algorithms

MDPI

Article

Adaptive Compensation for Robotic Joint Failures Using Partially Observable Reinforcement Learning

Tan-Hanh Pham [1,*], Godwyll Aikins [1], Tri Truong [2] and Kim-Doang Nguyen [1,*]

[1] Department of Mechanical and Civil Engineering, Florida Institute of Technology, Melbourne, FL 32901, USA
[2] Department of Fundamentals of Machine Design, HCMC University of Technology and Education, HCM City 700000, Vietnam; tri.truongquang@hcmute.edu.vn
* Correspondence: tpham2023@my.fit.edu (T.-H.P.); knguyen@fit.edu (K.-D.N.)

Abstract: Robotic manipulators are widely used in various industries for complex and repetitive tasks. However, they remain vulnerable to unexpected hardware failures. In this study, we address the challenge of enabling a robotic manipulator to complete tasks despite joint malfunctions. Specifically, we develop a reinforcement learning (RL) framework to adaptively compensate for a nonfunctional joint during task execution. Our experimental platform is the Franka robot with seven degrees of freedom (DOFs). We formulate the problem as a partially observable Markov decision process (POMDP), where the robot is trained under various joint failure conditions and tested in both seen and unseen scenarios. We consider scenarios where a joint is permanently broken and where it functions intermittently. Additionally, we demonstrate the effectiveness of our approach by comparing it with traditional inverse kinematics-based control methods. The results show that the RL algorithm enables the robot to successfully complete tasks even with joint failures, achieving a high success rate with an average rate of 93.6%. This showcases its robustness and adaptability. Our findings highlight the potential of RL to enhance the resilience and reliability of robotic systems, making them better suited for unpredictable environments.

Keywords: deep reinforcement learning; inverse kinematics; partial observability; fault-tolerant control

Citation: Pham, T.-H.; Aikins, G.; Truong, T.; Nguyen, K.-D. Adaptive Compensation for Robotic Joint Failures Using Partially Observable Reinforcement Learning. *Algorithms* **2024**, *17*, 436. https://doi.org/10.3390/a17100436

Academic Editor: Antonio Della Cioppa

Received: 21 August 2024
Revised: 26 September 2024
Accepted: 29 September 2024
Published: 1 October 2024

1. Introduction

Robotic manipulators are transforming industries across the board, from manufacturing and logistics to healthcare and agriculture. Their precision, versatility, and ability to handle complex tasks make them indispensable in modern automation. As artificial intelligence and sensing technologies advance, these robots are becoming increasingly adaptable, promising even greater impact in the future. The global market for industrial robots reached 16.5 billion USD in 2020, with manipulators being the most widely adopted type [1].

However, like any complex system, robotic manipulators are prone to faults. These faults can lead to performance issues, task failures, or even safety hazards. In critical applications such as medical surgery or space exploration, a malfunction could have catastrophic consequences. For instance, incidents involving collaborative robots in automotive plants led to temporary halts in production and raised concerns about the safety of human–robot interactions [2]. This vulnerability underscores the importance of fault-tolerant control (FTC) in robotic systems. FTC is crucial for ensuring continuous operation despite faults or failures, enhancing robotic manipulators' reliability and safety. As these machines become more prevalent in high-stakes environments, the need for robust fault-tolerance mechanisms becomes increasingly vital.

FTC approaches can be broadly categorized into traditional model-based methods and emerging learning-based techniques [3,4]. Model-based FTC relies on mathematical representations of the robotic system to detect and address faults [5]. These methods

typically employ observers or estimators to monitor performance and identify deviations from expected behavior, signaling potential faults. While effective, they may struggle with highly complex systems due to the need for detailed fault models [6]. In contrast, learning-based FTC leverages machine learning algorithms to enhance fault tolerance [7]. This approach can adapt to new and unforeseen faults by learning from data, potentially offering better scalability for complex systems. By utilizing large datasets and sophisticated algorithms, learning-based methods can potentially overcome some limitations of traditional approaches, especially in handling intricate or unexpected fault scenarios.

Learning-based FTC methods show significant potential, but their current implementation often involves layering machine learning algorithms onto existing control systems [8–10]. These algorithms monitor system performance and adjust control actions based on fault detection and diagnosis. This approach offers advantages in modularity, allowing for easier updates or replacements of individual components without overhauling the entire control system. However, this integration strategy has limitations. Response times may be slower due to the additional communication layer between modules [11]. The system also requires extensive tuning and training for various operating conditions, potentially limiting its adaptability. Moreover, its effectiveness can be compromised if the underlying mathematical model of the robotic system is inaccurate.

To address these challenges, we propose an innovative end-to-end learning-based framework for fault-tolerant control of robotic manipulators. This approach harnesses the power of deep reinforcement learning (DRL) to create a unified, adaptive, and efficient control system capable of dynamically handling faults. Our proposed system learns to manage faults directly from raw sensory inputs, eliminating the need for separate fault detection, diagnosis, and control modules.

Importantly, we frame this problem as a partially observable Markov decision process (POMDP). In a POMDP, the agent does not have full information about the state of the environment. This partial observability is particularly relevant in our scenario, where joint malfunctions may occur without explicit notification. The robot must infer the state of its joints from its observations and actions, making decisions under uncertainty.

This integration offers several advantages:

1. Unified approach: By combining fault detection, diagnosis, and control into a single system, we potentially reduce complexity and improve response times.
2. Adaptability: The DRL agent can learn to handle a wide range of faults, including those not explicitly modeled during training, enhancing the system's robustness.
3. Efficiency: The direct processing of raw sensory data eliminates the need for complex feature engineering or intermediate representations.
4. Scalability: As the complexity of the robotic system increases, the DRL approach can potentially scale more effectively than traditional methods when given sufficient training data and computational resources.
5. Continuous learning: The system is designed to update its policies in real time, allowing for ongoing adaptation to new fault scenarios or changing operating conditions.

Our novel framework aims to push the boundaries of fault-tolerant control in robotics, potentially offering a more robust and flexible solution for increasingly complex robotic systems. The rest of the paper is organized as follows. Section 2 reviews the related work on fault-tolerant control strategies and RL in robotic systems. Section 3 presents the details of our proposed methodology, including the problem formulation, the DRL algorithm for fault-tolerant control, and the simulation setup. Compared against the traditional method, Section 4 illustrates the failure of inverse kinematic control when one of the joints of a robot is broken. Section 5 describes the experimental results obtained from various fault scenarios, demonstrating the effectiveness of our approach. Finally, Section 6 provides concluding remarks and discusses potential future work.

2. Literature Review

The increasing complexity and autonomy of robotic systems necessitate robust FTC strategies to ensure reliable operation, even in the presence of faults [5]. FTC encompasses strategies and algorithms that enable robotic systems to adapt to faults and continue operation, potentially with degraded performance, rather than experiencing catastrophic failure [12]. Faults can arise from various sources within a robotic system. Actuator faults, including partial or complete loss of effectiveness, stuck actuators, and total actuator failures, can significantly impair a robot's ability to execute desired movements and interact with its environment [13]. Sensor faults, ranging from bias, noise, and drift to complete sensor failures, can compromise a robot's perception of its surroundings, leading to inaccurate localization and decision making [14]. Structural faults, such as physical damage or degradation to robot components, can induce changes in dynamic behavior, affecting stability and control performance [15]. The core objective of FTC is to detect, isolate, and accommodate these faults to maintain overall system stability, performance, and safety. Broadly, FTC approaches are classified into two main categories. Passive FTC methods leverage robust control techniques to design controllers that are inherently tolerant to a pre-defined range of faults [16]. While not requiring explicit fault detection and isolation (FDI), their fault tolerance capacity is often limited. Active FTC methods, on the other hand, hinge on real-time FDI to accurately identify and isolate faults [3]. Based on the detected fault information, the controller is reconfigured or adapted online to preserve stability and achieve the desired performance under the given fault conditions.

FTC strategies have been successfully applied to a wide range of robotic systems, demonstrating their versatility in mitigating the impact of various fault types. Sun et al. [17] proposed an innovative incremental nonlinear FTC method for quadcopters experiencing a catastrophic failure: the complete loss of two opposing rotors. This critical scenario typically renders such aerial vehicles inoperable. However, by implementing this advanced FTC strategy, the quadcopter could maintain stable flight and control, albeit with reduced maneuverability, preventing otherwise catastrophic failure.

Ali et al. [18] proposed an innovative FTC approach that simultaneously addressed both actuator and sensor faults. Their method employed nonlinear backstepping control coupled with friction compensation, enhancing the manipulator's ability to maintain precise movements and positioning even when faced with sensor inaccuracies or actuator inefficiencies. Researchers tackled the complex challenge of controlling uncrewed underwater vehicles in the presence of multiple system uncertainties and disturbances. They developed a novel sensor-active FTC scheme that achieved trajectory tracking without relying on linear and angular velocity measurements [19].

Traditional fault-tolerant control systems have several limitations, including their reliance on accurate system models, their difficulty in handling complex nonlinear systems, and their limited adaptability to unforeseen faults or changing conditions. These systems often struggle with uncertainty and may fail when encountering scenarios not explicitly accounted for in their design [20]. In contrast, learning-based methods offer a more robust and flexible approach to fault-tolerant control. By leveraging data-driven techniques, learning-based FTC can adapt to new situations, learn from experience, and handle complex, nonlinear systems more effectively [21].

Machine learning (ML) has significantly impacted various areas, greatly improving efficiency and capabilities. In healthcare, ML has enhanced diagnostic accuracy and treatment personalization by analyzing vast patient data and medical images [22–24]. In finance, it aids fraud detection and risk assessment, leading to more secure transactions and investment decisions [25,26]. In transportation, machine learning enables autonomous vehicles to navigate complex environments and optimize traffic flow for improved efficiency [27–29]. Additionally, machine learning has revolutionized agriculture, enabling precision farming techniques that optimize resource allocation and crop yield prediction [30–33]. Overall, machine learning's ability to learn from data and make intelligent predictions has greatly enhanced fault-tolerant control strategies for robotic systems, offering improved accuracy

and adaptability in fault detection, isolation, and accommodation. These methods can potentially identify and respond to a wider range of faults, including those not anticipated during the design phase, and they can continuously improve their performance over time [7]. Additionally, machine learning approaches can better handle the high dimensionality and uncertainty inherent in many modern control systems, making them particularly well-suited for applications in areas such as autonomous vehicles, robotics, and advanced manufacturing [7].

Researchers in [34] demonstrated the use of radial basis neural networks for detecting faults in robotic manipulators, achieving high accuracy by training the network on data from normal and faulty operations. In their work, Ref. [35] proposed a fault identification method that utilizes multiple source domains to enhance diagnostic accuracy in real-world scenarios. The method effectively learns and transfers generalized diagnostic knowledge from these diverse sources, improving the model's ability to identify faults in new, unseen scenarios.

Supervised and unsupervised ML methods have primarily been used for fault detection, diagnosis, and isolation in robotics. Reinforcement learning is used for developing adaptive control policies that can respond to faults in real time. However, RL has emerged as a powerful tool for developing adaptive control policies that can respond to faults in real time. RL involves training an agent to make decisions by rewarding desired behaviors and penalizing undesired ones, making it particularly suitable for fault-tolerant control. Researchers compared an RL-based fault-tolerant controller with a model predictive controller (MPC) on a C-130 aircraft fuel tank model. They aimed to test the controllers' adaptability to evolving system changes during operation. Their experiments revealed that the RL-based controller performed more robustly than the MPC under various challenging conditions, including faults, partially observable system models, and fluctuating sensor noise levels [36].

Recent research has demonstrated the effectiveness of RL in various robotic applications. For instance, in [37], researchers proposed an adaptive curriculum RL algorithm with dynamics randomization to train a quadruped robot to adapt to random actuator failures. Similarly, Zhu et al. [38] presented a model-free adaptive fault-tolerant control algorithm based on RL for a multi-joint Baxter robot. Their approach used parameter estimation and neural networks to identify and compensate for actuator faults and spring interference, thereby improving the robot's tracking performance. RL has also shown promise in optimizing control parameters in the presence of faults. In [11], researchers developed an innovative method to optimize proportional–integral control coefficients specifically for motor position control. This RL-based approach demonstrated superior performance, computational efficiency, and user-friendly implementation compared to the traditional Ziegler–Nichols method, making it accessible even to nonexperts. Expanding the application of RL to aerial robotics, researchers in [39] developed a fault-tolerant control system for UAV landing on a moving target. Their approach, which combines robust policy optimization and long short-term memory neural networks, effectively handled sensor failures and noise during the critical landing phase.

While the examples highlight the versatility and effectiveness of machine learning in addressing various aspects of fault-tolerant control in robotics, most learning-based FTC approaches fuse machine learning algorithms with existing model-based control systems. This integration, although beneficial in many aspects, still presents several challenges and limitations. The hybrid approach relies on the accuracy of the underlying model-based control system. If the initial model is flawed or oversimplified, the machine learning component may struggle to compensate fully for these inaccuracies.

This study addresses several key limitations and gaps in previous fault-tolerant control research for robotic manipulators. While prior work has explored model-based approaches and machine learning techniques layered on top of existing control systems, this study proposes a novel end-to-end reinforcement learning framework that directly learns adaptive control policies from raw sensory inputs. Unlike traditional methods that rely on accurate system models or separate fault detection and diagnosis modules, our approach

unifies fault handling and control into a single adaptive system. By framing the problem as a POMDP and leveraging the state-of-the-art RL algorithm, we enable the robot to dynamically compensate for both permanent and intermittent joint failures without requiring explicit fault models. Furthermore, our extensive evaluation in both seen and unseen failure scenarios, including challenging cases of partial joint functionality, demonstrates the robustness and generalizability of the proposed approach. By comparing against traditional inverse kinematics methods, we highlight the superior adaptability of our learning-based solution.

3. Methodology

3.1. Problem Formulation

Markov decision processes (MDPs) are mathematical frameworks used to make optimal decision in situations where outcomes are partly random and partly under the control of a decision maker [40]. An MDP is defined by a set of states, a set of actions, a transition function that determines the probability of moving from one state to another given an action, and a reward function that assigns a numerical reward for each action taken in a given state. The goal in an MDP is to find a policy—a mapping from states to actions—that maximizes the expected sum of rewards over time.

In many real-world scenarios, however, the agent does not have full observability of the environment's state. This leads to the framework of partially observable Markov decision processes [41]. POMDPs extend MDPs by incorporating a set of observations and an observation function. This function provides a probability distribution over possible observations, given the actual state and action taken. The agent maintains a belief state, a probability distribution over all possible states, based on the history of actions and observations. This belief state serves as a sufficient statistic for making decisions in a POMDP.

In this study, we frame the problem of a robotic manipulator completing tasks with joint malfunctions as a POMDP. This formulation allows us to account for uncertainties in joint functionality and train the robot to adapt to varying conditions. Our experimental platform utilizes the Isaac Lab environment, which provides a high-fidelity simulation for robotic manipulation tasks [42]. The platform allows for precise control and the measurement of the robot's movements, as well as the ability to simulate various joint failure scenarios. The robot used in our experiments is Franka Emika's Panda, a 7-DOF robotic manipulator known for its precision and dexterity. The Franka robot modeled in Isaac Lab has been accurately simulated with high fidelity to match the physical characteristics and kinematic properties of the real robot. The task environment was set up to simulate a typical industrial workspace where the Franka robot would be required to open a drawer. The following subsections describe details of the RL framework and reward functions.

3.1.1. Observation Space

The observation s_t at time t comprises the current joint angles and velocities of both the robot and the drawer, as well as the distance between the robot's gripper and the drawer. Formally, the observation can be represented as follows:

$$s_t = [\theta_1, \theta_2, \ldots, \theta_9, \dot{\theta}_1, \dot{\theta}_2, \ldots, \dot{\theta}_9, \delta_d, \dot{\delta}_d, \bar{x}_b, \bar{y}_b, \bar{z}_b], \tag{1}$$

where θ_i and $\dot{\theta}_i$ represent the angle and velocity of the ith joint of the robot, respectively. δ_d and $\dot{\delta}_d$ are the position and velocity of the drawer. $\bar{x}_b$, $\bar{y}_b$, and $\bar{z}_b$ are the distance between the robot gripper and the drawer in the x, y, and z directions, respectively.

3.1.2. Action Space

The action a_t at time t consists of the control inputs to the robot's joints and the end-effector positions. The action space is defined as follows:

$$a_t = [\theta_1, \theta_2, \ldots, \theta_9]. \tag{2}$$

It is important to note that we cannot apply any arbitrary angle or velocity to the joints. Therefore, we need to trim the angles to their respective upper and lower limits for each joint. To ensure that the angles remain within their allowed range, we apply the following constraints:

$$\theta_i^{\min} \leq \theta_i \leq \theta_i^{\max} \quad \text{for} \quad i = 1, 2, \ldots, 9, \tag{3}$$

where $\theta_i^{\min}$ is the minimum allowed angle for the ith joint. $\theta_i^{\max}$ is the maximum allowed angle for the ith joint. By applying these constraints, we ensure that the control inputs are feasible and within the physical limitations of the robot's joints.

3.1.3. Reward Function

The distance reward is designed to encourage the robot to minimize the distance between the robot's gripper and the drawer's handle. It is calculated as follows:

$$d = \|\mathbf{p}_{\text{gripper}} - \mathbf{p}_{\text{drawer}}\|_2, \tag{4}$$

$$r_{\text{dist}} = \frac{1}{1 + d^2}, \tag{5}$$

where $\mathbf{p}_{\text{gripper}}$ is the position of the robot's grasp, $\mathbf{p}_{\text{drawer}}$ is the position of the drawer's handle, and $\|\cdot\|_2$ denotes the Euclidean distance.

The rotation reward is designed to align the robot's gripper orientation with the drawer's handle. It is calculated using the dot products of the forward and up axes:

$$\text{dot}_1 = (\mathbf{a}_1 \cdot \mathbf{a}_2) \tag{6}$$

$$\text{dot}_2 = (\mathbf{a}_3 \cdot \mathbf{a}_4), \tag{7}$$

where $\mathbf{a}_1$ and $\mathbf{a}_2$ are the gripper forward axis of the Franka and the inward axis of the drawer, respectively. $\mathbf{a}_3$ and $\mathbf{a}_4$ are the gripper upper axis and the drawer, respectively. Eventually, the reward for matching the orientation of the hand to the drawer is expressed as follows:

$$r_{\text{rot}} = 0.5\Big(\text{sign}(\text{dot}_1) \times \text{dot}_1^2 + \text{sign}(\text{dot}_2) \times \text{dot}_2^2\Big). \tag{8}$$

The around-handle reward ensures that the robot's fingers are positioned appropriately around the drawer's handle. If the left finger of the Franka is above the drawer handle and the right below the drawer, we bonus a value of 0.5 for the reward function.

$$r_{\text{handle}} = \begin{cases} 0.5 & \text{if } \mathbf{p}_{\text{left\_finger},z} > \mathbf{p}_{\text{drawer},z} \text{ and } \mathbf{p}_{\text{right\_finger},z} < \mathbf{p}_{\text{drawer},z} \\ 0 & \text{otherwise,} \end{cases} \tag{9}$$

where $\mathbf{p}_{\text{left\_finger},z}$ and $\mathbf{p}_{\text{right\_finger},z}$ are the z coordinates of the left and right fingers, respectively. $\mathbf{p}_{\text{drawer},z}$ is the z coordinate of the drawer's grasp position.

The open reward is designed to encourage the robot to open the drawer, which means how far the cabinet has been opened out.

$$r_{\text{open}} = \text{pos}_{\text{drawer\_top}} \times r_{\text{handle}} + \text{pos}_{\text{drawer\_top}}, \tag{10}$$

where $\text{pos}_{\text{drawer\_top}}$ is the position of the drawer that we want the robot to open.

The overall reward function is a combination of the above components scaled by their respective factors:

$$r = w_{\text{dist}} \cdot r_{\text{dist}} + w_{\text{rot}} \cdot r_{\text{rot}} + w_{\text{handle}} \cdot r_{\text{handle}} + w_{\text{open}} \cdot r_{\text{open}}, \tag{11}$$

where $w_{\text{dist}}, w_{\text{rot}}, w_{\text{handle}}, w_{\text{open}}$ are the scaling factors for the distance reward, rotation reward, around-handle reward, and open reward, respectively.

This reward structure ensures that the robot is incentivized to minimize the distance to the drawer handle, align its gripper orientation correctly, position its fingers around the handle, and ultimately open the drawer.

3.2. Reinforcement Learning Framework

3.2.1. PPO Algorithm

In this study, we leveraged the Proximal Policy Optimization (PPO) algorithm to train the Franka robot for robust tasks, enabling robust task completion even in the presence of joint malfunctions. PPO is an on-policy RL algorithm that balances ease of implementation with strong empirical performance. It achieves stable learning by optimizing a clipped surrogate objective, which prevents large, potentially destabilizing updates to the policy.

For PPO, we employed two neural networks—actor and critic networks—as illustrated in Figure 1. The policy network outputs the robot's action (a_t), which represents the angles for each joint (θ_i) of the robot given its current state (s_t). Meanwhile, the value network estimates the expected return $V_\phi(s_t)$ from a given state (s_t). Both networks share the same architecture comprising three fully connected layers. The hidden dimensions of each layer are 256, 128, and 64, respectively, followed by ReLU activations. The final layer's output is either the action or the expected return, depending on whether it is the actor or critic network, and it uses a Tanh activation function [43].

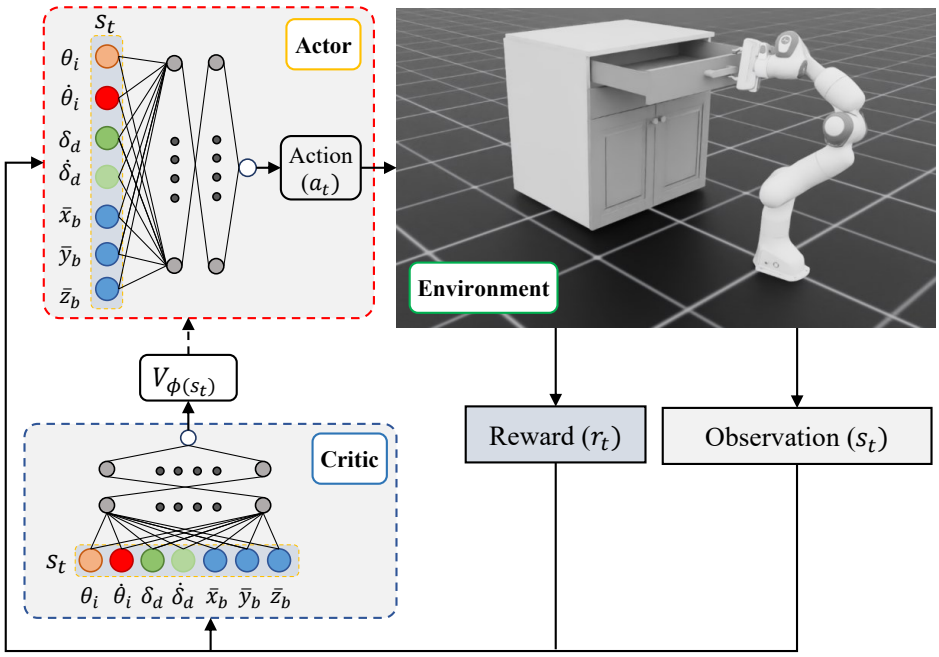


Figure 1. Our framework addresses the issue of joint malfunction in robot manipulators using the PPO algorithm. The actor takes in the observation from the environment and outputs action for joints. The critic estimates the value of a state, helping the actor learn by providing feedback on the quality of its actions.

3.2.2. Agent Training

The agent's training process involves several key steps, with each being critical for optimizing the policy and value networks. The workflow for training the agent using the PPO algorithm is as follows:

Initialization: Initialize the policy network π_ϑ with parameters ϑ. Initialize the value network V_ϕ with parameters ϕ. Set the initial parameters for the learning rate, discount factor γ, and clipping parameter ϵ.

Collecting Trajectories: Interact with the environment to collect trajectories of states s_t, actions a_t, and rewards r_t. The interaction involves executing actions sampled from the policy network and observing the resulting states and rewards.

Computing Returns and Advantages: Calculate the discounted returns R_t from each time step t:

$$R_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k}, \tag{12}$$

where R_t is the return at time step t, γ is the discount factor, and r_{t+k} is the reward at time step $t+k$.

Compute the advantage estimates A_t using the Generalized Advantage Estimation (GAE):

$$A_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \ldots + (\gamma\lambda)^{T-t+1}\delta_T, \tag{13}$$

where

$$\delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t). \tag{14}$$

Here, A_t is the advantage estimate at time step t, λ is the GAE smoothing parameter, δ_t is the temporal difference error, $V_\phi(s_t)$ is the value estimate at state s_t, and r_t is the reward at time step t.

To calculate the objective loss function of the policy, first of all, we need to calculate the ratio of action probabilities $r_t(\vartheta)$ between the new and old policies:

$$r_t(\vartheta) = \frac{\pi_\vartheta(a_t|s_t)}{\pi_{\vartheta_{\text{old}}}(a_t|s_t)} \tag{15}$$

where $\pi_\vartheta(a_t|s_t)$ is the probability of action a_t given state s_t under the new policy, and $\pi_{\vartheta_{\text{old}}}(a_t|s_t)$ is the probability under the old policy. The clipped surrogate objective $L^{CLIP}(\vartheta)$ is as follows:

$$L^{CLIP}(\vartheta) = \mathbb{E}_t[\min(r_t(\vartheta)A_t, \text{clip}(r_t(\vartheta), 1-\epsilon, 1+\epsilon)A_t)] \tag{16}$$

where $\mathbb{E}_t$ denotes the expectation over time steps t, and $\text{clip}(r_t(\vartheta), 1-\epsilon, 1+\epsilon)$ restricts $r_t(\vartheta)$ to the range $[1-\epsilon, 1+\epsilon]$.

The total loss for PPO includes both the policy loss (clipped objective) and the squared-error value function loss, as well as an entropy bonus to encourage exploration. The squared-error value function loss is defined as

$$L^{value}(\phi) = \mathbb{E}_t\Big[(V_\phi(s_t) - R_t)^2\Big], \tag{17}$$

$$\mathcal{L}(\vartheta) = \mathcal{L}^{\text{CLIP}}(\vartheta) - c_1 L^{value}(\phi) + c_2 \mathbb{E}_t[\text{Entropy}(\pi_\vartheta(s_t))]. \tag{18}$$

Here, c_1 and c_2 are coefficients that balance the value loss and entropy bonus.

After calculating the loss function, we update the policy network parameters by computing the gradients of the total loss $\mathcal{L}(\vartheta)$. Similar to the policy network, we also update the parameters in the value network based on the value loss $L^{value}(\phi)$.

$$\vartheta \leftarrow \vartheta - \alpha_\vartheta \nabla_\vartheta \mathcal{L}(\vartheta) \tag{19}$$

$$\phi \leftarrow \phi - \alpha_\phi \nabla_\phi \mathcal{L}^{value}(\phi). \tag{20}$$

Repeat the update process for a fixed number of epochs or until convergence.

3.3. Simulation Setup and Evaluation

We utilized NVIDIA's Isaac Lab as the simulation environment for our robotic manipulation tasks. Isaac Lab provides a high-fidelity physics simulation that allows us to accurately model the Franka Emika Panda robot and its interactions with the environment. The entire simulation and training process is implemented within Isaac Lab. We leverage PyTorch to implement the PPO algorithm for training the RL agent. The experiments are

conducted on a high-performance computing system equipped with an NVIDIA RTX 2070 Super GPU to accelerate the training process.

To evaluate the robustness of our RL framework, we simulated two realistic types of joint malfunctions:

- **Permanently broken joint**: One of the robot's joints is completely nonfunctional throughout the task execution.
- **Intermittently functioning joint**: One of the robot's joints operates intermittently, randomly switching between functional and nonfunctional states.

The trained policy was evaluated under both seen and unseen joint failure scenarios to assess its robustness and adaptability. In addition to the intermittent function scenario, we considered two additional test cases for a faulty functioning joint. In the first test case, the joint is set to be nonfunctional during the first half of the testing period. In the second test case, the joint is set to be nonfunctional during the second half of the testing period.

The primary metrics for evaluation include the following:

- The success rate of completing the task (opening the drawer).
- The time taken to complete the task.

These metrics provide a comprehensive evaluation of the robot's performance under different joint malfunction scenarios. In addition, the performance was compared with a traditional inverse kinematics-based control method to demonstrate the effectiveness of our RL approach.

4. Inverse Kinematics

In this section, we describe the kinematic modeling and analysis of the Franka Emika Panda robot used in our experiments. The Denavit–Hartenberg (DH) convention was employed to derive the kinematic equations and inverse kinematics solution for the robotic manipulator. We demonstrate the impact of a joint malfunction by fixing one of the robot's joints and evaluating its ability to perform the task of reaching the desired trajectory, as well as to open a drawer.

4.1. Denavit–Hartenberg Parameters and Inverse Kinematics Solving

The Franka Emika Panda robot is a 7-DOF manipulator. Each joint is associated with a DH parameter set (θ, d, a, α), which defines the transformations between consecutive links. The DH parameters for Franka Emika's Panda are summarized in Table 1.

Table 1. DH Parameters for the Franka Emika Panda.

Joint i	d_i (m)	a_i (m)	α_i (rad)	θ_i (rad)
1	0.333	0	0	1.157
2	0	0	$-\pi/2$	-1.066
3	0.316	0	$\pi/2$	-0.155
4	0	0.0825	$\pi/2$	-2.239
5	0.384	-0.0825	$-\pi/2$	-1.841
6	0	0	$\pi/2$	1.003
7	0	0.088	0	0.469

The forward kinematics of the manipulator is obtained by multiplying the homogeneous transformation matrices of each link, derived from the DH parameters. The transformation matrix A_i for the ith joint is given by the following:

$$A_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \cos\alpha_i & \sin\theta_i \sin\alpha_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\theta_i \cos\alpha_i & -\cos\theta_i \sin\alpha_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{21}$$

The overall transformation from the base frame to the end-effector frame is as follows:

$$T_0^7 = A_1 A_2 A_3 A_4 A_5 A_6 A_7. \tag{22}$$

The inverse kinematics involve finding the joint angles θ_i given the desired position and orientation of the end effector. This is typically a more complex problem than forward kinematics due to nonlinearity and requires iterative numerical methods or analytical solutions. For the Franka robot, we used an iterative approach based on the Jacobian pseudoinverse method [44,45]. The goal is to minimize the error between the current end-effector position/orientation and the desired position/orientation. The update rule for the joint angles is given by the following:

$$\Delta\theta = J^{+}(x_d - x), \tag{23}$$

where $\Delta\theta$ is the change in joint angle, J^{+} is the pseudoinverse of the Jacobian matrix J, x_d is the desired end-effector position/orientation, and x is the current end-effector position/orientation.

4.2. Inverse Kinematics Simulation and Joint Failure Scenario

In comparison with the DRL algorithm, we modeled the problem of the end effector of a robotic manipulator moving from its initial position to a drawer, grasping it, and pulling it open. In the scenario where one of the joints is malfunctioning, the robot deviates from the calculated trajectory. The desired end-effector motion is formulated as the path connecting three key points—the initial position (p_i), the drawer position (p_d), and the pulled-out position (p_p)—ignoring the internal motion of grippers. This setup is compatible with the motion of the robot in the simulation environment.

The robot's parameters are shown in Table 1. The robot's base is located at the coordinates of $X = 0$ and $Y = 0$. The end effector moves through the initial position, the drawer position, and the pulled-out position, sequentially. The positions and Euler angles representing the configuration of the end effector at each point are listed in Table 2. The end-effector trajectory was solved using the iterative Newton–Raphson numerical method with a step size of 0.01 and a convergence tolerance of 1×10^{-4}.

Table 2. Position of the three points constructing the end-effector trajectory: the initial position (p_i), the drawer position (p_d), and the pulled-out position (p_p). Euler angles (E_x, E_y, E_z) and translation vectors (x, y, z).

	E_x (rad)	E_y (rad)	E_z (rad)	x	y	z
p_i	0.5π	0	π	0.500	0	0.625
p_d	0	0	0.5π	0.750	0	0.317
p_p	0.5π	0	π	0.371	0	0.317

To demonstrate the impact of a joint malfunction, we simulated the scenario where one of the joints was fixed (the third joint), i.e., it was not actuated during the task. This is equivalent to constraining the corresponding joint angle to a constant value. We then analyzed the reachability and performance of the robot in completing the task of opening a drawer. The kinematic analysis was performed by solving the inverse kinematics and evaluating whether the end effector could reach the desired position and complete the expected trajectory.

5. Results and Discussion

5.1. RL Training Results

We trained the robot for 24,000 episodes, as described in Section 3.3 utilizing an NVIDIA GeForce 2070 Super GPU. The training, which took approximately 40 min, aimed to optimize the robot's performance on drawer-opening tasks under various joint malfunction

scenarios. The reward plot of the robot training is shown in Figure 2, in which we show the four reward functions that we describe in Section 3.1.3.

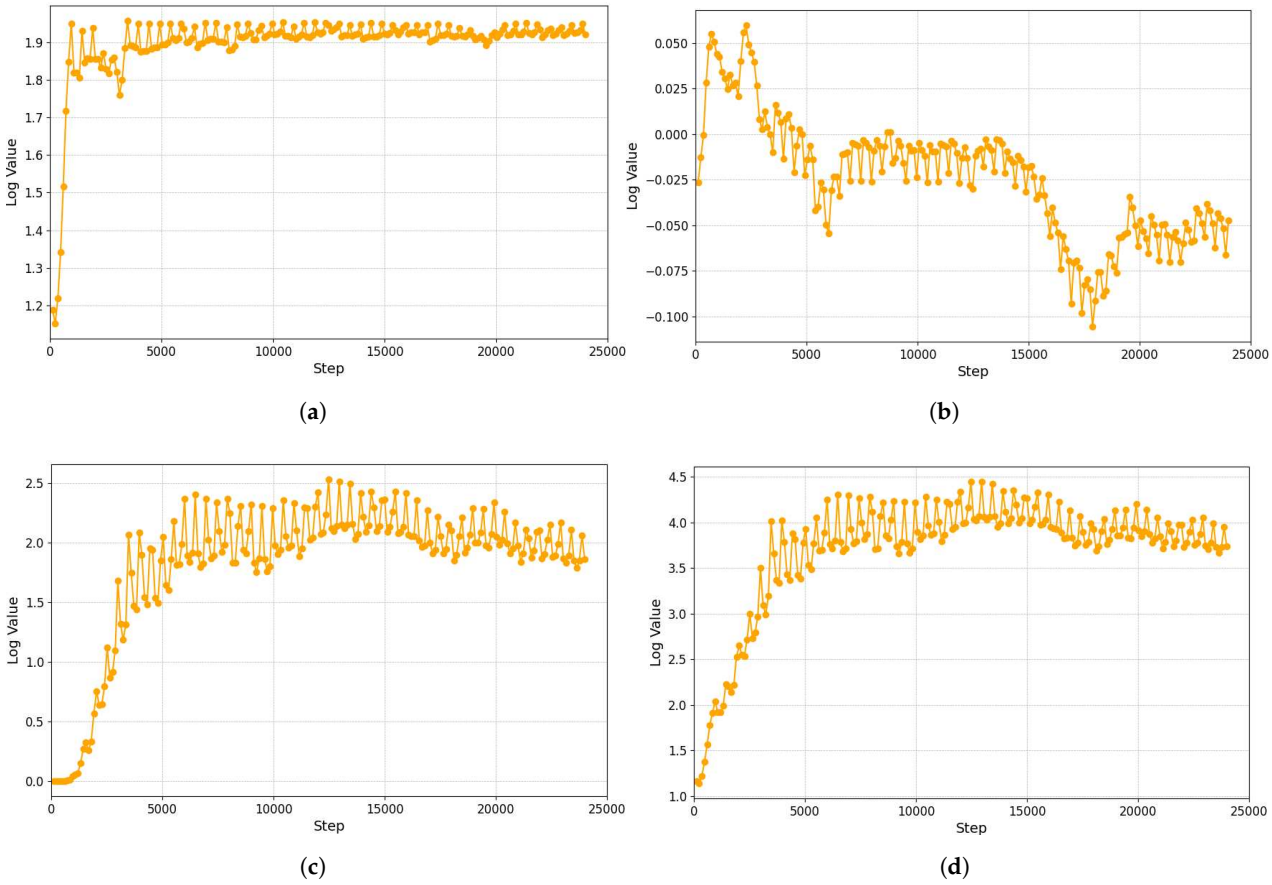


Figure 2. Reward from the training. (**a**) Distance reward. (**b**) Rotation reward. (**c**) Opening reward. (**d**) Total reward.

As illustrated in Figure 2a, the distance reward showed a sharp increase initially, stabilizing around the 1.8 to 1.9 log value range after approximately 2000 steps. This indicates that the robot quickly learned to minimize the distance required to open the drawer, achieving and maintaining high performance early in the training process. However, the rotation reward showed a more variable trend, with an initial increase peaking around 0.05 log value, followed by a decrease and some fluctuations, as shown in Figure 2b. The value eventually stabilized around -0.075 to -0.025 log value after 20,000 steps. This variability suggests that the robot had more difficulty optimizing the rotational aspect of the task compared to the distance, which is likely due to the complexity of the rotation movements required for the task.

Figure 2c shows that the opening reward showed a consistent increase, stabilizing around a log value of 2.0 after 7500 steps. This indicates a steady improvement in the robot's ability to perform the opening action of the drawer, reflecting the effectiveness of the training in teaching the robot this aspect of the task. The total reward shown in Figure 2d, which likely aggregated the individual rewards, demonstrated a steady increase, stabilizing around a log value of 3.5 to 4.0 after 20,000 steps. This overall upward trend indicates successful training, with the robot improving its performance across all aspects of the task over time.

In addition to showing task completion rewards, we conducted experiments under different fault conditions to evaluate the performance of our RL framework under various joint malfunction scenarios. Following previous research [46], we calculated the successful task completion rate under different fault scenarios, including permanently broken joints,

intermittently functioning joints, the joint working in the first half of testing and not working in the second half of testing, and vice versa. For the task completion rate, this metric was used to measure the effectiveness of the robot's ability to complete the drawer-opening task, even in the presence of joint malfunctions. A higher completion rate indicates a more reliable and adaptable system, demonstrating the robot's capacity to overcome joint failures. Additionally, we analyzed the time taken to complete the task to assess the robot's efficiency. A lower time suggests more efficient performance, while an increase in time indicates that the robot had to adjust its strategy to compensate for joint malfunctions. The success rate and average completion time for each scenario are presented in Table 3.

Table 3. Task completion performance under different fault scenarios.

Fault Scenario	Success Rate (%)	Average Completion Time (s)
No Fault	98.00	3.54
Permanently Broken Joint	96.00	4.62
Intermittently Functioning Joint	96.00	3.77
Joint Works First Half	96.00	4.11
Joint Works Second Half	82.00	8.02

The results presented in Table 3 provide valuable insights into the robustness and efficiency of our RL framework when subjected to various joint malfunction scenarios. In the no-fault scenario, the RL algorithm demonstrated a high success rate of 98.00% with an average completion time of 3.54 s. This served as the baseline for our experiments, indicating the algorithm's effectiveness in completing the task efficiently when there were no joint malfunctions.

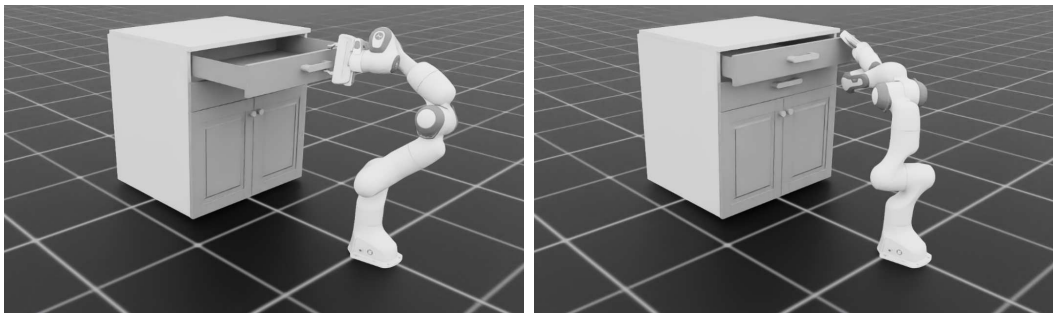
In addition, when a joint was permanently broken, the success rate remained high at 96.00%, although the average completion time increased to 4.62 s. This increase suggests that while the algorithm compensated for the loss of functionality by adjusting its strategy, it did so at the cost of increased time.

In the case of an intermittently functioning joint, the success rate was also 96.00%, with a slightly higher average completion time of 3.77 s compared to the no-fault scenario. This indicates that the algorithm could quickly adapt to intermittent faults and maintain efficiency without significant delays.

When the joint functioned only during the first half of the task, the success rate remained at 96.00%, with an average completion time of 4.11 s. This result implies that the RL algorithm effectively utilized the functional period of the joint to complete the task, demonstrating its ability to optimize performance under partial functionality.

However, the scenario where the joint worked only in the second half presented the most significant challenge. The success rate dropped to 82.00%, and the average completion time increased substantially to 8.02 s. The reduced success rate and longer completion time indicate that the algorithm struggled more when the joint only functioned in the latter part of the task. This difficulty likely arose from the initial lack of joint functionality in the first half, requiring the algorithm to employ more complex strategies to compensate and leading to longer task completion times and lower success rates. Figure 3 shows two cases: one where the robot successfully opened the cabinet and the other where the robot failed to open the cabinet.

Generally, our framework demonstrated strong adaptability and robustness across different fault scenarios, maintaining high success rates in most cases. The completion time varied depending on the type and timing of the joint malfunction, with intermittent faults and partial functionality scenarios resulting in moderate increases in time. The scenario where the joint worked only in the second half posed the greatest challenge, highlighting an area for potential improvement in the algorithm's adaptability to late-functioning components. However, these results underscore the effectiveness of the RL algorithm in handling joint malfunctions, ensuring reliable task completion even under adverse conditions.

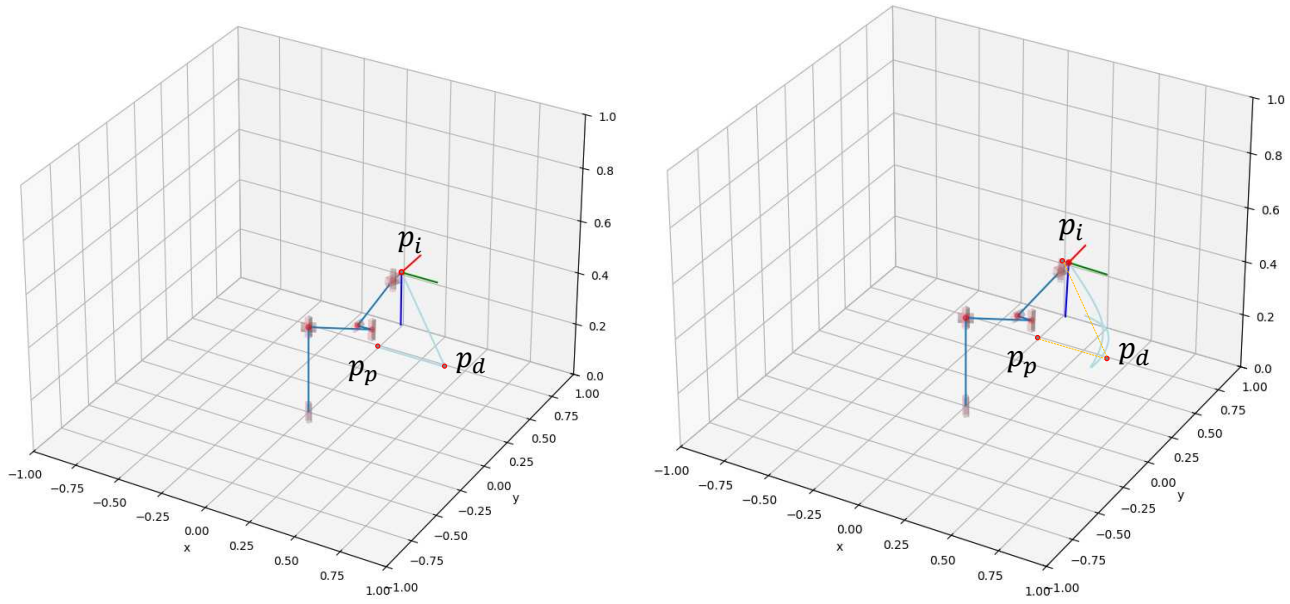


(**a**) (**b**)

Figure 3. Example of successful task completion and failed task completion. (**a**) The robot successfully opened the cabinet. (**b**) The robot failed to open the cabinet.

5.2. Kinematics Results

This section also demonstrates that when a joint malfunctions, the robot's end effector is unable to follow the programmed trajectory with a traditional inverse kinematic controller. The details of the experimental setup are discussed in Section 4.2, and intuitive results are presented in Figure 4. Figure 4a shows that the robot followed the trajectory from the initial position to the drawer position and opened it, which went through the three points p_i, p_d, and p_p. However, as shown in Figure 4b, when one joint was faulty (its angle was fixed)—typically joint 3—the robot was unable to reach the desired end-effector trajectory, hence failing to open the drawer.



(**a**) (**b**)

Figure 4. Comparison of the trajectories of the end effector when one of the joints of the robot is broken and when all the joints work properly. The robot body is illustrated in blue color, and the orientation of the end-effector is indicated at p_i with the red color being the x direction, the green color being the y direction, and the dark blue color being the z direction. When the robot operates properly, it must follow the desired trajectory light blue color in (**a**) or orange lines in (**b**) constructed by three joints: the initial position (p_i), the drawer position (p_d), and the pulled-out position (p_p). (**a**) The end effector followed the desired trajectory when all joints of the robot worked properly. (**b**) The end effector failed to follow the expected trajectory when one of the joints was broken as illustrated in light blue color.

The results demonstrate that the traditional inverse kinematics controller required all joints to be operational properly in order to complete a task. The kinematic analysis using DH parameters and inverse kinematics highlights the critical role of each joint in the Franka robot's ability to perform complex tasks. The demonstrated impact of a joint malfunction underscores the importance of developing robust control algorithms that can compensate for such failures [21]. By developing a DRL algorithm based on our proposed framework as described in the previous sections, we demonstrate that joint failures can be effectively handled for robotic manipulators.

6. Conclusions and Future Work

In this study, we addressed the challenge of enabling a robotic manipulator to complete tasks despite joint malfunctions by developing an RL framework. Our experimental platform, a Franka robot with seven degrees of freedom, was used to test the framework under various joint failure conditions, including permanently broken and intermittently functioning joints. By framing the problem as a partially observable Markov decision process, we successfully trained the robot to adapt to these varying conditions.

In the downstream task, we tested the RL algorithm in both seen and unseen cases, including the joint working in the first half of the time and vice versa. To evaluate the robot's performance, we calculated the task completion rate and the time taken. Our results demonstrate that the RL-trained robot was able to complete the drawer opening task even with joint failures, achieving a high success rate, with an average rate of 93.6% and an average operation time of 3.8 s. The results highlight the adaptability and resilience of our approach. Furthermore, the RL algorithm showed strong performance in both the seen and unseen failure scenarios, indicating its potential for real-world applications where unexpected hardware failures are common. Compared to the traditional inverse kinematic controller, if one of the joints is nonfunctional, the robot's end effector is unable to follow the desired trajectory, leading to task failure.

This study underscores the potential of RL to enhance the reliability of robotic systems, making them better suited for unpredictable environments. By integrating RL with advanced simulation environments like Isaac Lab, we have shown that it is possible to create robust and adaptable robotic solutions that can maintain functionality despite hardware malfunctions. While our approach shows promise for practical applications, there are still limitations to deploying RL-based systems in real-world manufacturing environments. The computational complexity and training time required for RL can present challenges, especially for completing complex tasks.

In the future, we aim to transition our framework from simulation to real-world applications. Furthermore, future efforts could extend this approach to other types of robotic tasks such as pick-and-place objects or part assembly. The integration of more sophisticated RL algorithms and neural networks such as Transformer-based models or CNN-based models could be considered to further improve performance and adaptability.

Author Contributions: T.-H.P.: Conceptualization, Investigation, Methodology, Formal analysis, Validation, Visualization, Writing—original draft, Writing—review and editing. G.A.: Conceptualization, Formal analysis, Writing—original draft. T.T.: Investigation, Formal analysis, Writing—review and editing. K.-D.N.: Conceptualization, Methodology, Formal analysis, Writing—review and editing, Funding acquisition. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the U.S. National Science Foundation grant number #2138206.

Data Availability Statement: The data that support the findings of this study are available online. https://hanhpt23.github.io/franka-IK/ (accessed on 28 September 2024).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. International Federation of Robotics. *World Robotics*; International Federation of Robotics: Frankfurt, Germany, 2021.
2. Vasic, M.; Billard, A. Safety issues in human-robot interactions. In Proceedings of the 2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, 6–10 May 2013; pp. 197–204. [CrossRef]
3. Zhang, Y.; Jiang, J. Bibliographical review on reconfigurable fault-tolerant control systems. *Annu. Rev. Control* **2008**, *32*, 229–252. [CrossRef]
4. Venkatasubramanian, V.; Rengaswamy, R.; Yin, K.; Kavuri, S.N. A review of process fault detection and diagnosis: Part I: Quantitative model-based methods. *Comput. Chem. Eng.* **2003**, *27*, 293–311. [CrossRef]
5. Blanke, M.; Kinnaert, M.; Lunze, J.; Staroswiecki, M. *Diagnosis and Fault-Tolerant Control*; Springer: Berlin/Heidelberg, Germany, 2006. [CrossRef]
6. Ding, S.X. *Model-Based Fault Diagnosis Techniques: Design Schemes, Algorithms, and Tools*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2008.
7. Amin, A.A.; Sajid Iqbal, M.; Hamza Shahbaz, M. Development of Intelligent Fault-Tolerant Control Systems with Machine Learning, Deep Learning, and Transfer Learning Algorithms: A Review. *Expert Syst. Appl.* **2024**, *238*, 121956. [CrossRef]
8. Piltan, F.; Prosvirin, A.E.; Sohaib, M.; Saldivar, B.; Kim, J.M. An SVM-based neural adaptive variable structure observer for fault diagnosis and fault-tolerant control of a robot manipulator. *Appl. Sci.* **2020**, *10*, 1344. [CrossRef]
9. Fei, F.; Tu, Z.; Xu, D.; Deng, X. Learn-to-recover: Retrofitting uavs with reinforcement learning-assisted flight control under cyber-physical attacks. In Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA), Paris, France, 31 May–31 August 2020; IEEE: New York, NY, USA, 2020; pp. 7358–7364.
10. Wang, Y.; Wang, Z. Model free adaptive fault-tolerant tracking control for a class of discrete-time systems. *Neurocomputing* **2020**, *412*, 143–151. [CrossRef]
11. Sardashti, A.; Nazari, J. A learning-based approach to fault detection and fault-tolerant control of permanent magnet DC motors. *J. Eng. Appl. Sci.* **2023**, *70*, 109. [CrossRef]
12. Chen, J.; Patton, R.J. *Robust Model-Based Fault Diagnosis for Dynamic Systems*; The International Series on Asian Studies in Computer and Information Science; Springer: New York, NY, USA, 1999. [CrossRef]
13. Yao, X.; Tao, G.; Ma, Y.; Qi, R. An adaptive actuator failure compensation scheme for spacecraft with unknown inertia parameters. In Proceedings of the 2012 IEEE 51st IEEE Conference on Decision and Control (CDC), Maui, HI, USA, 10–13 December 2012; pp. 1810–1815. [CrossRef]
14. Zhuo-Hua, D.; Zi-Xing, C.; Jin-Xia, Y. Fault diagnosis and fault tolerant control for wheeled mobile robots under unknown environments: A survey. In Proceedings of the 2005 IEEE International Conference on Robotics and Automation, Barcelona, Spain, 18–22 April 2005; IEEE: New York, NY, USA, 2005; pp. 3428–3433.
15. Ahmed, S.; Azar, A.T.; Tounsi, M. Adaptive Fault Tolerant Non-Singular Sliding Mode Control for Robotic Manipulators Based on Fixed-Time Control Law. *Actuators* **2022**, *11*, 353. [CrossRef]
16. Zhou, K.; Ren, Z. A new controller architecture for high performance, robust, and fault-tolerant control. *IEEE Trans. Autom. Control* **2001**, *46*, 1613–1618. [CrossRef]
17. Sun, S.; Wang, X.; Chu, Q.; Visser, C.D. Incremental Nonlinear Fault-Tolerant Control of a Quadrotor with Complete Loss of Two Opposing Rotors. *IEEE Trans. Robot.* **2021**, *37*, 116–130. [CrossRef]
18. Ali, K.; Mehmood, A.; Iqbal, J. Fault-tolerant scheme for robotic manipulator—Nonlinear robust back-stepping control with friction compensation. *PLoS ONE* **2021**, *16*, e0256491. [CrossRef]
19. Wang, X. Active Fault Tolerant Control for Unmanned Underwater Vehicle with Sensor Faults. *IEEE Trans. Instrum. Meas.* **2020**, *69*, 9485–9495. [CrossRef]
20. Blanke, M.; Christian Frei, W.; Kraus, F.; Ron Patton, J.; Staroswiecki, M. What is Fault-Tolerant Control? *IFAC Proc. Vol.* **2000**, *33*, 41–52. [CrossRef]
21. Abbaspour, A.; Mokhtari, S.; Sargolzaei, A.; Yen, K.K. A Survey on Active Fault-Tolerant Control Systems. *Electronics* **2020**, *9*, 1513. [CrossRef]
22. Esteva, A.; Kuprel, B.; Novoa, R.A.; Ko, J.; Swetter, S.M.; Blau, H.M.; Thrun, S. Dermatologist-level classification of skin cancer with deep neural networks. *Nature* **2017**, *542*, 115–118. [CrossRef] [PubMed]
23. Kim, J.W.; Zhao, T.Z.; Schmidgall, S.; Deguet, A.; Kobilarov, M.; Finn, C.; Krieger, A. Surgical Robot Transformer (SRT): Imitation Learning for Surgical Tasks. *arXiv* **2024**, arXiv:2407.12998 .
24. Pham, T.H.; Li, X.; Nguyen, K.D. seUNet-Trans: A Simple Yet Effective UNet-Transformer Model for Medical Image Segmentation. *IEEE Access* **2024**, *12*, 122139–122154. [CrossRef]
25. Whiting, D.G.; Hansen, J.V.; McDonald, J.B.; Albrecht, C.; Albrecht, W.S. Machine learning methods for detecting patterns of management fraud. *Comput. Intell.* **2012**, *28*, 505–527. [CrossRef]
26. Al Ayub Ahmed, A.; Rajesh, S.; Lohana, S.; Ray, S.; Maroor, J.P.; Naved, M. Using Machine Learning and Data Mining to Evaluate Modern Financial Management Techniques. In Proceedings of the Second International Conference in Mechanical and Energy Technology: ICMET 2021, Greater Noida, India, 28–29 October 2021; Springer: Berlin/Heidelberg, Germany, 2022; pp. 249–257.
27. Muhammad, K.; Ullah, A.; Lloret, J.; Del Ser, J.; de Albuquerque, V.H.C. Deep learning for safe autonomous driving: Current challenges and future directions. *IEEE Trans. Intell. Transp. Syst.* **2020**, *22*, 4316–4336. [CrossRef]

28. Aloufi, N.; Alnori, A.; Basuhail, A. Enhancing Autonomous Vehicle Perception in Adverse Weather: A Multi Objectives Model for Integrated Weather Classification and Object Detection. *Electronics* **2024**, *13*, 3063. [CrossRef]
29. Aikins, G.; Jagtap, S.; Gao, W. Resilience analysis of deep q-learning algorithms in driving simulations against cyberattacks. In Proceedings of the 2022 1st International Conference on AI in Cybersecurity (ICAIC), Victoria, TX, USA, 24–26 May 2022; IEEE: New York, NY, USA, 2022; pp. 1–6.
30. Pham, T.H.; Nguyen, K.D. Enhanced Droplet Analysis Using Generative Adversarial Networks. *arXiv* **2024**, arXiv:2402.15909.
31. Sharma, A.; Jain, A.; Gupta, P.; Chowdary, V. Machine learning applications for precision agriculture: A comprehensive review. *IEEE Access* **2020**, *9*, 4843–4873. [CrossRef]
32. Pham, T.H.; Nguyen, K.D. Soil Sampling Map Optimization with a Dual Deep Learning Framework. *Mach. Learn. Knowl. Extr.* **2024**, *6*, 751–769. [CrossRef]
33. Pham, T.H.; Acharya, P.; Bachina, S.; Osterloh, K.; Nguyen, K.D. Deep-learning framework for optimal selection of soil sampling sites. *Comput. Electron. Agric.* **2024**, *217*, 108650. [CrossRef]
34. Eski, I.; Erkaya, S.; Savas, S.; Yildirim, S. Fault detection on robot manipulators using artificial neural networks. *Robot. Comput. Integr. Manuf.* **2011**, *27*, 115–123. [CrossRef]
35. Zheng, H.; Wang, R.; Yang, Y.; Li, Y.; Xu, M. Intelligent Fault Identification Based on Multisource Domain Generalization Towards Actual Diagnosis Scenario. *IEEE Trans. Ind. Electron.* **2020**, *67*, 1293–1304. [CrossRef]
36. Ahmed, I.; Khorasgani, H.; Biswas, G. Comparison of model predictive and reinforcement learning methods for fault tolerant control. *IFAC-PapersOnLine* **2018**, *51*, 233–240. [CrossRef]
37. Okamoto, W.; Kera, H.; Kawamoto, K. Reinforcement Learning with Adaptive Curriculum Dynamics Randomization for Fault-Tolerant Robot Control. *arXiv* **2021**, arXiv:2111.10005 .
38. Zhu, J.W.; Dong, Z.Y.; Yang, Z.J.; Wang, X. A New Reinforcement Learning Fault-Tolerant Tracking Control Method with Application to Baxter Robot. *IEEE/ASME Trans. Mechatron.* **2024**, *29*, 1331–1341. [CrossRef]
39. Aikins, G.; Jagtap, S.; Nguyen, K.D. A Robust Strategy for UAV Autonomous Landing on a Moving Platform under Partial Observability. *Drones* **2024**, *8*, 232. [CrossRef]
40. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
41. Albrecht, S.V.; Christianos, F.; Schäfer, L. *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*; MIT Press: Cambridge, MA, USA, 2024.
42. Mittal, M.; Yu, C.; Yu, Q.; Liu, J.; Rudin, N.; Hoeller, D.; Yuan, J.L.; Singh, R.; Guo, Y.; Mazhar, H.; et al. Orbit: A Unified Simulation Framework for Interactive Robot Learning Environments. *IEEE Robot. Autom. Lett.* **2023**, *8*, 3740–3747. [CrossRef]
43. Dubey, S.R.; Singh, S.K.; Chaudhuri, B.B. Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing* **2022**, *503*, 92–108. [CrossRef]
44. Dulęba, I.; Opałka, M. A comparison of Jacobian-based methods of inverse kinematics for serial robot manipulators. *Int. J. Appl. Math. Comput. Sci.* **2013**, *23*, 373–382. [CrossRef]
45. Whitney, D.E. Resolved motion rate control of manipulators and human prostheses. *IEEE Trans. Man Mach. Syst.* **1969**, *10*, 47–53. [CrossRef]
46. Zhao, T.Z.; Kumar, V.; Levine, S.; Finn, C. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv* **2023**, arXiv:2304.13705.