Proposed Methods

In general, the aim of this research is to simulate the transfer of an object into a hole using a robotic arm, and imitation learning and reinforcement learning methods are utilized to solve the problem. For simulation in discrete environments, the Q-Learning algorithm is employed, and in this thesis, the algorithm's performance is enhanced by adding guidance paths extracted from recorded videos. In Section 3-1 of this chapter, we describe the Q-Learning algorithm as the base model for solving the problem in discrete environments. In Section 3-2, we also explain the addition of guidance paths to the Q-Learning algorithm as part of the proposed method in this thesis.

The PILCO algorithm [3] is introduced as a framework for solving problems in cases where designing the robotic model's dynamics is complex or when the dynamic model is uncertain. This algorithm has been used in various problems before but has not been applied to solve the problem of transferring an object into a hole using a simulated robotic arm. Considering the inherent uncertainty in this problem, the PILCO algorithm can serve as a suitable solution for solving this problem in continuous environments. In Section 3-3, the use of the PILCO algorithm in solving the object transfer problem into a hole in continuous environments is further elaborated as another part of the proposed method in this thesis. Additionally, we explain how modifications to the cost function are made to account for state space constraints, such as walls or obstacles, in the PILCO model.

3-1- Q-Learning

Q-learning is a simple learning method for agents to optimally act in controlled Markov environments based on their experiences and the consequences of their actions, without the need to construct environment maps. This method is an incremental approach to dynamic programming that imposes limited computational requirements. It operates by iteratively improving its evaluations of specific action qualities in specific states.

The Q-Learning algorithm operates similarly to the Temporal Difference (TD) learning method. In this approach, the agent tries an action in a specific state, receives immediate rewards or penalties, and estimates the value of the state it transitions to. By repeatedly trying all actions in all states, the agent learns which cases have better discounted long-

term rewards. Q-learning is a fundamental form of learning but can serve as the foundation for much more complex approaches on its own.

Consider a computational agent moving in a finite discrete world and selecting one action from a finite set of actions at each time step. The world consists of a controlled Markov process with the agent as the controller. At step n, the agent can record state $x_n(\epsilon X)$ and choose action $a_n(\epsilon A)$ accordingly. The agent receives a probabilistic reward r_n, with the average value $R_{x_n}(a_n)$ depending only on the state and action. The world state probabilistically transitions to y_n according to equation (3-1):

Eq.(3-1) $$Prob[y_n = y|x_n, a_n] = P_{x_n y}[a_n].$$

The task an agent is confronted with is to determine an optimal policy that maximizes the expected discounted cumulative reward. By "discounted cumulative reward," we mean the rewards received at time step s. Therefore, the rewards received at present are valued less with a discount factor $(0 < \gamma < 1)\ \gamma^s$. Under policy π, the value of state x is given by equation (3-2):

Eq.(3-2) $$V^{\pi}(x) = R_x\big(\pi(x)\big) + \gamma \sum_y P_{xy}[\pi(x)]V^{\pi}(y)$$

According to this equation, the agent expects to immediately receive the reward $R_x\big(\pi(x)\big)$ for the execution of the policy π it has proposed and then, with a probability of $P_{xy}[\pi(x)]$, move to a state that has the value $V^{\pi}(y)$. The principles of dynamic programming assure us that there is at least one stable and optimal policy π^* that can be expressed as equation (3-3).

Eq.(3-3) $$V^*(x) = V^{\pi^*}(x) = \max_a \big\{R_x\big(\pi(a)\big) + \gamma \sum_y P_{xy}[\pi(a)]V^{\pi^*}(y)\big\}$$

While this relationship may appear recursive, it is, in fact, well-defined and, considering we know $R_x(a)$ and $P_{xy}[a]$, dynamic programming provides several methods to calculate V^* and a π^*. The challenge a Q-learner faces is determining π^* without knowing these values initially. Traditional methods for learning $R_x(a)$ and $P_{xy}[a]$ simultaneously with dynamic programming exist, but any assumption of certainty equivalence, meaning the computation of actions as if the current model is exact, can be prohibitively expensive during the initial stages of learning. Watkins categorized Q-

learning as incremental dynamic programming because it determines the optimal policy step by step.

For a policy π, the values of Q are defined by equation (3-4).

Eq. (3-4) $Q^{\pi}(x,a) = R_x(a) + \gamma \sum_y P_{xy}[\pi(x)]V^{\pi}(y)$

In other words, the value of Q represents the expected discounted cumulative reward for taking action a in state x and then following the policy π afterward. In Q-learning, the goal is to estimate the values of Q for an optimal policy. For simplicity, for each x and a, $Q^*(x,a) \equiv Q^{\pi^*}(x,a)$ is defined. It can be easily shown that $V^*(x) = \max_a Q^*(x,a)$, and if a^* is an action that achieves the maximum, an optimal policy can be expressed as $\pi^*(x) \equiv a^*$. The usefulness of the Q values lies in the fact that if an agent can learn them, it can easily decide what the optimal actions are. Although there may be more than one optimal policy or a^*, the Q^* values are unique.

In Q-learning, the agent's experience consists of a sequence of episodes or distinct segments. In the 'n'-th episode, the following process occurs for the agent:

- The agent observes its current state, denoted as x_n,
- Selects and executes action a_n,
- Observes the next state, denoted as y_n,
- Receives an immediate reward, denoted as r_n.
- It adjusts the Q values, denoted as Q_{n-1}, using the learning rate α_n based on equation (3-5):

Eq.(3-5)

$$Q_n(x,a) = \begin{cases} (1-\alpha_n)Q_{n-1}(x,a) + \alpha_n[r_n + \gamma V_{n-1}(y_n)] & if\ x = x_n\ and\ a = a_n, \\ Q_{n-1}(x,a) & otherwise, \end{cases}$$

The best thing the agent thinks it can do in state y can be expressed as the equation (3-6).

Eq. (3-6) $V_{n-1}(y_n) \equiv \max_b \{Q_{n-1}(y,b)\}$

However, in the initial stages of learning, the values of Q may not accurately reflect the policy they implicitly define. It is assumed that the initial values of Q or $Q_0(x,a)$ are given for all states and actions.

3-2- Proposed Method: Using Guided Paths in Q-Learning

One of the most essential components of the Q-Learning algorithm is the matrix Q. This matrix determines the value of each action in all possible states. In the typical Q-Learning algorithm, the matrix Q is initialized with zero values. However, setting an appropriate initial value for the matrix Q can have a significant impact on the robot's performance. For example, when the matrix Q is initialized with zeros, in state spaces larger than 20x20, the robot cannot reasonably move the object from an initial position outside the hole to a target position inside the hole. The proposed method presented in this thesis introduces a framework that can guide the Q-Learning algorithm to improve its performance and reduce execution time through appropriate matrix Q initialization.

The foundation of the proposed framework for solving this problem is the use of guided paths. These guided paths are included in the Q-Learning algorithm in the form of initial values for the matrix Q. Guided paths can be provided to the algorithm in two ways. In the first case, the paths that lead the object to the goal are manually designed by a human agent and provided to the algorithm. Using this method is complex, especially in state spaces with large dimensions.

In the second case, these guided paths are extracted from recorded videos and using object tracking algorithms. In the following sections, we will first explain how recorded videos can be used to create guided paths in section 3-2-1, and then in section 3-2-2, we will provide detailed explanations of how to incorporate these guided paths as initial values for the matrix Q in the Q-Learning algorithm.

3-2-1- Creating Guided Paths from Recorded Videos

Guided paths can be created in two different ways. In the first method, the path for moving the object from the initial point to the goal is defined by a human agent. This method is not very efficient in environments with large dimensions and imposes significant complexities on the human agent. Therefore, a second method is proposed in which the path for moving the object from the initial point to the goal is automatically

determined using recorded videos (demos). In this section, the goal is to explain how to use recorded videos to determine the guided path.

To create videos for the purpose of extracting the guided path, it is necessary to record videos in a space similar to the robot's state space. Considering this, an environment should be designed, which includes an object, a hole, and the space for the object's movement. The object should be moved reasonably into the hole, and a video of this process should be recorded. In Figure 3-1, a sample of four views of the recorded video is shown, with the object and the hole included.

After preparing the videos, it is necessary to automatically extract the path of the object's movement from the initial position to the goal from these videos. For this purpose, object tracking algorithms in a two-dimensional environment are used. In such algorithms, the object's position within the environment is detected in each frame and provided as coordinates in a two-dimensional space.

Fig 3-1 : Four different views from one of the recorded demo videos of the guided path. The top-right figure shows the start of the object's movement, the top-left figure depicts the object passing through the hole, the bottom-right figure shows the object's movement toward the goal in the middle of the hole, and the bottom-left figure illustrates the object's placement at the target point .

3-2-2- Initializing the Q-Matrix Using Guided Paths

After extracting the coordinates of the object from different frames of the video, it is necessary to initialize the matrix Q based on the object's coordinates in the recorded videos. For this purpose, initially, all the elements of the matrix Q are set to zero, and then those elements of the matrix Q that in the video frame the object is observed in receive higher values.

In the next step, by considering a potential field around each of the valuable points (points where the object has been observed in the video), the values of other points are determined based on their placement within the potential field. Points adjacent to the potential field centers receive higher values and other points assign themselves lower values based on their distance from the potential field centers.

3-3- Using the PILCO Model in Solving the Problem of Moving an Object into a Hole in Continuous Environments

In this section, we explain how a simulated robotic arm can learn to autonomously place a block inside a hole. We make use of the following assumptions: First, since grasping the object is not the focus, we assume that the block is placed in the robot's gripper. Second, joint angles and velocities are not measured, only the position and velocity of the block's center in the robot's gripper are measured. Third, there is no pre-defined path or desired trajectory. Fourth, we assume that the initial and target positions of the block in the robot's gripper are fixed.

Simply following a direct path between the initial and target positions may not be successful due to obstacles. For long-term planning and controller learning, this uncertainty needs to be taken into account.

At each time step, the robot uses a discrete control signal $u \in \mathbb{R}^2$ to calculate from the center of the block in its gripper. Wrist rotation and gripper opening/closing are not learned. The reinforcement learning algorithm uses the 2D center of the object and its velocity as the state $x \in \mathbb{R}^4$.

3-3-1- Learning Policy with State Space Constraints

In the following, we briefly describe the PILCO framework [3] for learning a suitable closed-loop policy (state feedback controller) $\pi : \mathbb{R}^4 \to \mathbb{R}^2, x \mapsto u$. Here, x is referred to as the state, defined as the coordinates and velocity of the block's center (x_c, y_c, x_c', y_c') in the gripper. We intend to learn this policy from scratch, or in other words, with very general prior knowledge about the task and ourselves. Furthermore, we aim to find π with as few trials as possible, or in other words, we need an efficient data-driven learning approach.

As a criterion for evaluating the performance of the controller π, we can use the expected long-term return of a trajectory $(x_0, \cdots, x_T)$ under the application of π, which can be formulated as equation (3-7).

Eq. (3-7) $$J^\pi = \sum_{t=0}^{T} \mathbb{E}_{x_t}[c(x_t)]$$

In equation (3-7), T is the prediction horizon, and $c(\mathrm{x}_t)$ is the instantaneous cost of being in state x at time t. Unless stated otherwise, throughout this thesis, we use a saturating cost function $c = 1 - \exp(-d^2/\sigma_c^2)$ that penalizes Euclidean distances d of the block from the target position x_{target} in the end effector. We assume that the policy π is parameterized by ψ. PILCO learns an appropriate parameterized policy using the algorithm in section (2-7).

3-3-1-1- Probabilistic Dynamic Model

To avoid the assumption of certainty equivalence on the learned model, PILCO considers model uncertainties during planning. Therefore, it might be necessary to have a (posterior) distribution over the dynamic models. We currently use GPs [102] for inferring this posterior distribution from the available experience.

Following [102], we briefly introduce the notation and standard predictive models for GPs, which are used to infer the distribution over a latent function f from noisy observations $y_i = f(\mathrm{x}_i) + \varepsilon$. In this thesis, $\varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2)$ is considered as the i.i.d noise of the system. A GP is fully characterized by a mean function $m(.)$ and a positive semi-definite covariance function $k(.\ ,\ .)$ called the kernel. Throughout this thesis, we consider a zero mean function $m \equiv 0$ and the squared exponential (SE) kernel with automatic relevance determination defined as follows:

$$\textbf{Eq.(3-8)} \quad k(x, x') = \alpha^2 \, exp\left(-\frac{1}{2}(x - x')^T \Lambda^{-1}(x - x')\right)$$

Here, α^2 is defined as the variance of the latent function f, and $\Lambda := diag([\ell_1^2, \cdots, \ell_D^2])$, which depends on characteristic length scales ℓ_i. With access to n training inputs $\mathrm{X} = [\mathrm{x}_1, \cdots, \mathrm{x}_n]$ and their corresponding training targets $\mathrm{y} = [y_1, \cdots, y_n]^{\mathrm{T}}$, the posterior GP hyperparameters (length scales ℓ_i, signal variance α^2, and noise variances σ_ε^2) are learned by evidence maximization [102].

The predictive posterior distribution $p(f_*|\mathrm{x}_*)$ of the value function $f_* = f(\mathrm{x}_*)$ for an arbitrary but known input x_* is Gaussian with mean and variance described by equations (3-9) and (3-10) respectively.

$$\textbf{Eq.(3-9)}\quad m_f(x_*) = \mathbb{E}_f[f_*] = k_*^T(K + \sigma_\varepsilon^2)^{-1}y = k_*^T\beta$$

$$\textbf{Eq.(3-10)}\quad \sigma_f^2(x_*) = var_f[f_*] = k_{**} - k_*^T(K + \sigma_\varepsilon^2 I)^{-1}k_* + \sigma_\varepsilon^2$$

where $k_* := k(\mathrm{X}, \mathrm{x}_*)$ and $k_{**} := k(\mathrm{x}_*, \mathrm{x}_*)$ with $\beta := (K + \sigma_\varepsilon^2)^{-1}\mathrm{y}$, where K is the kernel matrix with elements $K_{ij} = k(\mathrm{x}_i, \mathrm{x}_j)$.

In our robotic system, the GP models the function $f: \mathbb{R}^6 \to \mathbb{R}^4, (\mathrm{x}_{t-1}, \mathrm{u}_{t-1}) \mapsto \Delta_t := \mathrm{x}_t - \mathrm{x}_{t-1} + \varepsilon_t$, where $\varepsilon_t \in \mathbb{R}^4$ is i.i.d Gaussian noise. The inputs and training targets for the GP model are, respectively, tuples $(\mathrm{x}_{t-1}, \mathrm{u}_{t-1})$ and their corresponding differences Δ_t.

3-3-1-2- Long-Term Planning via Approximate Inference

Minimizing and evaluating J^π in equation (3-7) requires long-term predictions of state changes and evolutions. To obtain the state distributions $p(\mathrm{x}_1), \cdots, p(\mathrm{x}_T)$, we concatenate one-step predictions consecutively. Properly performing this task requires mapping uncertain input samples through the GP dynamic model. In the following, we assume that these input samples have Gaussian distributions and extend the results from reference [3] to long-term planning in stochastic systems with control inputs.

To predict x_t from $p(\mathrm{x}_{t-1})$, we need the joint distribution $p(\mathrm{x}_{t-1}, \mathrm{u}_{t-1})$. To compute this distribution, we use $\mathrm{u}_{t-1} = \pi(\mathrm{x}_{t-1})$, which means control is a function of the state. First, we calculate the control predictor distribution $p(\mathrm{u}_{t-1})$ and then the cross-covariance $cov[\mathrm{x}_{t-1}, \mathrm{u}_{t-1}]$. Finally, we approximate $p(\mathrm{x}_{t-1}, \mathrm{u}_{t-1})$ as a Gaussian distribution with the correct mean and covariance. These calculations depend on the parameterization ψ of the policy π. In this thesis, we assume $\mathrm{u}_{t-1} = \pi(\mathrm{x}_{t-1}) = \mathrm{A}\mathrm{x}_{t-1} + \mathrm{b}$ with $\psi = \{\mathrm{A}, \mathrm{b}\}$. With $p(\mathrm{x}_{t-1}) = \mathcal{N}(\mathrm{x}_{t-1}|\mu_{t-1}, \Sigma_{t-1})$, applying standard results from linear Gaussian models yields the following two equations:

$$p(\mathrm{u}_{t-1}) = \mathcal{N}(\mathrm{u}_{t-1}|\mu_u, \Sigma_u)$$

$$\mu_u = \mathrm{A}\mu_{t-1} + \mathrm{b}\,, \quad \Sigma_u = \mathrm{A}\Sigma_{t-1}\mathrm{A}^T$$

In this example, π is a linear function of x_{t-1}, and therefore, the desired joint distribution $p(\mathrm{x}_{t-1}, \mathrm{u}_{t-1})$ is precisely Gaussian, and it is determined as in equation (3-11).

Eq. (3-11) $$\mathcal{N}\left(\begin{bmatrix}\mu_{t-1}\\ A\mu_{t-1}+b\end{bmatrix},\begin{bmatrix}\Sigma_{t-1} & \Sigma_{t-1}A^T\\ A\Sigma_{t-1} & A\Sigma_{t-1}A^T\end{bmatrix}\right)$$

In this equation, the cross-covariance $cov[\mathrm{x}_{t-1},\mathrm{u}_{t-1}]=\Sigma_{t-1}\mathrm{A}^T$. For many other interesting controller parameterizations, the mean and covariance can be computed analytically [118], although $p(\mathrm{x}_{t-1},\mathrm{u}_{t-1})$ may not be exactly Gaussian.

From now on, we assume a joint Gaussian distribution $p(\tilde{\mathrm{x}}_{t-1})=\mathcal{N}\left(\tilde{x}_{t-1}\middle|\tilde{\mu}_{t-1},\tilde{\Sigma}_{t-1}\right)$, at time $t-1$, where $\tilde{\mathrm{x}}:=[\mathrm{x}^T\ \mathrm{u}^T]^T$, and $\tilde{\mu}$ and $\tilde{\Sigma}$ are the mean and covariance of this augmented variable. When predicting the distribution shown in equation (3-12), we integrate over the random variable $\tilde{\mathrm{x}}_{t-1}$.

Eq. (3-12) $$p(\Delta_t)=\int p(f(\tilde{x}_{t-1})|\tilde{x}_{t-1})p(\tilde{x}_{t-1})d\tilde{x}_{t-1}$$

The transition probability $p(f(\tilde{\mathrm{x}}_{t-1})|\tilde{\mathrm{x}}_{t-1})$ is obtained from the GP posterior distribution. Calculating the exact predictive distribution in equation (3-12) is analytically tractable. Therefore, we approximate $p(\Delta_t)$ with a Gaussian distribution with exact mean and variance (moment-matching). Figure (3-2) illustrates this scenario. Note that to calculate the mean μ_Δ and variance σ_Δ^2 of the predictive distribution, the standard GP predictive distribution (refer to equations (3-9) and (3-10)) is insufficient because $\tilde{\mathrm{x}}_{t-1}$ is not deterministically given.

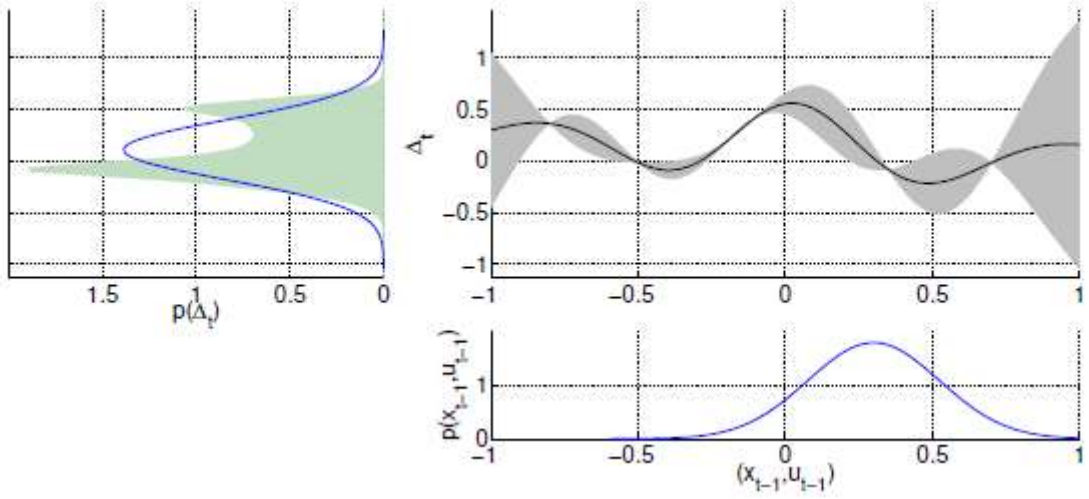

Fig 3-2: GP prediction in the presence of uncertain input. The input distribution $p(\mathrm{x}_{t-1},\mathrm{u}_{t-1})$ is assumed to be Gaussian (bottom right panel).

Propagating it through the GP model (top right panel) results in the shaded distribution $p(\Delta_t)$ in the top left panel. $p(\Delta_t)$ is approximated with a Gaussian distribution with exact mean and variance (bottom left panel) . [108]

We assume that the mean μ_Δ and covariance Σ_Δ of the prediction distribution $p(\Delta_t)$ are known. Then, a Gaussian approximation of the desired state distribution $p(\mathrm{x}_t)$ has a mean and covariance, respectively, as given by equations (3-13) and (3-14).

$$\textbf{Eq. (3-13)} \quad \mu_t = \mu_{t-1} + \mu_\Delta$$

$$\textbf{Eq. (3-14)} \quad \Sigma_t = \Sigma_{t-1} + \Sigma_\Delta + cov[x_{t-1}, \Delta_t] + cov[\Delta_t, x_{t-1}]$$

$$\textbf{Eq. (3-15)} \quad cov[x_{t-1}, \Delta_t] = cov[x_{t-1}, u_{t-1}]\Sigma_u^{-1} cov[u_{t-1}, \Delta_t]$$

Calculating the required cross-covariances in equation (3-15) depends on the policy parameterization but can often be computed analytically.

Next, we calculate the mean μ_Δ and variance σ_Δ^2 of the prediction distribution $p(\Delta_t)$ (refer to equation (3-12)).

1. Mean: By following the law of iterated expectations, we obtain:

$$\textbf{Eq. (3-16)} \quad \mu_\Delta = \mathbb{E}_{\tilde{\mathrm{x}}_{t-1}}\left[\mathbb{E}_f[f(\tilde{\mathrm{x}}_{t-1})|\tilde{\mathrm{x}}_{t-1}]\right] = \mathbb{E}_{\mathrm{x}_*}\left[m_f(\tilde{\mathrm{x}}_{t-1})\right]$$

$$= \int m_f(\tilde{x}_{t-1})\mathcal{N}\left(\tilde{x}_{t-1}\middle|\tilde{\mu}_{t-1}, \tilde{\Sigma}_{t-1}\right)d\tilde{x}_{t-1} = \beta^T q$$

where $\beta = (K + \sigma_\varepsilon^2 \mathrm{I})^{-1} y$ and $\mathrm{q} = [q_1, \cdots, q_n]^{\mathrm{T}}$. The elements of $\mathrm{q} \in \mathbb{R}^n$ are as follows:

$$q_i = \int k(\mathrm{x}_i, \mathrm{x}_*)\mathcal{N}\left(\tilde{x}_{t-1}\middle|\tilde{\mu}_{t-1}, \tilde{\Sigma}_{t-1}\right)d\tilde{\mathrm{x}}_{t-1}$$

$$= \frac{\alpha_a^2}{\sqrt{\left|\tilde{\Sigma}_{t-1}\Lambda^{-1} + \mathrm{I}\right|}} \exp\left(-\frac{1}{2}(\mathrm{x}_i - \tilde{\mu}_{t-1})^T\left(\tilde{\Sigma}_{t-1} + \Lambda\right)^{-1}(\mathrm{x}_i - \tilde{\mu}_{t-1})\right).$$

2. Variance: Using the law of total variance, we obtain:

$$\textbf{Eq. (3-17)} \quad \sigma_\Delta^2 = \mathbb{E}_{\tilde{\mathrm{x}}_{t-1}}\left[m_f(\tilde{\mathrm{x}}_{t-1})^2\right] - \mathbb{E}_{\tilde{\mathrm{x}}_{t-1}}\left[m_f(\tilde{\mathrm{x}}_{t-1})\right]^2 + \mathbb{E}_{\tilde{\mathrm{x}}_{t-1}}\left[\sigma_f^2(\tilde{\mathrm{x}}_{t-1})\right]$$

$$= \beta^T Q\beta + \alpha^2 - tr((K + \sigma_\varepsilon^2 I)^{-1}Q) - \mu_\Delta^2 + \sigma_\varepsilon^2$$

where $tr(.)$ refers to the sum of elements on the main diagonal of the matrix. The elements $Q \in \mathbb{R}^{n\times n}$ are as follows:

$$Q_{ij} = k(\mathrm{x}_i, \tilde{\mu}_{t-1})k(\mathrm{x}_j, \tilde{\mu}_{t-1})\left|2\tilde{\Sigma}_{t-1}\Lambda^{-1} + \mathrm{I}\right|^{-\frac{1}{2}} \times \exp\left(\frac{1}{2} z_{ij}^T \left(2\tilde{\Sigma}_{t-1}\Lambda^{-1} + \mathrm{I}\right)\tilde{\Sigma}_{t-1} z_{ij}\right)$$

where $\varsigma_i := (\mathrm{x}_i - \tilde{\mu}_{t-1})$ and $z_{ij} := \Lambda^{-1}(\varsigma_i + \varsigma_j)$.

It is worth noting that both μ_Δ and σ_Δ^2 practically depend on the mean μ_u and covariance Σ_u of the control signal due to $\tilde{\mu}_{t-1}$ and $\tilde{\Sigma}_{t-1}$ (refer to in equations (3-16) and (3-17)). These equations show that the uncertainty regarding the hidden function f is integrated, explicitly accounting for model uncertainty.

3-3-1-3- Controller Learning Through Indirect Policy Search

From section 3-3-1-2, we know how to compute one-step predictions sequentially to obtain Gaussian approximations of the predictive distributions $p(\mathrm{x}_1), \cdots, p(\mathrm{x}_T)$. To evaluate the expected return J^π in equation (3-7), it remains to compute the expected values of cost functions c with respect to the predictive state distributions.

Eq. (3-18) $$\mathbb{E}[c(x_t)] = \int c(x_t)\mathcal{N}(x_t|\mu_t, \Sigma_t)dx_t \quad , t = 0, \cdots, T$$

We assume that the cost function c has been chosen in such a way (e.g., polynomial) that equation (3-18) can be analytically solved.

To apply gradient-based policy search to find the parameters ψ that minimize J^π (refer to equation (3-7)), we first change the order of differentiation and summation in equation (3-7). With $\mathcal{E}_t := \mathbb{E}_{\mathrm{x}_t}[c(\mathrm{x}_t)]$, we obtain:

Eq. (3-19) $$\frac{d\mathcal{E}_t}{d\theta} = \frac{d\mathcal{E}_t}{d\mu_t}\frac{d\mu_t}{d\psi} + \frac{d\mathcal{E}_t}{d\Sigma_t}\frac{d\Sigma_t}{d\psi}$$

The partial derivatives of the mean μ_t and the covariance Σ_t for $p(\mathrm{x}_t)$ with respect to the policy parameters ψ can be computed analytically by repeatedly applying the chain rule to equations (3-13), (3-14), (3-15), (3-16), and (3-17). This process also includes the computation of partial derivatives $d\mu_t/d\psi$ and $d\Sigma_t/d\psi$. Here, we have omitted further details, but these derivatives have been analytically calculated in the paper [118]. This

work enables the use of gradient-based non-convex optimization methods such as CG or L-BFGS which return the optimized parameter vector ψ^*.

3-3-1-4- State Space Constrained Planning

In classical reinforcement learning, the assumption is that the learner is unaware of any constraints in the state space and needs to discover obstacles and constraints by colliding with them and incurring heavy penalties. In robotics, this general but not necessarily desirable assumption is problematic because the robot can incur damage.

If constraints (e.g., obstacles) in the state space are known in advance, it is preferable to incorporate this prior knowledge directly into the planning and policy learning. We recommend defining obstacles as "undesirable" regions, in other words, areas that the robot is supposed to avoid. The concept of "undesirability" is encoded as a penalty in the instantaneous cost function c. Therefore, the cost function c is defined as follows:

$$\textbf{Eq. (3-20)} \quad c(x) = \sum_{k=1}^{K} c_k^+(x) - \sum_{j=1}^{J} \iota_j c_j^-(x)$$

Where c_k^+ represents desirable states (e.g., goal states), and c_j^- represents undesirable states (e.g., obstacles) weighted by ι_j with $\iota_j \geq 0$. Larger values of ι_j make the policy avoid specific undesirable states to a greater extent. In this thesis, we always set $\iota_j = 1$. For c_k^+ and c_j^-, we choose negative exponential squared terms that create a balance between exploration and exploitation during averaging according to the state distribution [118]. The squared exponential terms have potential different widths Σ_k^+ and are not normalized. The widths of individual constraints c_j^- determine how "soft" the constraints are. Hard and soft constraints are described with squared exponential terms c_j^- that are very sharp and $\iota_j \to \infty$. This idea comes from the reference [119], where planning with fully known dynamics and piecewise linear controllers was performed.

Figure (3-3) illustrates Equation (3-20) with two penalties c_j^- and one reward c_k^+. This figure shows that if an undesirable state and a desirable state are close to each other, the total cost c creates a balance between the two objectives. Furthermore, unlike the past, the optimal state $x_* \in \underset{x}{\operatorname{argmin}}\, c(x)$ is not associated with $x_*^+ \in \underset{x}{\operatorname{argmin}}\, c^+(x)$: Optimal

behavior involves moving somewhat away from the goal state (away from the undesirable state).

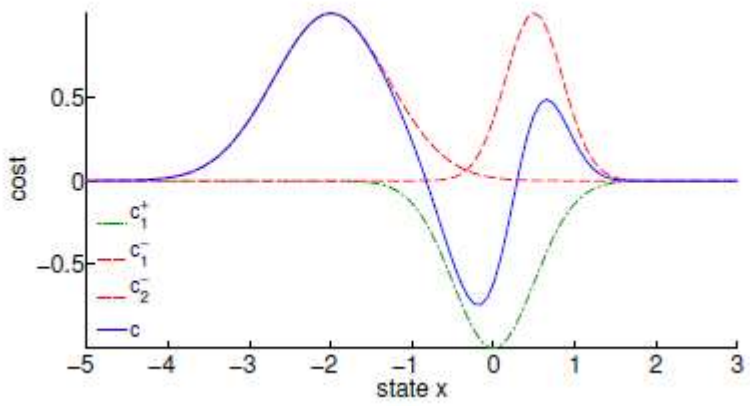

Fig 1-3 The text explains a cost function that considers constraints (e.g., obstacles) by designating them as "undesirable." The dashed lines in the curves represent individual components c_k^+ and c_j^-. Refer to Equation (3.20) for details. The solid line in the graph represents their sum c. [108]

The expected costs in Equation (3-20) and derivatives with respect to the mean μ_t and covariance Σ_t of the state distribution $p(\mathrm{x}_t)$ can be computed for each c_k^+ and c_j^-. Then, according to Equation (3-19), the chain rule is applied for gradient-based policy search.

Expressing constraints as "undesirable" in the cost function in Equation (3-20) still allows for fully probabilistic long-term planning and enables guiding the robot in the state space without experiencing collisions with obstacles.

Collisions within the Bayesian inference framework can be avoided but are not strictly eliminated. This doesn't mean that averaging out uncertainties is a mistake; instead, it tells us that we shouldn't expect constraints to be violated with specific certainty. An honest description of predictable uncertainty is often more valuable than claiming complete and, at times, unwarranted constraint violations.

3-4- Summary

In this chapter, we discussed basic and proposed methods for solving the problem of transferring an object into a hole using a robotic arm. To simulate in discrete environments, we used the Q-Learning algorithm, and the proposed method in this thesis

was described as adding guiding paths extracted from recorded videos. For simulation in continuous environments, we utilized the PILCO algorithm and introduced the capability of considering state space constraints (such as walls and obstacles) as the proposed method to the PILCO model.