ROBOTIC FORCE CONTROL VIA A REINFORCEMENT LEARNING APPROACH

by
DOĞANAY KARAKIŞ

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfilment of
the requirements for the degree of Master of Science

Sabancı University
July 2025

ROBOTIC FORCE CONTROL VIA A REINFORCEMENT LEARNING APPROACH

Approved by:

Assoc. Prof. KEMALETTİN ERBATUR .....
(Thesis Supervisor)

Asst. Prof. BESTE BAHÇECİ .....

Asst. Prof. MELİH TÜRKSEVEN .....

Date of Approval: July 23, 2025

ABSTRACT

ROBOTIC FORCE CONTROL VIA A REINFORCEMENT LEARNING APPROACH

DOĞANAY KARAKIŞ

Mechatronics Engineering, M.Sc. Thesis, July 2025

Thesis Supervisor: Assoc. Prof. Kemalettin Erbatur

Keywords: reinforcement learning, q-learning, elbow manipulator, force control

This thesis investigates the use of reinforcement learning (RL) for robotic force control using a planar elbow manipulator tasked with applying force to a rigid surface. Inspired by simplified but effective simulation environments such as GridWorld, our approach leverages Q-learning to train the manipulator in a discrete action–continuous state setting. The robot agent learns through interaction to establish end-effector contact without prior knowledge of the system's dynamics. The proposed control strategy is implemented in a simulated environment that captures the manipulator's kinematics and the interaction forces with a two-layered wall. Unlike traditional force control methods, which often require accurate dynamic models and extensive offline tuning, our approach enables fast learning with fewer than 10,000 training iterations. This allows for real-time or near-real-time application potential in physical systems. Simulation results demonstrate that the Q-learning agent successfully converges to a stable and effective contact establishment policy. The study contributes to bridging foundational reinforcement learning algorithms and practical robotic control problems, highlighting the feasibility of lightweight, model-free learning architectures for force-regulated interaction tasks.

ÖZET

PEKİŞTİRMELİ ÖĞRENME YAKLAŞIMIYLA ROBOTİK KUVVET KONTROLÜ

DOĞANAY KARAKIŞ

Mekatronik Mühendisliği, Yüksek Lisans Tezi, Temmuz 2025

Tez Danışmanı: Doç. Dr. Kemalettin Erbatur

Anahtar Kelimeler: pekiştirmeli öğrenme, q-öğrenme, dirsek manipülatörü, kuvvet kontrolü

Bu tez, sert bir yüzeye kuvvet uygulamakla görevli düzlemsel bir dirsek manipülatörü kullanarak robotik kuvvet kontrolü için pekiştirmeli öğrenmenin (RL) kullanımını araştırmaktadır. GridWorld gibi basitleştirilmiş fakat etkili simülasyon ortamlarından ilham alan yaklaşımımız, manipülatörü ayrık eylem – sürekli durum ortamında eğitmek için Q-öğrenmeyi kullanmaktadır. Robot ajan, sistemin dinamiklerine dair önceden bilgi sahibi olmadan etkileşim yoluyla uç efektör temasını sağlamayı öğrenir. Önerilen kontrol stratejisi, manipülatörün kinematiğini ve iki katmanlı bir duvar ile etkileşim kuvvetlerini yakalayan simüle edilmiş bir ortamda uygulanmıştır. Doğru dinamik modeller ve kapsamlı çevrimdışı ayarlamalar gerektiren geleneksel kuvvet kontrol yöntemlerinin aksine, yaklaşımımız 10.000'den az eğitim iterasyonu ile hızlı öğrenmeyi mümkün kılar. Bu da fiziksel sistemlerde gerçek zamanlı veya gerçek zamana yakın uygulama potansiyelini sağlar. Simülasyon sonuçları, Q-öğrenme ajanının kararlı ve etkili bir temas kurma politikası üzerinde başarılı bir şekilde yakınsadığını göstermektedir. Çalışma, temel pekiştirmeli öğrenme algoritmaları ile pratik robotik kontrol problemlerini birleştirmeye katkı sağlamakta ve kuvvet kontrollü etkileşim görevleri için hafif, model gerektirmeyen öğrenme mimarilerinin uygulanabilirliğini vurgulamaktadır.

ACKNOWLEDGMENTS

First of all, I would like to express my deepest gratitude to my advisor, Assoc. Prof. Kemalettin Erbatur, for his continuous guidance, encouragement, and invaluable support throughout my research and academic journey. His insights and advice have been essential for the completion of this thesis and for my personal and professional development.

I would also like to thank the thesis jury members for their valuable time, constructive feedback, and helpful suggestions, which significantly contributed to the improvement of this study.

Finally, I would like to express my heartfelt gratitude to my beloved family. Their unconditional love, patience, and unwavering support have always been my greatest source of strength and motivation in every stage of my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CAD Computer-Aided Design 9

DDPG Deep Deterministic Policy Gradient 3

DOF Degree of Freedom 8

DPG Deterministic Policy Gradient 3

DRL Deep Reinforcement Learning 3, 4

LfD Learning from Demonstration 4, 6

MLPs Multi-Layer Perceptrons 4

RL Reinforcement Learning ..... iv, v, 1, 2, 3, 4, 5, 6, 7

SCARA Selective Compliance Assembly Robotic Arm ..... ix, 10

1. INTRODUCTION

Robotic systems frequently perform tasks involving direct contact with their environment—such as polishing, pushing, or compliant assembly—which require precise force regulation. In many safety-critical domains, including industrial assembly, surgical robotics, and human–robot collaboration, the ability to maintain controlled interaction forces directly affects both performance quality and operator safety (Bicchi & Tonietti, 2004; Hogan, 1985). Establishing contact from zero force into a stable force-regulated mode is particularly challenging, since any overshoot or instability can cause damage to parts, injuries to humans, or failure of the task. These challenges highlight the fundamental need for adaptive and reliable force-control strategies in modern robotic applications.

Classical control methods like impedance and hybrid position/force control depend heavily on accurate dynamic models and carefully tuned parameters (Hogan, 1985; Raibert & Craig, 1981). However, these traditional approaches often fail to generalize to new contexts or adapt to uncertainties in real time. For example, small modeling errors or parameter variations may cause instability or degraded performance, making such controllers unsuitable for highly variable or unstructured environments (Whitney, 1987). These limitations have motivated researchers to investigate learning-based and adaptive approaches that can offer robustness without requiring exact system knowledge. Limitations underscore the importance of adaptive, data-driven strategies, naturally leading to reinforcement learning as a compelling framework for robotic control.

Reinforcement learning (RL) presents a model-free alternative, allowing an agent to learn through experience by maximizing a reward signal over time. RL encompasses a broad range of algorithms that enable agents to learn optimal behaviors through trial-and-error interactions with their environment. By continuously updating value functions or policies based on feedback, RL methods can handle stochastic dynamics and delayed rewards, which are common in real-world robotic tasks. By framing the control task as a Markov Decision Process (MDP), RL enables robotic agents to optimize their actions based on observed outcomes. Within this spectrum, Q-learning

represents one of the foundational algorithms, notable for its ability to estimate the optimal action-value function without requiring a model of the environment. Its convergence properties and simplicity make it particularly suitable for discrete action spaces, serving as a baseline for more advanced RL approaches (Kaelbling et al., 1996; Sutton & Barto, 2018; Watkins & Dayan, 1992).

In this work, we propose a Q-learning-based force control strategy for a planar elbow manipulator. The manipulator must learn to find a two-staged wall of unknown position and establish contact with steady force without having access to its own dynamics or those of the environment. Quantized joint positions of the manipulator are employed as RL state variables; the search direction in the Cartesian workspace is the quantized action variable. Once the steady contact is established, the control system can be switched into a classical force control algorithm. Our learning design is inspired by grid-based environments, where agents learn optimal actions over discrete steps. The novelty of this approach lies in its ability to achieve stable contact using fewer than 10,000 training iterations—addressing a critical barrier in robotic RL systems: learning speed. This lightweight framework is promising for real-world implementations where computational and time budgets are limited.

The thesis is organized as follows. The next chapter presents a literature survey of reinforcement learning in the field of contact and force control. Chapter 3 introduces the dynamic model of a planar elbow manipulator. The learning algorithms are run on this particular robot model in this thesis work. The Q-learning approach proposed is discussed in Chapter 4. Chapter 5 presents simulation results. Conclusions are drawn in Chapter 6. References and an appendix for the developed code follow lastly.

2. LITERATURE SURVEY

2.1 Reinforcement Learning (RL) and its Foundations in Robotics

Reinforcement Learning (RL) is a branch of machine learning that models intelligent decision-making in dynamic environments through interaction (Sutton & Barto, 2018). It is typically formalized using the Markov Decision Process (MDP), a mathematical framework defined by a tuple (S,A,P,R,γ), where S represents the state space, A the action space, P the state transition probabilities, R the reward function, and $\gamma \in [0, 1)$ the discount factor. The agent seeks to learn a policy $\pi(a \mid s)$ that maximizes the expected cumulative discounted reward over time (Sutton & Barto, 2018).

Policy optimization in RL can be approached in two major ways: value-based and policy-based methods. Value-based approaches such as Q-learning approximate the expected reward for each state-action pair and derive policies indirectly, whereas policy-based methods like REINFORCE or actor-critic models directly parameterize and optimize the policy. For robot manipulators, value-based methods struggle with high-dimensional or continuous action spaces. This limitation motivated the development of deterministic policy gradient (DPG) methods, where policies map directly from states to continuous actions (Lillicrap et al., 2015). However, algorithms like Q-learning can be used effectively when the action space is discrete or can be discretized.

The integration of deep learning into RL led to the emergence of Deep RL (DRL), where neural networks approximate value functions or policies. Deep Q-Networks (DQN) provide a foundational structure for controlling manipulators when the action space is discretized. To overcome the limitations of DQN in continuous control problems, algorithms like Deep Deterministic Policy Gradient (DDPG) (Fujimoto

et al., 2018) and Twin Delayed DDPG (TD3) were developed. These methods combine actor-critic architectures with deep networks and experience replay, enabling stable learning in continuous action spaces. These architectures are particularly effective in controlling continuous outputs like joint torques or velocities for a planar manipulator (Lillicrap et al., 2015).

One of the most significant challenges in applying RL to real-world robots is sample inefficiency. Traditional model-free RL methods often require hundreds of thousands of interactions, which is impractical in physical systems. Researchers have proposed multiple strategies to address this issue: Experience Replay, Reward Shaping, Curriculum Learning, and Learning from Demonstration (LfD) (Argall et al., 2009).

The architecture of the policy and value networks in DRL greatly influences learning stability and generalization. For controlling a planar manipulator, shallow networks may underfit complex dynamics, while very deep networks risk overfitting to simulation noise. Common design practices involve multi-layer perceptrons (MLPs) with two or three hidden layers, using activation functions like ReLU or tanh. Normalization techniques like LayerNorm or BatchNorm help stabilize training, and parameter noise or entropy regularization can encourage exploration (Lillicrap et al., 2015).

Sim2Real transfer refers to transferring a policy trained in simulation to a physical robot without requiring retraining. This is crucial in robotics, where simulation allows for fast, parallelized, and risk-free learning. Mismatches between simulated and real environments (the "reality gap") degrade policy performance. Techniques like domain randomization, where physical parameters are varied during simulation, improve robustness by exposing the policy to a broad distribution of dynamics (Tobin et al., 2017). Another method is progressive networks, where policies trained in simulation are frozen and augmented with new trainable layers that adapt to real-world dynamics (Rusu et al., 2017). One of the primary concerns when deploying RL in physical robotic systems is safety. To address this, researchers have explored safe exploration strategies, including constraint-aware policy optimization, shielding mechanisms, and risk-sensitive RL (Sutton & Barto, 2018).

While model-free RL offers flexibility and generality, classical model-based control methods remain dominant in industrial automation due to their predictability, efficiency, and theoretical guarantees. Inverse kinematics and impedance control are widely used for planar manipulators to follow a known trajectory or apply a specific force (Ikeura & Inooka, 1995; Whitney, 1982; Yamamoto & Fujita, 2017). However, these methods require accurate modeling of dynamics and contact forces (Huang et al., 2020; Jiang et al., 2021; Park et al., 2019). RL, in contrast, can implicitly learn such relationships from data without requiring explicit modeling (Xu et al.,

2019). Hybrid strategies are emerging as a practical solution, where a model-based controller ensures stability while an RL agent fine-tunes performance (Sutton & Barto, 2018).

A major advantage of RL is its capacity to handle multimodal sensory data. In a task involving a planar manipulator and a surface, these sensors include Force/Torque (F/T) sensors, joint torque inference, and proprioceptive signals (joint angles and velocities) (Chen et al., 2022; Huang et al., 2020; Kim et al., 2016; Zhao et al., 2019). This sensor fusion allows for more robust and adaptive behavior, particularly when detecting the exact moment of contact (Chen et al., 2022; Huang et al., 2020; Kim et al., 2016; Takahashi & Fujita, 2020; Zhao et al., 2019).

Despite significant progress, challenges remain, such as sample efficiency, reward design, and transfer learning across different robot configurations or tasks. Promising directions include Meta-RL, Hierarchical RL (Jeong et al., 2020), and Self-supervised RL.

2.2 Robotic Manipulators and Surface Contact: Classical and Contemporary Perspectives

The task of a manipulator touching a flat surface involves several complexities: dynamic uncertainties, surface interaction, and the need for safe contact (Luo et al., 2018).

Classical approaches for contact tasks include Inverse Kinematics and Position Control, Impedance Control (Ikeura & Inooka, 1995), and Hybrid Force-Position Control (De Luca & Mattone, 2005). These methods require accurate models of the manipulator and surface dynamics, along with carefully tuned parameters.

Sensing plays a pivotal role in contact tasks. Force/Torque (F/T) sensors provide direct feedback on the contact state. Joint torque sensors enable indirect estimation of external forces (De Luca & Mattone, 2005). Proprioceptive sensors determine the robot's kinematic state. Sensor fusion techniques, which combine data from multiple sensors, demonstrate improved robustness in contact detection and control (Chen et al., 2022; Huang et al., 2020; Zhao et al., 2019).

A robust system must be able to handle errors that may occur during the contact task. This includes contact failure detection and automated retries (Beltran-Hernandez et al., 2020; Fan et al., 2019).

2.3 Reinforcement Learning in Manipulator Contact Tasks

Traditional control strategies often fail under uncertainties such as friction variations or surface roughness (Ajoudani et al., 2018). In contrast, Reinforcement Learning (RL) offers a data-driven, model-agnostic, and adaptive approach capable of learning contact strategies through interaction (Xu et al., 2019). This flexibility makes RL highly suitable for applications where manipulators need to interact with complex and unknown environments (Ajoudani et al., 2018). Q-learning can be used to learn this task when the manipulator's action space is discretized.

Model-free RL is the most common approach applied to the planar manipulator contact problem. Algorithms like Q-learning can be used successfully by discretizing the action space (e.g., increase torque, decrease torque, or hold it constant). Within this framework, a Q-learning agent can learn a controlled approach trajectory, the amount of force to apply at contact, and decisions on moving or staying on the surface.

A key advantage of RL is its ability to process multimodal sensory input, such as joint angles, velocities, and force/torque data (Duan et al., 2017; Sadeghi et al., 2016). This sensor fusion helps to accurately detect and react to the moment of contact, thereby preventing damage and optimizing task success (Kim et al., 2016; Zhao et al., 2019).

To overcome the slow convergence of RL, Learning from Demonstration (LfD) can be used. Expert trajectories can be recorded and used to initialize the RL agent's policy or guide its exploration, enabling the robot to learn safely and efficiently (Argall et al., 2009; Nagabandi et al., 2018).

Sim2Real transfer is critical for the planar manipulator. Domain randomization can be used to bridge the reality gap by training the agent with a wide range of simulation parameters, making the policy more robust to real-world uncertainties (Andrychowicz et al., 2020; Tobin et al., 2017).

Open research problems include the inefficiency of Q-learning in continuous action spaces, real-time response limitations, and generalization to different surfaces. Hybrid approaches that combine model-free RL, model-based planning, and classical control are also an area of active research (Xu et al., 2019).

3. DYNAMIC MODEL OF A PLANAR ELBOW MANIPULATOR

This chapter presents the physical modeling of the planar elbow manipulator. Planar manipulators are frequently preferred in robotics research due to their motion being confined to a two-dimensional workspace. This restriction simplifies their kinematic and dynamic analysis compared to three-dimensional robotic systems. In particular, executing contact tasks on a flat surface constitutes a fundamental performance test for such mechanisms.

The manipulator investigated in this work is a custom-built, two-degree-of-freedom (DOF) direct-drive prototype developed in the Robotics Laboratory at Sabancı University. A photograph of the setup is provided in Fig. 3.1. The control system is implemented on a dSPACE 1102 DSP board, with a PC-based graphical interface. Servo control routines are coded in C, compiled within the dSPACE environment, and then downloaded to the DSP for execution. Both joints—the base and the elbow—are driven by Yokogawa Dynaserv direct-drive motors, each equipped with high-resolution position encoders capable of 1,024,000 pulses per revolution. The maximum torque capacity of the base motor is 200 Nm, while the elbow motor provides up to 40 Nm.

The general dynamic equation of the system can be expressed as:

$$\begin{bmatrix} J_1 & 0 \\ 0 & J_2 \end{bmatrix} \begin{Bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{Bmatrix} + D(q_1, q_2) \begin{Bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{Bmatrix} \Big(C(q_1, q_2, \dot{q}_1, \dot{q}_2) + \begin{bmatrix} B_1 & 0 \\ 0 & B_2 \end{bmatrix} \Big) \begin{Bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{Bmatrix} + \begin{Bmatrix} F_{c1} \\ F_{c2} \end{Bmatrix} + J_M^T \begin{Bmatrix} F_{e_x} \\ F_{e_z} \end{Bmatrix} = \begin{Bmatrix} \tau_1 \\ \tau_2 \end{Bmatrix} \tag{3.1}$$

In Eq.(3.1), q_1 and q_2 denote the angular positions of the base and elbow joints, respectively, as illustrated in Fig. 3.2. The constants J_1 and J_2 represent rotor inertias for the corresponding joints. The matrix $D(q_1, q_2)$ is the manipulator's inertia matrix, while $C(q_1, q_2, \dot{q}_1, \dot{q}_2)$ accounts for centripetal and Coriolis effects. The viscous

friction coefficients for the joints are B_t values, and F_{c1}, F_{c2} correspond to Coulomb friction torques. The vector $[F_{e_x}, F_{e_z}]^T$ contains the environmental interaction forces at the end-effector in the base frame's x and z directions. The Jacobian J_M relates Cartesian velocities to joint velocities and, for this planar case, is a 2×2 matrix. Joint torques τ_1 and τ_2 serve as control inputs. Due to the manipulator's horizontal configuration, gravitational effects on joint torques are negligible.

The inertia matrix D and Coriolis/centripetal matrix C are given explicitly as:

$$D(q_1, q_2) = \begin{bmatrix} m_1 l_1^2 + m_2 \left(l_1^2 + l_2^2 + 2 l_1 l_2 \cos q_2\right) + I_1 + I_2 & m_2 \left(l_2^2 + l_1 l_2 \cos q_2\right) + I_2 \\ m_2 \left(l_2^2 + l_1 l_2 \cos q_2\right) + I_2 & m_2 l_2^2 + I_2 \end{bmatrix} \tag{3.2}$$

$$C(q_1, q_2, \dot{q}_1, \dot{q}_2) = (m_2 l_1 l_{c_2} \sin q_2) \begin{bmatrix} -\dot{q}_2 & -(\dot{q}_1 + \dot{q}_2) \\ \dot{q}_1 & 0 \end{bmatrix} \tag{3.3}$$

The parameters used in Eqs. (3.2) and (3.3)—including link masses, lengths, inertia values, and center-of-mass locations—are summarized in Table 3.1. The inertia and center-of-mass coordinates are obtained from CAD models of the links (Fig. 3.1), while rotor inertias J_1 and J_2 are specified by the manufacturer. Both matrices are derived using the Euler–Lagrange formulation. Friction parameters, particularly Coulomb friction, are challenging to estimate precisely; therefore, viscous friction coefficients $\hat{B}_1$ and $\hat{B}_2$ listed in Table 3.1 are based on experimental measurements with force sensors.

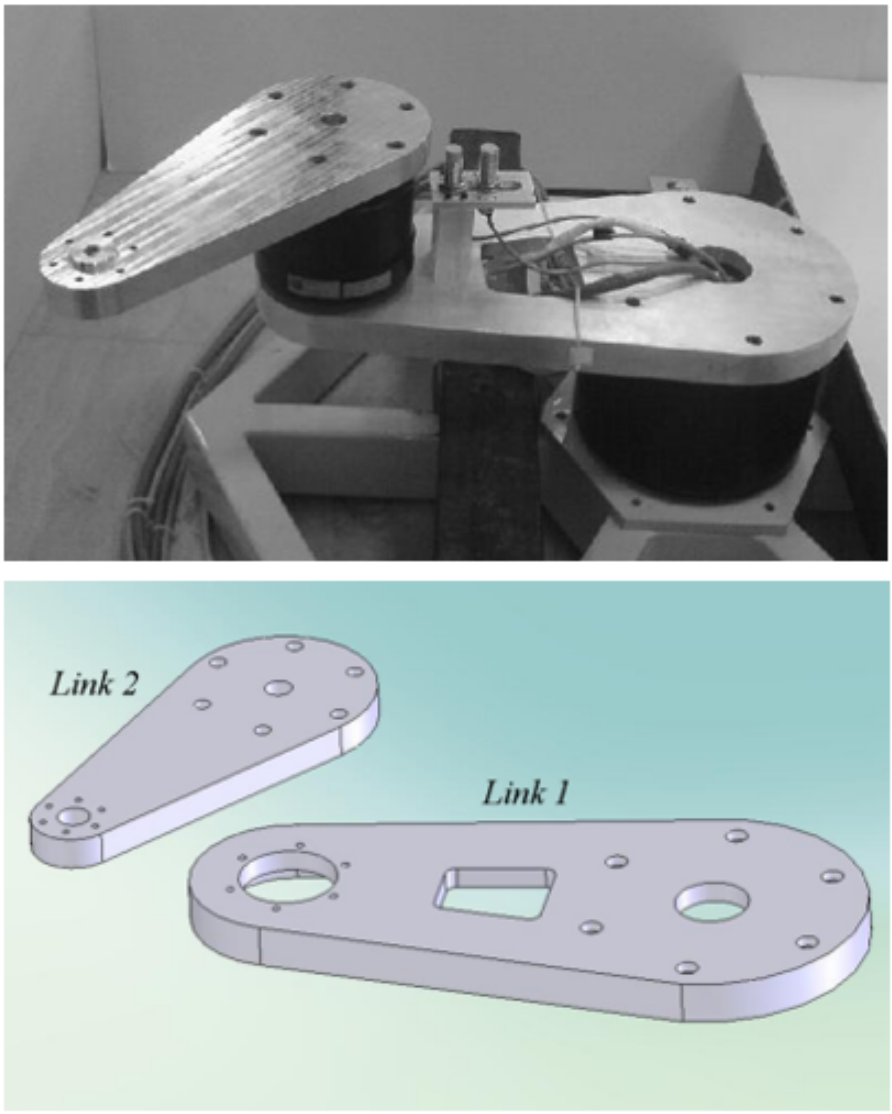

Figure 3.1 A direct-drive (SCARA) robot arm along with its corresponding CAD link representations

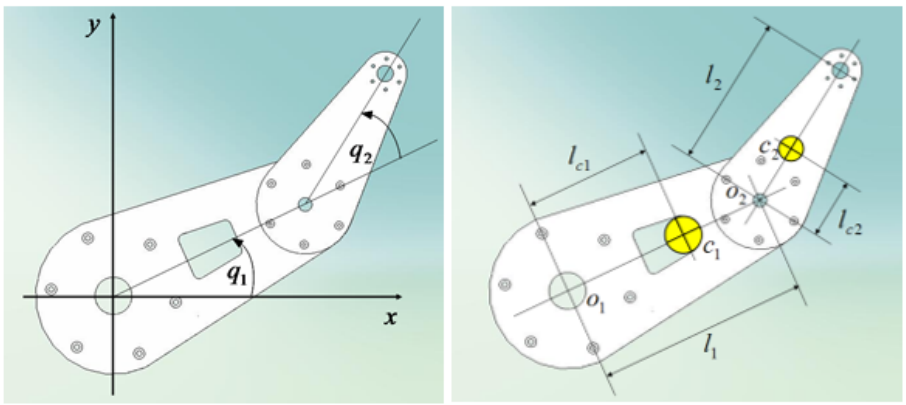

Figure 3.2 Definitions of the robot's joint angles and its link length parameters

Table 3.1 Dynamic parameters of the robot

Parameter	**Value**
Link 1 mass m_1 (inc. elbow motor)	17.9 kg
Link 1 inertia I_1 (inc. elbow motor)	0.54 kg $\cdot$ m^2
Motor 1 rotor inertia J_1	0.167 kg $\cdot$ m^2
Link 1 length l_1 (joint center to joint center)	0.40 m
Link 1 COM distance l_{c1}	0.277 m
Joint 1 viscous friction coefficient $\hat{B}_1$	3 N $\cdot$ m $\cdot$ s/rad
Link 2 mass m_2	3.25 kg
Link 2 inertia I_2	0.04 kg $\cdot$ m^2
Motor 2 rotor inertia J_2	0.019 kg $\cdot$ m^2
Link 2 length l_2 (joint center to tool center)	0.28 m
Link 2 COM distance l_{c2}	0.09 m
Joint 2 viscous friction coefficient $\hat{B}_2$	0.6 N $\cdot$ m $\cdot$ s/rad

The following chapter focuses on the Q-learning algorithm applied to this robotic platform.

4. Q-LEARNING ALGORITHM

4.1 Problem Definition

The problem definition revolves around employing Q-learning, a reinforcement learning method, to enable a planar elbow manipulator with two degrees of freedom to achieve a specific task. This manipulator, characterized by its two joints—a shoulder joint and an elbow joint—must learn to establish contact with a wall whose position and orientation are unknown. The wall itself is two-staged, featuring a recessed area where contact must be established for the task to be considered successful. The learning process involves discretizing both the state variables and the action space to simplify the problem and make it computationally feasible. The state variables correspond to the joint angles of the manipulator, with the shoulder joint angle quantized into 25 discrete values representing a complete rotation of 2π radians, and the elbow joint angle similarly quantized into 25 values. This discretization allows the manipulator to represent its position in a finite number of states, facilitating the application of Q-learning for decision-making.

The action space is also discretized, consisting of 37 possible actions that correspond to directions spanning 2π radians in the x-y Cartesian plane. Each action represents a direction in which a Cartesian control force of constant magnitude is applied to the manipulator's tool tip. The application of this force drives the manipulator in the chosen direction, with the ultimate goal of making contact with the recessed area of the wall. To translate the Cartesian control force into joint control torques, a Jacobian Transpose relation is employed, which ensures that the manipulator's movements are consistent with the applied forces and the geometry of its structure. This approach allows the manipulator to explore its environment effectively and adjust its joint angles to achieve the desired contact.

The learning process hinges on the manipulator's ability to interact with the environment and receive feedback based on its actions. Through the iterative updates of Q-learning, the manipulator learns an optimal policy by associating state-action pairs with rewards. When the manipulator successfully applies a steady force within the wall's recessed area for more than 0.1 seconds, the task is regarded completed. The challenge lies in the manipulator's need to explore the environment to locate the wall and its recess, which requires a balance between exploration and exploitation during learning. Additionally, the discretization of states and actions must be carefully designed to ensure that the manipulator can learn efficiently while maintaining the precision required to achieve the task. The framework also addresses the challenges of stability during force application, as the manipulator must maintain steady contact within the recess to fulfill the task requirements. The constant magnitude of the applied force is crucial for ensuring consistent interactions with the environment, while the Jacobian Transpose relation provides the necessary control over joint torques to achieve the desired movements. Overall, this framework combines the principles of reinforcement learning, state-action discretization, and robotic control to enable the planar elbow manipulator to perform a complex task in an unknown environment, demonstrating the potential of Q-learning in robotic applications.

The task description in the x-y plane is illustrated in Figures 4.1-4.4.

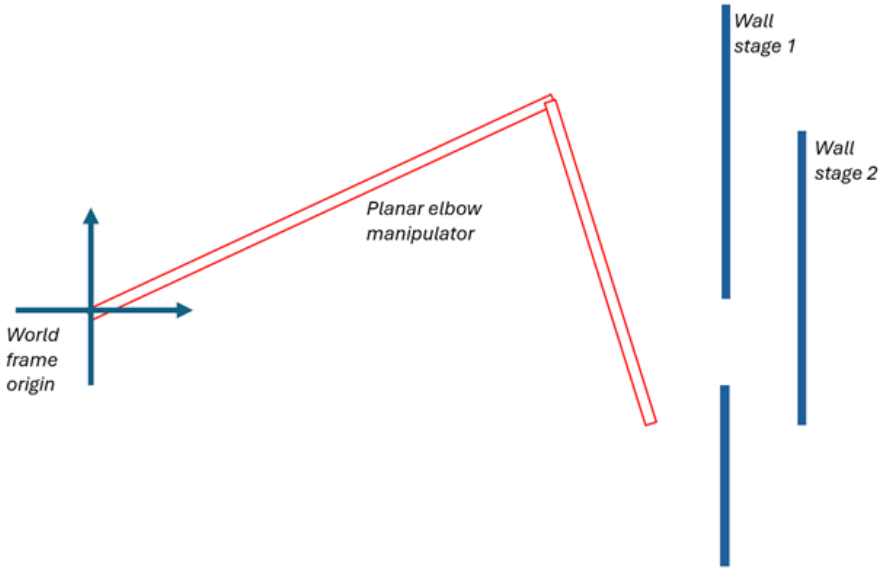

Figure 4.1 Typical manipulator and world setting

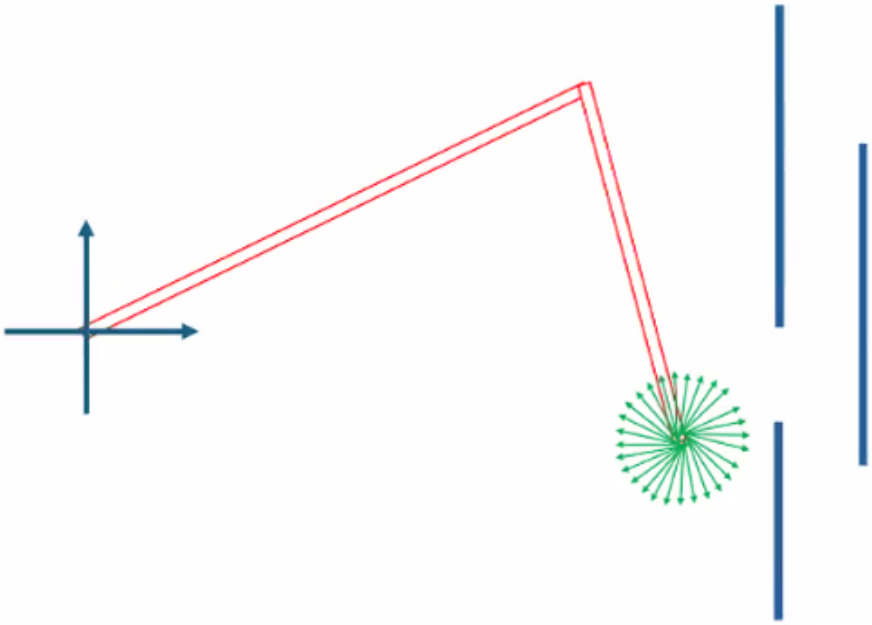

Figure 4.2 Direction search under fixed Cartesian force magnitude

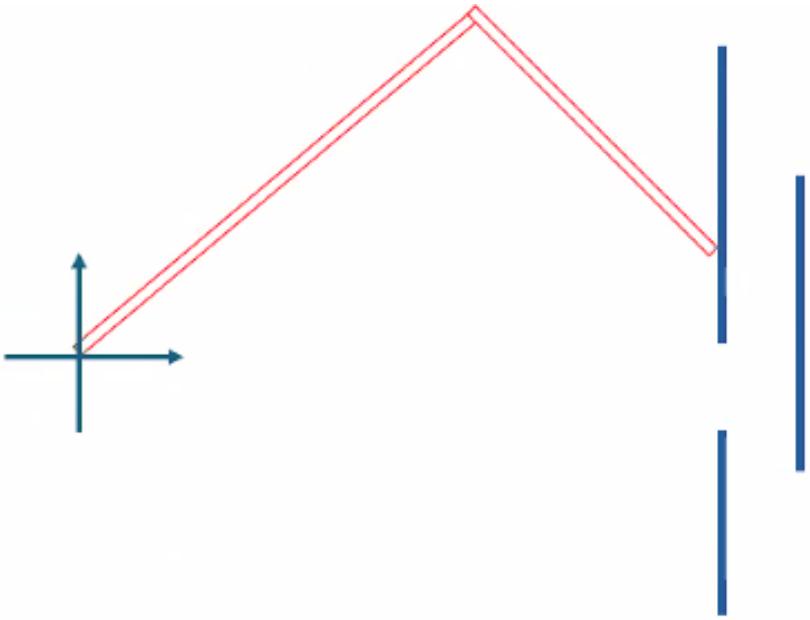

Figure 4.3 Contact with the first stage of the wall

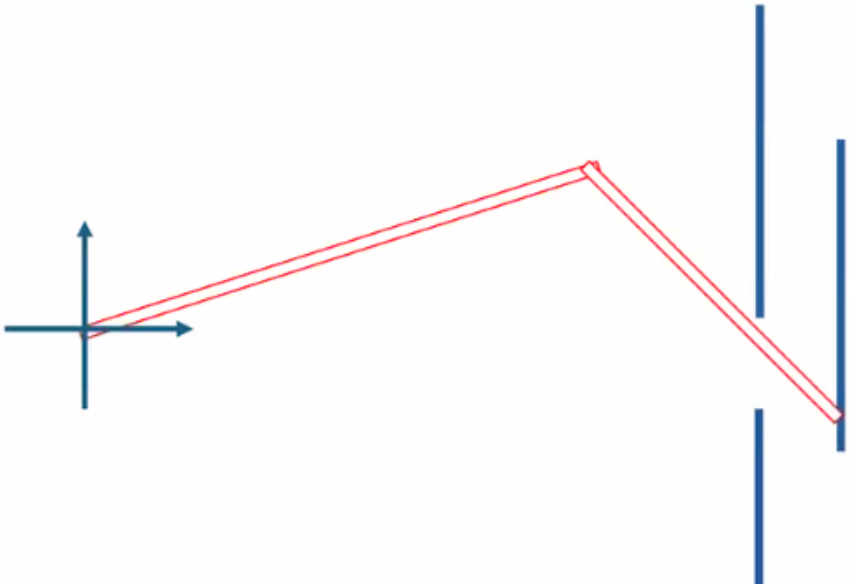

Figure 4.4 Contact with the recess (second-stage wall), which represents the goal configuration

4.2 Learning Strategy

In this study, a Q-learning algorithm is employed in conjunction with robot dynamic simulations to achieve a specific learning task. The simulations involve applying a Cartesian force with a fixed magnitude of 10 N, while its direction is determined by the learning algorithm. This force is reflected into joint torques using the tool Jacobian relation, and these torques serve as the sole control signals for the robot:

$$\tau = J^T F_{\text{Cartesian},} \tag{4.1}$$

$$F_{\text{Cartesian}} = 10 \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix}. \tag{4.2}$$

Here θ is the angle of search and is in the range of $(-\pi,+\pi)$. The angle is obtained from the discrete angle parameter in the set of [1,2,...,37] by linear interpretation. 19, for example, corresponds to θ=0.

During each learning step, the dynamic simulation is executed for a fixed simulation time, which is set to 0.1 seconds in this work. At the end of this simulation time, the Q-value corresponding to the initial state and action pair is updated.

The learning process is structured into episodes, each consisting of 30 steps. At the beginning of an episode, the initial joint positions are set randomly. However, in subsequent steps within the same episode, the destination joint configuration from the previous step becomes the initial joint configuration for the next step. After each step, in addition to updating the Q-value for the current state-action pair, the action with the maximum Q-value is identified and recorded in a table. This table is later used as a guide for determining the Cartesian direction in which the "guide" Cartesian force is applied to establish wall contact.

During the learning phase, an epsilon-greedy approach is employed to balance exploration and exploitation. At each step, a random number between 0 and 1 is generated. If this random number is greater than the epsilon value (fixed at 0.9 in this study), the action with the maximum Q-value is chosen for the current state. Otherwise, a random action is selected. This approach ensures that the algorithm explores different actions.

The described methodology combines dynamic simulation, reinforcement learning principles, and robotic control strategies to achieve effective learning and decision-making for the task at hand. By employing the epsilon-greedy approach and updating Q-values iteratively, the system learns to optimize its actions based on the accumulated knowledge and feedback from the environment.

4.3 Q-Value Update

In this thesis, the Q-learning algorithm is implemented with a specific reward mechanism. After each step, as described previously, the reward is defined as the maximum x-directional component of the robot tool tip's position during the 0.1 sec simulation. This design assumes that the wall structure is located in the positive x-direction relative to the origin. Importantly, the precise location and orientation of the wall are unknown to the learning algorithm. By defining the reward in this manner, the robot is incentivized to move toward the wall and actively search for the recess. This reward mechanism drives the robot to explore its environment effectively, motivating it to move closer to the wall structure without relying on

explicit positional data. The learning algorithm uses this reward signal to update its Q-values, gradually improving its policy for selecting actions that maximize the reward. The update rule for the Q-value incorporates this reward, along with the learning rate and discount factor, ensuring that the algorithm balances immediate rewards with long-term gains. Through this approach, the robot learns to navigate towards the wall structure and adapt its movements based on the feedback received from the environment, demonstrating the effectiveness of reinforcement learning in scenarios with limited prior knowledge of the surroundings.

The update rule is as follows:

$$Q(s_t, a_t) = (1-\alpha)Q(s_t, a_t) + \alpha\left(r_t + \gamma \max_a Q_(s_{t+1}, a)\right) \tag{4.3}$$

The update rule for the Q-value in this thesis is formulated as a discrete low-pass filter difference equation. Specifically, the Q-value of a state-action pair is updated by combining the current reward and the discounted future maximum reward, both weighted by the learning coefficient. In this equation, $Q(s_t, a_t)$ represents the quality value of taking action a_t at state s_t. Once the action a_t is executed, the system transitions to the next state, denoted as s_{t+1}.

The reward obtained from performing the action a_t is r_t. Additionally, the potential reward in the next state s_{t+1}, reached by selecting the action with the highest reward, is multiplied by the discount factor γ. This discounted future reward is then added to the current reward to provide a comprehensive measure of the action's effectiveness.

The learning constant, α, plays a crucial role in this update process. It acts as a weighting factor that determines the influence of the new information on the Q-value. From a signal processing perspective, α can also be interpreted as a low-pass filter constant, as it helps smooth the updates and reduce the impact of sudden changes in rewards.

This formulation ensures that the Q-learning algorithm balances immediate rewards with long-term gains, enabling the robot to progressively refine its policy and improve its decision-making capabilities over time.

The next chapter discusses simulation results with this algorithm.

5. SIMULATION RESULTS

5.1 Dynamics Related Routines

The simulation is based on the dynamics of a two-link planar elbow manipulator. The equations of motion, derived using the Euler-Lagrange formulation, are given by:

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + B\dot{q} + g(q) = \tau - J(q)^T F_E \tag{5.1}$$

The terms in equation 5.1 and their corresponding functions in the simulation code are:

- $D(\mathbf{q})$**:** The inertia matrix, computed by a function (see Appendix, d_computation_fnc.m).
- $C(\mathbf{q}, \dot{\mathbf{q}})$**:** The Coriolis and centrifugal matrix, computed by a function (see Appendix, c_computation_fnc.m).
- **B:** The viscous friction matrix.
- $g(\mathbf{q})$**:** The gravity vector, computed by a function (see Appendix, g_computation_fnc.m).
- $J(\mathbf{q})$**:** The Jacobian matrix, which relates joint velocities to end-effector velocity. It is computed by a function (see Appendix, j_computation_fnc.m).
- $\mathbf{F}_E$**:** The external force vector, representing the contact force from the wall, computed by a function (see Appendix, external_force_computation_fnc.m).

The forward and inverse kinematics of the manipulator are handled by dedicated functions (see Appendix, forward_kinematics_fnc.m and inverse_kinematics_fnc.m).

5.2 Q-Learning Framework and Parameters

Instead of relying on the dynamic model for control, the system learns a policy using a model-free Q-learning algorithm. The core of this approach is the discretization of the continuous state space into a manageable grid-like structure.

- **State Space:** The robot's state is a four-dimensional vector $[q_1, q_2, \dot{q}_1, \dot{q}_2]$, representing the two joint angles and their velocities. In our final work, the velocity components are dropped due to size of the space to be explored and q1 and q2 are used as stated variables. This continuous state is mapped to a discrete grid position using a discretization function. The Q-table is defined with a grid size of $25 \times 25 \times 25 \times 25$, as specified in the parameters file (see Appendix, `q_learning_ke_parat.m`).
- **Action Space:** The action space is discrete, with a size of 37 possible actions.
- **Reward Function:** The reward signal, calculated in the episode script (see Appendix, `grid_episode_ke.m`), is designed to encourage the robot to establish and maintain contact with the wall.
- **Algorithm Parameters:** The parameters file (see Appendix, `q_learning_ke_parat.m`) defines key parameters including the learning rate (α), the discount factor (γ), and the exploration rate (ϵ). The training loop (see Appendix, `episode_iteration_loop.m`) uses an epsilon-greedy policy where ϵ decays over time.

5.3 Simulation Architecture

The simulation is structured around a main training loop (see Appendix, `episode_iteration_loop.m`) that iterates for a set number of episodes. In each episode, the `grid_episode_ke.m` script is called to simulate a fixed number of steps. The `one_second_simulator_fnc.m` serves as the physics engine, calculating the robot's dynamic response to the chosen action. The Q-table is updated at each step using the Q-learning update rule implemented in `q_updater_ke.m` (see Appendix). After the training is complete, the learned policy is evaluated using a dedicated test simulator (see Appendix, `test_simulator_fnc.m`). The results are then visualized using a plotting script (see Appendix, `result_plotter.m`).

This section will present the quantitative and visual results of the Q-learning simulation. The plots are generated by `result_plotter.m`.

- **Force Convergence:** A plot showing the end-effector's contact force magnitude over time. This figure will demonstrate the agent's ability to approach the wall, make contact, and regulate the force to a stable target value.
- **End-Effector Trajectory:** A plot of the end-effector's position in the Cartesian (x-y) plane, visually representing the path taken by the robot from its initial position to the point of contact with the wall.

5.4 Simulations

Simulation results are presented in Figures 5.1–5.4. A learning process spanning a total of 1000 individual episodes was conducted. Within each of these episodes, a sequence of dynamic simulation-based searches was performed, with each episode comprising 30 steps. Each step was executed over a brief duration of 0.1 seconds. The learning algorithm employed specific hyperparameters to guide its optimization process. One of these parameters, ϵ, which is commonly associated with exploration–exploitation trade-offs in reinforcement learning, was set to a value of 0.9, encouraging a higher degree of exploration during the learning procedure. Additionally, the learning rate, denoted as α, was assigned a value of 0.1, ensuring gradual updates to the system's knowledge base. The discount factor, γ, which plays a critical

role in determining the importance of future rewards, was configured to be 0.8, striking a balance between immediate and long-term reward considerations. This entire process, including the computationally intensive simulations and parameter adjustments, was executed on an Intel i-9 processor, which enabled the code to run efficiently. The total time required for completing all 1000 episodes, along with their associated computations, amounted to approximately 300 seconds.

Figure 5.1 shows the heatmap of learned actions on the discrete shoulder angle–elbow angle plane. Shades of blue indicate the 37 actions over the 25×25 state plane. Upon closer examination, it becomes evident that the heatmap reveals a pattern of varying intensities, with darker and lighter shades distributed across different locations within the discrete state space. These shades represent the relative values or characteristics of specific regions within the state space, offering a visual depiction of how the system behaves or responds under certain conditions. However, a noteworthy observation is that a significant number of cells within the discrete state space exhibit action values that are relatively average or moderate, rather than displaying extreme values. This phenomenon can be attributed to the dynamics of the learning process itself, particularly during its initial stages. Early in the learning procedure, a solution was discovered for establishing contact with the second segment or gate of the wall. This discovery had a profound impact on the subsequent evolution of the learning process, shaping the way the system interacted with the environment and influencing the distribution of action values across the state space. As a result, many regions within the state space settled into a pattern where their action values remained within an average range, reflecting the influence of the early solution on the learning trajectory. This insight underscores the importance of initial discoveries in shaping the overall behavior of a system during a learning process and highlights how specific solutions can contribute to the stabilization of certain regions within the state space.

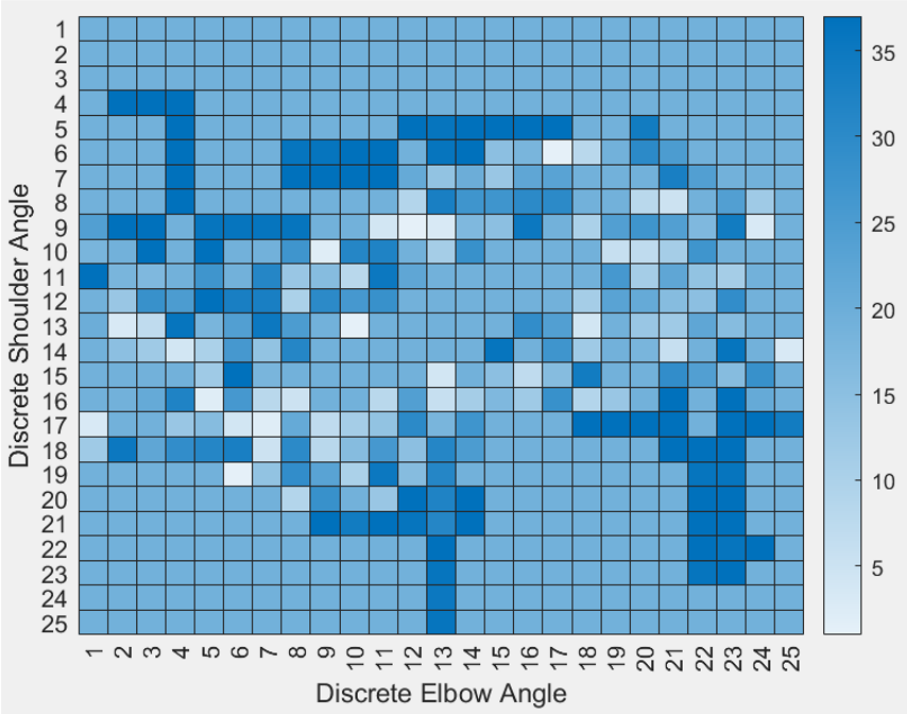

Figure 5.1 The heatmap of learned actions on the shoulder angle-elbow angle plane. The shades of blue indicate the actions with the highest Q-value.

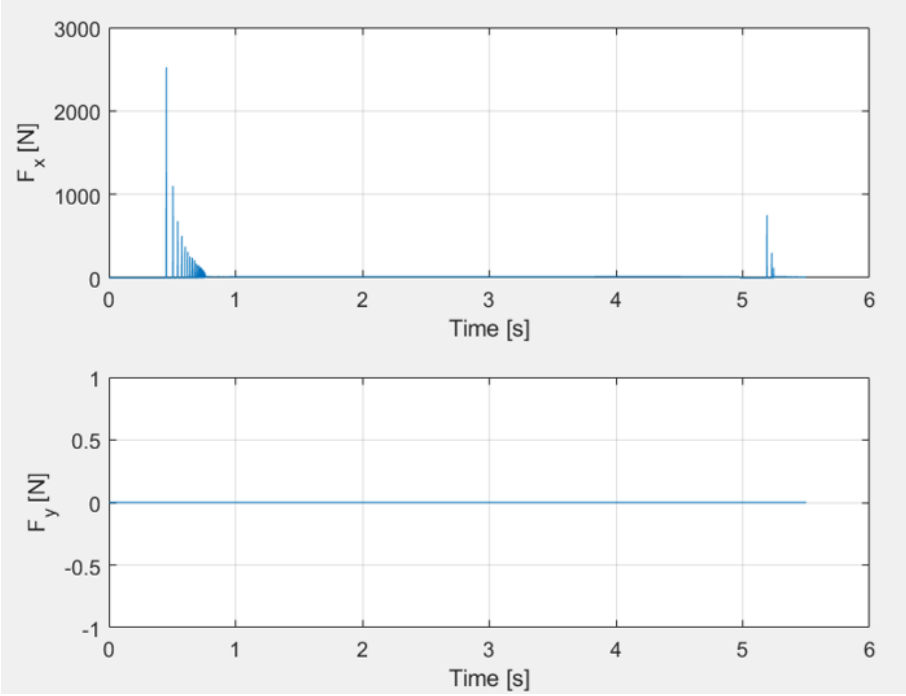

Figure 5.2 Contact force components in the test simulation. The y-directional component of contact force is not modelled. This corresponds to frictionless walls.

The x-directional force component exhibits two distinct regions characterized by the woodpecker effect, with one region centered around 0.7 seconds and the other around 5.2 seconds. The first region is more pronounced due to the longer distance of acceleration following the initial standstill, resulting in a greater force impact. During the first stage of interaction with the wall, the tool tip comes to a temporary halt in the x direction, before the second stage of contact is established. In this second stage, the contact force peaks are smaller, and the number of knocks is reduced, indicating a less intense interaction compared to the initial stage.

In the simulations, the first stage of the wall is positioned at a distance of 0.6 meters from the origin, oriented perpendicular to the x-axis. This section includes a gap spanning 20 centimeters, located between $y = 0.1$ meters and $y = -0.1$ meters, effectively creating an opening within these boundaries. The second stage is aligned parallel to the first wall, maintaining a consistent orientation, and is situated at a distance of 0.65 meters from the origin. This arrangement ensures a structured layout with precise spatial relationships between the two stages.

Figure 5.3 provides detailed time plots illustrating the movement of the robot tool tip along both the x and y coordinates. The trajectory in the x-direction ultimately comes to a halt when it encounters the recessed section of the wall, effectively stopping its forward motion. In contrast, the movement in the y-direction behaves differently, as the tool tip does not come to a stop but instead slides along the surface of the wall, maintaining a tangential path relative to its structure.

Figure 5.4 illustrates the x-y trajectory of the tool tip as it progresses through its motion. Initially, the tool tip interacts with the surface of the first wall stage, sweeping across it before transitioning into the gap located between the two stages. Following this movement, the tool tip ultimately establishes contact with the second wall stage, completing its trajectory.

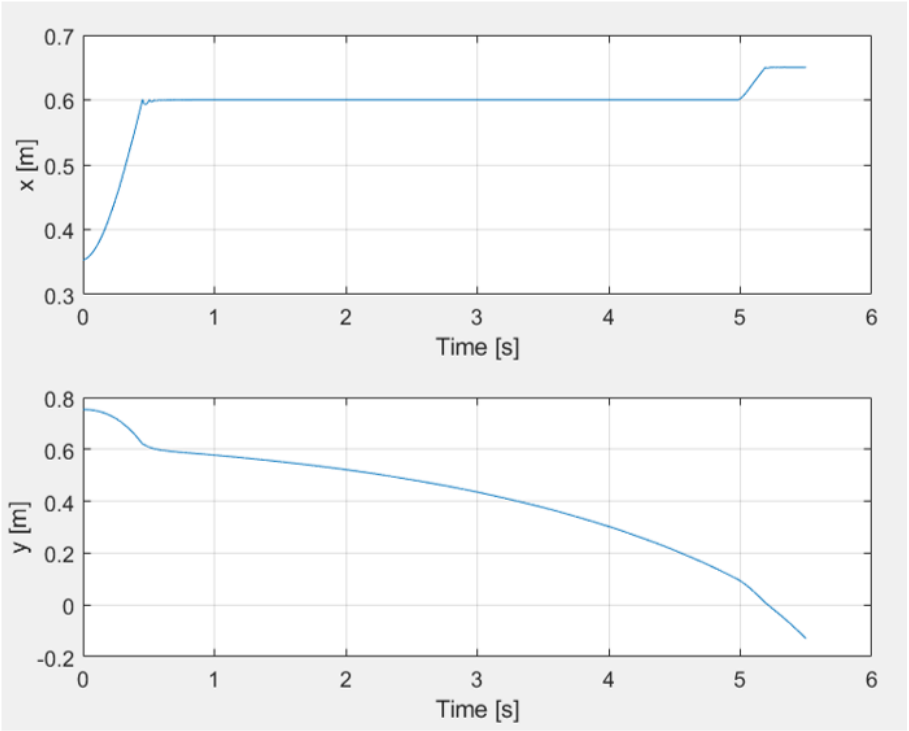

Figure 5.3 Cartesian x and y trajectories

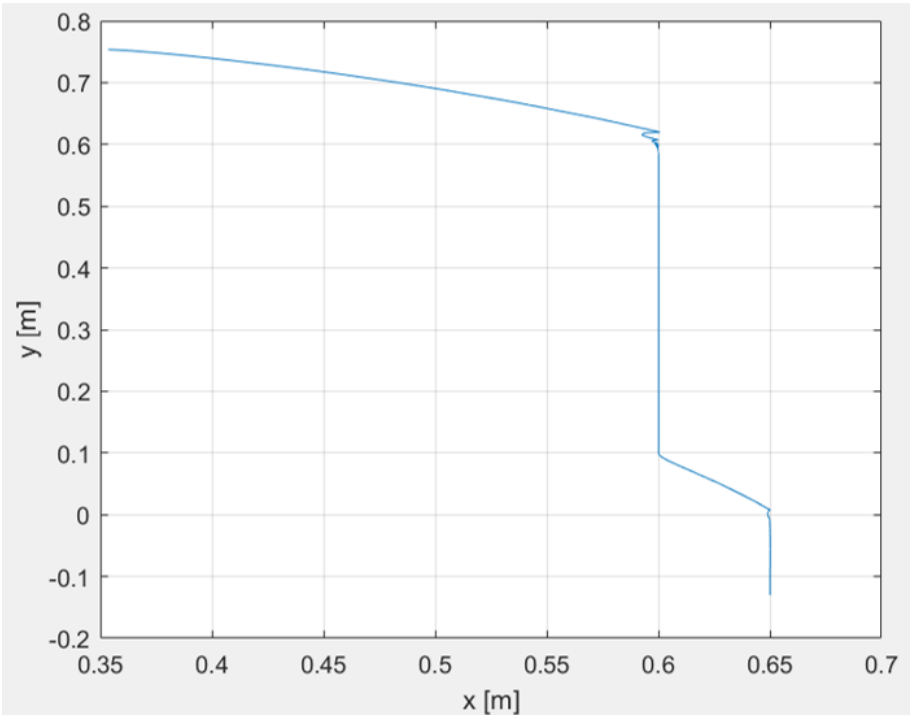

Figure 5.4 x-y trajectory obtained with the learned actions

6. CONCLUSION

This thesis explores the application of Q-learning, a reinforcement learning technique, in the domain of robotic force control for contact tasks, specifically focusing on a planar elbow manipulator. By leveraging the principles of reinforcement learning within a grid-inspired framework, the study successfully demonstrates how adaptive and model-free regulation of contact establishment can be achieved with minimal training requirements. The approach emphasizes simplicity and efficiency, showcasing the ability of a lightweight controller to deliver effective performance with rapid convergence rates. This characteristic positions the controller as a practical solution for real-time systems, where responsiveness and adaptability are critical. The primary contribution of this work lies in validating that such a streamlined learning mechanism can meet the demands of dynamic environments, offering a promising alternative to more complex, resource-intensive methods.

In addition to demonstrating feasibility, the proposed framework contributes to ongoing efforts in bridging the gap between fundamental reinforcement learning methods and practical robotic applications. Unlike classical force control approaches, which often rely on accurate models and extensive offline tuning, the results of this study highlight the potential of model-free learning strategies in uncertain and variable conditions. This makes the approach particularly relevant for tasks in industrial assembly, surface processing, and human–robot collaboration, where safe and adaptive force regulation is crucial.

Nevertheless, the study has been limited to a simulation-based environment. The transition to physical robotic platforms will inevitably introduce additional challenges such as sensor noise, friction uncertainties, and communication delays. Future research should therefore investigate sim-to-real transfer strategies, to improve robustness in real-world implementations. Moreover, the integration of lean deep reinforcement learning methods could provide the ability to handle higher-dimensional state and action spaces, thereby extending the applicability of the approach to more complex manipulation tasks.

In summary, this thesis shows that a relatively simple reinforcement learning framework can achieve rapid and stable learning in robotic contact tasks. The findings indicate that lightweight, model-free controllers can be both efficient and effective.

Future research efforts could focus on applying the proposed methodology to a physical robotic system. This would involve transitioning from simulation-based studies to practical implementation, enabling the validation of theoretical findings in real-world scenarios.

BIBLIOGRAPHY

Ajoudani, A., Argiolas, F., Baldi, T., Catalano, M., Gabiccini, M., Malfatti, M., & Bicchi, A. (2018). Progress and prospects of the human–robot collaboration. *Autonomous Robots*, *42*, 1–28.

Andrychowicz, M., Rahtizadeh, R., & Zaremba, W. (2020). What matters in on-policy reinforcement learning? *arXiv preprint arXiv:2006.05990*.

Argall, B. D., Chernova, S., Veloso, M., & Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, *57*(5), 462–473.

Beltran-Hernandez, V., Pliam, J., Komeil, K., & Juergen, S. (2020). Variable compliance control for robotic peg-in-hole assembly. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 7709–7715.

Bicchi, A., & Tonietti, G. (2004). Fast and "soft-arm" tactics: Dealing with the safety–performance trade-off in robot arms design and control. *IEEE Robotics & Automation Magazine*, *11*(2), 22–33. https://doi.org/10.1109/MRA.2004.1310939

Chen, Z., Ma, Y., Han, S., & Li, Z. (2022). Position/force visual-sensing-based sheet-like peg-in-hole assembly. *Sensors*, *22*(1), 324.

De Luca, A., & Mattone, R. (2005). Sensorless robot collision detection and hybrid force/motion control. *IEEE Transactions on Robotics*, *21*(4), 549–561.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., & Abbeel, P. (2017). One-shot imitation learning. *Advances in Neural Information Processing Systems*, 1087–1097.

Fan, M., Gu, S., & Zhang, J. (2019). Ddpg with guided policy search for precision insertion. *arXiv preprint arXiv:1904.09506*.

Fujimoto, S., van Hoof, H., & Meger, D. (2018). Addressing function approximation error in actor-critic methods. *International Conference on Machine Learning (ICML)*, 1656–1665.

Hogan, N. (1985). Impedance control: An approach to manipulation. part i—theory, part ii—implementation, part iii—applications. *Journal of Dynamic Systems, Measurement, and Control*, *107*(1), 1–24. https://doi.org/10.1115/1.3140702

Huang, H., Li, T., & Chen, Y. (2020). Fusion of force and visual sensing for robotic insertion tasks. *Journal of Field Robotics*, *37*(4), 551–564.

Ikeura, R., & Inooka, H. (1995). Variable impedance control of a robot for cooperative motion with a human. *1995 IEEE International Conference on Robotics and Automation*, 3093–3098.

Jeong, D., Park, S., & Lee, J. (2020). Hierarchical reinforcement learning for industrial insertion tasks. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 8171–8177.

Jiang, W., Han, C., Li, C., & Zhang, J. (2021). Hybrid visual sensing with coarse-to-fine view switching for accurate positioning. *Robotics and Computer-Integrated Manufacturing*, *69*, 102120.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–285. https://doi.org/10.1613/jair.291

Kim, Y., Lim, H., & Kim, J. (2016). Flexible peg-in-hole assembly strategy using learning and visual feedback. *IEEE Transactions on Robotics*, *32*(4), 1017–1025.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Luo, R., Xu, S., & Li, R. (2018). Learning peg-in-hole on deformable surfaces using guided policy search. *arXiv preprint arXiv:1807.03714*.

Nagabandi, A., Konolige, K., & Levine, S. (2018). Neural network dynamics for model-based deep rl with model-free fine-tuning. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 8251–8258.

Park, D., Choi, M., & Kim, Y. (2019). Learning compliant peg-in-hole assembly with multimodal feedback. *2019 IEEE International Conference on Robotics and Automation (ICRA)*, 3704–3710.

Raibert, M. H., & Craig, J. J. (1981). Hybrid position/force control of manipulators. *Journal of Dynamic Systems, Measurement, and Control*, *103*(2), 126–133. https://doi.org/10.1115/1.3139652

Rusu, A. A., Colmenarejo, S. G., Gulcehre, C., Desjardins, G., Kirilov, T., & van Hoof, H. (2017). Sim-to-real transfer for rl with progressive networks. *arXiv preprint arXiv:1702.08343*.

Sadeghi, F., Legg, J., & Levine, S. (2016). Cad2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.08503*.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT Press.

Takahashi, T., & Fujita, M. (2020). Passive alignment mechanism for deformable thin-ring insertion. *Robotics and Computer-Integrated Manufacturing*, *62*, 101880.

Tobin, J., Fong, G., Ray, A., Schneider, J., Zaremba, W., Abbeel, P., et al. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 23–30.

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*(3-4), 279–292. https://doi.org/10.1007/BF00992698

Whitney, D. E. (1987). Historical perspective and state of the art in robot force control. *International Journal of Robotics Research*, *6*(1), 3–14. https://doi.org/10.1177/027836498700600101

Whitney, D. E. (1982). Quasi-static assembly of compliantly supported rigid parts. *Journal of Dynamic Systems, Measurement, and Control*, *104*(1), 65–77.

Xu, J., Liang, S., & Wu, Y. (2019). Compare contact model-based control and contact model-free learning. *2019 IEEE International Conference on Automation Science and Engineering (CASE)*, 2060–2065.

Yamamoto, A., & Fujita, M. (2017). Contact force-based precision insertion using a position-controlled robot arm. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 642–647.

Zhao, D., Ma, G., & Liu, Y. (2019). Compliant control of dual-arm robots for assembly. *IEEE Access*, *7*, 137397–137407.

APPENDIX

Software Listings

Listing A1

addrun.m

```
function addrun()
    episode_iteration_beginning_no=number_of_episodes+1;
    number_of_episodes=number_of_episodes+number_of_episodes_to_add

    episode_performance_list=[episode_performance_list, zeros(1,
       number_of_episodes_to_add)];
    filtered_episode_performance_list=[
       filtered_episode_performance_list, zeros(1,
       number_of_episodes_to_add)];
    heavily_filtered_episode_performance_list=[
       heavily_filtered_episode_performance_list, zeros(1,
       number_of_episodes_to_add)];

    epsilon_list=[epsilon_list, zeros(1,number_of_episodes_to_add)
       ];

    %epsilon=0.9;

    episode_iteration_loop
    result_plotter
end
```

Listing A2

c_computation_fnc.m

```
function result = c_computation_fnc(theta, theta_dot)

global l1 lc1 w1 h1 m1 I1
global l2 lc2 w2 h2 m2 I2

sinq2 = sin(theta(2));
h = -m2*l1*lc2*sinq2;
C=zeros(2,2);
C(1, 1) = h * theta_dot(2);
C(1, 2) = h * (theta_dot(1)+theta_dot(2));
C(2, 1) = -h * theta_dot(1);
C(2, 2) = 0;
result=C;
```

Listing A3

d_computation_fnc.m

```
function result = d_computation_fnc(theta)

global l1 lc1 w1 h1 m1 I1
global l2 lc2 w2 h2 m2 I2

theta;
cosq2 = cos(theta(2));

D=zeros(2,2);
D(1,1) = m1 * lc1^2 ...
       + m2 * (l1^2 + lc2^2 +2*l1*lc2*cosq2) ...
       + I1 ...
       + I2;
D(1,2) = m2 * (lc2^2 + l1*lc2*cosq2) ...
       + I2;
D(2,1) = D(1, 2);
D(2,2) = m2 * lc2^2 ...
       + I2;

result=D;
```

Listing A4

episode_iteration_loop.m

```
for episode_iteration_no=episode_iteration_beginning_no:
   number_of_episodes

    episode_iteration_no

    %epsilon=epsilon-epsilon_decrement
    if epsilon < epsilon_lower_limit
        epsilon=epsilon_lower_limit;
    end
    epsilon_list(episode_iteration_no)=epsilon;

    episode_result=grid_episode_ke();

    total_episode_reward=0;
    step_counter=0;
    for iteration_index=1:episode_length
        if episode_result(iteration_index,11)~=333
            total_episode_reward ...
                =total_episode_reward ...
                +episode_result(iteration_index,10);  %1 apr 2025
                %+episode_result(iteration_index,7);  %1 apr 2025

            step_counter=step_counter+1;
        end
    end
    average_episode_reward=total_episode_reward/step_counter;
    episode_performance_list(episode_iteration_no) ...
        =average_episode_reward;

    filtered_average_episode_reward ...
        =average_episode_reward*display_lambda ...
        +filtered_average_episode_reward*(1-display_lambda);
    filtered_episode_performance_list(episode_iteration_no) ...
        = filtered_average_episode_reward;

    heavily_filtered_average_episode_reward ...
        =average_episode_reward*diplay_heavy_lambda ...
        +heavily_filtered_average_episode_reward*(1-
           diplay_heavy_lambda);
    heavily_filtered_episode_performance_list(episode_iteration_no)
        ...
        = heavily_filtered_average_episode_reward;

end
```

Listing A5

external_force_computation_fnc.m

```
function result= external_force_computation_fnc(tool_cartesian_pos,
    tool_cartesian_vel)

global x_wall b_wall k_wall
global hole_size_wall
global wall_model_active

    p_tool_x=tool_cartesian_pos(1);
    p_tool_y=tool_cartesian_pos(2);
    v_tool_x=tool_cartesian_vel(1);

    f_tool_vector =zeros(2,1);

    if p_tool_x > x_wall
        f_spring = k_wall*(p_tool_x - x_wall);
    else
        f_spring = 0;
    end

    if p_tool_x > x_wall && v_tool_x > 0
        f_damper=b_wall*v_tool_x;
    else
         f_damper=0;
    end

    f_tool_vector(1) = f_spring + f_damper;
    f_tool_vector(2) = 0;

    %hole
    if (abs(p_tool_y) < hole_size_wall/2) && (p_tool_x > x_wall)
        wall_model_active=0;
        % f_tool_vector(1) =0;
        % f_tool_vector(2) =0;
    end

    if wall_model_active==0
        f_tool_vector(1) =0;
        f_tool_vector(2) =0;
    end

    result=f_tool_vector;
```

Listing A6

forward_kinematics_fnc.m

```
function result=forward_kinematics_fnc(theta)

global l1 lc1 w1 h1 m1 I1
global l2 lc2 w2 h2 m2 I2

    cosq1 = cos(theta(1));
    cosq12 = cos(theta(1)+ theta(2));
    sinq1 = sin(theta(1));
    sinq12 = sin(theta(1)+ theta(2));

    p_vector =zeros(2,1);
    p_vector(1) = l1 * cosq1 + l2 * cosq12;
    p_vector(2) = l1 * sinq1 + l2 * sinq12;

    result=p_vector;
```

Listing A7

g_computation_fnc.m

```
function result=g_computation_fnc(theta)

global l1 lc1 w1 h1 m1 I1
global l2 lc2 w2 h2 m2 I2
global g_c

    cosq1 = cos(theta(1));
    cosq12 = cos(theta(1)+ theta(2));

    g=zeros(2,1);
    g(1) = ( (m1*lc1 + m2 *l1)*cosq1 + m2*lc2*cosq12 )*g_c;
    g(2) = m2*lc2*cosq12*g_c;

    result=g;
```

Listing A8

grid_episode_ke.m

```
function result=grid_episode()

global start_position
global end_position
global episode_length
global list_of_forbidden_locations
global number_of_forbidden_locations
global gamma
global number_of_actions
global policy_seed_ke
global episode_iteration_no
global epsilon
global max_q_action_array
global alpha
global q_array
global x_wall
global grid_x_mid grid_x2_mid
global grid_y_mid grid_y2_mid

result=zeros(episode_length,11);

x_limit=0.6;
p_forw=forward_kinematics_fnc([0;0]);
x_forw=p_forw(1);

while (x_forw > x_limit) || (x_forw < 0.1)
    random_p(1)=pi-rand*2*pi;
    random_p(3)=pi-rand*2*pi;
    p_forw=forward_kinematics_fnc([random_p(1);random_p(3)]);
    x_forw=p_forw(1);
end
random_p(2)=0;
random_p(4)=0;
step_initial_position=random_p';

result(1:episode_length,11)=333*ones(episode_length,1);

for step_iteration_no=1:episode_length
    if step_initial_position ~=[333;333;333;333]

        [action_number,action_value]=policy_ke_fnc(
           step_initial_position);

        [step_final_position,reward]=one_second_simulator_fnc(
           step_initial_position(1), ...
```

```
            step_initial_position(2), ...
            step_initial_position(3), ...
            step_initial_position(4), ...
            action_number);

        result(step_iteration_no,1:4)=step_initial_position';
        result(step_iteration_no,5)=action_number;
        result(step_iteration_no,6:9)=[step_final_position(1)
           grid_y_mid step_final_position(3) grid_y2_mid];
        result(step_iteration_no,10)=reward;

        step_initial_position=step_final_position;

    end

    if step_final_position ~=[333;333;333;333]
        dummy_aaa=q_updater_ke(result(step_iteration_no,:));
    end

end

reward_column_with_extra_row=[result(:,10);333];
for step_iteration_no=episode_length:-1:1
    if    reward_column_with_extra_row(step_iteration_no+1)==333
       ...
       && reward_column_with_extra_row(step_iteration_no)  ~=333
        last_reward=reward_column_with_extra_row(step_iteration_no)
           ;
        result(step_iteration_no,11)=last_reward;
    end
    if    reward_column_with_extra_row(step_iteration_no+1)~=333
       ...
       && reward_column_with_extra_row(step_iteration_no)  ~=333
        result(step_iteration_no,11)=reward_column_with_extra_row(
           step_iteration_no)...
            +gamma*result(step_iteration_no+1,11);
    end
end
```

Listing A9

inverse_kinematics_fnc.m

```
function result=inverse_kinematics_fnc(p_input)

global l1 lc1 w1 h1 m1 I1
global l2 lc2 w2 h2 m2 I2

x_input=p_input(1);
y_input=p_input(2);

D_variable=(x_input^2+y_input^2-l1^2-l2^2)/(2*l1*l2)

if norm(D_variable) > 1
    result=[555;555];
else
    theta2_output=atan2(sqrt(1-D_variable^2),D_variable);
    theta1_output=atan2(y_input,x_input) ...
                  -atan2(l2*sin(theta2_output),l1+l2*cos(
                     theta2_output));
    result=[theta1_output;theta2_output];
end
```

Listing A10

j_computation_fnc.m

```
function result=j_computation_fnc(theta)

global l1 lc1 w1 h1 m1 I1
global l2 lc2 w2 h2 m2 I2

    cosq1 = cos(theta(1));
    cosq12 = cos(theta(1)+ theta(2));
    sinq1 = sin(theta(1));
    sinq12 = sin(theta(1)+ theta(2));

    J=zeros(2,2);
    J(1, 1) = -l1*sinq1 - l2*sinq12;
    J(1, 2) = -l2*sinq12;
    J(2, 1) =  l1*cosq1 + l2*cosq12;
    J(2, 2) =  l2*cosq12;

    result=J;
```

Listing A11

q_learning_ke_parat.m

```
clear all
close all

global start_position
global end_position
global grid_x_size grid_y_size grid_x2_size grid_y2_size
global episode_length
global list_of_forbidden_locations
global number_of_forbidden_locations
global gamma
global number_of_actions
global new_return_contribution_coefficient
global policy_seed_ke
global episode_iteration_no
global epsilon
global max_q_action_array
global q_array
global alpha

global l1 lc1 w1 h1 m1 I1
global l2 lc2 w2 h2 m2 I2
global g_c
global x_wall b_wall k_wall
global step_time stop_time no_of_iterations
global Kp Kd
global B
global wall_contact_counter
global mid_action_number
global Force_coefficient
global grid_x_mid grid_y_mid grid_x2_mid grid_y2_mid
global attack_force
global hole_size_wall
global wall_model_active

%simulation parameters
step_time = 0.001;
stop_time = 0.1;
no_of_iterations = floor(stop_time/step_time + 1);

no_of_links = 2;
l1 = 0.3;
lc1 = 0.15;
w1 = 0.02;
h1 = 0.02;
m1 = 2.5;
```

```
I1 = m1*(w1^2+h1^2)/12;

l2 = 0.3;
lc2 = 0.15;
w2 = 0.02;
h2 = 0.02;
m2 = 2.5;
I2 = m2*(w2^2+h2^2)/12;

g_c=9.81;

x_wall=0.4;
b_wall=10;
k_wall=100;
hole_size_wall=0.05;

B = [0.1 0; 0 0.1];
Kp = [200 0 ; 0 130];
Kd = [25 0 ; 0 8];

Force_coefficient=1;
attack_force=10;

%learning parameters
gamma=0.9;
display_lambda=0.01;
diplay_heavy_lambda=0.001;
alpha=0.1;
epsilon_upper_limit=0.9;
epsilon_lower_limit=0.1;
epsilon=0.9;
epsilon_decrement=0.001;

grid_x_size=25;
grid_y_size=25;
grid_x2_size=25;
grid_y2_size=25;

grid_x_mid=1+(grid_x_size-1)/2;
grid_y_mid=1+(grid_y_size-1)/2;
grid_x2_mid=1+(grid_x2_size-1)/2;
grid_y2_mid=1+(grid_y2_size-1)/2;

start_position=x_y_4_discretization_fnc([1;0;1;0]);

episode_length=30;
number_of_episodes=100;
```

```
number_of_episodes_to_add=800;

number_of_actions=37;
mid_action_number=1+(number_of_actions-1)/2;

policy_seed_ke=ones(1,number_of_actions)/number_of_actions;

q_array=zeros(grid_x_size,grid_y_size,grid_x2_size,grid_y2_size,
   number_of_actions);
max_q_action_array=ceil(number_of_actions*rand(grid_x_size,
   grid_y_size,grid_x2_size,grid_y2_size));

episode_performance_list=zeros(1,number_of_episodes);
filtered_episode_performance_list=zeros(1,number_of_episodes);
heavily_filtered_episode_performance_list=zeros(1,
   number_of_episodes);
epsilon_list=zeros(1,number_of_episodes);

filtered_average_episode_reward=0;
heavily_filtered_average_episode_reward=0;

wall_contact_counter=0;
wall_model_active=1;

episode_iteration_beginning_no=1;
episode_iteration_loop
result_plotter
```