeroudis, Konstantinos, and Jean-Baptiste Mouret. "Using Parameterized Black-Box Priors to Scale Up Model-Based Policy Search for Robotics." *arXiv preprint arXiv:1709.06917* (2017).

[22] J. A. Nelder and R. Mead, "A simplex method for function minimization," *Computer Journal*, vol. 7, pp. 308–313, 1965.

[23] A. Y. Ng and M. Jordan, "Pegasus: A policy search method for large MDPs and POMDPs," in *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pp. 406–415, 2000.

[24] Peshkin, Leonid. "Reinforcement learning by policy search." (2003).

[25] Kamthe, Sanket, and Marc Peter Deisenroth. "Data-Efficient Reinforcement Learning with Probabilistic Model Predictive Control." *arXiv preprint arXiv:1706.06491* (2017).

[26] Hansen, Nikolaus. "Benchmarking a BI-population CMA-ES on the BBOB-2009 function testbed." *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers.* ACM, 2009.

[27] Polydoros, Athanasios S., and Lazaros Nalpantidis. "Survey of Model-Based Reinforcement Learning: Applications on Robotics." *Journal of Intelligent & Robotic Systems* 86.2 (2017): 153-173.

[28] Amend, John R., et al. "A positive pressure universal gripper based on the jamming of granular material." *IEEE Transactions on Robotics* 28.2 (2012): 341-350.

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

[30] V. M. C. E. Brochu and N. de Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," CoRR, vol. abs/1012.2599, 2010.

[31] Pautrat, Rémi, Konstantinos Chatzilygeroudis, and Jean-Baptiste Mouret. "Bayesian Optimization with Automatic Prior Selection for Data-Efficient Direct Policy Search." *arXiv preprint arXiv:1709.06919* (2017).

[32] Cully, Antoine, et al. "Robots that can adapt like animals." *Nature* 521.7553 (2015): 503-507.

[33] Lizotte, Daniel J., et al. "Automatic Gait Optimization with Gaussian Process Regression." *IJCAI.* Vol. 7. 2007.

[34] Calandra, Roberto, et al. "Bayesian optimization for learning gaits under uncertainty." *Annals of Mathematics and Artificial Intelligence* 76.1-2 (2016): 5-23.

[35] M. Blum and M. A. Riedmiller, "Optimization of Gaussian process hyperparameters using Rprop," in *Proc. of ESANN*, 2013.

[36] G. Johnson Steven, "The NLopt nonlinear-optimization package."

[37] Cutler, Mark, and Jonathan P. How. "Efficient reinforcement learning for robots using informative simulated priors." *Robotics and Automation (ICRA), 2015 IEEE International Conference on.* IEEE, 2015.

[38] Saveriano, Matteo, et al. "Data-Efficient Control Policy Search using Residual Dynamics Learning." *Proc. of IROS.* 2017.

[39] Shimoga, Karun B. "Robot grasp synthesis algorithms: A survey." *The International Journal of Robotics Research* 15.3 (1996): 230-266.

[40] Peters, Jan, and Stefan Schaal. "Natural actor-critic." *Neurocomputing* 71.7 (2008): 1180-1190.

[41] Williams, Ronald J. "Simple statistical gradient-following algorithms for connectionist reinforcement learning." *Machine learning* 8.3-4 (1992): 229-256.

[42] Mediano, Pedro Antonio Martınez. "Data-Efficient Reinforcement Learning for Autonomous Helicopters." (2014).

[43] McKenzie, Mark, et al. "Competitive Reinforcement Learning in Atari Games." *Australasian Joint Conference on Artificial Intelligence.* Springer, Cham, 2017.

[44] V. Verma, G. Gordon, R. Simmons, and S. Thrun, "Real-time fault diagnosis," *Robotics & Automation Magazine*, IEEE, vol. 11, no. 2, pp. 56–66, 2004.

[45] Vladimir Vapnik, *Statistical learning theory*, Wiley, 1998.

[46] *http : //www.resibots.eu/_static/mid_res/omnigrasper.jpg*

[47] *https : //blog.openai.com/better − exploration − with − parameter − noise/*