SELF-REFLECTIVE EXPERIENTIAL LEARNING

FOR PERSISTENT AUTONOMY

by

Xu Zhou

A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Doctor of Philosophy (Mechanical Engineering).

Golden, Colorado
Date _____

Signed: _____
Xu Zhou

Signed: _____
Dr. Xiaoli Zhang
Thesis Advisor

Golden, Colorado
Date _____

Signed: _____
Dr. John Berger
Professor and Head
Department of Mechanical Engineering

ABSTRACT

Robots have been expected to achieve persistent autonomy for a long time, in which robots are required to safely operate in unknown environments for extended lengths of time while without human interventions. Reinforcement learning holds the promise for persistent autonomy because it can adapt to dynamic and unstructured environments by automatically learning optimal policies from the interactions between robots and environments. However, failures can be unavoidable in the learning process as reinforcement learning can learn the outcome of an action only by executing the action itself. These failures can cause damages to robots or environments in practical applications and hence hinder persistent autonomy. In general, human interventions are usually needed to avoid or resolve the learning failures, but they can be unavailable in practical applications such as space exploration, search and rescue, and underground or underwater construction. Based on a multi-level architecture for persistent autonomy, this dissertation proposes new self-reflective, experiential strategies and methods, aiming at achieving safe adaptions to different environments with minimum human interventions. At the strategic level, while understanding learning is not always necessary and beneficial, this dissertation adds a high-level sophistication of whether and when to learn to reduce failures during learning. At the tactical level, this dissertation proposes a new self-recoverable reinforcement learning algorithm that consists of a multi-state recovery strategy and a failure-prevention strategy. The multi-state recovery strategy improves the learning's own capability of handling already occurred failures and the failure-prevention strategy learns from failures that are usually ignored and abandoned before to generate a more effective strategy of preventing future similar failures. At the operational level, this dissertation proposes a new multi-objective-optimization-based auto-tuning method to adjust control parameters for robustly achieving the upper-level learning behaviors.

TABLE OF CONTENTS

ABSTRACT iii

LIST OF FIGURES viii

LIST OF TABLES xii

ACKNOWLEDGEMENTS xiv

CHAPTER 1 INTRODUCTION 1

1.1 Persistent Autonomy 1

1.2 Inherent Difficulties in Persistent Autonomy 2

1.3 Limitations of State-of-the-Art Techniques in Persistent Autonomy 3

1.4 Dissertation Goals 7

1.5 References 9

CHAPTER 2 SELF-REFLECTIVE LEARNING STRATEGY FOR PERSISTENT AUTONOMY OF AERIAL MANIPULATORS 13

2.1 Abstract 13

2.2 Introduction 14

2.3 Related Work 16

2.4 Self-Reflective Learning Strategy 17

2.4.1 New Situation Determination 18

2.4.2 Self-Reflective Adjustment 19

2.4.3 Self-Evaluation for Termination 22

2.5 Validation Simulations 23

2.6 Results 25

2.6.1 New Situation Determination 25

2.6.2 Self-Reflective Adjustment 26

2.6.3 Self-Evaluation for Termination 27

2.6.4 Task Performance 29
2.6.5 Failure Costs and Computational Costs 31
2.6.6 Experimental Approach Stability Analysis 32
2.7 Discussions 33
2.8 Conclusion 34
2.9 References 35
CHAPTER 3 VALUE OF LEARNING: WHEN TO LEARN AND WHEN NOT IN FAILURE-AWARE LEARNING-BASED AUTONOMY 37
3.1 Abstract 37
3.2 Introduction 38
3.3 Related Work 40
3.4 Preliminaries 42
3.5 Methodology 43
3.5.1 Environment Model 44
3.5.2 Learning Gain Indicator 46
3.5.3 Learning Cost Indicator 47
3.5.4 Value of Learning 48
3.5.5 Failure Check 48
3.6 Results 49
3.6.1 Experiment Setup 49
3.6.2 Environment Model 50
3.6.3 Value of Learning Performances 51
3.6.4 Important Factors for Decision to Learn 53
3.7 Discussions 56
3.8 Conclusion 58
3.9 References 58

CHAPTER 4 A SELF-RECOVERABLE RESET STRATEGY FOR REINFORCEMENT-LEARNING-BASED PERSISTENT AUTONOMY 61

4.1 Abstract 61

4.2 Introduction 62

4.3 Related Work 64

4.4 Preliminaries 66

4.5 Methodology 68

4.5.1 Multi-State Recovery 69

4.5.2 Failure Prevention 71

4.5.3 Algorithm Summary 72

4.6 Evaluation 73

4.6.1 Experiment Setup 73

4.6.2 Experiment Results 75

4.7 Discussions 84

4.8 Conclusion 85

4.9 References 86

CHAPTER 5 MULTI-OBJECTIVE-OPTIMIZATION-BASED CONTROL PARAMETERS AUTO-TUNING FOR AERIAL MANIPULATORS 88

5.1 Abstract 88

5.2 Introduction 89

5.3 Mathematical Modeling 92

5.3.1 Kinematics for the Quadrotor-Arm System 92

5.3.2 Dynamics for the Quadrotor-Arm System 94

5.4 Multi-Objective-Optimization-Based PID Control Scheme 96

5.4.1 PID Control Structure for Quadrotor-Arm System 96

5.4.2 MOO-Based PID Control Parameters Auto-Tuning 97

5.5 Validation Simulations 99
5.6 Results 102
5.7 Conclusion 112
5.8 References 113
CHAPTER 6 CONCLUSION AND FUTURE WORK 117
6.1 Contributions 117
6.2 Future Work 118
6.2.1 An Extension from Deterministic Environment to Stochastic, Noisy Environment 118
6.2.2 An Extension from discrete MDP to continuous MDP 118

LIST OF FIGURES

Figure 1.1 Hierarchical persistent autonomy architecture..... 9

Figure 2.1 Overall self-reflective learning scheme 18

Figure 2.2 Visualization of the object-carry task 24

Figure 2.3 New situation determination for different object weights. For 0.1kg and 0.15kg, there is no need of self-reflection as they are similar to pre-trained situations. For 0.25kg and 0.3kg, self-reflection is needed in all different parameter settings as they are greatly different from pre-trained situations. For 0.2kg, the necessity degree value is less with loose constraints than that with strict constraints. If given the conservative threshold, only the case with strict constraints needs self-reflection. But given an aggressive threshold, both cases with strict and loose constraints, respectively, need self-reflection..... 26

Figure 2.4 Maximum episode for different object weights 27

Figure 2.5 Link1's movement for different object weights. The movements with 0.15kg and 0.2kg are still acceptable using the pre-trained strategy for 0.1kg. But the movement with 0.2kg is close to unstable due to the large overshoot and long setting time. Because self-reflection is applied for 0.25kg, the movement is stable again with a smaller overshoot and shorter setting time. The movement with 0.3kg becomes worse with using the strategy for 0.25kg but without self-reflection. 30

Figure 2.6 Task performances for different object weights. If self-reflection is not triggered, the task performance decreases as the object weight increases from 0.1kg to 0.2kg. Once self-reflection is needed and applied at 0.25kg due to an unacceptable performance drop, the task performance increases back to be acceptable. When addressing 0.3kg, the task performance drops again because self-reflection is determined to be not used. 31

Figure 3.1 When should the robot learn in a dynamically changing environment? The robot needs to stack blocks in a prior unknown order by trial and error, but some trials may cause blocks to fall over and the blocks can be damaged after a certain number of times. Hence, in addition to the gains, the learning also has costs of experiencing failures. Both the learning gains and costs should be considered to decide when the robot should learn. 39

Figure 3.2 Value of Learning method. Given the prediction of the environmental change from an online learned environment model, both learning gains and learning costs are calculated with respect to the current optimal policy. Based on the learning gains and costs, the personalized decision-maker uses three different strategies including an optimistic, realistic, and pessimistic strategy to decide

if reinforcement learning should be activated to find a new policy. If the current policy is kept but a failure exists, reinforcement learning must be activated. 44

Figure 3.3 Estimation of the environment change probability using different methods. All the methods can converge to the ground truth within 40 iterations. The NN method performs best with a fast convergence speed with using only 4 iterations. 51

Figure 3.4 Comparison of learning gain, cost, and decisions to learn between MLE-based VOL (left) and NN-based VOL (right). 52

Figure 3.5 Comparison of learning gains, costs, and decisions to learn with respect to different environment change probabilities. The random exploration probability is fixed to be 0.1. Decision = 1 means a decision to learn while decision = 0 means a decision to not learn. 54

Figure 3.6 Comparison of learning gains, costs, and decisions to learn with respect to different random exploration probabilities. The environment change probability is fixed to be 0.1. Decision = 1 means a decision to learn while decision = 0 means a decision to not learn. 55

Figure 4.1 Can reinforcement-learning-based robot do the reset itself? The robot needs to stack four blocks from four pre-defined positions into a target position in a prior unknown order. If the stacking order does not match the expected one, the blocks fall over. When such a failure occurs, human interventions are required to reset the state to the initial state if traditional reinforcement learning is used for the robot. However, manual resets can be unavailable and hence what should the robot do? 63

Figure 4.2 Self-recoverable reset strategy. It consists of the multi-state recovery strategy and the failure prevention strategy. The multi-state recovery strategy provides the robot with a capability of self-recovering back to any state with reversely executing the safe actions stored in an experience buffer. In addition, the multi-state recovery method decides which state is the best for the robot to recover back to for an efficient learning. The failure prevention strategy learns from already occurred failures and predict future failures so that the robot can avoid them, which also avoids resets. 68

Figure 4.3 Multi-state recovery strategy: recover from an occurred failure to an ideal previous state to continue learning to achieve the training goal in each episode. The robot should recover from the gray state to the red junction state instead of the light blue and dark blue states which are still in the dead end. 71

Figure 4.4 Failure prevention strategy: predict and avoid potential failures based on the recorded failure experiences. This strategy can be as simple as exactly preventing the occurred failures, and can also be as complex as using additional knowledge to prevent similar failures. 72

Figure 4.5 The simulation environment: a 4*4 (left) and 11*11 (right) maze. The robot is supposed to find an optimal path with a largest reward from the initial state (S) to the goal state (G). The black grids represent obstacles such as walls and the robot is not allowed to pass these obstacles. The robot can take four actions: go north, go south, go east, and go west. 75

Figure 4.6 The relationship between number of resets and recovery limits N. When the environment is simple (a 4*4 maze), even only recovering back to the latest state can guarantee no resets. However, when the environment is more complex (a 11*11 maze), a larger recovery limit N (e.g., 3) could be more effective, especially given enough maximum steps per episode. 80

Figure 4.7 The relationship between number of resets and maximum steps per episode. The number of resets reduces as the maximum steps per episode increases. When maximum steps per episode remains large, a larger N can be better in reducing the number of resets. 81

Figure 4.8 The capability of self-recovering back to any states. Human interventions are required in traditional reinforcement learning to reset a failure state to the initial state. In contrast, our self-recoverable reset strategy can recover back to any states with reversely executing the safe actions recorded in the experience buffer. Even if our method cannot decide a better state to recover back to, it can at least recover back to the initial state, which is equivalent to the manual reset. 82

Figure 5.1 Coordinate configuration of the aerial manipulator system..... 93

Figure 5.2 Multi-objective-optimization-based PID control structure for the quadrotor-arm system 98

Figure 5.3 Visualization of simulated movements. The quadrotor is designed to follow a circular trajectory. The arm is designed to move from configuration A to configuration B when the quadrotor starts flying. 102

Figure 5.4 Pareto front for multi-objective optimization in terms of integrated time square error and control rate. The solutions to the left of point A focus more on minimizing the integrated time square error while the solutions to the right of point B focus more on minimizing the control rate. The solutions between point A and B are more balanced between the integrated time square error and control rate. 106

Figure 5.5 Link performance comparison using PID parameter set 1 (manual-tuned), 4 (best manual-tuned), 5 (auto-tuned), and MPC. The auto-tuned performances are comparable to MPC because the differences between the red solid lines and green dash-dot lines are small. Although the best manual-tuned performances (purple dotted lines) are close to auto-tuned and MPC performances for links 2 and 3, the best manual-tuned link 1 performance is notably worse than the

auto-tuned and MPC performance. The manual-tuned performances (blue dashed lines) are worst with significant oscillations..... 109

Figure 5.6 Quadrotor trajectories using PID parameter set 1 (manual-tuned), 4 (best manual-tuned), 5 (auto-tuned), and MPC. The auto-tuned parameter set 5 has comparable performances with MPC, but better performances than manual tuned parameter sets in terms of less deviations and faster stabilization time.... 110

Figure 5.7 Performance comparison of PID parameter set 1 (manual-tuned), 4 (best manual-tuned), 5 (auto-tuned), and MPC. All performances including the overshoot, stabilization time, quadrotor position deviation, and quadrotor angle deviation are considered better with smaller values. The auto-tuned and MPC performances are comparable while they are better than the manual-tuned performances. 111

LIST OF TABLES

Table 2.1 Quadrotor simulation parameters 24

Table 2.2 Self-reflective control strategy parameters 25

Table 2.3 Termination criterion value and episodes for different parameters 29

Table 2.4 Failure costs and computational costs 32

Table 3.1 Statistical results of VOL performances by using different learning strategies ... 53

Table 4.1 Comparisons with previous results. Either of our three strategies outperform the reset policy proposed by Eysenbach et al. in terms of both number of resets and number of total steps. 76

Table 4.2 Full comparisons among the standard Q-learning, Q-learning algorithm with failure prevention (FP) strategy, Q-learning with multi-state recovery (MSR) strategy, and Q-learning with self-recoverable reset (SRR) strategy. Either FP or MSR can individually outperform the standard Q-learning, although the combination of FP and MSR to SRR has the best performance in all the four methods. MSR can almost guarantee a zero reset even in a complex 11*11 maze because it avoids the resets from the perspective to fixing failures rather than just avoiding failures. 77

Table 4.3 Failure comparisons between the standard Q-learning and our method. While MSR only reduces the failures in a small amount, FP can significantly reduce the failures because it can strictly restrict the random explorations in the Q-learning to safe actions. However, the failures in MSR do not result in resets while the failures in the standard Q-learning and FP must require corresponding resets. 78

Table 4.4: Full comparisons among the block stacking results using standard Q-learning, Q-learning algorithm with FP strategy, Q-learning with MSR strategy, and Q-learning with SRR strategy. Either FP or MSR can individually outperform the standard Q-learning, although the combination of FP and MSR to SRR has the best performance in all the four methods. MSR and SRR can guarantee a zero reset. 81

Table 4.5 Total training time with respect to different recovery limits and different maximum steps per episode 83

Table 5.1 Quadrotor simulation parameters 100

Table 5.2 Parameters used for NSGA-II 100

Table 5.3 Manually tuned PID parameter values and corresponding performances 103

Table 5.4 PID parameter values and corresponding performances for different arm settings 104

Table 5.5 Performances of auto-tuned PID parameter sets 5 and 6 107

Table 5.6 Performance comparison of PID parameter set 1, 5 (auto-tuned) and MPC 108

ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere gratitude to my advisor, Professor Xiaoli Zhang, for the continued support of my Ph.D. study. Her enthusiasm and vision had a great influence on how I learned to do my own research. Her patience and trust made me enjoy doing my research with the greatest freedom. Her immense knowledge and insights guided me in all the time of research and writing of this dissertation.

In addition, I would like to thank Professor Tyrone Vincent, Professor Anthony Petrella, and Professor Gongguo Tang, for their professional services in my thesis committee, and, more importantly, for their invaluable comments that greatly strengthened and widened my research from various perspectives.

I would like to thank my fellow lab mates, Rui, Songpo, Miachel, Lingfeng, Sen and Noopur for the stimulating discussions and their helps during my research. I would also like to thank my friends, Jincheng, Dehui, Hongjie, Qiuwei, and Zhihui, for their precious support and all the memorable time we have shared.

Lastly, I am deeply thankful to my parents for their support, love and sacrifices in my whole life. Without them, I might have never had the chance to study abroad and pursue my dream. Special thanks is reserved for my fiancée Shanshan Wang, who showed immense patience and support during my study. Her warm encouragement and love will keep me strong and going all the time.

CHAPTER 1

INTRODUCTION

1.1 Persistent Autonomy

Autonomous robots generally perform well when carrying out well-defined tasks in well-known environments. This reliable performance is achieved because all the tasks and environments can be exactly pre-programmed with appropriate parameters [1][2] without uncertainties. There are various applications such as stabilizing at a location [3], following designed trajectories [4], or moving around objects of interest [5].

However, the true autonomy (persistent autonomy) [6] requires more than the above well-defined contexts of robot configurations, tasks, and environments because practical operations may need to place different robots in new environments to finish a new task. The new robot configurations and task descriptions may not be precisely specified or known, and/or the new environments may not be encountered previously. When state of the art autonomous robots recognize they are in such situations beyond their sensing or reasoning abilities, they easily get stuck and typically seek human operator assistance. This often imposes unwanted operational constraints that are against the idea of autonomy and the initial motivation of using robots.

In order to realize the persistent autonomy, robots are required to have greater flexibilities in the way of sensing, reasoning, and acting to safely operate in unknown environments for extended lengths of time while without human interventions. As shown in Figure 1.1, the architecture of persistent autonomy generally includes four following levels.

The highest level is the strategic level, which adds a level of sophistication to the simple switching of low-level behaviors in the traditional autonomy architecture. From the general system's perspective, this sophistication makes reasonable and global decisions by considering

more complex criteria such as general task semantic vagueness, the stability, risks, and energy consumptions of different low-level behaviors, and the conflicts between low-levels.

The tactical level generally includes traditional robot motion planning and path planning with using machine learning or optimal control techniques. While following the global-view instructions from the strategic level, this level uniquely addresses the system adaptability and safety within the scope of the learning or control algorithms themselves. In other words, this level aims to improve the system adaptability and safety by improving an algorithm itself rather than through choosing alternative better algorithms.

The operational level generally designs for robots robust controllers that can reliably follow a given trajectory from the tactical level. Various traditional control problems, such as the robustness, control stability, and the tuning of control parameters, are expected to be solved in this level.

The lowest level is the execution level, in which the robot hardware (e.g. actuators) physically execute the commands from the upper levels.

1.2 Inherent Difficulties in Persistent Autonomy

The persistent autonomy is a challenging problem due to the difficulties in (1) uncertain and complex robot configurations and (2) dynamic and unstructured environment.

(1) Uncertain and Complex Robot Configurations: It is common for robot models to be inaccurate due to system uncertainties and simplification assumptions. For example, the aerodynamics is ignored in most dynamic modeling of quadrotors, which does not matter because many quadrotor applications are executed with low speed. However, if the quadrotor is supposed to move aggressively with a fast speed [7], the aerodynamics has a significant influence on the quadrotor dynamics. If still not considered, the quadrotor tends to crash. Also, robot models could be complicated due to strong coupling effects and high nonlinearities. For instance, the quadrotor

is an underactuated nonlinear system with strong coupling effects between the position and attitude [8]. Both the model uncertainties and complexities inherently increase the difficulty in robot control so that robots are too easily unstable to maintain persistent autonomy.

(2) Dynamic and Unstructured Environments: Most real-world environments are dynamic and unstructured due to an increased level of uncertainties and disturbances, such as undocumented maintenances, moving obstacles, reaction forces between robots and environments, and so on. In such environments, tasks tend to have more flexibilities and perhaps vagueness in descriptions. Robots may have longer periods of time without direct sensor feedback from environment interactions. In general, robots can only have a partially known, or even totally unknown, prior knowledge about these environments. The missing of useful information about environments makes it difficult or even impossible for robots to make correct and timely decisions. Without such decisions, robots easily fail tasks or get damaged, which obviously hinders the realization of persistent autonomy.

1.3 Limitations of State-of-the-Art Techniques in Persistent Autonomy

As persistent autonomy demands higher requirements on the system's safety, adaptability, robustness, and efficiency at the same time, traditional autonomy architectures [9][10] that usually focus on solving a specific problem are no longer applicable. Instead, a multi-level architecture like the popular four-level computational architecture [6] that can achieve multiple goals is needed. However, this architecture is not compatible with the fast-developed robot learning technique. Given that robot learning has already shown great successes and potentials in adapting to various environments, even unknown environments, it could be necessary to embed robot learning in the current persistent autonomy architecture to achieve a true persistent autonomy. A main challenge of such an integration is that robot learning usually requires numerous trainings but not all of them

are safe. The unsafe trainings can lead to unacceptable consequences such as the crash of robots. Thus, it is urgent to develop safe robot learning algorithms that can be easily used in the current persistent autonomy architecture to comprehensively enhance the performance. Because this dissertation focuses on reinforcement learning based persistent autonomy, the open problems are specifically as follows:

(a) At the strategic level, no high-level, sophisticated mechanism has been reported to address the failures related to reinforcement learning. Here, the "high-level" means addressing the failure problem from a broader perspective such as the whole system's perspective rather than the learning algorithm's own perspective. For example, although some safe reinforcement learning algorithms can improve the learning safety, it could be easier and more efficient to do this by deciding whether or when to learn. In some dynamic environments with small changes, the previously learned policy may work for a long time. Keeping this policy may completely avoid learning failures while safe reinforcement learning methods cannot actually guarantee no failures during their learning phases. Although some meta-learning methods [11][12] have been recently proposed for reinforcement learning to enhance the capability to decide which policy to be used in different situations, they were actually designed to further improve the efficiency of reinforcement learning rather than specifically focusing on the learning safety. In fact, the meta-learning may even increase the failures during the learning because frequently switching policies may have more chances to make the system unstable. Thus, a high-level, safety-centered mechanism is imperative for reinforcement learning to be used for persistent autonomy.

(b) At the tactical level, two fundamental tendencies have been reported to deal with the safety and/or risks during the reinforcement learning and/or deployment process. The first one consists of transforming the traditional optimization criteria that maximizes the expectation of the return to

more comprehensive criteria respecting learning safety such as the worst-case criterion [13][14], the risk-sensitive criterion [15][16], the constrained optimization criterion [17][18], and other optimization criteria in financial engineering [19][20]. Since exploratory actions could have serious consequences, the second tendency improved the safety with modifying the exploration process in two ways: (1) through the incorporation of external knowledge such as providing initial knowledge [21], deriving a policy from a finite set of demonstrations [22][23], and providing teach advice [24][25][26], and (2) through the use of a risk-directed exploration metric [27][28]. However, the first method may be too strict to rule out some good solutions that actually do not cause failures. The criterion may also be specifically designed and hence cannot be flexible to different learning situations. When the situation changes, re-designing a new criterion is necessary and it usually needs human interventions. Similarly, in the second method, extensive human interventions are also required to obtain useful external knowledge for different situations. More importantly, both methods can only prevent failures but has no idea about how to address the already occurred failures.

Because learning failures usually exist in the form of system failures, many failure recovery methods [29][30][31] can be used in robots to handle these already occurred failures. However, these methods are actually external to reinforcement learning algorithms, which can only fix a failure whenever it occurs but cannot analyze why it occurs in reinforcement learning. Thus, it is possible for reinforcement learning to repeat these failures in the future. This may waste considerable resources in robots especially when the repeat is frequent. Furthermore, the recovery plan could conflict with reinforcement learning itself so that it is totally useless. Thus, it is necessary to integrate failure recovery into reinforcement learning's own framework to essentially enhance reinforcement learning's own capability to address failures.

To this end, novel safe reinforcement learning algorithms that can automatically recover from learning failures and effectively reduce learning failures with flexibilities on different situations are needed, but there are rare such algorithms at present.

(c) At the operational level, many current controllers need exhaustive manual tunings of control parameters to achieve desirable performances in different situations because one set of control parameters is usually suitable for a specific situation. Taking the simple but popular proportional-integral-derivative (PID) controller for example, various tuning methods have been developed, including Ziegler-Nichols method [32], Tyreus-Luyben method [33], C-H-R method [34], Cohen and Coon method [35], Fertik method [36], Ciancone-Marline method [37], Internal Model Control (IMC) method [38], minimum error criteria method [39], and so on. All these methods can achieve satisfied tuning results, but a major problem of these methods is that the tuning process must be done by human operators or experts with a considerable length of time. One way to avoid the controller tuning is to design more advanced and complicated controllers (e.g., model predictive control) in which the control parameters rarely have to be changed once determined. Because this method uses its own robustness to handle uncertainties, it may still need human interventions when the uncertainty is beyond its maximum robustness. Since human interventions are not expected in persistent autonomy, auto-tuning methods become necessary.

A systematic way to perform automatic tuning is utilizing an optimization method to find the optimal parameters. During the past years, the single-objective optimization has been widely researched and applied to the tuning problem [40][41][42][43][44][45]. However, this method is limited to simple systems with linear or linearized models. Another problem is that a single objective may not meet practical expectations because robots could be required to simultaneously satisfy two or more possibly conflicting control goals such as a minimal overshoot and a shortest

stabilization time. Considering these two limitations, a more advanced multi-objective-optimization-based auto-tuning method is necessary for the operational level of persistent autonomy. However, such techniques are still under development in current persistent autonomy.

1.4 Dissertation Goals

Motivated by the problems above, this dissertation aims to endow robots with smarter and safer adaptabilities to different robot configurations, tasks, and environments, helping robots move towards to the true persistent autonomy. In general, inspired by human's self-reflection, we propose new strategies and methods for different levels of persistent autonomy as shown in Figure 1.1. Humans usually repeat a self-reflection process in their daily life with keeping asking themselves: Should I change? When should I change? How should I change? Similarly, the self-reflection in persistent autonomy evaluates the necessity of changing lower-level behaviors, the timing of changing lower-level behaviors from the whole system's perspective with considering both theoretical and practical constrains, and how to robustly change to desired behaviors. Based on current behavior performances, it decides whether and when to trigger learning for adjustments at the strategic level. At the tactical level, it aims to efficiently and safely find an optimal behavior using learning. At the operational level, it decides how to robustly adjust low-level operations to achieve the higher-level behaviors. More specifically,

(i) At the strategic level, two new, sophisticated reasoning approach, named "self-reflective learning" and "value of learning" are developed for robots to make reasonable strategies from the whole system's perspective. As the tactical level mainly investigates reinforcement learning in this dissertation, this level can be regarded as a "learn to learn". It believes that learning is not always necessary and beneficial for practical robot systems because learning also has costs such as learning can lead to system failures. While the self-reflective learning method and value of learning

method both aim to reduce the learning failures, the self-reflective learning method is mainly through the utilization of heuristic safety knowledge and the value of learning method is differently through a more complicated analysis of learning gains and learning costs with respect to a self-learned environment model. The details can be seen in Chapter 2 and Chapter 3, respectively.

(ii) At the tactical level, a new self-recoverable reinforcement learning algorithm is proposed to enhance the reinforcement learning's capability of dealing with failures. This new algorithm consists of two individual strategies: a multi-state recovery strategy and a failure-prevention strategy.

The multi-state recovery strategy focuses on how to recover from the already occurred non-fatal failures, avoiding the failure-induced manual resets to the initial state. The recovery is achieved by allowing robots to go back to previous safe states according to a replay of previous experience. More importantly, it further investigates which previous state is the best to recover back to from the current failure state for future efficient learning. The failure-prevention strategy focuses on how to prevent failures from happening in the learning phase. Unlike traditional methods that are inflexible and still need human interventions, this strategy avoids failures by self-learning safety constraints from its own past experiences, especially the failure experiences. The details can be seen in Chapter 4.

(iii) At the operational level, a new multi-objective-optimization-based auto-tuning method for control parameters is proposed to robustly achieve higher-level behaviors. It uses multi-objective optimization to automatically tune control parameters of a conventional proportional-integral-derivative (PID) controller. By setting the integrated time square error and control rate as two objectives, NSGA (Non-Dominated Sorting in Genetic Algorithms)-II algorithm, an

evolutionary algorithm, is used to find an optimal PID control parameter set. The details can be seen in Chapter 5.

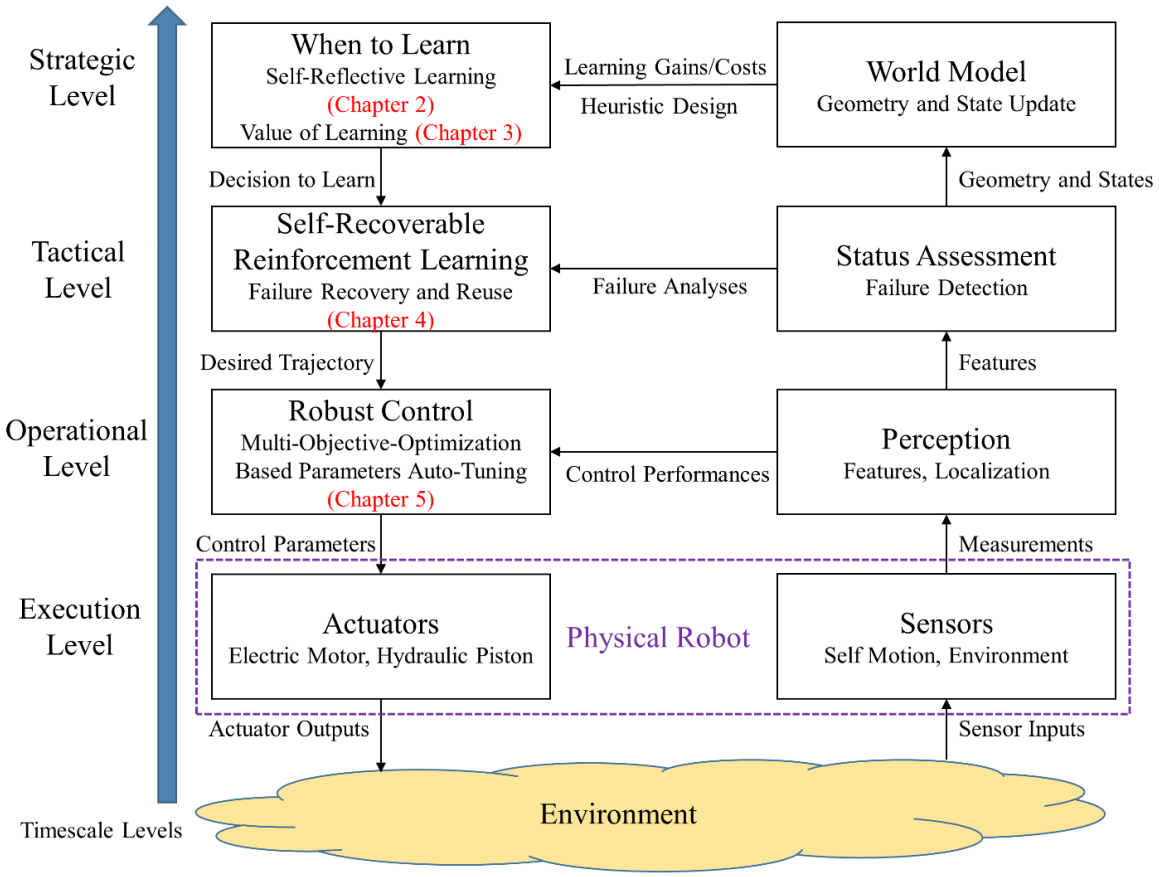

Figure 1.1 Hierarchical persistent autonomy architecture

1.5 References

[1] P. Newman, "MOOS: Mission Oriented Operating Suite," Technical Report OE2007-07, Department of Ocean Engineering, Massachusetts Institute of Technology, 2003.

[2] J. Evans, P. Patrón, B. Smith, and D. M. Lane, "Design and evaluation of a reactive and deliberative collision avoidance and escape architecture for autonomous robots," *Autonomous Robots*, vol. 24, no. 3, pp. 247-266, 2008.

[3] O. Santos, H. Romero, S. Salazar, and R. Lozano, "Real-time Stabilization of a Quadrotor UAV: Nonlinear Optimal and Suboptimal Control," *Journal of intelligent & robotic systems*, vol. 70, nos. 1-4, pp. 1-13, 2013.

[4] S. Bouabdallah and R. Siegwart, "Full control of a quadrotor," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Diego, CA, October 29 - November 2, 2007.

[5] J. Yu, S. Karaman, and D. Rus, "Persistent monitoring of events with stochastic arrivals at multiple stations," *IEEE Transactions on Robotics*, vol. 31, no. 3, pp. 521–535, 2015.

[6] D. M. Lane, "Persistent autonomy artificial intelligence or biomimesis?" in *IEEE/OES Autonomous Underwater Vehicles (AUV)*, Southampton, UK, September 24-27, 2012.

[7] H. Huang, G. M. Hoffmann, S. L. Waslander, and C. J. Tomlin, "Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering," in *IEEE International Conference on Robotics and Automation (IRCA)*, Kobe, Japan, May 12-17, 2009.

[8] J. Xiong and E. Zheng, "Position and attitude tracking control for a quadrotor UAV," *ISA Transactions*, vol. 53, no. 3, pp. 725-731, 2014.

[9] F. Ruggiero, et al.., "A multilayer control for multirotor UAVs equipped with a servo robot arm," in *IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, USA, May 26-30, 2015.

[10] M. Liu, Y. Tan, and V. Padois, "Generalized hierarchical control," *Autonomous Robots*, vol. 40, no. 1, pp. 17-31, 2016.

[11] A. Nagabandi, I. Clavera, S. Liu, et al., "Learning to adapt in dynamic, real-world environments via meta-reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2019.

[12] A. Gupta, B. Eysenbach, C. Finn, and S. Levine, "Unsupervised meta-learning for reinforcement learning," *arXiv preprint arXiv:1806.04640*, 2018.

[13] R. S. Sutton, A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

[14] J. Garcıa and F. Fernandez, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[15] M. Sato, H. Kimura, and S. Kobayashi, "Td algorithm for the variance of return and mean-variance reinforcement learning," *Transactions of the Japanese Society for Artificial Intelligence*, vol. 16, no. 3, pp. 353–362, 2001.

[16] P. Geibel and F. Wysotzki, "Risk-sensitive reinforcement learning applied to control under constraints," *Journal of Artificial Intelligence Research*, vol. 24, pp. 81–108, 2005.

[17] Y. Kadota, M. Kurano, and M. Yasuda, "Discounted markov decision processes with utility constraints," *Computers & Mathematics with Applications*, vol. 51, no. 2, pp. 279–284, 2006.

[18] T. M. Moldovan and P. Abbeel, "Safe exploration in markov decision processes," in *Proceedings of the 29th International Conference on International Conference on Machine Learning (ICML)*, pp. 1451-1458, 2012.

[19] H. Kashima, "Risk-sensitive learning via minimization of empirical conditional value-at-risk," *IEICE TRANSACTIONS on Information and Systems*, vol. 90, no. 12, pp. 2043–2052, 2007.

[20] Luenberger, D. G. *Investment science*. Oxford University Press, 2013.

[21] K. Driessens and S. Dˇzeroski, "Integrating guidance into relational reinforcement learning," *Machine Learning*, vol. 57, no. 3, pp. 271–304, 2004.

[22] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010.

[23] J. Tang, A. Singh, N. Goehausen, and P. Abbeel, "Parameterized maneuver learning for autonomous helicopter flight," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1142– 1148, 2010.

[24] A. L. Thomaz, C. Breazeal, et al., "Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance," in *AAAI*, vol. 6, pp. 1000–1005, 2006.

[25] J. Garcia and F. Fernandez, "Safe exploration of state and action spaces in reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 45, pp. 515-564, 2012.

[26] P. Vidal, R. Iglesias Rodriguez, M. A. Rodriguez Gonzalez, and C. Vazquez Regueiro, "Learning on real robots from experience and simple user feedback," *Journal of Physical Agents*, vol. 7, no. 1, pp. 57-65, 2013.

[27] C. Gehring and D. Precup, "Smart exploration in reinforcement learning using absolute temporal difference errors," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems (AAMAS)*, pp. 1037-1044, 2013.

[28] Law, E. L. *Risk-directed Exploration in Reinforcement Learning*. McGill University, 2005.

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

[30] S. Lengagne, J. Vaillant, E. Yoshida, and A. Kheddar, "Generation of whole-body optimal dynamic multi-contact motions," *The International Journal of Robotics Research*, vol. 32, nos. 9-10, pp. 1104-1119, 2013.

[31] V. Vonásek, S. Neumann, D. Oertel, and H. Wörn, "Online motion planning for failure recovery of modular robotic systems," In *IEEE International Conference on Robotics and Automation (ICRA),* pp. 1905-1910, 2015.

[32] C. C. Hang., J. K. Astrom, W. K. Ho, "Refinements of Ziegler Nichols Tuning formula," *IEEE Proceedings*, vol. 138, no. 2, pp. 111, 1991.

[33] M. L Luyben and W. L. Luyben. *Essentials of Process Control*. New York: McGraw-Hill, 1997.

[34] K. J. Astrom and T. Hagllund. *PID controllers Theory, Design and Tuning*. Research Triangle Park, NC: Instrument Society of America,1995.

[35] G. H. Cohen and G. A. Coon, "Theoretical investigations of retarded control," *Transactions of the ASME*, vol. 75, pp. 827, 1953.

[36] J. Ribeiro, M. F. Santos, M. J. Carmo, and M. F. Silva, "Comparison of PID controller tuning methods: analytical/classical techniques versus optimization algorithms," In *18th International Carpathian Control Conference*, 2017.

[37] T. E. Marlin. *Process Control*. New York: McGraw-Hill, 1995.

[38] M. Morari and E. Zafirion. *Robust Process Control*. Englewood Cliffs, NJ: Prentice Hall, 1989.

[39] C. A. Smith and A. B. Copripio. *Principles and Practice of Automatic Process Control*. New York: John Wiley & Sons, 1985.

[40] R. Ciancone and T. Marlin, "Tune controllers to meet plant objectives," *Control May*, pp. 50-57, 1992.

[41] R. Gorez, "A survey of PID auto-tuning methods", *Journal A*, vol. 38, pp. 3-10, 1997.

[42] Y. Li, W. Feng, K. C. Tan, X. K. Zhu, X. Guan, and K. H. Ang, "PID easy and automated generation of optimal PID controllers," in P*roceedings of 3rd Asia Pacific Conference Control and Measurement*, Dunhuang, China, 1998.

[43] R. Evins, "A review of computational optimization methods applied to sustainable building design," *Renewable and Sustainable Energy Reviews*, vol. 22, pp. 230-245, 2013.

[44] M. Palonen, A. Hasan, and K. Siren, "A genetic algorithm for optimization of building envelope and HVAC system parameters," in *IBPSA Conference on Eighth International Building Performance Simulation Association*, Glasgow, Scotland, July 27-30, 2009.

[45] Y. Lu, S. Wang, Y. Zhao, and C. Yan, "Renewable energy system optimization of low/zero energy buildings using single-objective and multi-objective optimization methods," *Energy and Buildings*, vol. 89, pp. 61-75, 2015.

CHAPTER 2

SELF-REFLECTIVE LEARNING STRATEGY FOR PERSISTENT AUTONOMY OF AERIAL MANIPULATORS

A paper accepted for publication on *ASME Dynamic Systems and Control Conference 2019*[1]

Xu Zhou[2], Jiucai Zhang[3], and Xiaoli Zhang[4]

2.1 Abstract

Autonomous aerial manipulators have great potentials to assist humans or even fully automate manual labor-intensive tasks such as aerial cleaning, aerial transportation, infrastructure repair, and agricultural inspection and sampling. Reinforcement learning holds the promise of enabling persistent autonomy of aerial manipulators because it can adapt to different situations by automatically learning optimal policies from the interactions between the aerial manipulator and environments. However, the learning process itself could experience failures that can practically endanger the safety of aerial manipulators and hence hinder persistent autonomy. In order to solve this problem, we propose for the aerial manipulator a self-reflective learning strategy that can smartly and safely finding optimal policies for different new situations. This self-reflective manner consists of three steps: identifying the appearance of new situations, re-seeking the optimal policy with reinforcement learning, and evaluating the termination of self-reflection. Numerical simulations demonstrate, compared with conventional learning-based autonomy, our strategy can significantly reduce failures while still can finish the given task.

---

[2] Primary researcher and author, graduate student, Department of Mechanical Engineering, Colorado School of Mines, 1610 Illinois Street, Golden, Colorado, 80401, USA.

[3] Co-author, Guangzhou Automotive Group R&D Center Silicon Valley Inc, Sunnyvale, California, 94085, USA.

[4] Corresponding author, Associate Professor, Department of Mechanical Engineering, Colorado School of Mines, 1610 Illinois Street, Golden, Colorado, 80401, USA.

2.2 Introduction

Autonomous aerial manipulators have great potentials to assist humans through manipulations in dangerous or remote locations, and to automate manual labor-intensive tasks such as aerial cleaning, aerial transportation, infrastructure construction and repair, and agricultural inspection and sampling. Due to the varieties of the system configurations, tasks, and environments, it is challenging for aerial manipulators to achieve persistent autonomy which requires aerial manipulators to safely operate in dynamic or unstructured environments for extended lengths of time with minimal human interventions.

Reinforcement learning [1] holds the promise for persistent autonomy of aerial manipulators because it can adapt to different situations by automatically learning optimal policies from the interactions between the aerial manipulator and environments. However, failures can be unavoidable during the learning process because reinforcement learning can only learn the outcome of an action by executing the action itself. These failures can lead to serious safety issues in practical systems such as the crash of aerial manipulators, which is not acceptable in persistent autonomy, especially for safety-critical systems. Although human interventions could address some failures, they are also not desirable in persistent autonomy. Thus, it is ideal to avoid these failures during the practical deployment of reinforcement learning on persistent autonomy.

While some safe reinforcement learning algorithms [2] could reduce failures by adding safety concerns into the optimization criteria or restricting to safe exploratory actions, they were still restricted within the scope of reinforcement learning itself. In other words, they assumed that they were always in the learning/training process to find a new optimal policy for a new situation. However, in persistent autonomy, the old policy may still work or even remain optimal when the situation changes to a new one. In this case, activating learning to re-find the optimal policy is

actually unnecessary and unbeneficial because re-learning can bring more failures. Hence, it is safer to not activate learning and use current policy to finish the task. Similarly, even if the learning is activated, it could be sometimes enough for reinforcement learning to only find a sub-optimal policy with fewer failures as long as this policy can finish the task. From this high-level perspective focusing on the system safety, the practical deployment of reinforcement learning on persistent autonomy actually raises a fundamental but unanswered question: ***when to start learning a new policy and when to terminate the learning?***

While this question is difficult to robots, humans can well solve this problem with their self-reflection capability [3][4]. More specifically, humans prefer doing nothing and remaining current behaviors if they are satisfied with the outcomes of their decisions or actions. If not satisfied, however, humans are highly likely to start changing behaviors and continue exploring new, alternative actions until the outcomes satisfy them again. Inspired by this self-reflection process, we present a novel self-reflective learning strategy for the aerial manipulator to smartly and safely operate in different situations. This strategy includes three steps. The first step is self-determining if the aerial manipulator is encountering a new situation by evaluating the necessity degree to change the control policy. If the necessity degree value is low, the strategy keeps its current policy because no new situation is identified. Once a new situation is detected, in the second step, the strategy makes new explorations about the new situation by using reinforcement learning. The maximum number of iterations is determined by the necessity degree value. The third step is deciding when to stop self-reflection. The termination criteria are related to task performances. With these three steps, the output policy is considered as safer for the new situation.

We consider our main contributions as: (1) Develop a new cognitive learning strategy with a self-reflective architecture for the aerial manipulator to step towards persistent autonomy. While

still maintaining the capability to finish a given task, this strategy focuses on improving the system safety by smartly deciding the timing of activating and terminating learning. (2) Demonstrate our strategy can experience substantially fewer failures when finishing a given task with simulations. The results also show that learning can be sometimes unbeneficial and unnecessary for a practical system to finish a given task when situation changes.

2.3 Related Work

In general, any unmanned aerial vehicle (UAV) equipped with any degree-of-freedom (DOF) robotic arm can be regarded as an aerial manipulator. For simplicity, this work investigates the quadrotor equipped with a 3-DOF robotic arm (the quadrotor-arm system) as a specific example due to its popularity in aerial manipulators. However, our strategy can be also used in other aerial manipulators or robots because it does not rely on specific robot dynamics.

While several methods [5][6] have been reported for the quadrotor-arm system to autonomously finish a task, they were limited to a well-defined situation without changes. If the situation such as the task or environment changes, their performances may reduce greatly. Reinforcement learning has shown great adaptabilities to different unknown situations in robotics [1], but it has not been really investigated in the aerial manipulator, which could be difficult due to the aerial manipulator's high complexity and nonlinearity. More importantly, failures in the learning process can result in serious consequences on the aerial manipulator like the crash.

With regard to safety and/or risks during the learning and/or deployment process, two fundamental tendencies of safe reinforcement learning methods [2] have been reported. The first one consists of transforming the traditional optimization criteria that maximizes the expectation of the return to more comprehensive criteria respecting learning safety such as the worst-case criterion [2], the risk-sensitive criterion [7], the constrained optimization criterion [8], and other

optimization criteria in financial engineering [9]. Since exploratory actions could have serious consequences, the second tendency improved the safety with modifying the exploration process in two ways: (1) through the incorporation of external knowledge such as providing initial knowledge [10], deriving a policy from a finite set of demonstrations [11], and providing teach advice [12], and (2) through the use of a risk-directed exploration metric [13]. However, these methods restricted themselves within the training process of reinforcement learning. In contrast, our strategy reduces failures from a more general system's perspective with understanding learning can be unnecessary and unbeneficial and deciding when to activate or terminate learning to avoid such unbeneficial learnings.

2.4 Self-Reflective Learning Strategy

As shown in Figure 2.1, the self-reflective control strategy includes three important steps. The first step is to determine if a new situation is identified and the necessity of adaptation to the new situation. The identification criteria are usually related to the system's maximum capability and situation change degree. If no new situation is detected, the self-reflective strategy believes the current policy is functional and no more action is needed. However, when a new situation is detected, the self-reflective strategy uses reinforcement learning technique to re-seek an optimal policy. New policies are tried with approximate value iteration (AVI) reinforcement learning method [14]. The maximum number of trials or iterations is proportional to the value of the necessity degree value to have more chances to find an optimal policy. During this process, the self-reflection can be terminated if a termination criterion is met, which is usually the satisfaction of new policy's performance.

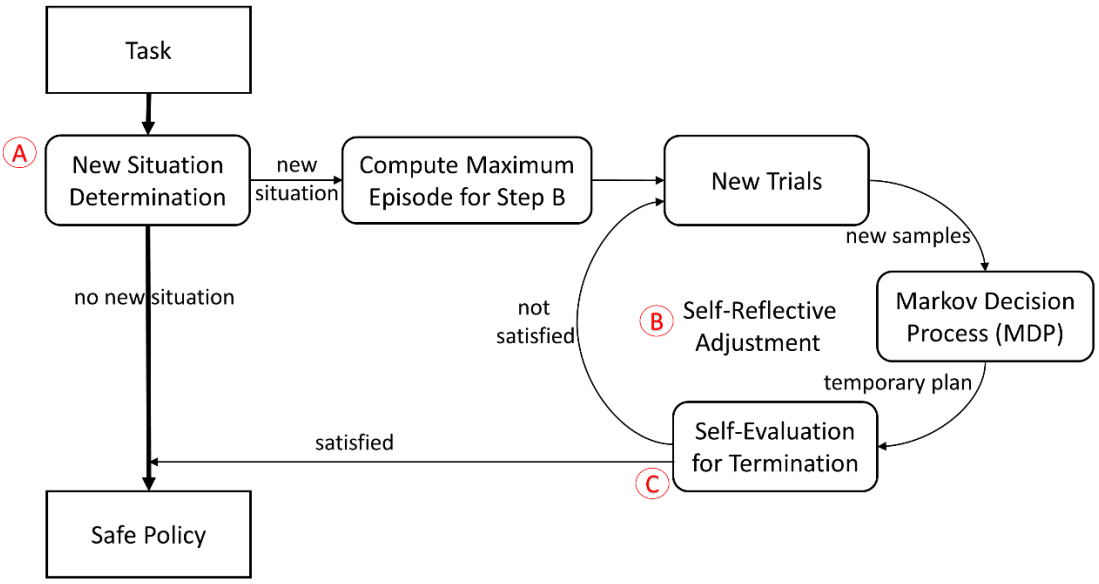

Figure 2.1 Overall self-reflective learning scheme

2.4.1 New Situation Determination

This step is to identify if there is a new situation that has not been met before. The basic idea is to compute a necessity degree value α based on the aerial manipulator system's maximum capability and situation change degree. As we use the quadrotor-arm system for example, the criteria can include the limit of the quadrotor's attitudes as well as the change of the quadrotor's CoG (center of gravity) and end-effector's position. The limit of quadrotor's attitudes represents the maximum capability of the quadrotor-arm system to remain stable. The CoG change and position change indirectly reflect situation changes as different situations result in different variations of quadrotor's CoG and end-effect's position. If either of the two parameters is out of a reasonable range, indicating that the system behavior is strange and unexpected, then the current situation should be changed to a new one. The necessity degree value α, which is between 0 and 1, is calculated based on the deviation of the criteria from reasonable ranges. A threshold α_{th} is defined to determine if a new task is encountered and its value can be empirically defined for

different manipulators. If $\alpha > \alpha_{th}$, then a self-reflection process is needed. Otherwise, the original policy is still considered as functional. Mathematically, α can be formulized as

$$\alpha = \text{sigmoid}\left(a_1(\phi - \phi_e) + a_2(\theta - \theta_e) + a_3\left(\Delta C_Q - \Delta C_{Q_e}\right) + a_4\frac{\Delta E_e}{\Delta E}\right), \qquad (2.1)$$

where ϕ and θ indicates the quadrotor's roll and pitch angles, ΔC_Q is the change of quadrotor's CoG, ΔE is the end-effector's position change, a_i are the coefficients to adjust each term's importance, and all variables with subscript e are the critical values the system should obey. For example, $\phi_e = 60$ degrees means the quadrotor's roll angle is not supposed to exceed 60 degrees, otherwise the system is considered unstable. $\Delta E_e = 0.01$m indicates the end-effector is supposed to move effectively with at least a 0.01m change. Each criterion is normalized first, and their sum is then transferred into [0,1] with a logistic sigmoid function (i.e., $f(x) = \frac{1}{1+e^{-x}}$). The reason choosing this sigmoid function is that any measurements close to the unsafe criteria in (2.1) can make α approach 1 quickly, which means a new situation is more likely to appear. This can help detecting an unusual or new situation effectively.

2.4.2 Self-Reflective Adjustment

We formulate the quadrotor-arm self-reflective adjustment process as a Markov Decision Process (MDP), specified by the 5-tuple (S, A, $\{P_{sa}\}$, r, γ). Let S be the set of system states, where each state is a vector containing the positions, poses, and corresponding velocities. Let A be the set of actions the system can take from any state, where an action is a set of inputs including forces and torques. $\{P_{sa}\}$ describes the unknown state transition probabilities, capturing the dynamics of the system. Specifically, $P_{sa}(s')$ is the probability of transitioning to state $s' \in S$, given current state $s \in S$ and applied action $a \in A$. The reward function is defined as $r: S \longrightarrow \mathbb{R}$, representing

the cost and reward structure of the state space. Let γ be a discount factor on rewards, chosen to be 0.99 in this case.

The appropriate action to take at a given state can be defined in terms of a fixed policy π, where $a = \pi(s)$. The total expected payoff (or value) of a given state can be defined as

$$V^{\pi}(s) = \mathbb{E}[r(s_0) + \gamma r(s_1) + \gamma^2 r(s_2) + \cdots | s_0 = s, \pi]. \tag{2.2}$$

Equation (2.2) can formulate in the so-called Bellman Equations [1]

$$V^{\pi}(s) = r(s) + \gamma \sum_{s' \in S} P_{sa}(s') V^{\pi}(s'). \tag{2.3}$$

The optimal value function can then be defined as

$$V^{*}(s) = r(s) + \gamma \max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^{*}(s'). \tag{2.4}$$

Value iteration is a commonly used RL technique where the optimal value function is iteratively found and used to compute an optimal policy [1]. The optimal policy can be obtained as

$$\pi^{*}(s) = \operatorname*{argmax}_{a \in A} \sum_{s' \in S} P_{sa}(s') V^{*}(s'). \tag{2.5}$$

However, the tabular representation of the value function is impractical or infeasible when the state and action spaces are large or continuous. Instead, we consider using approximate value iteration (AVI) method [14] to solve the above MDP problem. With this method, the value function is approximated with a linearly parameterized feature vector $F(s)$, which is specifically chosen for this problem including the quadrotor's displacement and its velocity magnitude, arm's end-effector displacement and its velocity magnitude. Mathematically, $F(s)$ can be expressed as $F(s) = [\|\boldsymbol{p}\|^2 \quad \|\dot{\boldsymbol{p}}\|^2 \quad \|\boldsymbol{p_e}\|^2 \quad \|\dot{\boldsymbol{p}}_{\boldsymbol{e}}\|^2]^T$. The reward function is designed to penalize the distance from the goal state for both the quadrotor and arm. The quadrotor's altitude is also penalized to provide a bounding box so that the whole system is enforced to stay above the ground. The form is $r(s) = [\beta_1 \quad \beta_2 \quad \beta_3][r_1(s) \quad r_2(s) \quad r_3(s)]^T$, where

$$r_1(s) = -\|p\|^2$$

$$r_2(s) = -\|p_e\|^2$$

$$r_3(s) = \begin{cases} -10000, & z < 0.3 \\ 0, & z \geq 0.3 \end{cases}$$

Because AVI is an iterative process, it only ends when the maximum iterative number (or maximum episode) is reached or the value function (2.3) is converged. The selection of maximum episode is important because it is better to be large enough to allow the value function to be converged. However, it cannot be too large if considering the practical availability. More specifically, when reinforcement learning cannot converge to an optimal policy for some new situations within the defined maximum episodes, increasing the maximum episode generally means a longer time, which may be not allowed in some situations. For example, a large maximum episode can give the aerial manipulator an ability to finish a new task in 15 minutes, but the task is required to be finished in 10 minutes. The sensitive 10-minutes requirement can make the selection of a large maximum episode meaningless. A convenient way to adjust the maximum episode can be based on an empirical constant and the necessity degree value α as in (2.6).

$$episode_{max}^{new} = \begin{cases} episode_{max}^{init} \times (1 + b\alpha), & if\ \alpha \geq \alpha_{th} \\ episode_{max}^{init}, & if\ \alpha < \alpha_{th} \end{cases}, \tag{2.6}$$

where $episode_{max}^{init}$ is empirically set to be 10000, which corresponds to about 17 minutes using a PC with Intel i7-4790 dual Core CPU and 8G RAM. If there is no self-reflection (i.e., $\alpha < \alpha_{th}$), no maximum episode is needed due to no need of re-learning. However, we still assign a value ($episode_{max}^{init}$) for this situation just in case the system accidently learns a policy again when no self-reflection is needed. The coefficient b before α expands the maximum episode corresponding to the new level of situations. If α is close to 1, it means the current situation is very unlikely to be met before and hence the system may need a totally different policy. A new situation usually needs

more trials, which means the convergence may be slower because more samples need to be collected to get the new situation well known. $\alpha = 1$ could increase the maximum episode to (1+b) of the empirical constant and we can still use a larger b to increase the maximum episode, which can be suitable for highly complex tasks that need a long time to learn out an optimal policy. In general, α_{th} mainly determines if re-finding control policy is needed. b and α decides how large the maximum episode will be.

2.4.3 Self-Evaluation for Termination

When MDP outputs a new policy, we also need a self-evaluation module to decide whether the self-reflection should stop. In this paper, we define the stop criterion as

$$TC = c_1\left(\left\|\boldsymbol{p_e} - \boldsymbol{p_{e_r}}\right\|^2 + \|\boldsymbol{p} - \boldsymbol{p_r}\|^2 + \|\boldsymbol{\Omega} - \boldsymbol{\Omega_r}\|^2\right) + c_2\left(t_s - t_{s_r}\right)^2, \tag{2.7}$$

where $\boldsymbol{p_e}$ is the end-effector's position, $\boldsymbol{p}$ is the quadrotor's position, $\boldsymbol{\Omega}$ is the quadrotor's attitude, t_s is the stabilization time, and all corresponding variables with subscript r are the desired values. c_1 and c_2 are two coefficients balancing the focus between the accuracy and stabilization time of quadrotor-arm system. If c_1 is larger, then the system focuses more on the accuracy of finishing a task. Otherwise, the system's stabilization time has a higher priority. It can be noticed that (2.7) and (2.1) have some overlapping terms such as ϕ and $\boldsymbol{\Omega}$. However, there are two major differences. One is the inclusion of more criteria about task performances such as t_s in (2.7). The other one is the different meaning of the similar comparison terms in (2.1) and (2.7). The comparison terms in (2.1) mean the system's maximum capability to keep stable, but those in (2.7) indicate a satisfactory performance of a given task. Equation (2.7) is generally stricter than (2.1). In addition, the termination criterion value (*TC*) has a value only if a new situation is considered, and the self-reflection can stop once *TC* is less than a threshold TC_{th}. If there is no new situation detected, *TC* is intentionally set to 0. The selection of TC_{th} is based on specific requirements of different users.

For example, some users may only require the system not to crash, but some may need the system to perform the tasks with good performances like smaller deviation and shorter stabilization time. In general, TC_{th} should be smaller if the requirement is stricter. The use of TC_{th} may scarify some task performances as the learned policy does not necessarily need to be theoretically optimal. Instead, as long as the new policy can meet the requirement of finishing a task, it can be considered as safely functional, which implicitly considers the system's safety as an optimization criterion as well. Whatever TC_{th} is, for any specific case, there is always a maximum TC_{th}, defined as TC_{th_max}. TC_{th_max} represents the loosest termination criterion to keep a particular system stable, i.e., not crash. For a given system, TC_{th_max} is fixed and only depends on the system. As long as $TC \leq TC_{th} \leq TC_{th\_max}$, the learned control strategy can output an optimal control policy.

2.5 Validation Simulations

In this work, all simulations are completed in MATLAB/Simulink. The parameters of the quadrotor-arm system [5] are listed in Table 2.1. More modeling details about the quadrotor-arm system can be also seen in [5].

In order to validate the self-reflective strategy, the quadrotor-arm system is supposed to carry an object from position A ($\boldsymbol{\eta} = [-45, 45, 45]$ (degrees)) to position B ($\boldsymbol{\eta} = [0, 45, 45]$ (degrees)) as shown in Figure 2.2. According to the parameters in Table 2.1, the quadrotor is supposed to be stabilized at a certain location ($\boldsymbol{p_r} = [0, 0, 0.3]$ (m), $\boldsymbol{\Omega_r} = [0, 0, 0]$ (degree)). The object weight can be 0.1kg, 0.15kg, 0.2kg, 0.25kg, and 0.3kg. An MDP controller is designed for a 0.1kg object beforehand, then the other weights can be considered as new tasks, depending on the self-reflective strategy's decision. Given the designed task, the parameters of the self-reflective strategy are listed in Table 2.2. TC_{th_max} is assigned with 0.95 to remain the system stable. However, the task that is

considered to be successfully finished should also include the object movement from A to B, which is why we choose TC_{th} to be 0.7.

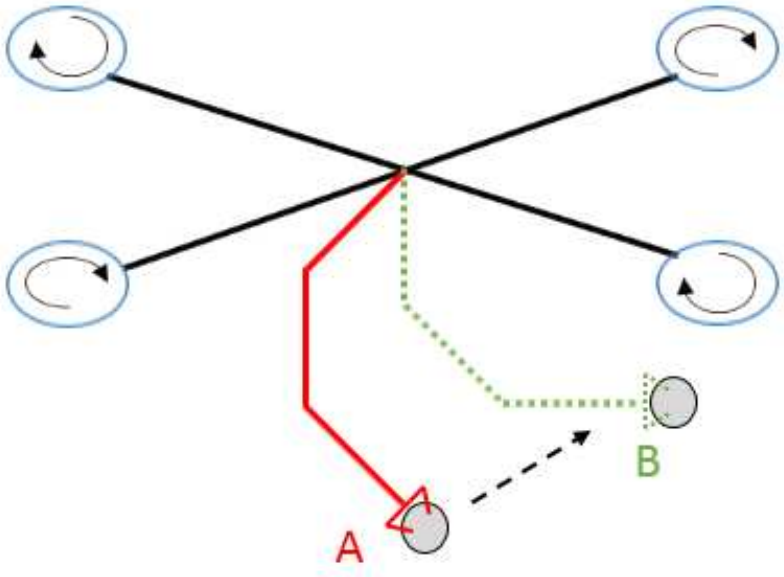

Figure 2.2 Visualization of the object-carry task

Table 2.1 Quadrotor simulation parameters

Parameters	Values	Descriptions
I_{bx}, I_{by}	1.24 kg · m^2	Moment of inertia around quadrotor's X, Y axis
I_{bz}	2.48 kg · m^2	Moment of inertia around quadrotor's Z axis
m_b	2 kg	Mass of quadrotor
γ_t	3.13*10^(-5)	Thrust factor
γ_d	7.5*10^(-7)	Drag factor
l	0.1 m	Distance between the rotor center and quadrotor center
m_1, m_2, m_3	0.1 kg	Mass of the first, second, and third link of the arm
L_1, L_2, L_3	0.1 m	Length of the arm's first, second, and third link
I_{x1}, I_{y1}	0.0025 kg · m^2	Moment of inertia around first link's X, Y axis
I_{z1}	0.0075 kg · m^2	Moment of inertia around first link's Z axis
I_{x2}, I_{y2}	0.0025 kg · m^2	Moment of inertia around second link's X, Y axis
I_{z2}	0.0075 kg · m^2	Moment of inertia around second link's Z axis
I_{x3}, I_{y3}	0.0025 kg · m^2	Moment of inertia around third link's X, Y axis
I_{z3}	0.0075 kg · m^2	Moment of inertia around third link's Z axis

Table 2.2 Self-reflective control strategy parameters

Parameters	Values
a_1, a_2, a_3, a_4	1
b	2
c_1, c_2	80, 20
γ	0.99
$\beta_1, \beta_2, \beta_3$	10000, 800, 50
α_{th}	0.5
TC_{th}	0.7
TC_{th_max}	0.95
ϕ_e, θ_e	60 degrees
ΔC_{Q_e}	0.15m
ΔE	0.01m
$\boldsymbol{p_{e_r}}$	[0.17, 0, 0.13] m
$\boldsymbol{p_r}$	[0, 0, 0.3] m
$\boldsymbol{\Omega_r}$	[0, 0, 0] degrees
t_{s_r}	10 s

2.6 Results

2.6.1 New Situation Determination

The results of new situation determination for different weights are shown in Figure 2.3. The new situation determination values (reflection necessity degree values) for five different weights are 0.06, 0.20, 0.41, 0.84, and 1, respectively. The default threshold α_{th} (conservative threshold) is 0.5, so only two cases (0.25kg and 0.3kg) need self-reflections. However, if we decrease the threshold α_{th} to 0.3 (aggressive threshold), the 0.2kg case needs self-reflection as well. This is because a smaller α_{th} means more situations are considered as new situations, which indicates self-reflection is triggered easier (i.e., self-reflection is more needed). Instead of decreasing the threshold value α_{th}, another way to trigger self-reflection easier is to adjust constraints on the designed parameters, ϕ_e, θ_e, ΔC_{Q_e}, ΔE_e, in (2.1) to be stricter. More specifically, ϕ_e, θ_e, and ΔC_{Q_e} should be smaller while ΔE_e should be larger to cause an easier trigger. The result of

reducing ΔC_{Q_e} from 0.15 to 0.05 can be found as the blue line in Figure 2.3. The necessity degree values of five weights are 0.09, 0.27, 0.55, 0.90, and 1, respectively. If using the default threshold 0.5, one more case (0.2kg) needs self-reflection if compared with the loose maximum quadrotor CoG change (red line with ΔC_{Q_e} = 0.15). It is hence convenient to adjust when a new situation should be considered or a self-reflection is needed.

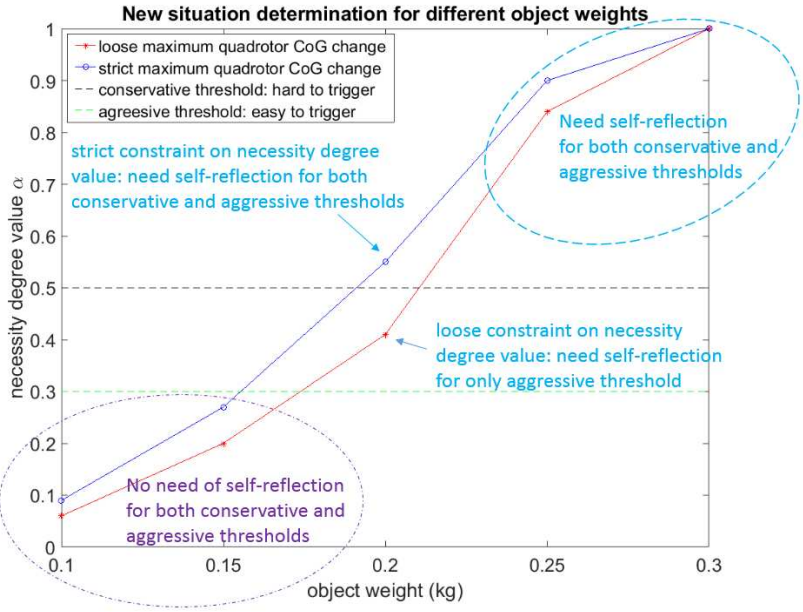

Figure 2.3 New situation determination for different object weights. For 0.1kg and 0.15kg, there is no need of self-reflection as they are similar to pre-trained situations. For 0.25kg and 0.3kg, self-reflection is needed in all different parameter settings as they are greatly different from pre-trained situations. For 0.2kg, the necessity degree value is less with loose constraints than that with strict constraints. If given the conservative threshold, only the case with strict constraints needs self-reflection. But given an aggressive threshold, both cases with strict and loose constraints, respectively, need self-reflection.

2.6.2 Self-Reflective Adjustment

The change of maximum episode for different object weights is shown in Figure 2.4. Both the 0.1kg and 0.15kg case do not need self-reflection, so their maximum episodes are intentionally set to be the empirical constant 10000 according to (2.6). For the 0.2kg case, selections of parameters

to calculate α and α_{th} are both important. If using the same α_{th}, a stricter selection of parameters in (2.1) needs to adjust its maximum episode because it needs self-reflection. Similarly, if keeping the selection of parameters the same, a small α_{th} needs to adjust its maximum episode too. For the 0.25kg case and 0.3kg case, they all need self-reflections and the maximum episodes of 0.3kg are all larger than 0.25kg because the necessity degree value α for 0.3kg is larger. In particular, the maximum episode of the 0.3kg case are all the same for four different combinations. This is because 0.3kg is too heavy for the quadrotor's current controller to handle, so ΔE in (2.1) equals to 0 and hence α is 1 and all maximum episodes become 30000.

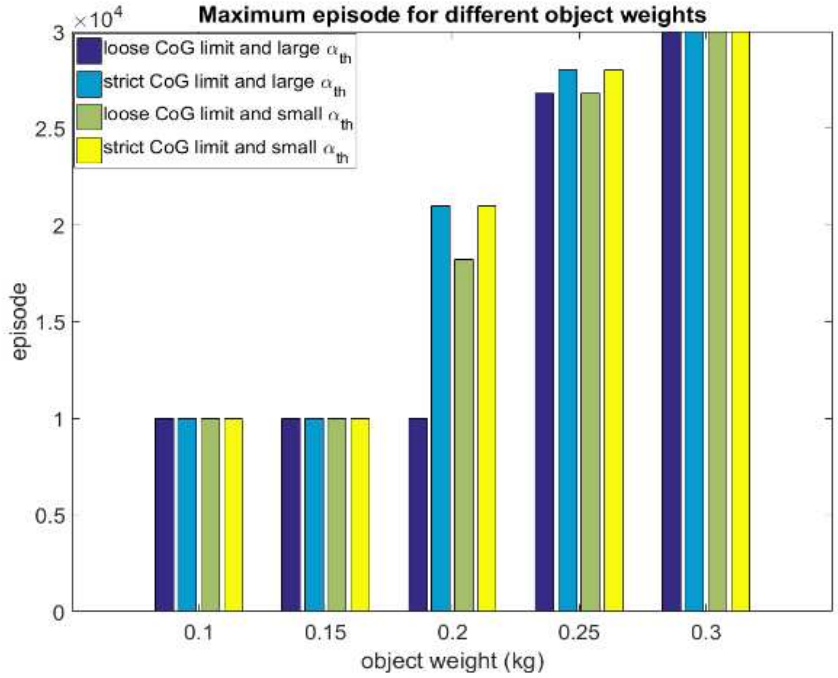

Figure 2.4 Maximum episode for different object weights

2.6.3 Self-Evaluation for Termination

Table 2.3 shows termination criterion values and episodes with respect to different parameters. If using the default parameters in Table 2.2, 0.1kg, 0.15kg and 0.2kg are not considered as new

situations. Thus, their termination values are 0 and episodes used to find the optimal control policy are 10000. On the contrary, the termination criterion value for 0.25kg and 0.3kg are 0.68 and 0.69, respectively. The corresponding episodes are 20336 and 28212. More episodes are needed for 0.3kg because this case deviates more from the original controller designed for 0.1 kg object than 0.25kg and is hence much more difficult for the system to handle.

TC_{th} has an influence on the self-reflection results including final termination criterion value and episodes used for learning a new policy. Changing $\boldsymbol{p_r}$, $\boldsymbol{\Omega_r}$, t_{s_r}, c_1 ,and c_2 can also affect TC and so self-reflection results can be different. Because of various combinations of the related parameters, we only change c_1 and c_2 for simple examples. Also, both changes are applied to 0.25kg and 0.3kg cases because these two cases are easily switched between task success and task failure. The new termination criterion values and episodes are displayed in Table 2.3 after re-doing the simulations with new parameters. Reducing TC_{th} from 0.7 to 0.6 and keeping other parameters unchanged means the current task can be considered as successful only with better performances. With this new setting, the 0.25kg case can still work but the 0.3kg case fails because the termination criterion value can reach only 0.66 within 30000 episodes. The failure means no optimal policy can be found with the given episodes and stricter performance criteria. If increasing the maximum episode number, the 0.3kg case may work again. For an additional test, we change c_1 to 20 and c_2 to 80, which means the controller focuses more on stabilization time. Although we use the same TC_{th} (0.7) as in Table 2.2, the termination criterial values are still different from c_1=80 and c_2=20. The 0.25kg case needs 4766 more episodes, and the 0.3kg case fails. This is reasonable because limited time is a stronger constraint than good accuracy for the designed task if using the conclusion from the original 0.3kg case that the accuracy can be satisfied if given unlimited time.

Table 2.3 Termination criterion value and episodes for different parameters

Object weight (kg)	Parameters	TC_{th_max}	TC_{th}	Termination criterion value	Episodes
0.1	c_1=80, c_2=20	0.95	0.7	0	10000
0.15	c_1=80, c_2=20	0.95	0.7	0	10000
0.2	c_1=80, c_2=20	0.95	0.7	0	10000
0.25	c_1=80, c_2=20	0.95	0.7	0.68	20336
0.3	c_1=80, c_2=20	0.95	0.7	0.69	28212
0.25	c_1=80, c_2=20	0.95	0.6	0.59	27482
0.3	c_1=80, c_2=20	0.95	0.6	0.66	30000
0.25	c_1=20, c_2=80	0.95	0.7	0.68	25102
0.3	c_1=20, c_2=80	0.95	0.7	0.74	30000

2.6.4 Task Performance

The task performances include both the quadrotor's performance and the arm's performance. For simplicity, only link 1's movements are shown in Figure 2.5 because link 1 has the most obvious movement. From this figure, it can be seen the original policy is good for 0.1kg. For the 0.15kg and 0.2kg cases, the overshoots and stabilization time increase greatly. Especially for the 0.2kg case, it is close to the unstable status. The unstable 0.25kg case is considered as a new situation and then a self-reflection starts. Although the performance after self-reflection cannot catch 0.1kg's performance, it is much better than 0.15kg and 0.2kg. Carrying a 0.3kg object also needs self-reflection but the performance is worse than that of 0.25kg. This indicates that the self-reflection has a maximum capability to deal with new situations. If the situation difference is out of a system's maximum capability, the self-reflective strategy may not work. We keep increasing the object weight until the self-reflective strategy totally fails with settings in Table 2.2. The corresponding maximum object weight is 0.34kg, which means the aerial manipulator fails to move an object over 0.34kg from A to B with an expected accuracy and time, but it does not mean that the aerial manipulator crashes since TC could still be smaller than TC_{th_max}.

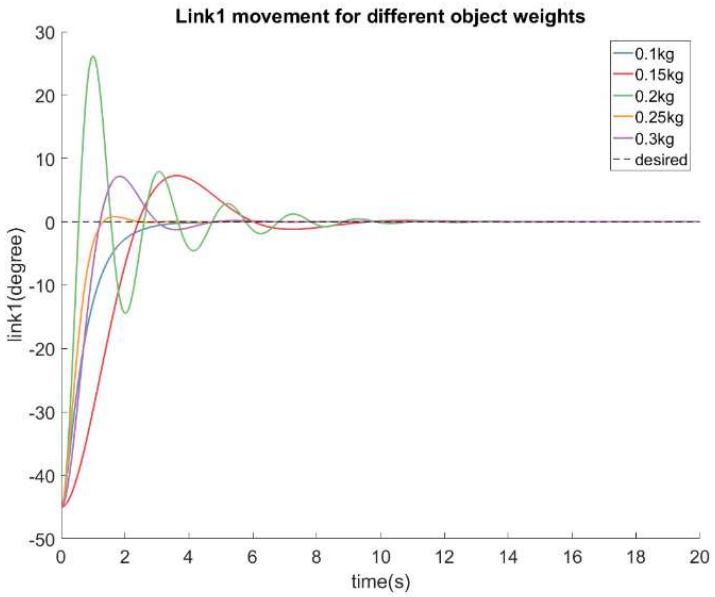

Figure 2.5 Link1's movement for different object weights. The movements with 0.15kg and 0.2kg are still acceptable using the pre-trained strategy for 0.1kg. But the movement with 0.2kg is close to unstable due to the large overshoot and long setting time. Because self-reflection is applied for 0.25kg, the movement is stable again with a smaller overshoot and shorter setting time. The movement with 0.3kg becomes worse with using the strategy for 0.25kg but without self-reflection.

In order to present the self-reflection process, the task performance is quantified with integral time squared error (ITSE). ITSE is a comprehensive evaluation criterion because it covers the overshoot, stabilization time, rising time, and the like. In detail, the task performance is calculated using $task\ performance = \sum_{dof=1}^{9} 100/\int_0^\infty te_{dof}(t)^2\,dt$, where 100 is the user-defined number to get the scale back into a reasonable range [0,1]. The summation is needed because ITSE can only apply to 1 DOF, and the aerial manipulator has 9 DOFs. The greater the ITSE value is, the better performance the system has. The task performances for different object weights can be seen in Figure 2.6. If not using the self-reflective strategy, the performance keeps decreasing as the object weight increases. According to (2.1), the performance drops from 0.1kg to 0.15kg and 0.15kg to 0.2kg are acceptable. But it is not acceptable with the drop from 0.2kg to 0.25kg, so this

is when self-reflection is needed. With applying self-reflection, the performance is increased back to the acceptable range. Since self-reflection is not kept used during the change from 0.25kg to 0.3kg, the performance drops again although it is still acceptable.

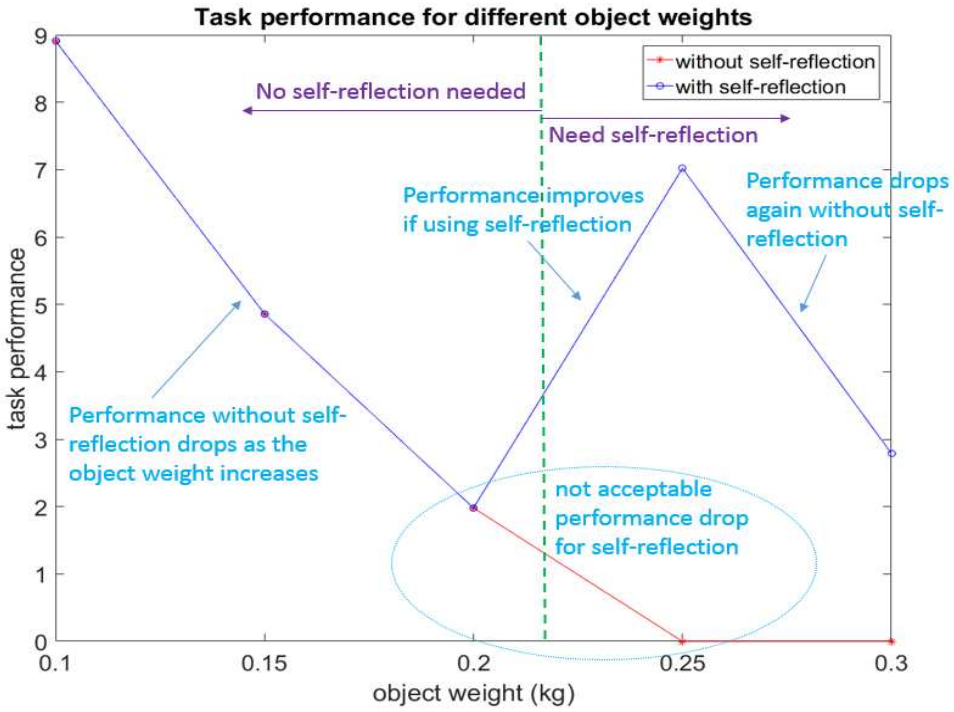

Figure 2.6 Task performances for different object weights. If self-reflection is not triggered, the task performance decreases as the object weight increases from 0.1kg to 0.2kg. Once self-reflection is needed and applied at 0.25kg due to an unacceptable performance drop, the task performance increases back to be acceptable. When addressing 0.3kg, the task performance drops again because self-reflection is determined to be not used.

2.6.5 Failure Costs and Computational Costs

Table 2.4 shows the failure costs and computational costs of two different methods: (a) always using reinforcement learning with the maximum episode to be 30000 and (b) self-reflective method with the default parameters in Table 2.2. Always using reinforcement learning means that reinforcement learning is always applied again for each time the situation changes. The definition of the failure is simplified as either ϕ or θ exceeds its limitation defined in Table 2.2, which can

be considered as a failure of physical instability. Also, a large deviation from the goal with $r(s) < -10000$ can be considered as a task failure. The computational cost is defined as the episodes used to converge to an optimal policy. The total numbers of failures and episodes using our method are much less than the corresponding numbers using always reinforcement learning. In most cases, our method has zero failures due to no need of re-learnings which are however necessary in always reinforcement learning. Even for the 0.25kg case that our method also re-learns, our method still has fewer failures and computational costs because the self-termination removes some unnecessary re-learnings. Given that both methods can finish the task, these re-learnings are actually unbeneficial because they result in more computational costs and failures.

Table 2.4 Failure costs and computational costs

Object Weight Change	Methods	Number of Failures	Episodes for Convergence
0.1kg to 0.15kg	Always reinforcement learning	3549	21448
	Self-reflective method	0	0
0.15kg to 0.2kg	Always reinforcement learning	4356	25633
	Self-reflective method	0	0
0.2kg to 0.25kg	Always reinforcement learning	5013	28702
	Self-reflective method	3208	20336
0.25kg to 0.3kg	Always reinforcement learning	5482	30000
	Self-reflective method	0	0

2.6.6 Experimental Approach Stability Analysis

In order to experimentally analyze the stability of our method, we conduct one more simulation with the following scenario. After the self-reflective method adapts to the 0.25kg object, we test the five different object weights (0.1kg to 0.3kg) again on the newly learned policy. The corresponding reflection necessity degree values are 0.95, 0.48, 0.25, 0.05, 0.19, respectively. According to the default threshold α_{th}=0.5, only the 0.1kg object is seen as the new situation. This

simulation experimentally shows that our method can be stable in dealing with new situations. In addition, it is interesting to note that a same weight change in the weight decreasing process has a larger necessity degree value than that in the weight increasing process. For example, the necessity degree value with the weight change from 0.25kg to 0.2kg is 0.06 larger than that with the change from 0.25kg to 0.3kg. This may indicate the weight decreasing situations can be more likely to be considered as new situations than the weight increasing situations. One possible reason is that the policy designed for the old, heavy objects could be too strong for the new, light objects, resulting in larger deviations in (2.1).

2.7 Discussions

Although our strategy uses reinforcement learning as the second step to ensure the system's adaptability to different situations, it more importantly improves the system's safety during the adaptation, which is important for practical safety-critical systems and results in significant differences between our strategy and traditional reinforcement learning methods [15]. The learned policies of traditional reinforcement learning based methods usually do not work for a new situation unless with a re-learning process. It is possible to keep reinforcement learning all the time to adapt to new situations, but the safety of adaptions may be compromised because introducing more learnings may lead to more failures due to possibly unavoidable unsafe explorations and intolerably high computational costs. Similarly, it is sometimes not safe to find a theoretically optimal policy for a task because the extra learnings from a sub-optimal policy that can already finish a task but may sacrifice some task performances to the theoretically optimal policy may also introduce more failures. With considering the safety from the whole system's perspective, our strategy can finish the task more safely by smartly avoiding the unnecessary and unbeneficial learnings with using self-detection (step one) and self-termination (step three),

One limitation of our work is that our strategy cannot guarantee zero failures because it still needs reinforcement learning to learn a new policy if the situation has a significant change. Considering that safe reinforcement learning methods [2] can reduce failures in the learning process, our strategy may be further improved by complementarily using safe reinforcement learning methods in the second step. In addition, introducing safe reinforcement learning may also relax the self-termination criteria to find a policy that is both theoretically optimal and safe. The combination of our strategy and safe reinforcement learning could be a promising way to achieve the persistent autonomy.

Another limitation of our work is the use of heuristic criteria for the self-detection and self-termination. This can require specific engineering knowledge for different aerial manipulators, although the criteria once designed can be used for a long time for different tasks and environments. This could be improved by automatically formulating different criteria using intrinsically-motivated learning [16]. Furthermore, some criteria that can be used for many situations may be stored in a memory so that it can be convenient to directly use them when the system encounters a similar situation in the future.

2.8 Conclusion

In this paper, a novel self-reflective learning strategy is developed for aerial manipulators to pursue persistent autonomy with mainly solving the problem of a smart and safe adaptation to new situations. While using reinforcement learning to achieve good self-adaptations, our method, more importantly, improves the system safety during the adaptations by determining the timing of triggering and terminating learning to reduce unnecessary and unbeneficial learnings. Numerical simulations using an aerial manipulator to carry different loads confirm the effectiveness of our method. In the future, we plan to apply our strategy in the physical aerial manipulator platform to

get further validations. We also expect to replace the heuristic self-reflective criteria with more sophisticated criteria that may be changed online.

2.9 References

[1] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[2] J. Garcıa and F. Fernandez, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[3] A. M. Grant, "Rethinking psychological mindedness: Metacognition, self-reflection, and insight," *Behaviour Change*, vol. 18, no. 1, pp. 8-17, 2001.

[4] M. D. Berzonsky and K. Luyckx, "Identity Styles, Self-Reflective Cognition, and Identity Processes: A Study of Adaptive and Maladaptive Dimensions of Self-Analysis," *An International Journal of Theory and Research*, vol. 8, no. 3, pp. 205-219, 2008.

[5] V. Lippiello and F. Ruggiero, "Cartesian impedance control of a UAV with a robotic arm," in *10th International IFAC Symposium on Robot Control*, Dubrovnik, Croatia, September 5-7, 2012.

[6] M. Orsag, C. Korpela, and P. Oh, "Modeling and control of mm-uav: mobile manipulating unmanned aerial vehicle," *Journal of Intelligent and Robotic Systems*, vol. 69, pp. 227–240, 2013.

[7] P. Geibel and F. Wysotzki, "Risk-sensitive reinforcement learning applied to control under constraints," *Journal of Artificial Intelligence Research*, vol. 24, pp. 81–108, 2005.

[8] T. M. Moldovan and P. Abbeel, "Safe exploration in markov decision processes," in *Proceedings of the 29th International Conference on International Conference on Machine Learning (ICML)*, pp. 1451-1458, 2012.

[9] H. Kashima, "Risk-sensitive learning via minimization of empirical conditional value-at-risk," *IEICE TRANSACTIONS on Information and Systems*, vol. 90, no. 12, pp. 2043–2052, 2007.

[10] K. Driessens and S. Dˇzeroski, "Integrating guidance into relational reinforcement learning," *Machine Learning*, vol. 57, no. 3, pp. 271–304, 2004.

[11] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010.

[12] J. Garcia and F. Fernandez, "Safe exploration of state and action spaces in reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 45, pp. 515-564, 2012.

[13] C. Gehring and D. Precup, "Smart exploration in reinforcement learning using absolute temporal difference errors," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems (AAMAS)*, pp. 1037-1044, 2013.

[14] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*, 1st edition, Boca Raton, FL, USA: CRC Press, Inc., 2010.

[15] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096-2103, 2017.

[16] G. Baldassarre and M. Mirolli. *Intrinsically motivated learning in natural and artificial system*, Springer, Germany, 2013.

CHAPTER 3

VALUE OF LEARNING: WHEN TO LEARN AND WHEN NOT IN FAILURE-AWARE LEARNING-BASED AUTONOMY

To be submitted to *IEEE International Conference on Robotics and Automation (ICRA) 2020*

Xu Zhou[5], Jiucai Zhang[6], and Xiaoli Zhang[7]

3.1 Abstract

Reinforcement learning holds the promise for long-term autonomy because it can adapt to dynamic or unknown environments by automatically learning optimal policies from the interactions between robots and environments. However, the learning process itself could experience failures that are catastrophic to safety-critical systems. Thus, the practical deployment of reinforcement learning raises a fundamental but unanswered question: when to learn and when not to achieve a low failure risk in the long-term autonomy? An early learning could potentially increase failures due to unnecessary and unsafe explorations in the learning process; a late learning may not avoid a potential failure caused by a substantial performance drop. To determine the best timing of learning with minimal failures, we develop a Value of Learning (VOL) approach that maximizes the difference between the learning gains and learning costs. While the learning gains indicate the performance improvements with respect to time, the learning costs measure the learning-induced failure risks, which are related to the level of environment changes and exploration strategies within learning mechanisms. A real-world block stacking experiment

[5] Primary researcher and author, graduate student, Department of Mechanical Engineering, Colorado School of Mines, 1610 Illinois Street, Golden, Colorado, 80401, USA.
[6] Co-author, Guangzhou Automotive Group R&D Center Silicon Valley Inc, Sunnyvale, California, 94085, USA.
[7] Corresponding author, Associate Professor, Department of Mechanical Engineering, Colorado School of Mines, 1610 Illinois Street, Golden, Colorado, 80401, USA.

verifies that our VOL approach can effectively reduce failures compared with “always learning” or “no learning”, which are two representative cases of the early or late learning.

3.2 Introduction

Reinforcement learning has been widely used for robot autonomy including decision making and task control in the past decades. By keeping learning from the interactions between a robot agent and its surrounding environment, reinforcement learning can autonomously find an optimal policy for a new environment such as exploring and finding an optimal trajectory in an unseen scenario. This capability of autonomously handling new tasks or environments has great potentials to enable long-term autonomy also named persistent autonomy, in which the robot can keep safe operations for a long time with minimal human interventions and self-adaptations to dynamic, uncertain environments.

However, the learning safety has been overlooked for a long time due to two possible reasons: (1) the testing robots are assumed not safety-critical or the working environment does not cause lethal damages to robots or to task contexts, and (2) some reinforcement learning algorithms can ensure robot safety by assuming complete safe state and environment knowledge at the training time. However, these assumptions are invalid in most practical applications. For example, as shown in Figure 3.1, the robot tries to stack blocks in the target position in a specific order, which is unknown to the robot beforehand. Although the four positions where the blocks can stay are fixed, which block in which position is also changing and unknown to the robot. Without these prior known knowledge, the reinforcement-learning-based robot needs to try all the possibilities to find out how to stack blocks. During the trials, however, the robot may experience inappropriate stacking sequences, which makes the blocks fall over and damaged after a certain number of times. Thus, in addition to possible benefits, learning can also have failure costs.

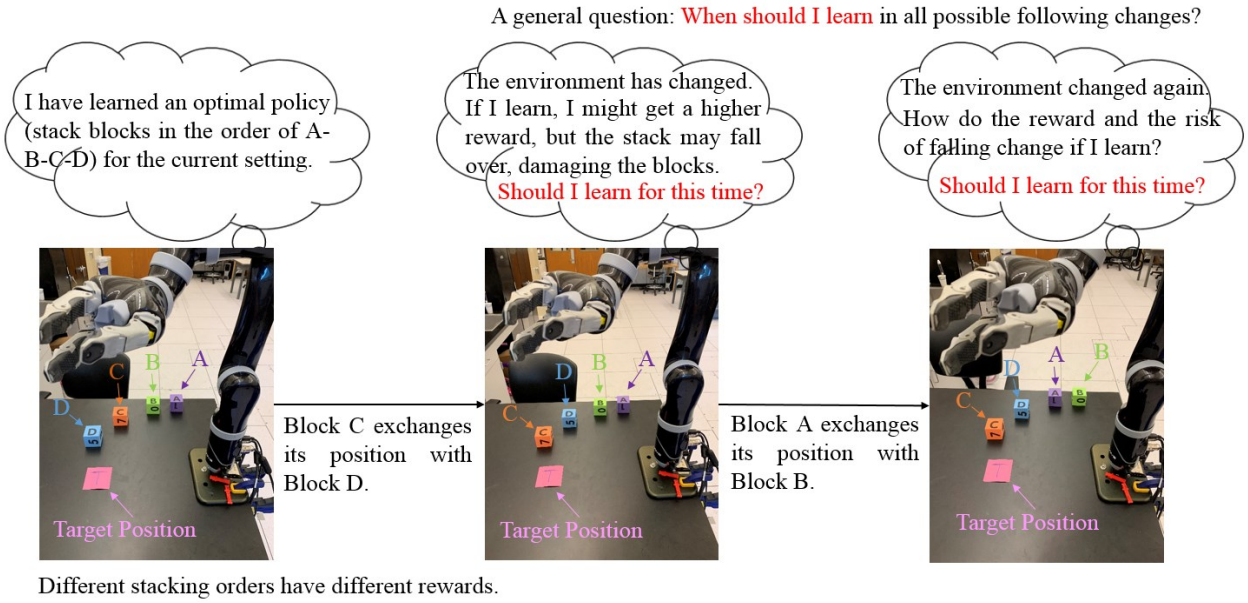

Figure 3.1 When should the robot learn in a dynamically changing environment? The robot needs to stack blocks in a prior unknown order by trial and error, but some trials may cause blocks to fall over and the blocks can be damaged after a certain number of times. Hence, in addition to the gains, the learning also has costs of experiencing failures. Both the learning gains and costs should be considered to decide when the robot should learn.

Given no reinforcement learning is failure-free, the practical deployment of reinforcement learning on safety-critical robots or tasks generally raises a fundamental but unanswered question: ***When to learn a new policy and when not (i.e., just act with the current policy)?*** If the current policy can still work with an acceptable performance, an early decision to learn a new policy may endanger the safety of robots or environments due to possible unsafe explorations during learning. In other words, the learning gain of improving the task performance is not significant; however, the learning cost of resulting in failures is high. When the performance keeps decreasing and the potential learning gains (performance improvements) become significantly greater than the learning costs, learning a new policy may be preferred. A late decision after such a timing may not be able to avoid a potential failure due to a substantial performance drop. Thus, it could be ideal

for reinforcement learning to be activated when the net gain, the difference between learning gains and learning costs, is maximal.

In order to determine when a new round of reinforcement learning would be needed for a safer, longer autonomy, we develop a Value of Learning (VOL) approach which maximizes the learning net gain, i.e., the difference between the learning gains and learning costs. More specifically, we introduce and investigate two indicators: a learning gain indicator and a learning cost indicator. The learning gain indicator considers possible performance improvements while the learning cost indicator is related to the learning's own exploration mechanisms and how the environment changes. Three learning strategies are proposed to combine the learning gain and cost together for a decision of whether or when to learn. Our main contributions are: (1) Develop a failure-aware learning approach that decides the best timing of learning for robot's persistent autonomy. In addition to potential performance improvements, this approach also takes into account the potential failure risks induced by learning. Three different learning strategies including the optimistic, realistic, and pessimistic strategy are used to evaluate both the learning gain and the learning cost, and the influence of their combinations on the decision to learn. (2) Validate our approach with a real-world experiment of stacking blocks in a prior-unknown order. The block positions are also changeable so that a learned policy cannot work all the time. The results verify that our approach can effectively reduce failures and hence improves the operational safety of learning-based autonomy.

3.3 Related Work

Reinforcement learning has been used in a wide variety of autonomous problems, ranging from task allocation [1] and path planning [2] to manipulation control [3]. There are two general methods: model-based methods and model-free methods. Although model-based methods [4] are

generally known to be more sample-efficient, model-free methods [5] have been particularly popular due to their simplicity and favorable computational properties. In this paper, we adopt a more easily implemented model-free method to investigate the essential problem about when to learn in long-term autonomy. However, our work could also be generalized to model-based reinforcement learning methods because the exploration mechanism which we are interested in can be the same in different reinforcement learning methods.

One main challenge to achieve long-term autonomy is that the systems must be capable of dealing with dynamic changes such as environmental changes or the incremental variations in their own performance capabilities. This is difficult for reinforcement learning because the agent cannot stay informed about the status of all dynamic objects in the environment. A promising solution is to take dynamic changes into account when determining a policy. By explicitly detecting dynamic changes [6][7], methods such as using a temporal model [8], instantiated information [9], and experience replay [10] have been reported to achieve good results in robotic navigation. While these works have improved the adaptability of autonomous robots to dynamic changes, they overlooked the learning-induced failure risk during the practical deployment of reinforcement learning.

With regard to safety and/or risks during the learning and/or deployment process [11], two fundamental tendencies have been reported in past decades. The first one consists of transforming the traditional optimization criteria that maximizes the expectation of the return to more comprehensive criteria respecting learning safety. For example, the worst-case criterion used the maximum worst-case return to mitigate the effects of the variability which can result in risk or unsafe situations. By defining the risk as the variance of the return [12] or the probability of entering into an error state [13], the risk-sensitive criterion introduced a subjective measure like a

linear combination to balance the return and the risk. With one or more constraints, the constrained optimization criterion maximized the return while keeping other types of expected measures higher or lower than some given bound [14][15]. The use of other optimization criteria in financial engineering, such as value-at-risk (VaR) [16][17][18], or the density of the return [19][20], were also reported. Since exploratory actions could have serious consequences, the second tendency improved the safety with modifying the exploration process in two ways: (1) through the incorporation of external knowledge such as providing initial knowledge [21], deriving a policy from a finite set of demonstrations [22][23], and providing teach advice [24][25][26], and (2) through the use of a risk-directed exploration metric [27][28]. Although our work could share a similar aim of improving learning safety with these works, we provide a more general perspective for the safety in long-term autonomy. More specifically, our work focuses on improving the operational safety through deciding the necessity and timing of learning, instead of assuming that learning is always necessary in previous works.

3.4 Preliminaries

Our VOL approach could be generalizable to different value-based learning algorithms as long as they can fit the indicator definitions of learning gains and costs discussed later. For simplicity, we adopt a standard reinforcement learning algorithm, ε-greedy Q learning [29], as our basic learning strategy. A reinforcement learning problem can be formulated as a Markov Decision Problem (MDP) [29], specified by the 5-tuple $(S, A, \{P_{sa}\}, r, \gamma)$. At each discrete time step $t \in \{0,1,2,3, \dots\}$ the robot is in a certain state $s_t \in S$. After the selection of an action, $a_t \in A$, the agent receives a reward signal from the environment, $r_{t+1} \in \mathbb{R}$, and passes into a successor state s'. The decision which action a is chosen in a certain state is characterized by a policy $a = \pi(s)$, which could also be stochastic $\pi(a|s) = P_{sa}\{a_t = a|s_t = s\}$. The cumulative reward R_t for any π by

starting in state s and selecting action a can be denoted as $Q^{\pi}(s,a) = E_{\pi}\{R_t|s_t = s, a_t = a\} = E_{\pi}\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s, a_t = a\}$, where γ is a discount factor on rewards. A policy that maximizes the cumulative reward is denoted as π^*.

A single-state MDP with n different actions is considered where each single action is associated with a stochastic reward distribution. After the selection of action a_t, the environment responds with the reward signal, r_{t+1}, by which the mean reward of action a can be estimated by

$$Q_{t+1}(s_t, a_t) \leftarrow (1-\alpha)Q_t(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1}, a)), \tag{2.1}$$

where α is the learning rate deciding how much the system trusts the new information. With the ε-greedy exploration strategy, the robot selects at each time step a random action with a fixed probability, $0 \leq \varepsilon \leq 1$, instead of greedily selecting one of the learned optimal actions with respect to the Q-function:

$$\pi(s) = \begin{cases} \text{random action from } A & \text{if } \xi < \varepsilon \\ \underset{a \in A}{\text{argmax}}\, Q(s_{t+1}, a) & \text{otherwise,} \end{cases} \tag{2.2}$$

where $0 \leq \xi \leq 1$ is a uniform random number drawn at each time step.

3.5 Methodology

As shown in Figure 3.1, our VOL approach is a high-level learning mechanism, which consists of estimating an unknown environment, computing learning gains and learning costs, determining whether to learn or when to learn, and checking failures of the decision to learn or not. As our approach does not change anything inside the reinforcement learning, it is easy for our approach to work with general reinforcement learning algorithms well.

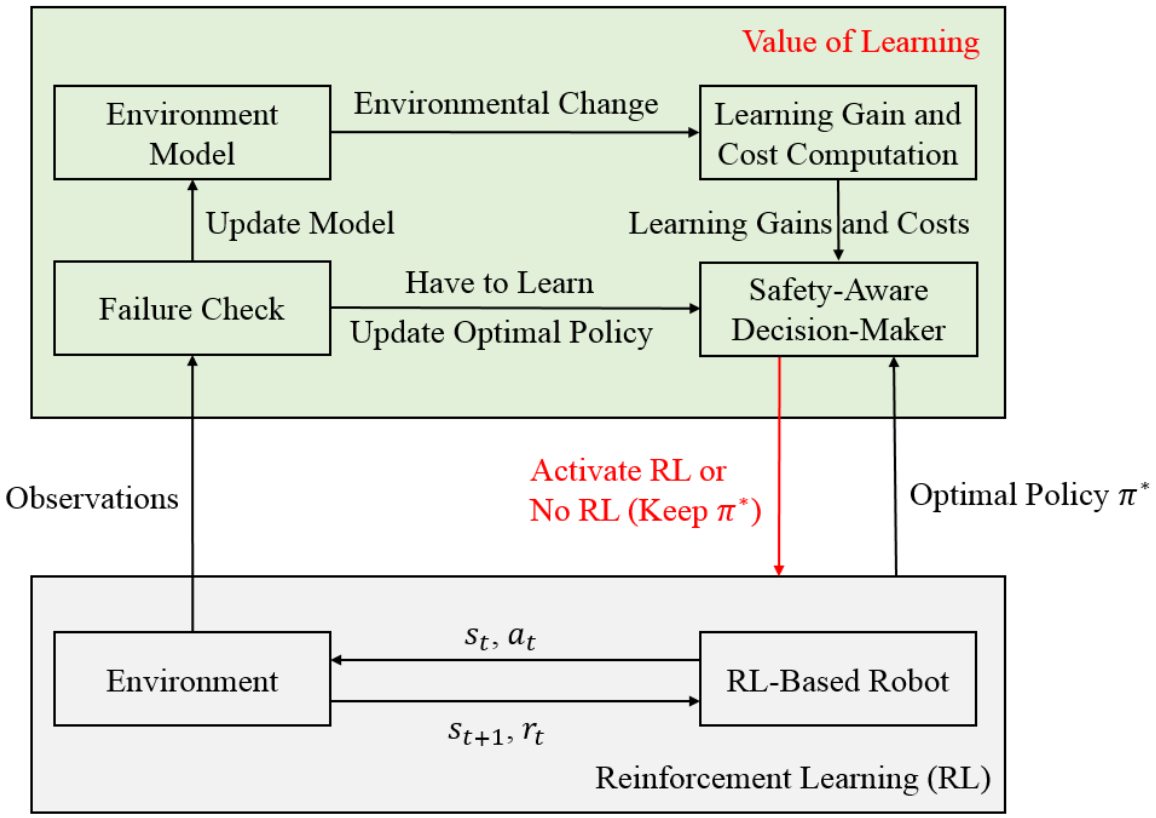

Figure 3.2 Value of Learning method. Given the prediction of the environmental change from an online learned environment model, both learning gains and learning costs are calculated with respect to the current optimal policy. Based on the learning gains and costs, the personalized decision-maker uses three different strategies including an optimistic, realistic, and pessimistic strategy to decide if reinforcement learning should be activated to find a new policy. If the current policy is kept but a failure exists, reinforcement learning must be activated.

3.5.1 Environment Model

Because both learning gains and learning costs are related to dynamic environmental changes, it is necessary for us to build an environment model so that we can predict the changes in the environment. It could be impossible for us to predict irregular changes, so we assume that the environmental change has a pattern in this work. To be more specific, the environment can have a constant change probability p_e, but this probability is unknown to the robot. Thus, building the environment model means estimating the change probability p_e.

A simple method is to use maximum likelihood estimation (MLE) [30]. MLE can estimate p_e based on the principle that if we observe the training data D, we should choose the value of p_e that makes D most probable. Therefore, the estimation of p_e in general is

$$\hat{p}_e^{MLE} = \underset{p_e}{\operatorname{argmax}} P(D|p_e) = \frac{n_c}{n_c+n_{uc}}, \tag{2.3}$$

where n_c means the number of changed environments and n_{uc} means the number of unchanged environments. The intuition underlying this principle is simple: we are more likely to observe data D if we are in a world where the appearance of this data is highly probable. Therefore, we should estimate p_e by assigning it whatever value maximizes the probability of having observed D.

Because the environment is sometimes much more complicated than obeying a fixed change probability, a more general method can be using neural networks. We can consider the changeable part in the environment as an opponent's behavior to the robot. This behavior such as the state of the opponent can be estimated by using neural networks. To learn an approximation of the environmental changes s_t^e which can be observed after the robot executes its action at time t, we use a neural network μ_t parameterized by θ_t which we optimize by minimizing the loss:

$$L(\theta_t) = (\mu_t(s_{t-1}^e, s_{t-1}, a_{t-1}) - s_t^e)^2. \tag{2.4}$$

The training samples are from the replay of the robot's past experiences $(s_{t-1}, a_{t-1}, s_{t-1}^e, s_{t-2}, a_{t-2}, s_{t-2}^e, \dots, s_{t-K}, a_{t-K}, s_{t-K}^e)$, where K indicates the length of the experience replay. The neural network model can theoretically predict any changes in the environment. However, since we assume a constant change in the environment for simplicity in this work, we need to further estimate p_e using μ_t. This can be also achieved by counting the number of changes in $(s_t^e, s_{t-1}^e, s_{t-2}^e, \dots, s_{t-K}^e)$ with MLE. In addition, we can speed up the estimation of p_e by randomly simulating more samples using the past experience

$(s^e_{t-1}, s^e_{t-2}, \dots, s^e_{t-K})$. The robot's states and actions are removed because the environment change is independent of them in our setting.

3.5.2 Learning Gain Indicator

Our VOL requires the capability of being aware that a better policy exists, thus we propose the gain indicator (I^t_G), a metric that quantifies the performance improvements of a policy with respect to time. The performance measures can be various and user-defined, but a general one is the reward of a policy because each policy can be evaluated by the reward. The performance improvement is then the difference between the reward of a new policy and that of the old policy. Given that the environment model is probabilistic, the learning gain is actually a probability-based value too. In this work, we define three types of I^t_G as follows.

Optimistic: $I^t_G = \max_m(p_e R^{new}_{t,m} + (1-p_e)R^{old}_{t,m}) - R^{old}_{t,m} = \max_m p_e(R^{new}_{t,m} - R^{old}_{t,m})$.

Realistic: $I^t_G = \frac{1}{M}\sum_m^M\left(p_e R^{new}_{t,m} + (1-p_e)R^{old}_{t,m}\right) - R^{old}_{t,m} = p_e\frac{1}{M}\sum_m^M(R^{new}_{t,m} - R^{old}_{t,m})$.

Pessimistic: $I^t_G = \min_m(p_e R^{new}_{t,m} + (1-p_e)R^{old}_{t,m}) - R^{old}_{t,m} = \min_m p_e(R^{new}_{t,m} - R^{old}_{t,m})$.

In the above three equations, M means the number of the opponent's all possible actions, m is one index of M, which means taking mth action and results in a specific environment. $R^{new}_{t,m}$ means the reward of a new, optimal policy in the environment after the opponent takes mth action, and $R^{old}_{t,m}$ means the reward of the current, old policy in the environment after the opponent takes mth action.

The optimistic learning gain takes the maximal improvement among all possible environmental changes, indicating that the robot tends to take a risky chance to achieve an upper bound. In contrast, the pessimistic learning gain takes the minimal improvement among all possible environmental changes, indicating that the robot tends to always guarantee a learning gain

if it decides to learn. The realistic learning gain is between the optimistic and pessimistic one, taking the expected improvement in all possible environment changes.

3.5.3 Learning Cost Indicator

The cost indicator (I_C^t) is a metric that quantifies the risks of failures if activating reinforcement learning. In order to be consistent with the learning gain indicator, the learning cost indicator is also measured by the reward like the learning gain. However, this reward is negative because it is actually a punishment of a failure. For reinforcement learning in dynamic environments, the failures can be caused by two cases. One case is that the random action selected during the learning phase results in a failure and the other case is that the action following the current, optimal policy results in a failure. Thus, similar to the learning gain indicator, we also define three types of I_C^t below.

Optimistic: $I_C^t = (\frac{|a_d|}{|A|}\varepsilon + \min_m p_e\, \delta(\pi_m^*(s) \in a_d))r_f$.

Realistic: $I_C^t = (\frac{|a_d|}{|A|}\varepsilon + p_e \frac{1}{M}\sum_m^M \delta(\pi_m^*(s) \in a_d))r_f$.

Pessimistic: $I_C^t = (\frac{|a_d|}{|A|}\varepsilon + \max_m p_e\, \delta(\pi_m^*(s) \in a_d))r_f$.

In the above three definitions, a_d means a dangerous action which will result in a failure, which can be easily determined based on the current, optimal policy. $|\cdot|$ means the dimension of a vector space. $\delta(\pi_m^*(s) \in a_d)$ is a binary function. $\delta(\pi_m^*(s) \in a_d) = 1$, if the action following the optimal policy π^* is a dangerous action, otherwise $\delta(\pi_m^*(s) \in a_d) = 0$. r_f is the negative reward of a failure.

The optimistic learning cost namely considers the minimal failure reward only so that using this type prefers learning more. In contrast, the pessimistic learning cost considers the maximal

failure reward so that this type prefers learning as least as possible. The realistic learning cost is a comprehensive evaluation of learning costs in all the environmental changes.

3.5.4 Value of Learning

As the VOL approach decides the timing of learning at the maximum learning net gain, the timing can be mathematically obtained by

$$t = \underset{t}{\operatorname{argmax}}\, I_G^t + I_C^t, \forall\, I_G^t + I_C^t \geq \epsilon, \tag{2.5}$$

where ϵ is an empirically defined safety threshold, which indicates the user's allowance of the minimal safety level. A large ϵ represents a conservative learning decision mechanism because such a decision to learn requires a significant difference between the learning gains and costs. The decision mechanism can be interpreted from two perspectives. For each specific moment, there is only a one-time change in the environment so that the mechanism actually decides whether to learn for once. However, if the change is time-continuous, the mechanism decides the best timing of learning. After re-learning, the old policy will be automatically replaced by the new policy. Then, this new policy will be regarded as an old policy for the subsequent moments in the remaining period. If the learning gains outweigh the learning costs, then the learning will be activated again. Thus, multiple learnings could exist in a time-continuous period.

3.5.5 Failure Check

If VOL decides to not learn, a failure check process is used to verify if any failure can occur according to the current policy. If a failure occurs, the environment model could be inaccurate so that the failure check process passes the current observations to the environment model for further improvements. In the meantime, the failure check process mandatorily activates reinforcement learning to find a new, optimal policy to replace the current, problematic policy. If VOL decides

to learn, then the failure check process only sends a positive signal to reinforce the environment model because the policy after reinforcement learning can never result in a failure.

3.6 Results

3.6.1 Experiment Setup

In this work, we use a real-world block stacking experiment as shown in Figure 3.1 to validate our approach. This experiment can been seen as a simplified scenario of an automatic industrial assembly task. In the assembling, the robot must follow a specific order or multiple orders that are acceptable with different levels. If not following these orders, the assembly task fails and the parts for assembling may be damaged. In practical assembly tasks, the parts for assembling may be stored at pre-planned places. These places are usually fixed, but which part in which place can be changeable and unknown until it is taken out for assembling.

Thus, in our experiment, the robot is supposed to pick four different blocks, block A (purple), block B (green), block C (orange), and block D (blue), from their original, pre-defined positions (position 1, 2, 3, and 4, respectively) and put them on a target position. Different stacking orders have different rewards. To more be specific, stacking order A-B-C-D (from the bottom to the top) has a reward of 1. Stacking order A-B-D-C, A-C-B-D, and A-D-C-B have a reward of 0.8, 0.6, and 0.4, respectively. We assume that block A must be at the bottom, otherwise the stack falls over and results in a failure. The reward of a failure is -1. All other stacking orders and individual movements during the stacking have a reward of 0. The robot can know these rewards only after exploring them by itself. We also assume the block positions vary at each time step with a constant probability p_e. For example, if $p_e = 0.1$, then block A may has 10% chance to exchange its position with block B at $t = 1$ and block C may have 10% chance to exchange its position with block D at $t = 2$. Because an accurate grasping is out of our scope, we assume all the pre-defined

positions to be fixed, which reduces the difficulty of grasping and removes unexpected influences of an inaccurate grasping. However, which specific block in which specific position is unknown to the robot beforehand. The discount factor of the ε-greedy Q-learning is set to be 0.99. The learning rate α is set to be 0.2 and the safety threshold ϵ is set to be 0.

In general, we want to answer two questions from our experiment: (1) what is the performance of our proposed method? and (2) what is the relationship between the decision to learn or not and the environment change probability or random exploration probability. In order to answer the first question, we allow the environment to change at every time step and the total number of changes is set to be 100. At each time step, the robot needs to decide whether to learn or not. For the second question, we assume the robot already knows the environment change probability, which could be achieved by allowing enough time for the environment estimation. Without the influence from the environment change probability, we vary both p and ε from [0,1] with a 0.1 increment.

3.6.2 Environment Model

Figure 3.3 shows the estimation of the environment change probability p_e using MLE method and neural network (NN) method. All methods can be considered as converged to 0.3 around 40 iterations (episodes). However, the MLE method is much more inaccurate at the beginning (from episode 1 to episode 20). In contrast, NN can quickly approach 0.3 using much less episodes (about 5-10 episodes) because NN can simulate K more samples to speed up the estimation. When K is 4, NN uses only 4 episodes to get the estimation close to 0.3, which could be sufficient for a real-time implementation.

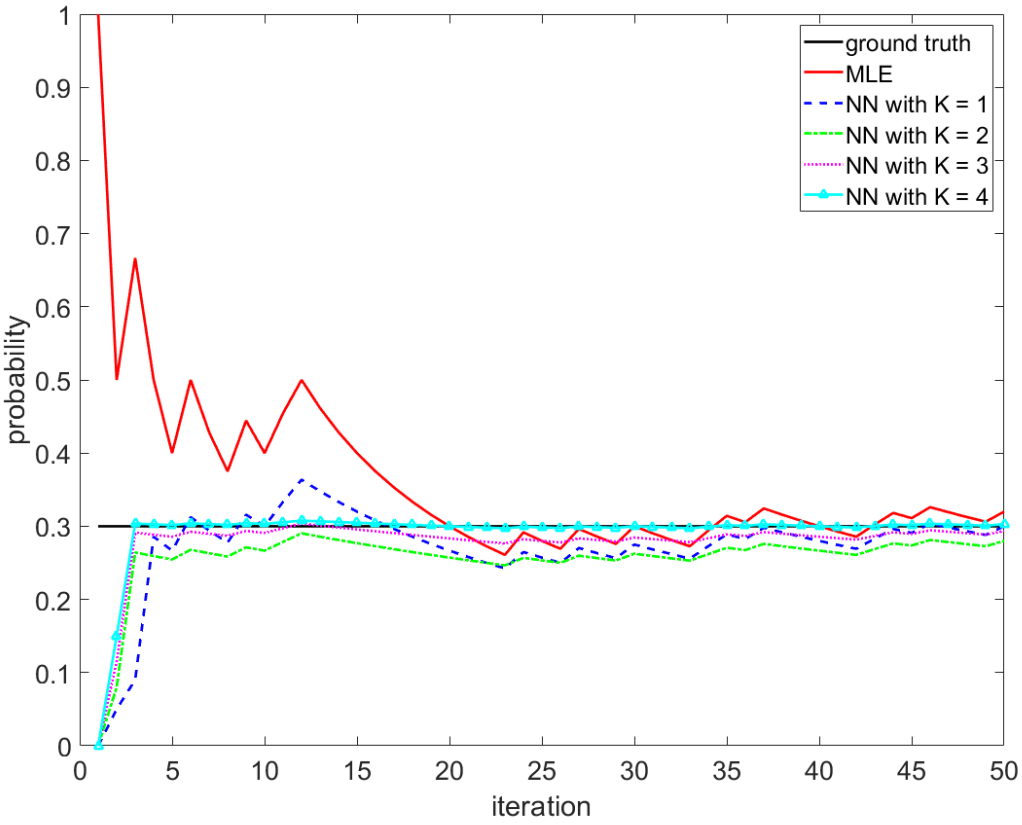

Figure 3.3 Estimation of the environment change probability using different methods. All the methods can converge to the ground truth within 40 iterations. The NN method performs best with a fast convergence speed with using only 4 iterations.

3.6.3 Value of Learning Performances

Figure 3.4 shows the learning gains, costs, and decisions to learn or not at each time step for the 100-time-step experiment with continuous changes. The left figure uses MLE to estimate the environment change probability while the right figure uses NN with K=4 to do the estimation. Because the reward is fixed, the learning gains and costs are mostly stable once the environment change probability is estimated. Due to the oscillations of MLE, the decisions also fluctuate. In most cases where there is no need to learn, MLE-based VOL also decides to learn. The comparison indicates the importance of a good environment model to VOL.

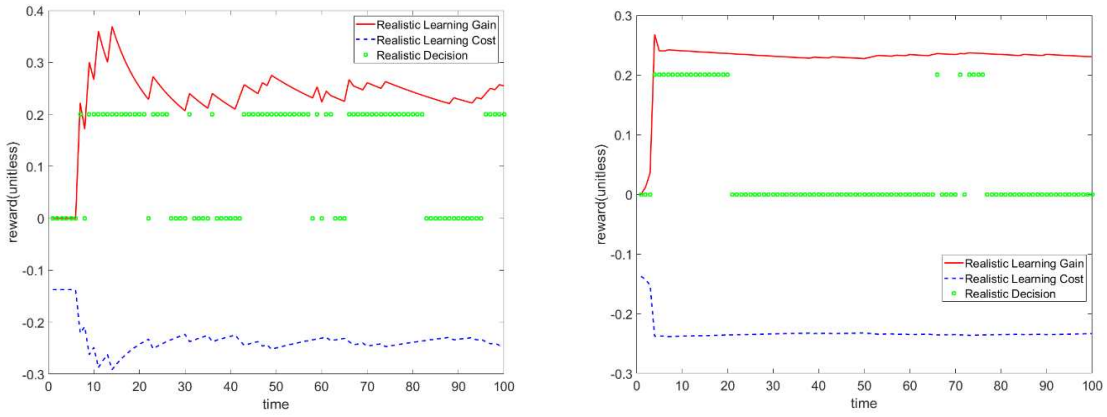

Figure 3.4 Comparison of learning gain, cost, and decisions to learn between MLE-based VOL (left) and NN-based VOL (right).

Table 3.1 shows the complete results, including the number of decisions to learn and the number of failures, for different learning strategies. Because our environment model is a probabilistic model, we repeat the 100-time-step experiment 20 times to get a statistical result. We also add the always learning and no learning for comparison. More specifically, always learning namely means always activating reinforcement learning for each time step no matter the environment actually changes or not. No learning indicates that the robot never learns until it encounters a failure and has to use standard reinforcement learning to find a new policy. The decision accuracy is the percentage of decisions that are consistent with human labeled decisions.

From Table 3.1, always learning has the lowest decision accuracy and the largest number of failures because there is no need to learn in most cases. Because MLE estimates the environment in a relatively slow speed, all MLE-based methods prefer to learn more than all NN-based methods, which is problematic. Due to the more learnings, the number of failures is also larger. Although all NN-based methods have a relatively lower decision accuracy than no learning, they experience

much fewer failures because our approach performs much better in the cases where learning is necessary. In such cases, because no learning decides to not learn, it must experience a failure for each change before it restarts learning, which can be totally avoided by our method. If there are more critical changes in the environment, always learning can perform better than no learning. However, our method can still experience fewer failures because our method avoids failures in those cases where learning is not needed.

Table 3.1 Statistical results of VOL performances by using different learning strategies

	Number of Decisions to Learn	Decision Accuracy	Number of Failures
Always Learning	100±0	11%±5%	253±59
No Learning	0±0	88%±6%	56±24
Optimistic Strategy with MLE	68±14	41%±18%	201±46
Optimistic Strategy with NN (K=4)	33±9	74%±10%	48±17
Realistic Strategy with MLE	59±10	54%±13%	183±31
Realistic Strategy with NN (K=4)	26±6	82%±6%	34±8
Pessimistic Strategy with MLE	52±13	56%±15%	185±39
Pessimistic Strategy with NN (K=4)	22±11	84%±11%	33±12

3.6.4 Important Factors for Decision to Learn

Figure 3.5 shows the decisions to learn with respect to different environment change probabilities. If the reward is fixed, we can see both the learning gains and learning costs increase as p_e increases. However, since the optimistic learning strategy always considers the best learning gain, the increasing speed of the optimistic learning gain is much faster than the optimistic learning cost, which results in decisions to learn with respect to all environment change probabilities except $p_e = 0$. If $p_e = 0$, it means the environment does not change. Thus, the learning gain is 0 and the learning cost is a small number only related to the random exploration risks. In contrast to the optimistic learning strategy, the pessimistic learning strategy always considers the worst case,

which results in no learning with respect to all environment change probabilities. The realistic learning strategy should be in the middle of the optimistic and pessimistic learning strategy, but the learning gain is still higher than the learning cost for each environment change in our setting. Thus, the realistic learning strategy also decides to learn for each environment change.

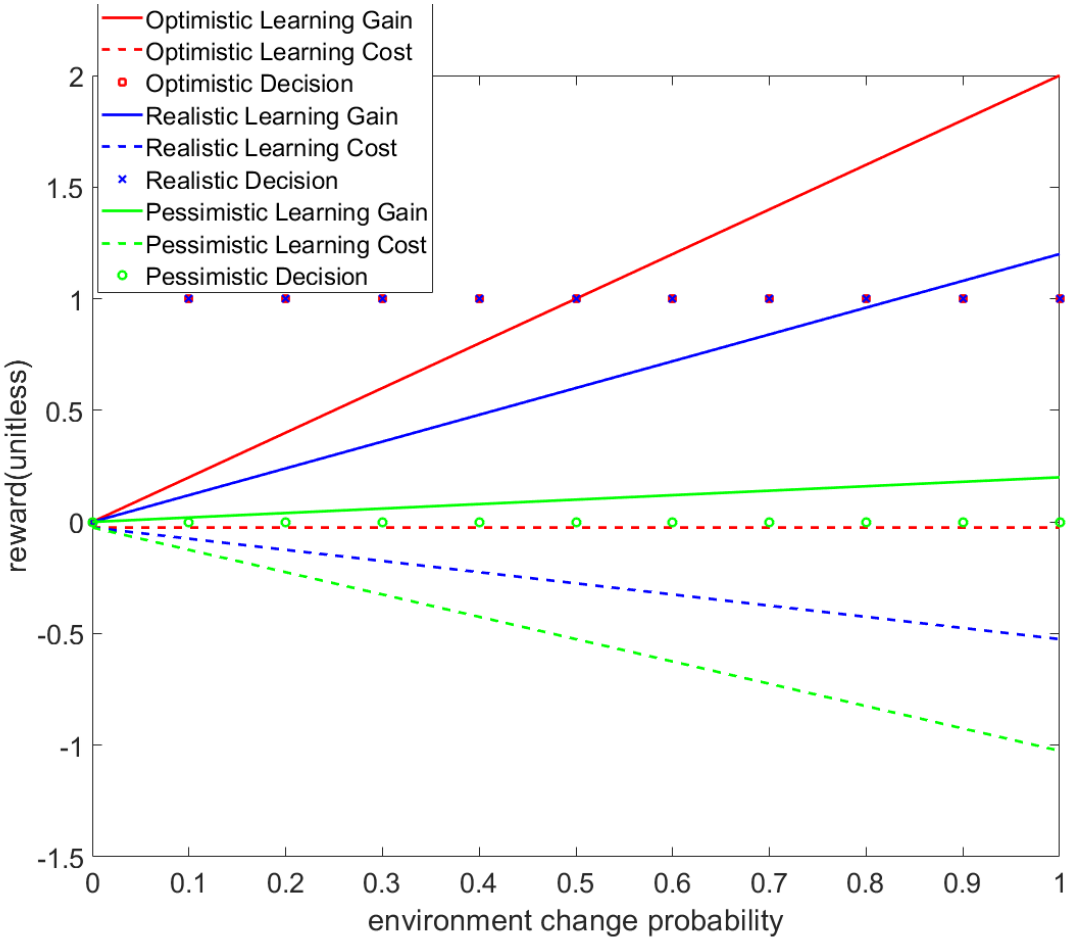

Figure 3.5 Comparison of learning gains, costs, and decisions to learn with respect to different environment change probabilities. The random exploration probability is fixed to be 0.1. Decision = 1 means a decision to learn while decision = 0 means a decision to not learn.

Figure 3.6 shows the decisions to learn with respect to different random exploration probabilities ε. In our experiment setting, the optimal reward will always be 1 and there is always a possibility that block A can be exchanged with other blocks, so the difference between the new

optimal policy and the old one is constantly fixed and hence the learning gain remains the same for each ε. If we set the reward to be changeable in the learning phase, the learning gain can then change. The learning costs increase as ε increases, which means the more the robot tends to randomly explores, the more risky the robot will be. Given the learning gain is a constant, most learnings are not preferred when ε reaches 0.8. The pessimistic learning strategy decides to not learn for all the cases while the optimistic learning strategy decides to learn at the first eight cases with ε from 0 to 0.7.

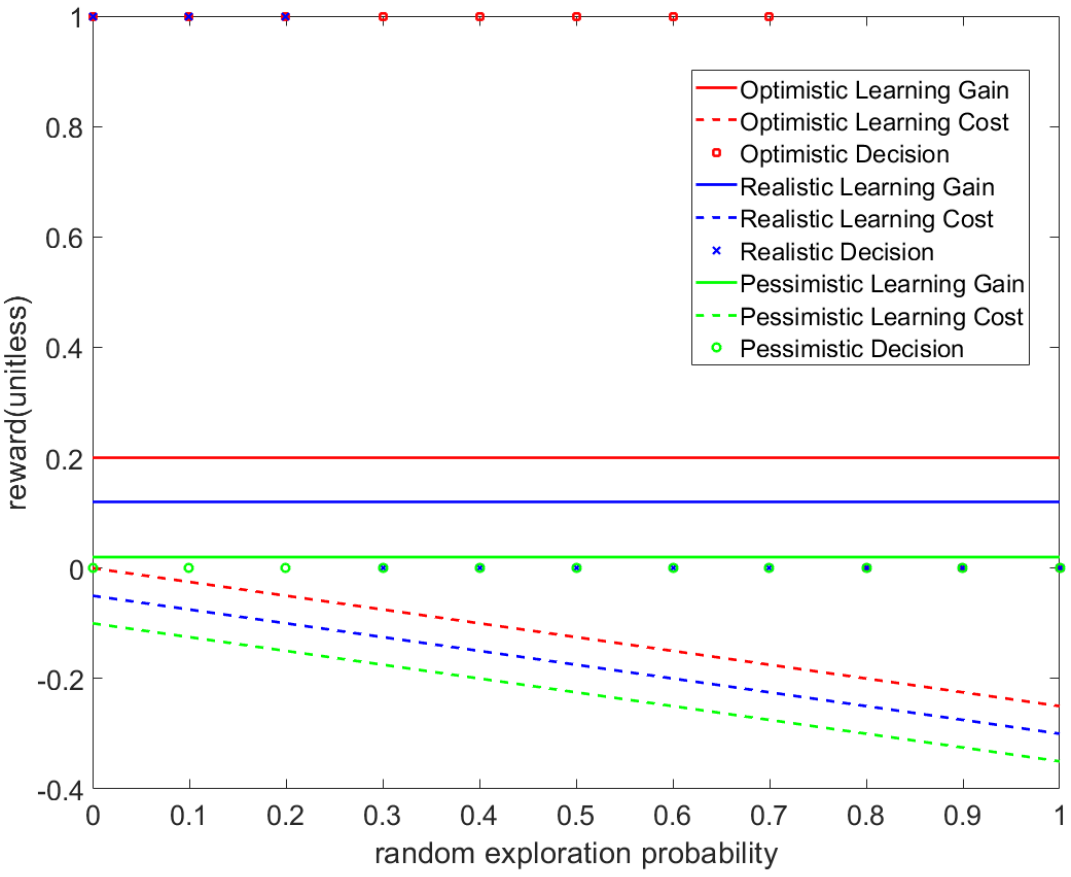

Figure 3.6 Comparison of learning gains, costs, and decisions to learn with respect to different random exploration probabilities. The environment change probability is fixed to be 0.1. Decision = 1 means a decision to learn while decision = 0 means a decision to not learn.

3.7 Discussions

Although our work can be seen as a solution addressing both the learning adaptation and learning safety problems at the same time, it could be more valuable in providing a novel approach to improve the "operational" safety, which further supports the learning-based long-term autonomy. Compared with previous works in improving safety in the learning/training process, we introduce a new perspective to improve the operational safety by deciding when to activate learning. In other words, we believe that not all the learning is necessary and beneficial in the long-term autonomy because learning has not only gains (i.e., performance improvements) but also costs related to potential failure risks. When the learning gains are less than learning costs, deactivating learning instead of keeping learning could improve the operational safety.

Another interpretation of our work can be a high-level exploration-exploitation trade-off from the system's perspective with focusing on the system safety. More specifically, the exploration is a learning or planning itself while the exploitation is a pure execution without a new learning. This trade-off is important in the long-term robot autonomy because the practical systems have more concerns such as safety and the practical availability of theoretic algorithms rather than the problems like the algorithm efficiency only in the algorithm itself. However, it should be noted that some traditional exploration-exploitation strategies might unintentionally improve the system safety due to some specific methodology designs. For example, some adaptive epsilon-greedy strategies such as the epsilon-first strategy [29], epsilon-decreasing strategy [29], and value-difference based exploration (VDBE) strategy [31] moderate epsilon with respect to time or uncertainty. More specifically, the portion of explorations is high at first and gradually decreases to a low value as time increases. In other words, random explorations rarely exist near the end of reinforcement learning. This is safer than keeping fixed random explorations due to more

confidences in the environment. When the environment only has small changes, the adaptive epsilon-greedy strategy could be comparable to or even have better performances than ours, especially in the situation where our method decides to not learn even if the obstacle blocks current path. However, if the environment change is large and frequent, our method should still outperform the adaptive epsilon-greedy strategy because our method can avoid considerable collision risks by deciding to not learn in the situations where changes are not important to robots.

To achieve long-term autonomy, the operational safety could be also maintained by methods that revert to a hand-designed safety controller if a failure is detected. These methods can be generally considered as posterior methods which mainly considers how to recover systems from failures. However, it is not always easy to accurately detect a failure in practical applications. Due to system or environment uncertainties, the detection system may erroneously conclude that a safe environment is unsafe or vice versa. Such inappropriate activations or deactivations of the hand-designed safety controller could easily result in failures. Furthermore, the activation of the hand-designed safety controller after the occurrence of a failure may not recover the system if the failure brings lethal damages to the agent. By contrast, our approach tries to activate learning before the failure happens. We also use a prediction model to estimate the environment changes so that the system can prepare in advance to change to a safer policy.

One limitation of our work is the assumption of a constant environment change probability. In some practical applications, the changes may be much more complex. For example, the probability may also change during different time periods. In order to solve this problem, more advanced environment modeling techniques such as an occupancy grid model [32] could be integrated in our method. Another issue the robot may meet in practice is the influence of noises

and systematic errors. These could be solved by considering them as non-deterministic actions in the MDP.

3.8 Conclusion

In this paper, we propose a VOL approach to address the timing of activating learning in learning-based autonomy. By quantifying the learning gains and learning costs in the dynamic environments, we calculate the net gain of the learning with different strategies. If the net gain is greater than a threshold, then learning should be activated. A block-stacking experiment verifies that our approach can reduce the failures compared to no learning and always learning. In the future, we plan to improve our method by adding noises and stochastics into our MDP and giving more complexities to the environmental changes.

3.9 References

[1] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pp. 50-56, 2016.

[2] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol.32, no. 11, pp. 1238-1274, 2013.

[3] M. P. Deisenroth, G. Neumann, and J. Peters, "A survey on policy search for robotics," *Foundations and Trends in Robotics*, vol. 2, nos. 1–2, pp. 1-142, 2013.

[4] M. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pp. 465-472, 2011.

[5] J. Peters and S. Schaal, "Policy gradient methods for robotics," In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2219-2225, 2006.

[6] P. Dayan and T. J. Sejnowski, "Exploration bonuses and dual control," *Machine Learning*, vol. 25, no. 1, pp. 5-22, 1996.

[7] A. Marinescu, I. Dusparic, A. Taylor, V. Cahill, and S. Clarke, "Decentralised multi-agent reinforcement learning for dynamic and uncertain environments," *CoRR* abs/1409.4561, 2014.

[8] J. A. Boyan and M. A. Littman, "Exact solutions to time-dependent MDPs," in *Advances in Neural Information Processing Systems (NIPS)*, pp. 1026-1032, 2001.

[9] M. A. Wiering, "Reinforcement learning in dynamic environments using instantiated information," in *Proceedings of the Eighteenth International Conference on Machine Learning: (ICML)*, pp. 585-592, 2001.

[10] M. Pieters and M. A. Wiering, "Q-learning with experience replay in a dynamic environment," in *IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1-8, 2016.

[11] J. Garcıa and F. Fernandez, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437-1480, 2015.

[12] M. Sato, H. Kimura, and S. Kobayashi, "Td algorithm for the variance of return and mean-variance reinforcement learning," *Transactions of the Japanese Society for Artificial Intelligence*, vol. 16, no. 3, pp. 353– 362, 2001.

[13] P. Geibel and F. Wysotzki, "Risk-sensitive reinforcement learning applied to control under constraints," *Journal of Artificial Intelligence Research*, vol. 24, pp. 81–108, 2005.

[14] Y. Kadota, M. Kurano, and M. Yasuda, "Discounted Markov decision processes with utility constraints," *Computers and Mathematics with Applications*, vol. 51, no. 2, pp. 279-284, 2006.

[15] T. M. Moldovan and P. Abbeel, "Safe exploration in markov decision processes," in *Proceedings of the 29th International Conference on International Conference on Machine Learning (ICML)*, pp. 1451-1458, 2012.

[16] H. Mausser, and D. Rosen, "Beyond VaR: from measuring risk to managing risk," in *Proceedings of the IEEE/IAFE 1999 Conference on Computational Intelligence for Financial Engineering*, pp. 163-178, 1998.

[17] H. Kashima, "Risk-sensitive learning via minimization of empirical conditional value-at-risk," *IEICE TRANSACTIONS on Information and Systems*, vol. 90, no. 12, pp. 2043–2052, 2007.

[18] D. G. Luenberger. *Investment science*. Oxford University Press, 2013.

[19] T. Morimura, M. Sugiyama, H. Kashima, H. Hachiya, and T. Tanaka, "Nonparametric return distribution approximation for reinforcement learning," in *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pp. 799-806, 2010.

[20] T. Morimura, M. Sugiyama, H. Kashima, H. Hachiya, and T. Tanaka, "Parametric return density estimation for reinforcement learning," in *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, pp. 368-375, 2010.

[21] K. Driessens and S. Dˇzeroski, "Integrating guidance into relational reinforcement learning," *Machine Learning*, vol. 57, no. 3, pp. 271–304, 2004.

[22] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010.

[23] J. Tang, A. Singh, N. Goehausen, and P. Abbeel, "Parameterized maneuver learning for autonomous helicopter flight," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1142– 1148, 2010.

[24] A. L. Thomaz, C. Breazeal, et al., "Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance," in *AAAI*, vol. 6, pp. 1000–1005, 2006.

[25] J. Garcia and F. Fernandez, "Safe exploration of state and action spaces in reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 45, pp. 515-564, 2012.

[26] P. Vidal, R. Iglesias Rodriguez, M. A. Rodriguez Gonzalez, and C. Vazquez Regueiro, "Learning on real robots from experience and simple user feedback," *Journal of Physical Agents*, vol. 7, no. 1, pp. 57-65, 2013.

[27] C. Gehring and D. Precup, "Smart exploration in reinforcement learning using absolute temporal difference errors," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems (AAMAS)*, pp. 1037-1044, 2013.

[28] E. L. Law. *Risk-directed Exploration in Reinforcement Learning*. McGill University, 2005.

[29] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

[30] S. J. Russell and P. Norvig. *Artificial intelligence: A modern approach*. Englewood Cliffs, NJ: Prentice-Hall, 1994.

[31] M. Tokic, "Adaptive ε-greedy exploration in reinforcement learning based on value differences," in *Annual Conference on Artificial Intelligence*, pp. 203-210, 2010.

[32] D. Meyer-Delius, M. Beinhofer, and W. Burgard, "Occupancy grid models for robot mapping in changing environments," In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

CHAPTER 4

A SELF-RECOVERABLE RESET STRATEGY FOR REINFORCEMENT-LEARNING-BASED PERSISTENT AUTONOMY

To be submitted to *IEEE International Conference on Robotics and Automation (ICRA)* 2020

Xu Zhou[8] and Xiaoli Zhang[9]

4.1 Abstract

Reinforcement learning holds the promise for persistent autonomy because it can adapt to dynamic or unknown environments by automatically learning optimal plans from the interactions between robots and environments. However, the practical deployment of many reinforcement learning algorithms often requires manually resetting the state of the system between training episodes when a failure occurs. Because these manual resets could be unavailable in practice, we propose a self-recoverable reset strategy to avoid such failure-induced resets. This strategy consists of a multi-state recovery strategy and a failure prevention strategy. The multi-state recovery strategy provides robots with the capability of recovering from failures by self-correcting its behavior in the problematic state and, more importantly, deciding which previous state is the best to recover back to for an efficient learning. The failure prevention strategy reduces potential failures by predicting possible dangerous actions in specific states. Both a simulation of a robot navigating in an unknown maze and a real-world experiment of stacking blocks in a prior unknown order are used to validate our strategy. The results show a significant reduction in the number of resets and failures during the learning with using our strategy.

[8] Primary researcher and author, graduate student, Department of Mechanical Engineering, Colorado School of Mines, 1610 Illinois Street, Golden, Colorado, 80401, USA.
[9] Corresponding author, Associate Professor, Department of Mechanical Engineering, Colorado School of Mines, 1610 Illinois Street, Golden, Colorado, 80401, USA.

4.2 Introduction

Reinforcement learning (RL) has shown success on robot autonomy in recent years. However, it is still challenging for the practical deployment of RL on autonomous robots because many RL algorithms require manually resetting the state of the system between training episodes [1][2][3]. For example, RL-based robots need human interventions to be reset to the same initial state for a new learning episode if they experience failures during the learning. In practice, the manual resets can be easily unavailable due to the complexity of the environments. The RL algorithm entirely stops working when the robot is waiting for such an unavailable manual reset. Thus, it is practically important to design an autonomous reset strategy for RL to avoid manual resets.

Consider an example scenario as shown in Figure 4.1, in which the robot needs to stack four blocks from four predefined positions into a target position in a specific order. If the order is wrong, then the blocks fall over, which is considered as a failure in the learning. This failure is not expected to happen many times because the blocks can be damaged after a certain number of fallings. When a failure occurs, the robot using traditional reinforcement learning waits for human interventions to solve the failure by manually resetting all the blocks to their initial positions. If human interventions are not available, the robot does not know what to do. The best case is the robot can keep moving between the pre-defined positions and the target position, but the blocks are not in these positions any more so that the failure keeps existing. In the case of no human interventions, we want to answer three questions in this work: (1) Is there an automatic recovery mechanism that can be integrated into reinforcement learning?, (2) Is it really necessary to recover back to the initial state? Would it be better to recover back to intermediate states?, and (3) How can we reduce as many resets as possible?

For the first question, it is possible to use external, pre-defined failure recovery methods to diagnose the failure and bypass it with alternative plans. However, this does not enhance RL's own capability to recover from failures. If the remedy plan proposed by an external recovery method is not appropriate or the recovery method itself no longer works, the RL-based robots will fail again. Thus, it is ideal for RL itself to have the capability of generating recovery plans to get rid of the problematic state.

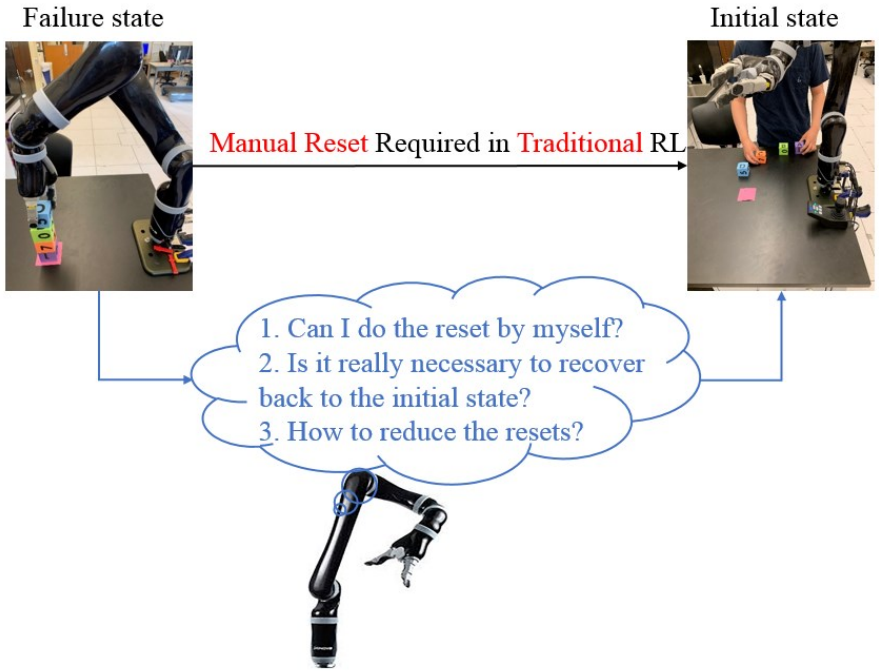

Figure 4.1 Can reinforcement-learning-based robot do the reset itself? The robot needs to stack four blocks from four pre-defined positions into a target position in a prior unknown order. If the stacking order does not match the expected one, the blocks fall over. When such a failure occurs, human interventions are required to reset the state to the initial state if traditional reinforcement learning is used for the robot. However, manual resets can be unavailable and hence what should the robot do?

More importantly, it may be neither necessary nor the best to recover back to the initial states. In the block stacking task, for example, it may be better to recover back to the state with block A already in the target position rather than placing all the blocks in their original positions, if block

A is required to be placed at the bottom. Starting from the state with block A in the target position may assist RL in finding the optimal policy more quickly. Thus, it could be more efficient for a RL-based robot to know which state it should recover back to.

There could be many reasons such as experiencing failures or the transition from the goal state of one episode to the initial state of the next episode, which can lead to resets. However, because we only focus on the failure-induced resets, it is then intuitive to reduce the resets by reducing the failures. Since avoiding failures is a promising way to reduce failures, it is better for RL to have the capability of predicting failures and avoiding actions that can result in the predicted failures.

To this end, we propose a self-recoverable reset strategy that can (1) self-recover from non-fatal failures back to any safe states to continue achieving the training goal in each episode, (2) determine which state the robot should be reset to for an efficient learning when encountering a failure, and (3) prevent failures that are similar to previously occurred ones from happening again. The contributions are two-fold. Firstly, a novel self-recoverable reset strategy is proposed to solve the manual reset problem in practical deployment of RL. This strategy can substantially make RL applicable to real-world autonomous robot planning without manual engineering. Secondly, we experimentally demonstrate that either of the two components of our strategy, i.e., the multi-state recovery strategy and failure prevention strategy, can individually reduce resets while their combination can further reduce resets in a more efficient way. Both the advantages and disadvantages of each strategy are discussed to provide insights of our work.

4.3 Related Work

Reinforcement learning (RL) has achieved success in robot planning [4][5][6] in recent years. In general, RL methods can be divided into two categories: model-based methods [7] and model-free methods [8]. Model-based methods are known to be more sample-efficient and model-free

methods are particularly popular due to their simplicity and favorable computational properties. However, the practical deployments of both methods often assume an episodic setting, which require manually resetting the state of the system between episodes [2]. Since the manual resets can be easily unavailable in practice, our work aims to solve this important problem by proposing an autonomous reset strategy, including self-recovering from failures and reducing potential failures.

Although the failure recovery [9][10][11] has already been used in traditional robot autonomy, it has been rarely investigated in RL-based robot autonomy, especially when it is expected to solve the failures within RL. This is different from solving the system failures from the robot system's perspective. For example, RL with an external failure recovery mechanism can still keep making the same mistake because the external failure recovery can only correct the mistake whenever it occurs but cannot analyze why it occurs in RL. Thus, it is necessary to integrate failure recovery into RL's own framework to essentially enhance RL's own capability to address failures and hence avoid corresponding failure-induced resets. This is actually a main goal of our work which specifically investigates the recovery from a problematic state to the best unproblematic state in RL.

Many safe reinforcement learning algorithms [12] have been proposed to reduce the failures during the learning process by modifying the optimization criteria to additionally respect the learning safety [13][14][15][16] and/or restricting safe explorations [17][18][19][20]. However, these methods usually require extensive human knowledge to design different criteria for different situations while our work assumes no such knowledge and only rely on RL itself to reduce failures. More importantly, reducing failures cannot completely solve the reset problem because they

cannot address the already occurred failures. As failures could be unavoidable in RL, our work proposes a multi-state recovery strategy to address the occurred failures.

The reset problem has been partially addressed in a few recent work. For example, learning multiple policies [1][2], namely a forward and a reset policy, has been used to address the safety, reset, and task complexity problems in autonomous RL. While such a reset policy can reduce resets through reducing potential failures, it has not shown an ability of recovering from an occurred failure, which is important because failures could be unavoidable in practical RL. A reset-free trial-and-error (RTE) algorithm [3] has been proposed to solve the robot damage recovery with formulizing it as an RL problem. Although our work shares the same goal of avoiding failures through failure recovery, we have an essential difference in the recovery mechanism. Specifically, the RTE algorithm recovers from failures in one state by one state with trying a best action in each state according to the practical feedback. In contrast, our work investigates a multi-state recovery, in which the system can go several states back to avoid some intermediate, meaningless states for an efficient purpose. In addition, the RTE algorithm needs an intact robot model to make recovery plans while our work can be totally model-free without any knowledge about the robot and environment.

4.4 Preliminaries

In this section, we discuss a standard Q-learning setup, which provides the basis to understand our proposed self-recoverable reset strategy. Although we use the Q-learning, a model-free RL method, as an example, our method is actually designed for general RL methods because the multi-state recovery and failure prevention are both based on general definitions of states and action selections.

The Q-learning problem is usually formulated as a Markov Decision Problem (MDP) that consists of a state space S, an action space A, transition dynamics $P(s_{t+1}|s_t, a)$, an initial state distribution $p_0(s)$, a scalar reward function $r(s, a)$, and a reward discount factor γ. In episodic, finite horizon tasks, the objective is to find the optimal policy $\pi^*(a|s)$ that maximizes the expected sum of γ-discounted returns, $Q^\pi(s, a) = E_\pi[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)]$, where $s_0 \sim p_0$, $a_t \sim \pi(a_t|s_t)$, and $s_{t+1} \sim P(s_{t+1}|s_t, a_t)$. The Q-values with multiple actions can be updated by

$$Q_{t+1}(s_t, a_t) \leftarrow (1 - \alpha) Q_t(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1}, a)), \tag{4.1}$$

where α is the learning rate deciding how much the system trusts the new information. With the ε-greedy exploration strategy, the robot selects at each time step a random action with a fixed probability, $0 \leq \varepsilon \leq 1$, instead of greedily selecting one of the learned optimal actions with respect to the Q-function:

$$\pi(s) = \begin{cases} \text{rand}(A) & \text{if } \xi < \varepsilon \\ \underset{a \in A}{\text{argmax}}\, Q(s_{t+1}, a) & \text{otherwise,} \end{cases} \tag{4.2}$$

where $0 \leq \xi \leq 1$ is a uniform random number drawn at each time step.

Typically, the RL training routines involve iteratively sampling new episodes, where at the end of each episode, a new starting state s_0 is sampled from a given initial state distribution p_0. For simplicity, we assume the initial state to be the same in each episode. In practice, this procedure can be done by executing some hard-coded reset policies. Although this may work for resetting from a successful episode in which the initial state and goal state are known, it is difficult to reset from failures where the terminate state is usually uncertain. Human interventions can well address the failure-induced resets, but they are not always available in practical applications. Thus, the goal of our work is to avoid these failure-induced resets by proposing a self-recoverable reset strategy.

4.5 Methodology

As shown in Figure 4.2, our method consists of two individual parts, multi-state recovery and failure prevention. The multi-state recovery is used by RL to recover from occurred failures with determining which previous state is the best to reset to. Instead of simply abandoning the failed episode with resetting to the initial state, the multi-state recovery aims to finish the training goal in each episode. The failure prevention is namely to prevent the failures from occurring during the learning process.

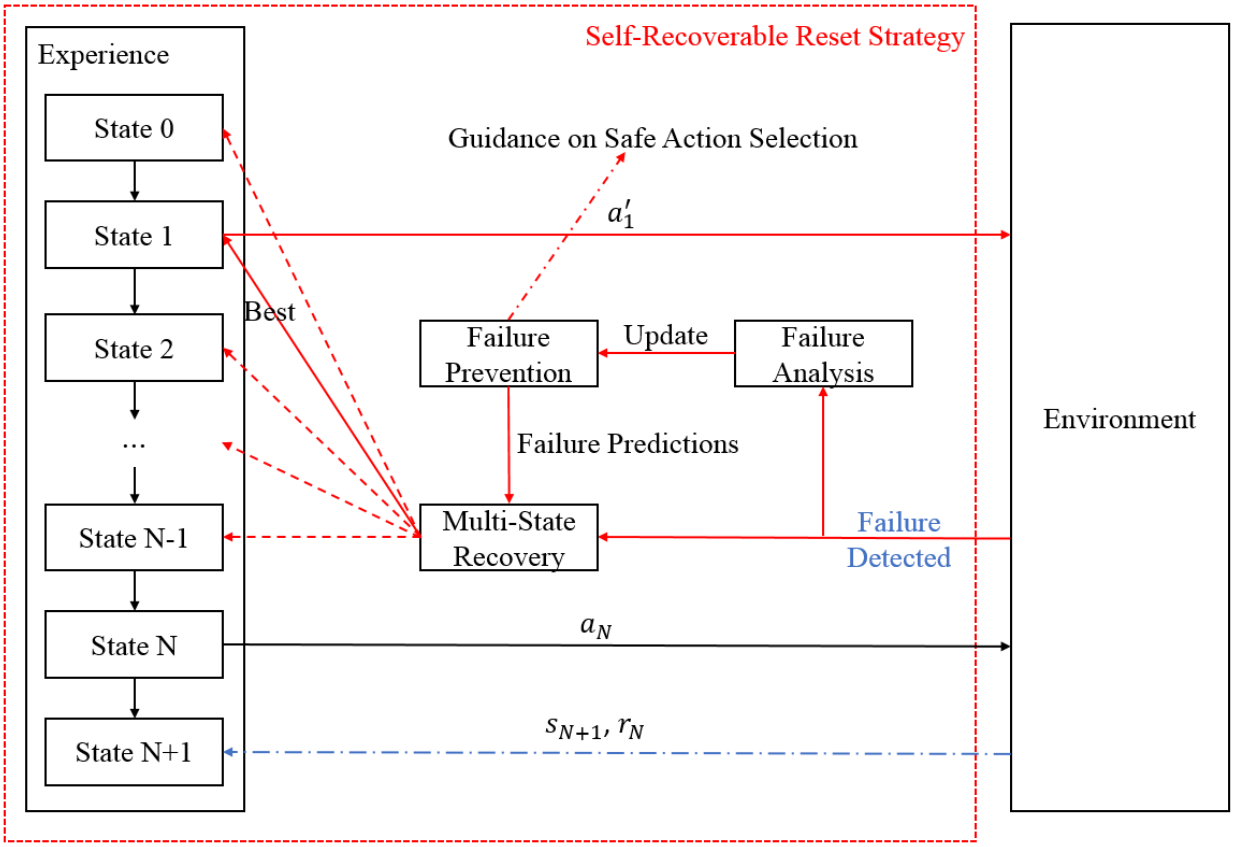

Figure 4.2 Self-recoverable reset strategy. It consists of the multi-state recovery strategy and the failure prevention strategy. The multi-state recovery strategy provides the robot with a capability of self-recovering back to any state with reversely executing the safe actions stored in an experience buffer. In addition, the multi-state recovery method decides which state is the best for the robot to recover back to for an efficient learning. The failure prevention strategy learns from already occurred failures and predict future failures so that the robot can avoid them, which also avoids resets.

4.5.1 Multi-State Recovery

When practical robots experience failures in a problematic state, it is possible to recover back to some previous unproblematic states and make a new plan. To make this recovery practically applicable, we make two weak assumptions. The first one is the availability of an accurate detection of a failure. In other words, the robot can know when a failure happens, which triggers the execution of the multi-state recovery strategy. The second assumption is that there exists an available method for the robot to execute the recovery plan such as going to a certain state. The first assumption can be achieved with powerful sensors and the second one can be achieved with some pre-defined low-level controllers, which are both out of our scope so that they are not investigated in this work.

An important question in the recovery is: *which state is the best to recover back to*? A promising answer is to assist the learning process as efficiently as possible. During a specific episode, for example, the robot is in a specific state and taking a problematic action "pick and place block D" as shown in Figure 4.3. Because the resultant state is "A-C-B-D", which violates a defined rule, i.e., block D cannot be at the top of block B, a failure will occur and be detected. It may then be ideal for the robot to explore other actions in the current position if it does not know the results of these alternative actions. Not doing this and directly going back to other states may sometimes miss an optimal policy. However, if the robot has already known the consequences of alternative actions as shown in Figure 4.3, then it is ideal to recover back to the junction state (the blue state) because the states in the dead end is meaningless in achieving the goal.

Inspired by this, we record a N-step learning process that includes the previous N states, $S_N = \{S_{t-N+1}, S_{t-N+2}, \dots, S_{t-1}, S_t\}$, and the corresponding N actions. The selection of N should be related to the complexity of the robot, task or environment and different values of N should affect

the performance of RL. Our goal is to experimentally investigate these relationships and also determine which state in these N states is ideal to recover back to when a failure occurs. We assign for each of these N states a recovery priority value $\theta(s)$, which indicates the priority of each state for the robot to reset to. For the intuition, when a state has more safe actions that do not cause failures, it should have a higher priority value. A safe action set in state s can be defined as

$$A_s(s) \triangleq \{a \in A(s) | a \notin A_d(s)\}, \tag{4.3}$$

where $A_d(s)$ means the dangerous action set in state s and it is updated based on the experience of RL. Because we assume the failure can be accurately detected, the dangerous action can be recorded immediately. In this work, we simply assume the detection as receiving a pre-defined large negative reward

$$A_d(s) \triangleq \{a | r(s,a) = r(failure)\}. \tag{4.4}$$

Thus, the priority $\theta(s)$ can be calculated by

$$\theta(s) = |A_s(s)|, \tag{4.5}$$

where $|\cdot|$ computes the size (number of elements) of a set.

If different states have the same maximal priority, then we will reset the robot to the nearest state because the nearest state may be more close to the goal state. Then, the best state S_r^* determined by the recovery strategy is

$$S_r^* = \{s^* \in S_N | s^* = \underset{s'}{\operatorname{argmin}}\, d(s', s_t), s' = \underset{s}{\operatorname{argmax}}\, \theta(s)\}, \tag{4.6}$$

where $d(s', s_t)$ means the distance between an arbitrary state with the largest priority in state space S_N and the state s_t. The recording of the N-step learning process is always running for the anytime execution of the recovery strategy. However, the computation of $\theta(s)$ and S_r^* is only activated when a failure occurs to save resources.

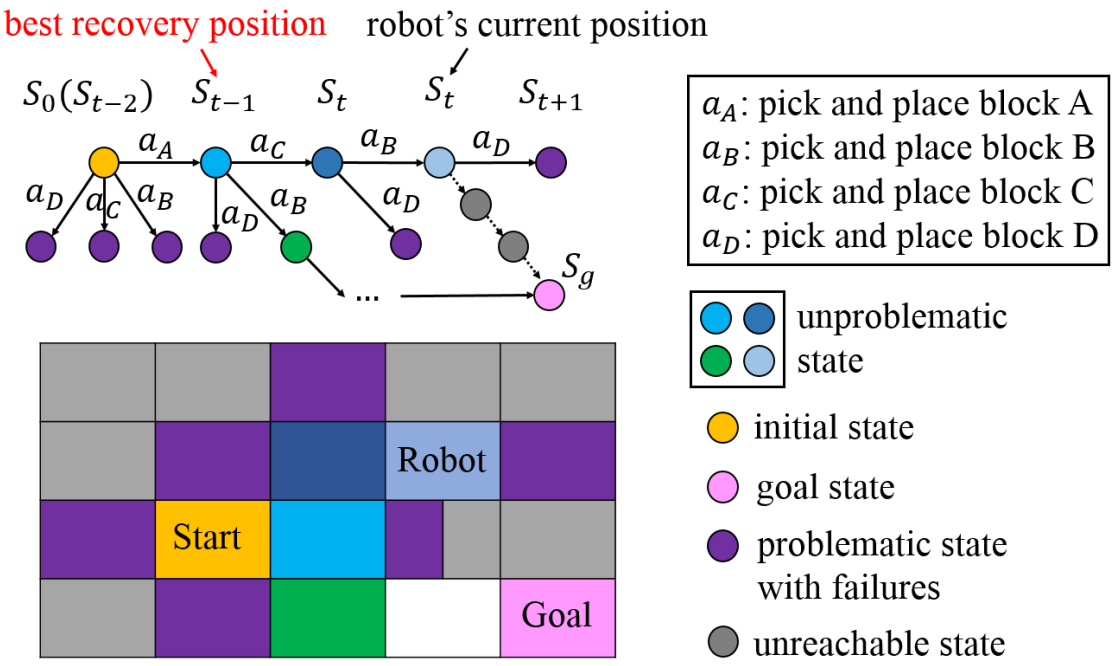

Figure 4.3 Multi-state recovery strategy: recover from an occurred failure to an ideal previous state to continue learning to achieve the training goal in each episode. The robot should recover from the gray state to the red junction state instead of the light blue and dark blue states which are still in the dead end.

4.5.2 Failure Prevention

Although failure prevention cannot address the occurred failures, it can be used as another way to reduce resets from a complementary perspective of reducing potential failures. In general, failures can provide useful information for the successive learning, so we record the failures and predict potential failures by analyzing these failures as shown in Figure 4.4.

A simple and practically available method is to exactly prevent the failures that have already occurred before. The standard Q-learning can actually achieve a similar performance by updating Q-value after each step, but it allows random explorations which may still make a mistake. However, the prediction can be extended to similar failures if some additional information about the environment can be provided. For example, if the robot takes an action of picking and placing block D in the light blue position in Figure 4.4, it will encounter a failure of blocks falling over. It

is sometimes possible to learn out a rule from this failure and similar failures or get a human's guidance, which is block D cannot stand on the top of block B. With this rule, the action of picking and placing block D in the green state will also result in a failure and this action will be forbidden.

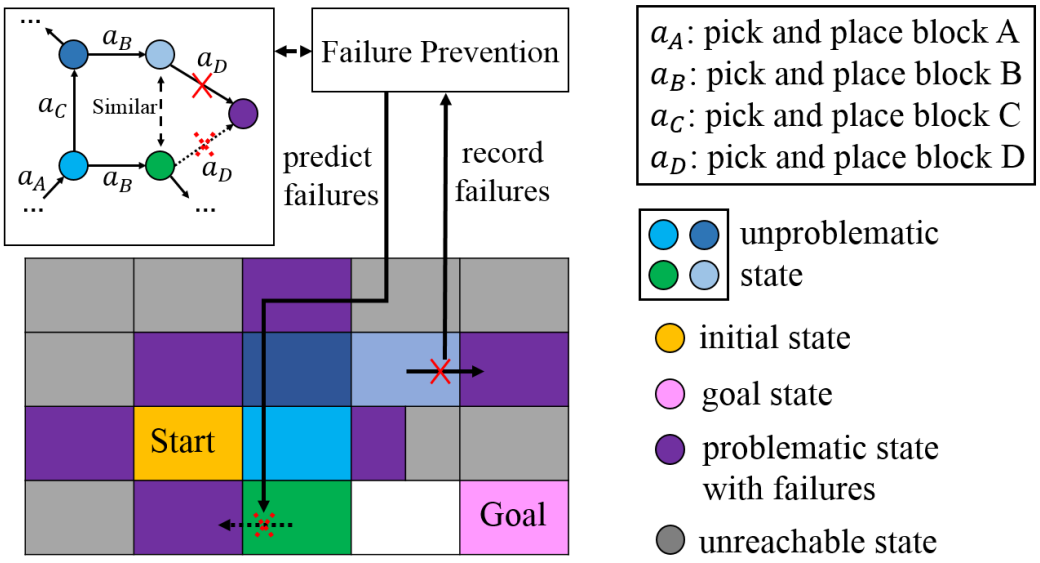

Figure 4.4 Failure prevention strategy: predict and avoid potential failures based on the recorded failure experiences. This strategy can be as simple as exactly preventing the occurred failures, and can also be as complex as using additional knowledge to prevent similar failures.

Because we assume no external information in this work, we use the first, simple method with exactly preventing the occurred failures, which can also be more realistic. Thus, the ε-greedy exploration strategy should be modified to

$$\pi(s) = \begin{cases} rand(A) | a \notin A_d(s) & \text{if } \xi < \varepsilon \\ \underset{a \in A}{\operatorname{argmax}}\, Q(s_{t+1}, a) \,| a \notin A_d(s) & \text{otherwise} \end{cases}. \tag{4.7}$$

4.5.3 Algorithm Summary

Our full algorithm (Algorithm 1) consists of multi-state recovery and failure prevention. Failure prevention is always running to reduce failures and the multi-state recovery is executed only when a failure occurs. In general, the multi-state recovery is executed mostly in the early

stage of RL. As the episode increases, RL can have more accurate knowledge about the environment and dangerous actions. This results in less occurrences of failures and hence less utilizations of the multi-state recovery.

The algorithm firstly initializes all the variables and then execute the modified action selection strategy described in equation (4.7). If a failure is detected (line 5), the failure is recorded to augment the dangerous action set $A_d(s)$ (line 6). Based on $A_d(s)$, the algorithm decides the ideal state to recover back to (line 9) and continues learning in the current episode until the final training goal or the maximal step number is achieved. If no failure occurs, our algorithm updates the Q value as the standard Q-learning does.

Algorithm 1: Q-learning algorithm with self-recoverable reset strategy	
1:	Initialize all variables
2:	**repeat**
3:	**for** *max_steps_per_episode or finish_goal* **do**
4:	$a \leftarrow \pi(s)$ ▷ failure prevention
5:	**if** $r(s,a) == r(failure)$ **then**
6:	$A_d(s) = A_d(s) \cup \{a\}$ ▷ record failures
7:	Update $A_s(s)$ ▷ equation (3)
8:	Update Q-value
9:	$s \leftarrow S_r^*$ ▷ multi-state recovery
10:	**else**
11:	$s \leftarrow P(s_{t+1}\|s_t, a_t)$
12:	Update Q-value
13:	**Return** the optimal policy

4.6 Evaluation

4.6.1 Experiment Setup

In this work, we use both a maze-navigation simulation and a real-world block stacking experiment to validate our algorithm. The simulation is about the robot navigation in an unknown maze with obstacles as our experiment. We set two maze environments with the maze size to be 4*4 and 11*11. Firstly, it is convenient for us to compare 4*4 maze results with a closely related

work [2] that learns a reset policy to reduce manual resets. In addition, the 11*11 maze can be used to investigate the performances of the multi-state recovery in a more complex environment and the influences of different selections of N on the performance of RL. As shown in Figure 4.5, the state space of the 4*4 maze consists of 16 grids and 4 of them are obstacles (walls). The 11*11 maze has 121 grids and 43 of them are obstacles. For simplicity, we assume these obstacles are static and the extensions to dynamic obstacles will be discussed in the discussion section. The robot can take four actions: go north, go south, go east, and go west. In general, the robot starts each episode in the upper-left corner of the maze and terminates if reaching the lower-right corner (goal) with a reward of 1 or hitting an obstacle with a reward of -5. For any other individual movements, the reward is -0.02. Parameters ε, γ, and α are set to be 0.1, 0.9, and 0.5, respectively.

As shown in Figure 4.1, we expect the robotic arm to stack four blocks, block A (purple), block B (green), block C (orange), block D (blue), in the target position from their original positions. Because the grasping is not the focus of our work, we assume the original position and the target position are fixed and known so that we can guarantee a high grasping accuracy. However, the stacking order is unknown to the robot, which needs reinforcement learning itself to find out. The stacking rules includes (1) block A must be at the bottom of all the other three blocks, (2) block D must be at the top of block C, (3) block D cannot be at the top of block B. If violating any of these rules, the blocks fall over, which is counted as a failure. The state space is defined as the block order at the target position. The robotic arm can take four actions: pick and place block A, block B, block C, and block D. The reward of reaching the goal (A-B-C-D) is 1 and the reward of a failure is -1. For any other individual action, the reward is 0. Parameters ε, γ, and α are set to be 0.1, 0.9, and 0.2, respectively.

The calculation of the number of resets includes two parts. The first one is adding one reset if the robot cannot achieve the goal state within the limited steps in an episode. We do not count the transition from the goal state in an episode to the initial state in the next episode as a reset because we focus on the failure-induced resets in this work. The other part comes from the occurrence of failures. This means a reset will be needed whenever a failure happens, which is the cases in the standard Q-learning without the mutli-state recovery strategy. In order to provide reliable results, the simulations and the real-world experiment in this work have been conducted 50 times and 20 times, respectively, to compute the average and standard deviation.

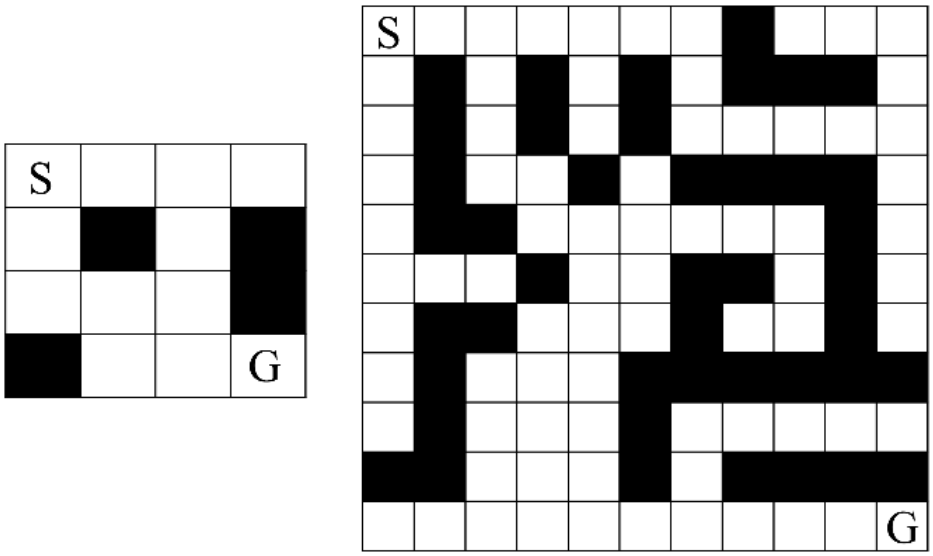

Figure 4.5 The simulation environment: a 4*4 (left) and 11*11 (right) maze. The robot is supposed to find an optimal path with a largest reward from the initial state (S) to the goal state (G). The black grids represent obstacles such as walls and the robot is not allowed to pass these obstacles. The robot can take four actions: go north, go south, go east, and go west.

4.6.2 Experiment Results

For a convenient expression and comparison, we use SRR to replace the formal name of our self-recoverable reset strategy in the following part. The multi-state recovery and failure

prevention are also abbreviated to MSR and FP. SRR is a complete strategy including both MSR and FP.

(1) Comparison with related work

Table 4.1 shows that either of our three strategies can reduce much more resets than the reset policy proposed by Eysenbach et al. We use the best result of their reset policy with q_{min}=0.4 for comparison. For our methods, N is chosen to be 3 and the maximum steps per episode is set to 200 because the 4*4 maze is relatively simple.

Table 4.1 Comparisons with previous results. Either of our three strategies outperform the reset policy proposed by Eysenbach et al. in terms of both number of resets and number of total steps.

Methods	Number of Resets (Mean)	Number of Total Steps (Mean)
Reset Policy [2]	55	6200
Q-learning with FP	10	1830
Q-learning with MSR	0	1823
Q-learning with SRR	0	1822

Although both Eysenbach's reset policy and our FP strategy reduce the resets by reducing potential failures, our FP strategy is still better because we can exactly avoid recorded failures that has occurred before while Eysenbach uses a threshold to partially abort some dangerous actions. Because MSR gives RL the ability to recover from failures, both MSR and SRR can completely avoid resets. In addition, our method requires much less steps to finish the training, which can be regarded as more efficient.

(2) MSR vs FP in SRR

As discussed before, either MSR or FP can individually reduce the number of resets. In order to investigate the effects in detail, we calculate the number of resets, number of total steps to converge, and number of episodes to converge in Table 4.2. For both the simple 4*4 maze and the

complex 11*11 maze, both MSR and FP can significantly reduce the number of resets compared to a standard Q-learning. While FP can only reduce resets, MSR can guarantee no resets in the simple maze and a mostly zero reset in the complex maze. This is because MSR avoids resets in a direct way that can substantially eliminate resets through recovering from failures. In contrast, FP only avoids resets by preventing failures, which can be easily restricted with a necessity of accurate knowledge about the environment. If combining MSR and FP to SRR, the performance keeps improving. Both FP and MSR can find the optimal path with less episodes and total steps than the standard Q-learning. However, MSR is more efficient, especially in the complex maze.

Table 4.2 Full comparisons among the standard Q-learning, Q-learning algorithm with failure prevention (FP) strategy, Q-learning with multi-state recovery (MSR) strategy, and Q-learning with self-recoverable reset (SRR) strategy. Either FP or MSR can individually outperform the standard Q-learning, although the combination of FP and MSR to SRR has the best performance in all the four methods. MSR can almost guarantee a zero reset even in a complex 11*11 maze because it avoids the resets from the perspective to fixing failures rather than just avoiding failures.

Methods	Num. Resets (Mean)	Num. Episodes (Mean)	Num. Total Steps (Mean)
Standard-4*4 maze	33.5	243.0	1796
FP-4*4 maze	9.0	217.6	1720
MSR-4*4 maze	0	211.8	1738
SRR-4*4 maze	0	207.8	1694
Standard-11*11maze	345.5	524.0	10610
FP-11*11 maze	95.1	327.6	10658
MSR-11*11 maze	0.9	234.2	9887
SRR-11*11 maze	0.7	235.0	9709

(3) Will MSR cause more failures to recover from failures?

One important concern for MSR is whether MSR causes more failures to recover from failures. The failure in our experiments is defined to be the collision between the robot and an obstacle.

Firstly, Table 4.3 shows that MSR does not really have such a concern. Although the number of total failures in MSR is close to that in the standard Q-learning, it is still less than the standard Q-learning. This is because MSR cannot change the essence of RL, i.e., "trial-and-error". MSR only changes the timing of exploring such failures, which is exploring more failures than the standard Q-learning in the first episodes. This expedites the learning process with less episodes, which can avoid the failures in the unnecessary episodes. Because most failures have been encountered in such episodes, Q-learning can also avoid them based on the Q-values. However, the random exploration strategy in the standard Q-learning can still cause a few failures, which explains the small failure difference between MSR and the standard Q-learning.

Table 4.3 Failure comparisons between the standard Q-learning and our method. While MSR only reduces the failures in a small amount, FP can significantly reduce the failures because it can strictly restrict the random explorations in the Q-learning to safe actions. However, the failures in MSR do not result in resets while the failures in the standard Q-learning and FP must require corresponding resets.

Methods	Number of Total Failures (Mean)
Standard-4*4 maze	33.5
FP-4*4 maze	9.0
MSR-4*4 maze	32.1
SRR-4*4 maze	10.0
Standard-11*11maze	345.5
FP-11*11 maze	95.1
MSR-11*11 maze	333.2
SRR-11*11 maze	114.7

Secondly, FP has much fewer failures than MSR because all the random explorations in FP are strictly restricted to safe actions. In other words, the same mistake is not allowed to be made again in FP. Although the number of failures caused by normal random explorations can be small,

but its multiplication with the number of episodes can be significant, which explains the large failure difference between FP/SRR and standard Q-learning/MSR.

Thirdly, although MSR has a certain number of failures, these failures do really not result in resets because they can be recovered by RL itself. For example, the MSR explores an average 32.1 failures, it does not need any reset in Table 4.2. In contrast, the number of failures equals to the number of resets for the standard Q-learning and FP because a reset is required whenever a failure occurs in these two methods.

(4) N and maximum steps per episode in MSR

As discussed in the multi-state recovery strategy, the selection of *N* and maximum steps per episode is important in MSR. We empirically allow the variation of *N* and maximum steps per episode to be 1 to 7 with a one-step increment of 1 and 100 to 1000 with a one-step increment of 100, respectively.

Figure 4.6 shows the relationship between the number of resets and the recovery limits *N*. When the environment is simple (the 4*4 maze), MSR can guarantee no resets. However, when the environment is more complex (the 11*11 maze), a larger recovery limit *N* could be better especially when allowing enough maximum steps in one episode. The limited maximum step per episode such as 400 for the 11*11 maze could be too small for MSR to be fully functional because recovering from failures need to cost episodes. Thus, when the maximum steps per episode is small (400), a lower recovery limit (*N*=1) outperforms other *N*'s. As the maximum step per episode increase to 600 or 800, however, *N*=2 or *N*=3 actually has the best performance. The differences between various *N*'s should be much more significant when the environment gets highly complex.

Figure 4.7 shows the relationship between the number of resets and maximum steps per episode. A general conclusion is that the number of resets reduces as the maximum steps per

episode increases, which provides enough time for MSR to recover from failures. In addition, Figure 4.7 also proves a larger *N* has better performances when the maximum steps per episode is larger.

(5) Block stacking results

As shown in Table 4.4, the block stacking results are consistent with the maze-navigation results. SRR still performs best with the least number of resets, the least number of episodes, and the least number of total steps. The number of resets is zero if MSR is used. FP can also reduce the number of resets, but it still has 11.6 resets in average.

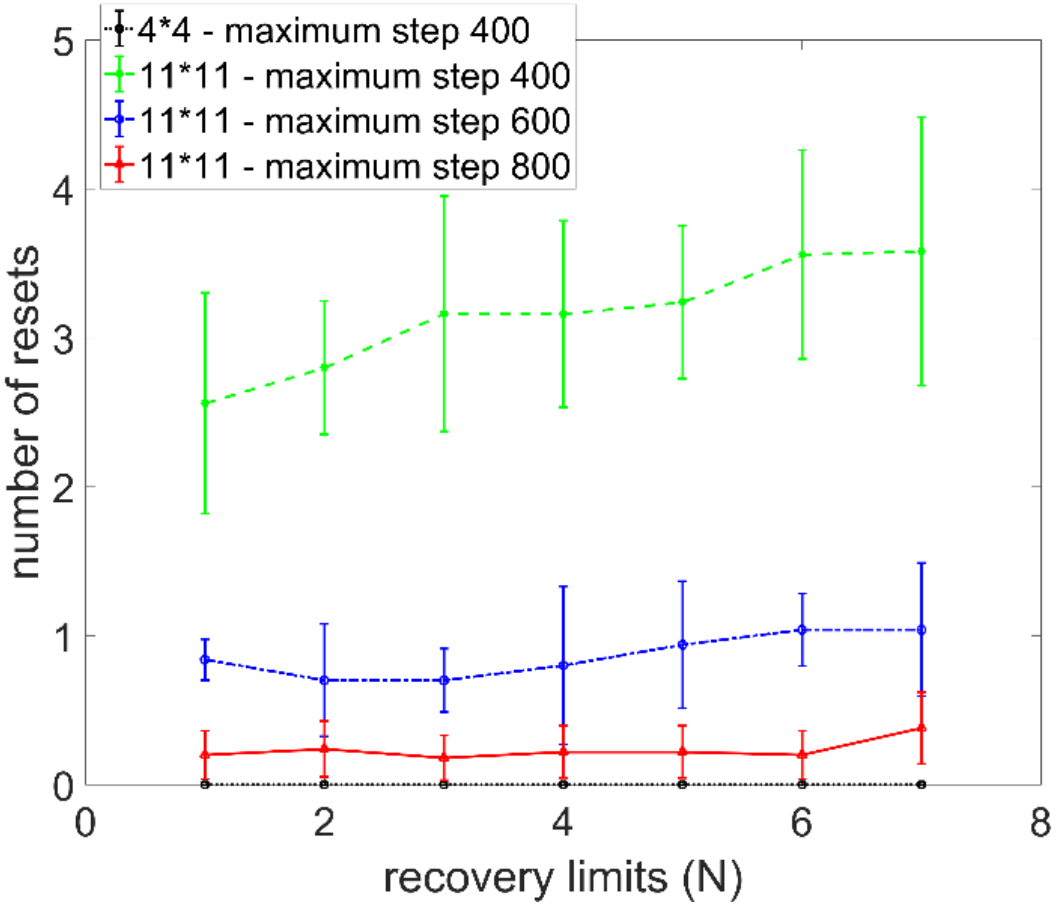

Figure 4.6 The relationship between number of resets and recovery limits N. When the environment is simple (a 4*4 maze), even only recovering back to the latest state can guarantee no resets. However, when the environment is more complex (a 11*11 maze), a larger recovery limit N (e.g., 3) could be more effective, especially given enough maximum steps per episode.

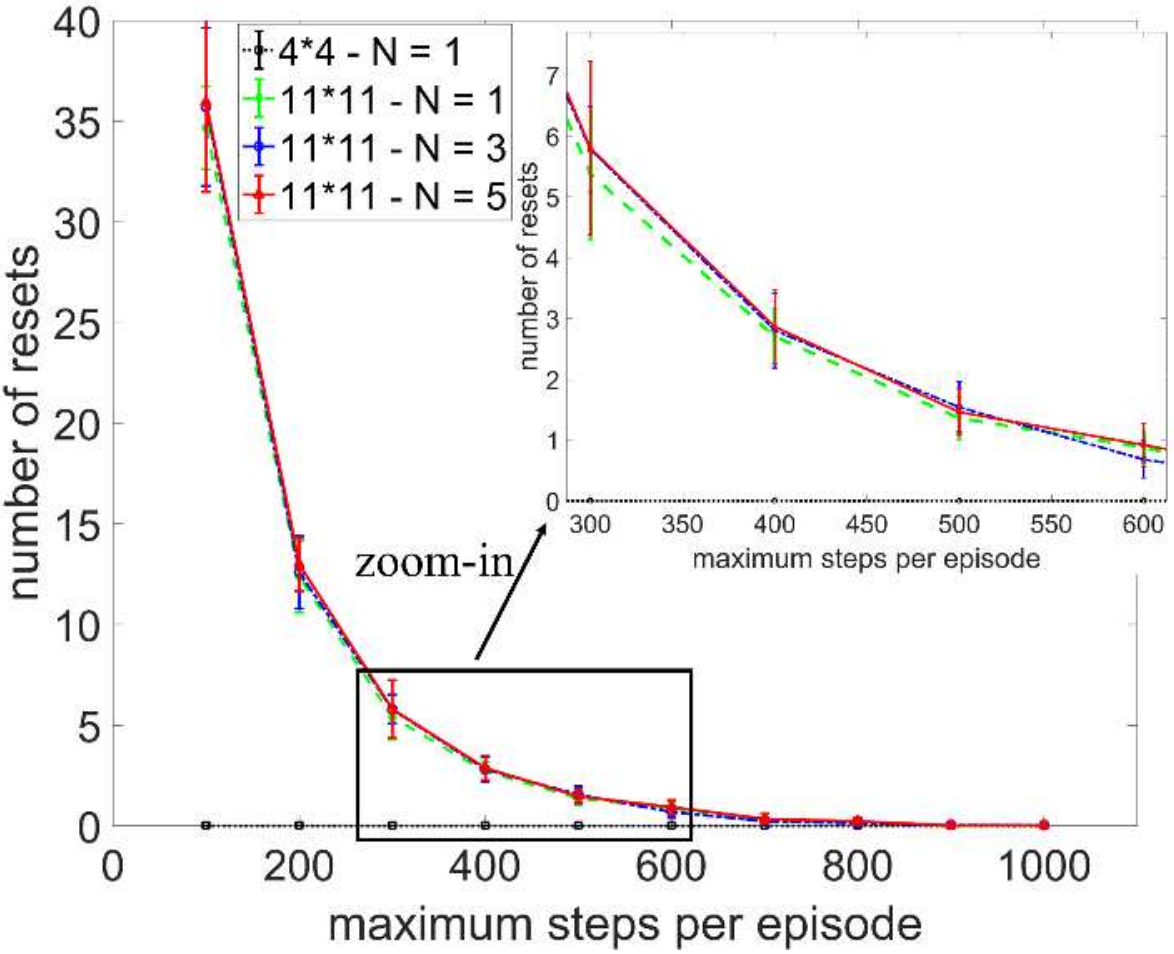

Figure 4.7 The relationship between number of resets and maximum steps per episode. The number of resets reduces as the maximum steps per episode increases. When maximum steps per episode remains large, a larger N can be better in reducing the number of resets.

Table 4.4: Full comparisons among the block stacking results using standard Q-learning, Q-learning algorithm with FP strategy, Q-learning with MSR strategy, and Q-learning with SRR strategy. Either FP or MSR can individually outperform the standard Q-learning, although the combination of FP and MSR to SRR has the best performance in all the four methods. MSR and SRR can guarantee a zero reset.

Methods	Num. Resets (Mean)	Num. Failures (Mean)	Num. Episodes (Mean)	Num. Total Steps (Mean)
Standard – Block Stacking	41.2	41.2	157.0	3824
FP – Block Stacking	11.6	11.6	133.2	2516
MSR – Block Stacking (N=3, maximum step per episode = 50)	0	39.8	124.3	2681
SRR – Block Stacking	0	12.2	120.7	2294

As shown in Figure 4.8, our method firstly provides the robot with a capability of self-recovering to any state in its stored experience. This completely avoids human interventions that are required in traditional reinforcement learning. Even if the robot does not know which state it should recover back to, it can at least recover back to the initial state, which is equivalent to the result of a manual reset. If the robot has more information from learning the failures or from external guidance, it can decide a better state to recover back to.

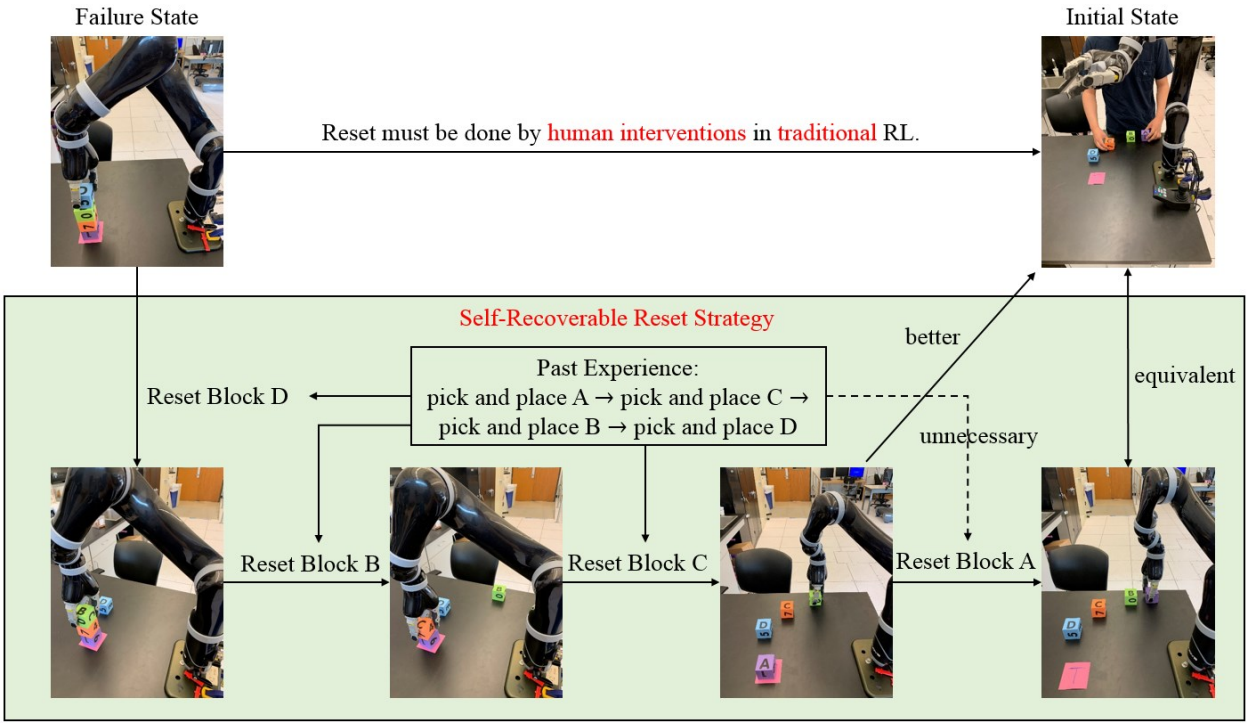

Figure 4.8 The capability of self-recovering back to any states. Human interventions are required in traditional reinforcement learning to reset a failure state to the initial state. In contrast, our self-recoverable reset strategy can recover back to any states with reversely executing the safe actions recorded in the experience buffer. Even if our method cannot decide a better state to recover back to, it can at least recover back to the initial state, which is equivalent to the manual reset.

Table 4.5 shows the time used to finish training with respect to different recovery limits N and maximum steps per episode. From the table, we can see our method is faster than the traditional reinforcement learning method, which means our method is more efficient than traditional

reinforcement learning method. This is another benefit in addition to our main goal of providing a self-recovery capability. Moreover, the total time generally increases as the recovery limits N increases expect for N = 4. This makes sense because it is obviously not the most efficient when the robot recovers four steps back, i.e. recovers back to the original state. As long as the robot picks block A first, it only needs to recover block B, block C, or block D back. Increasing the recovery limits can bring additional burdens to our method. Similarly, the total time also increases as the maximum steps per episode increases. However, an exception exists between 50 maximum steps per episode and 100 maximum steps per episode, where the mean total time is the same. This indicates 50 maximum steps per episode may be already sufficient for this experiment. Both cases indicate that it is important to select an appropriate recovery limit number and maximum step per episode for practical applications. This could be achieved by using additional human knowledge or mathematical optimization methods.

Table 4.5 Total training time with respect to different recovery limits and different maximum steps per episode

		Total Time (Mean)
Different Recovery Limits N (Maximum Steps per Episode = 50)	0	5.5 hours
	1	4.8 hours
	2	4.4 hours
	3	3.6 hours
	4	3.9 hours
Different Maximum Steps per Episode (Recovery Limits N = 3)	10	5.2 hours
	20	4.5 hours
	50	3.6 hours
	100	3.6 hours

4.7 Discussions

The general goal of our work is to assist RL-based robots being truly autonomous in practical applications with solving an unavoidable problem: robots require manual resets whenever encountering failures during the learning. While this can be solved by reducing failures as done in previous work, we, more importantly, propose a multi-state recovery strategy to fix the failures. This is similar to the advice for many complex systems: "we should not wonder ***if*** some mishap may happen, but rather ask ***what*** one will do about it when it occurs" [21]. We have also demonstrated that the combination of the multi-state recovery strategy and the failure-prevention strategy can achieve a better performance than either individual strategy with guaranteeing no resets in simple situations.

Within the self-recovery from failures, we specifically propose a multi-state recovery strategy with investigating which previous state is ideal to recover back to when a failure occurs. This is actually inspired by human's self-recovery from failures. For example, when humans experience a failure in a jigsaw puzzle, they do not really go back to start from the first piece or just revert the last piece, it is usual to take out several recently inserted pieces and restart from such a configuration. We achieve this by using a memory to record some past experiences such as a certain-length sequence of state-action pairs in RL. This could cost more resources from robots, but the increment of resources is believed to be affordable in practical systems because the amount of data is much smaller if compared to the memory prepared for RL. The storage of the matrix about dangerous state-action pairs can be another cost of memory, but this cost can be greatly reduced because the matrix is usually highly sparse. It should not cause a problem if using a compressed storage.

While we assume failures in this work are non-fatal and recoverable, manual resets could be certainly needed if the failures are fatal and unrecoverable. In addition, although we focus on the failure-induced resets because the occurrence of failures is a primary reason of manual resets in practice, it cannot be denied that manual resets can be also caused by other reasons. A representative example is the transition from the goal state in one episode to the initial state in the next episode. This could require significant resets if RL requires massive learning episodes, especially for highly complex tasks. However, unlike the failure-induced resets, this kind of resets does not necessarily need human interventions because it can be totally the same for a specific task. Robots may use pre-defined plans to autonomously go back to the initial state.

In this work, we have only discussed the performance of our method in a static environment, although it is unknown. A more challenging case can be a dynamic, unknown environment. While our work may still work if the time-varying change is small, it is better to design an advanced algorithm to completely address the dynamic changes. The difficulty is that the recorded experiences in our current work can be no longer accurate and useful due to the environment change. One possible solution is to use a reliable online learning method like the Bayes inference [22] to real-time estimate a dynamic environment model. Our method may then use this model to update its past experience by predicting the environment changes when it starts recovering.

4.8 Conclusion

In this work, we propose a self-recoverable reset strategy to avoid manual resets in the practical deployment of RL on autonomous robot planning. It includes a multi-state recovery strategy and a failure prevention strategy. Instead of simply abandoning the failed episode without further learning within it, the multi-state recovery focuses on recovering from failures so that RL can continue learning to succeed in finishing the training goal in the failed episode. The failure

prevention strategy predicts and avoids potential failures according to previous experiences. Although these two strategies can individually reduce resets, the combination of them is experimentally demonstrated to achieve less resets with using less episodes. Our method can guarantee no reset in simple environments and can also have a significantly small number of resets in complex environments. In the future, we plan to extend our applications from static, unknown environments to dynamic, unknown environments.

4.9 References

[1] W. Han, S. Levine, and P. Abbeel, "Learning compound multi-step controllers under unknown dynamics," In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6435-6442, 2015.

[2] B. Eysenbach, S. Gu, J. Ibarz, and S. Levine, "Leave no Trace: Learning to reset for safe and autonomous reinforcement learning," in *6th International Conference on Learning Representations (ICLR)*, 2017.

[3] K. Chatzilygeroudis, V. Vassiliades, and J. B. Mouret, "Reset-free trial-and-error learning for robot damage recovery," *Robotics and Autonomous Systems,* vol. 100, pp. 236-250, 2018.

[4] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol.32, no. 11, pp. 1238-1274, 2013.

[5] M. Pieters and A. Wiering, "Q-learning with experience replay in a dynamic environment," in *IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1-8, 2016.

[6] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, "Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1-8, 2018.

[7] M. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pp. 465-472, 2011.

[8] J. Peters and S. Schaal, "Policy gradient methods for robotics," In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2219-2225, 2006.

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

[10] S. Lengagne, J. Vaillant, E. Yoshida, and A. Kheddar, "Generation of whole-body optimal dynamic multi-contact motions," *The International Journal of Robotics Research*, vol. 32, nos. 9-10, pp. 1104-1119, 2013.

[11] V. Vonásek, S. Neumann, D. Oertel, and H. Wörn, "Online motion planning for failure recovery of modular robotic systems," In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1905-1910, 2015.

[12] J. Garcıa and F. Fernandez, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437-1480, 2015.

[13] P. Geibel and F. Wysotzki, "Risk-sensitive reinforcement learning applied to control under constraints," *Journal of Artificial Intelligence Research*, vol. 24, pp. 81–108, 2005.

[14] Y. Kadota, M. Kurano, and M. Yasuda, "Discounted Markov decision processes with utility constraints," *Computers and Mathematics with Applications*, vol. 51, no. 2, pp. 279-284, 2006.

[15] T. Morimura, M. Sugiyama, H. Kashima, H. Hachiya, and T. Tanaka, "Nonparametric return distribution approximation for reinforcement learning," in *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pp. 799-806, 2010.

[16] T. M. Moldovan and P. Abbeel, "Safe exploration in markov decision processes," in *Proceedings of the 29th International Conference on International Conference on Machine Learning (ICML)*, pp. 1451-1458, 2012.

[17] K. Driessens and S. Dˇzeroski, "Integrating guidance into relational reinforcement learning," *Machine Learning*, vol. 57, no. 3, pp. 271–304, 2004.

[18] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010.

[19] J. Tang, A. Singh, N. Goehausen, and P. Abbeel, "Parameterized maneuver learning for autonomous helicopter flight," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1142– 1148, 2010.

[20] C. Gehring and D. Precup, "Smart exploration in reinforcement learning using absolute temporal difference errors," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems (AAMAS)*, pp. 1037-1044, 2013.

[21] F. Corbato, "On building systems that will fail," *Comminutions of ACM*, vol. 34, no. 9, pp. 72-81, 1991.

[22] S. Ishii, W. Yoshida, and J. Yoshimoto, "Control of exploitation–exploration meta-parameter in reinforcement learning," *Neural Networks*, vol. 15, nos. 4-6, pp. 665-687, 2002.

CHAPTER 5

MULTI-OBJECTIVE-OPTIMIZATION-BASED CONTROL PARAMETERS AUTO-TUNING FOR AERIAL MANIPULATORS

A paper published on *International Journal of Advanced Robotic Systems*[10]

Xu Zhou[11] and Xiaoli Zhang[12]

5.1 Abstract

The aerial manipulator has recently attracted much research attention due to its wide applications such as aerial cleaning, aerial transportation, and aerial manipulation. It is important to design a reliable controller for the aerial manipulator to robustly perform aerial tasks with different settings. However, current controllers still employ manual parameters tuning methods, which is mostly limited to a specific setting like a fixed aerial manipulator configuration or an unchanged environment. In fact, there could be diverse configurations of aerial manipulators and uncertain environments in practice, which requires the manual tuning process to be frequently repeated. This repetition is easy to be unavailable due to its significant cost of time and expensive involvements of control-tuning experts. To solve these problems, a novel multi-objective-optimization based control parameters auto-tuning method is proposed for the aerial manipulator. Based on a conventional proportional-integral-derivative (PID) control structure, an evolutionary-algorithm-based optimization is used to automatically find optimal PID control parameters to satisfy conflicting objectives such as minimizing the integrated time square error and the control

[10] Reprinted with permission from *International Journal of Advanced Robotic Systems*, vol. 16, no. 1, pp. 1-13, 2019. DOI: 10.1177/1729881419828071.

[11] Primary researcher and author, graduate student, Department of Mechanical Engineering, Colorado School of Mines, 1610 Illinois Street, Golden, Colorado, 80401, USA.

[12] Corresponding author, Associate Professor, Department of Mechanical Engineering, Colorado School of Mines, 1610 Illinois Street, Golden, Colorado, 80401, USA.

rate. Simulation results prove that the proposed method can achieve better control performances like smaller overshoots and faster stabilization time than manual tuning methods.

5.2 Introduction

Thanks to the physical interactions with surrounding environments, the capacities of unmanned aerial vehicles (UAVs) have been extended from "passive" tasks such as inspection [1], remote sensing [2] and surveillance [3] to "active" tasks like aerial manipulation [4-20]. Although many tools including grippers [4][5], cables [6-9], screw drivers [10], and brushes [11] have been used for the important physical interactions, attaching a robotic arm to a UAV, named an aerial manipulator [12-20], has lately been considered as the most efficient and promising way. The aerial manipulator does not restrict the attitude of the payload or the reachable space of the end-effector as a stationary gripper-based platform does. It is also able to directly regulate the movement of the payload, of which the cable-based platform is not capable.

It is important to design a reliable controller for the aerial manipulator to robustly and accurately finish tasks. Some such controllers have been reported for different tasks. For example, a Cartesian impedance controller [12] was proposed for a quadrotor equipped with a three-degrees-of-freedom (3-DOFs) robotic arm to execute dexterous manipulation tasks. A novel hierarchical motion control scheme [13], including a top layer of an inverse kinematics algorithm and a bottom layer of a motion control algorithm, was introduced for a quadrotor with a 5-DOFs robotic arm to follow circular helix trajectories. A sliding mode controller [14] was designed to transport objects using a quadrotor with a 2-DOFs robotic arm. For the ARCAS project, a multilayer control architecture [15], including a novel battery movement compensation, arm static compensation, and external generalized forces estimation and compensation, was proposed for an octorotor with a 6-DOFs robotic arm. To allow outdoor operations, a stable backstepping based controller and an

admittance controller were developed for an octorotor and its 7-DOFs robotic arm respectively [16]. To fly with unknown objects, an augmented passivity-based controller [17] was designed for a hexacopter with a 2-DOFs robotic arm by online estimating unknown parameters of the payload. The attitude control of a hexacopter equipped with a 2-DOFs robotic arm was achieved by a quaternion-based backstepping method [18]. A variable parameter integral backstepping controller [19] was presented for an unmanned helicopter equipped with multi-link arms to perform structure assembly and manipulation tasks. An advanced controller, including an attitude and a rate command augmentation system, a feedforward torque compensation system, and an L_1 adaptive augmentation law, was designed to simultaneously address challenges of maximizing workspace in constrained spaces and achieving stable flights when lifting payloads with unknown masses [20].

However, these controllers employed manual tuning of control parameters, which may severely hinder their applications in practice. One reason is that the robot configuration or the environment is not always unchanged in practice. As a well-tuned control parameter set is generally tuned for a specific dynamic model case, the practical time-varying dynamics could easily make the current controller no longer suitable. To address this problem, the manual tuning needs to be frequently repeated; however, it requires a large amount of time like days and expert knowledge. In addition, practical applications often require the controller to satisfy multiple but often conflicting goals like both a good task accuracy and a short completion time. This is much more difficult for the manual tuning compared with satisfying only one single objective. Even if the previous requirements are all achievable and acceptable, the manual tuning could be still difficult or even impossible due to human's own limit, named bounded rationality [21][22], in handling complex systems. For example, humans need to tune 27 parameters for a 6-DOFs quadrotor equipped with a 3-DOFs robotic arm using a proportional-integral-derivative (PID)

controller; however, the bounded rationality argues that humans begin to struggle when 4 variables are involved in problem solving and more than 5 variables to be solved at the same time nearly impossible.

One way to solve the manual tuning problems is automatically tuning parameters based on multi-objective optimization (MOO) and its effectiveness has been proven by other practical applications such as automatic voltage regulator system [23], chemical titration and neutralization plant [24], and linear brushless DC motor [25]. Although the optimization of gains control is not a new idea, it has never been studied for aerial manipulators. As a useful tool for practical aerial tasks, it is meaningful to formulize the specific MOO based auto-tuning method for the aerial manipulator and validate its feasibility since the aerial manipulator is much more complex than previous systems [23][24][25]. Thus, we propose a new MOO based control parameters auto-tuning method for the complex aerial manipulator system with specific problem formulizations. It is tested in two conventional parallel PID controllers that are designed for motion controls of a quadrotor and a 3-DOFs robotic arm, respectively. Although the PID controller is basic and simple, it is proved to be useful with good performances in practical applications. For each PID controller, an evolutionary algorithm based optimization, NSGA (Non-Dominated Sorting in Genetic Algorithms)-II, is used to automatically find optimal control parameters, avoiding the time-consuming manual tuning process. Two conflicting objectives, reliable static-dynamic performance and smooth control, are considered as an example of possibly conflicting performance criteria. The main contributions are considered as: (1) The work provides a new multi-objective-optimization based automatic parameters tuning in order to stably control the highly nonlinear and complex aerial manipulator system with possible dynamic changes in many different situations. (2) This method can be easily transferred to similar aerial manipulators with

different UAVs and robotic arms using any controllers. This work does not need structure modifications or additional mechanical designs [15], as the problem is fully solved through the control algorithms. (3) This work investigates meanings of multiple equivalent optimal solutions, as well as the consequences of adding new objectives. This could be used as a starting point for users to better understand how to select an appropriate set of control parameters according to their own requirements.

5.3 Mathematical Modeling

As the aerial manipulator modeling is based on a fundamental but common Euler-Lagrange method, we only explain necessary kinematic and dynamic models, which have been studied in previous work [12-14, 16-18], for the following controller design.

5.3.1 Kinematics for the Quadrotor-Arm System

With the aerial manipulator system consisting of a quadrotor and a robotic arm shown in Figure 5.1, O_w, O_b, and O_i represent the world frame (inertial frame), body frame, and link i frame, where $i = 1, 2, 3$ denotes the link number. If the inertial frame is arbitrarily chosen while the body frame and link frames are located at the center of gravity (CoG) of their respective rigid bodies, a vector including all the generalized coordinate variables can be defined as

$$\boldsymbol{q} = [\boldsymbol{p}^T \ \boldsymbol{\Omega}^T \ \boldsymbol{\eta}^T]^T, \tag{5.1}$$

where $\boldsymbol{p} = [x \ y \ z]^T$ represents the position of the CoG of quadrotor in the world frame, $\boldsymbol{\Omega} = [\phi \ \theta \ \psi]^T$ indicates the Euler angles of the quadrotor, and $\boldsymbol{\eta} = [\eta_1 \ \eta_2 \ \eta_3]^T$ means the joint angles of the 3-DOFs manipulator, which are defined about the positive y_b axis.

The relationship between the velocities in different frames is given as follows

$$\dot{\boldsymbol{p}} = \boldsymbol{R}\dot{\boldsymbol{p}}^b, \tag{5.2}$$

$$\boldsymbol{\omega} = \boldsymbol{R}\boldsymbol{\omega}^b, \tag{5.3}$$

$$\boldsymbol{\omega} = \boldsymbol{T}\dot{\boldsymbol{\Omega}}, \tag{5.4}$$

$$\boldsymbol{\omega}^b = \boldsymbol{R}\boldsymbol{T}\dot{\boldsymbol{\Omega}} = \boldsymbol{Q}\dot{\boldsymbol{\Omega}}, \tag{5.5}$$

where $\dot{\boldsymbol{p}}$ and $\dot{\boldsymbol{p}}^b$ represent the translational velocity of the CoG of the quadrotor in the world and body-fixed frame, respectively. They are related by the rotation matrix $R \in SO(3)$. $\boldsymbol{\omega} = [\omega_x\ \omega_y\ \omega_z]$ and $\boldsymbol{\omega}^b = \left[\omega_x^b\ \omega_y^b\ \omega_z^b\right]$ mean the angle velocity of the quadrotor in the world and body fixed frame, respectively. They are also related by $\boldsymbol{R}$. Also, $\dot{\Omega}$ can be mapped to $\boldsymbol{\omega}$ by the transformation matrix $\boldsymbol{T}$. $\boldsymbol{Q} = \boldsymbol{R}\boldsymbol{T}$ is the relationship matrix between $\boldsymbol{\omega}^b$ and $\dot{\boldsymbol{\Omega}}$.

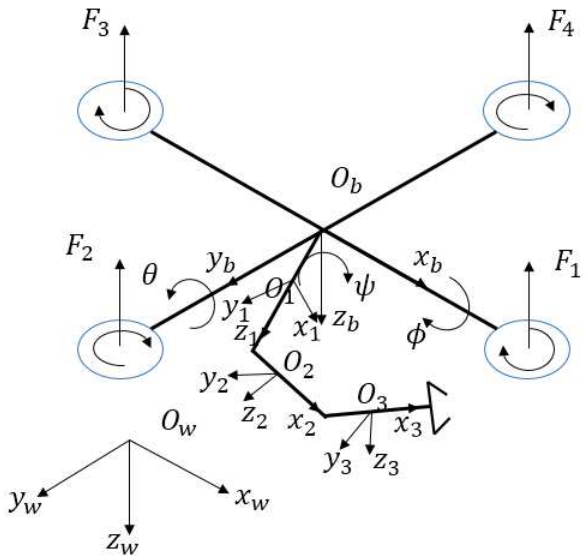

Figure 5.1 Coordinate configuration of the aerial manipulator system

Let p_i^b be the position of the CoG of link $i = 1, 2, 3$ in the body-fixed frame O_b. Then, $\boldsymbol{p}_i$, which is the position of the CoG of the link i in the world frame O_w, is related to $\boldsymbol{p}_i^b$ by $\boldsymbol{p}_i = \boldsymbol{p} + \boldsymbol{R}\boldsymbol{p}_i^b$. Also, if defining two Jacobian matrices $\boldsymbol{J}_t \in \mathbb{R}^{3\times3}$ and $\boldsymbol{J}_r \in \mathbb{R}^{3\times3}$, which are used to relate the translational and angular velocity of each manipulator link with $\dot{\boldsymbol{\eta}}$ by $\dot{\boldsymbol{p}}_i^b = \boldsymbol{J}_t\dot{\boldsymbol{\eta}}$ and $\boldsymbol{\omega}_i^b = \boldsymbol{J}_r\dot{\boldsymbol{\eta}}$, then we can get the following equations

$$\dot{\boldsymbol{p}}_i = \dot{\boldsymbol{p}} + \dot{\boldsymbol{R}}\boldsymbol{p}_i^b + \boldsymbol{R}\dot{\boldsymbol{p}}_i^b, \tag{5.6}$$

$$\dot{\boldsymbol{p}}_i = \dot{\boldsymbol{p}} + \widehat{\boldsymbol{\omega}}_b\boldsymbol{R}\boldsymbol{p}_i^b + \boldsymbol{R}\boldsymbol{J}_t\dot{\boldsymbol{\eta}}, \tag{5.7}$$

$$\boldsymbol{\omega}_i = \boldsymbol{\omega} + \boldsymbol{R}\boldsymbol{J}_r\dot{\boldsymbol{\eta}}. \tag{5.8}$$

For simplicity, (5.2), (5.4), (5.7), and (5.8) can be rewritten in the following matrix form [14].

$$\dot{\boldsymbol{p}} = [\boldsymbol{I}_{3\times3}\ \boldsymbol{0}_{3\times3}\ \boldsymbol{0}_{3\times3}]\dot{\boldsymbol{q}} = \boldsymbol{M}_{t,b}\dot{\boldsymbol{q}}, \tag{5.9}$$

$$\boldsymbol{\omega} = [\boldsymbol{0}_{3\times3}\ \boldsymbol{T}\ \boldsymbol{0}_{3\times3}]\dot{\boldsymbol{q}} = \boldsymbol{M}_{r,b}\dot{\boldsymbol{q}}, \tag{5.10}$$

$$\dot{\boldsymbol{p}}_i = \left[\boldsymbol{I}_{3\times3} - \left(\boldsymbol{R}\boldsymbol{p}_i^b\right)^{\wedge}\boldsymbol{T}\ \boldsymbol{R}\boldsymbol{J}_{t,i}\right]\dot{\boldsymbol{q}} = \boldsymbol{M}_{t,i}\dot{\boldsymbol{q}}, \tag{5.11}$$

$$\boldsymbol{\omega}_i = \left[\boldsymbol{0}_{3\times3}\ \boldsymbol{T}\ \boldsymbol{R}\boldsymbol{J}_{r,i}\right]\dot{\boldsymbol{q}} = \boldsymbol{M}_{r,i}\dot{\boldsymbol{q}}, \tag{5.12}$$

where the subscript $\cdot\times\cdot$ represents the size of $\boldsymbol{I}$ and $\boldsymbol{0}$. $\wedge$ is the operator that converts a vector into a skew-symmetric matrix. $\boldsymbol{J}_{t,i}$ and $\boldsymbol{J}_{r,i}$ are previously defined Jacobian matrices for link i. Through equations (5.9) - (5.12), $\dot{\boldsymbol{q}}$ can be easily mapped into the translational and angular velocities of the quadrotor and each link in the inertial coordinate frame.

5.3.2 Dynamics for the Quadrotor-Arm System

Given the Euler-Lagrange method provides an easy way to derive the system dynamics, the following Lagrange-D' Alembert equation is used.

$$\frac{d}{dt}\frac{\partial\mathcal{L}}{\partial\dot{q}} - \frac{\partial\mathcal{L}}{\partial q} = u + u_{ext}, \tag{5.13}$$

$$\mathcal{L} = \mathcal{K} - \mathcal{U}, \tag{5.14}$$

where $\mathcal{K}$ and $\mathcal{U}$ are the total kinetic and potential energy of the combined system, $\boldsymbol{u}$ represents the generalized force, and $\boldsymbol{u_{ext}}$ indicates external disturbance applied to the system, such as wind and the interaction with an external object.

The total kinetic energy is contributed by the quadrotor and each individual link, so the total kinetic energy $\mathcal{K}$ is expressed as follows:

$$\mathcal{K} = \frac{1}{2}\dot{\boldsymbol{p}}^T m_b \dot{\boldsymbol{p}} + \frac{1}{2}\dot{\boldsymbol{\Omega}}^T \boldsymbol{T}^T \boldsymbol{R} \boldsymbol{I}_b \boldsymbol{R}^T \boldsymbol{T} \dot{\boldsymbol{\Omega}} + \sum_{i=1}^{3} \frac{1}{2}\dot{\boldsymbol{p}}_i^T \boldsymbol{m}_i \dot{\boldsymbol{p}}_i + \frac{1}{2}\boldsymbol{\omega}_i^T (\boldsymbol{R}\boldsymbol{R}_i)\boldsymbol{I}_i(\boldsymbol{R}\boldsymbol{R}_i)^T \boldsymbol{\omega}_i, \quad (5.15)$$

where m is the mass and $\boldsymbol{I}$ is the inertia matrix. The subscripts b and i indicate the corresponding values with respect to O_b and O_i respectively. $\boldsymbol{R}_i$ is the coordinate transformation matrix from link i of the manipulator to the quadrotor body-fixed coordinates.

Similarly, the total potential energy is contributed by the potential energy of the quadrotor and each link, which is described by the equation below

$$\mathcal{U} = m_b g \boldsymbol{e}_3^T \boldsymbol{p} + \sum_{i=1}^{3} m_i g \boldsymbol{e}_3^T (\boldsymbol{p} + \boldsymbol{R}\boldsymbol{p}_i^b), \quad (5.16)$$

where the first and last terms are the potential energies of the quadrotor and link i, respectively. e_3 is the unit vector $[0\ 0\ 1]^T$, and g is the gravity coefficient.

By computing (5.13) and considering the Christoffel symbols of the first type [26], the dynamics equation which includes all components as one system can be derived as the following

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = u + u_{ext}, \quad (5.17)$$

where $\boldsymbol{M}(\boldsymbol{q}) \in \mathbb{R}^{9\times 9}$ is the inertia matrix which is positive definite and symmetric. $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})$ is the Coriolis matrix, and $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}}$ represents the Coriolis and Centrifugal forces. In addition, $\boldsymbol{G}(\boldsymbol{q})$ includes gravity effects at each joint [27]. $\boldsymbol{u}$ is the vector of inputs, i.e., $\boldsymbol{u} = \left[\boldsymbol{u}_f\ \boldsymbol{u}_\mu\ \boldsymbol{u}_\tau\right]^T = \left[\boldsymbol{R}\boldsymbol{f}_b^b\ \boldsymbol{R}^T\boldsymbol{T}\boldsymbol{\mu}_b^b\ \boldsymbol{\tau}\right]^T$, $\boldsymbol{\tau} \in \mathbb{R}^3$ is the vector of the manipulator joint torques, while $\boldsymbol{f}_b^b \in \mathbb{R}^3$ and $\boldsymbol{\mu}_b^b \in \mathbb{R}^3$ are the forces and torques generated by the four motors of the quadrotor, expressed in frame O_b.

The input vectors $\boldsymbol{f}_b^b$ and $\boldsymbol{\mu}_b^b$ can be expressed as $\boldsymbol{f}_b^b = [0\ 0\ f_z]^T$ and $\boldsymbol{\mu}_b^b = \left[\mu_\phi\ \mu_\theta\ \mu_\psi\right]^T$, where f_z is the total thrust applied by the rotors along z axis. Both f_z and $\boldsymbol{\mu}_b^b$ are related to the quadrotor's four motor actuation forces $\boldsymbol{f}$ via the following relation [28]

$$\begin{bmatrix} f_z \\ \mu_b^b \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & l & 0 & -l \\ -l & 0 & l & 0 \\ c & -c & c & -c \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} = \boldsymbol{\Xi f}, \tag{5.18}$$

where $l > 0$ is the distance from each motor to the vehicle center of mass, $c = \gamma_d/\gamma_t$, and γ_d, γ_t are the drag and thrust coefficient, respectively, f_i is the force generated by rotor i.

The matrices $\boldsymbol{M}(\boldsymbol{q})$, $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$, $\boldsymbol{G}(\boldsymbol{q})$ in equation (5.17) can be expressed by following equations. The derivation details can be found in previous work [12][13].

$$\boldsymbol{M}(\boldsymbol{q}) = \boldsymbol{M}_{t,b}^T m_b \boldsymbol{M}_{t,b} + \boldsymbol{M}_{r,b}^T \boldsymbol{R} \boldsymbol{I}_b \boldsymbol{R}^T \boldsymbol{M}_{r,b} + \sum_{i=1}^{3} \boldsymbol{M}_{t,i}^T m_i \boldsymbol{M}_{t,i} + \boldsymbol{M}_{r,i}^T (\boldsymbol{R}\boldsymbol{R}_i) \boldsymbol{I}_i (\boldsymbol{R}\boldsymbol{R}_i)^T \boldsymbol{M}_{r,i}, \tag{5.19}$$

$$c_{kj} = \sum_{i=1}^{9} \frac{1}{2} \left\{ \frac{\partial m_{kj}}{\partial \boldsymbol{q}_i} + \frac{\partial m_{ki}}{\partial \boldsymbol{q}_j} - \frac{\partial m_{ij}}{\partial \boldsymbol{q}_k} \right\}, \tag{5.20}$$

$$\boldsymbol{G}(\boldsymbol{q}) = \frac{\partial \mathcal{U}}{\partial \boldsymbol{q}}, \tag{5.21}$$

where $m_{\alpha\beta}$ indicates the element $\alpha\beta$ of the inertia matrix $\boldsymbol{M}(\boldsymbol{q})$.

5.4 Multi-Objective-Optimization-Based PID Control Scheme

This section focuses on explaining the MOO based auto-tuning method with conventional PID control designs. The combination of the PID controller and parameters auto-tuning method results in the multi-objective-optimization based PID control method for aerial manipulators.

5.4.1 PID Control Structure for Quadrotor-Arm System

Since we only discuss the stabilization of the quadrotor-arm system in this paper, the external disturbance/force in (5.17) is assumed to be zero. This assumption will not affect the validness of the proposed method because the external disturbance/force term can be moved to the left side of (5.17), which makes the design of control input $\boldsymbol{u}$ the same. In order to globally linearize the closed-loop dynamics, the following control input $\boldsymbol{u}$ can be considered

$$\boldsymbol{u} = \boldsymbol{M}(\boldsymbol{q})\boldsymbol{\sigma} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}), \tag{5.22}$$

where the auxiliary input $\boldsymbol{\sigma}$ can be partitioned, according to equation (5.1), i.e., $\boldsymbol{\sigma} = [\boldsymbol{\sigma_p}^T \ \boldsymbol{\sigma_\Omega}^T \ \boldsymbol{\sigma_\eta}^T]^T$, with $\boldsymbol{\sigma_\Omega} = [\sigma_\phi^T \ \sigma_\theta^T \ \sigma_\psi^T]^T$. Hence, the globally linearized closed-loop dynamics is given by $\ddot{\boldsymbol{q}} = \boldsymbol{\sigma}$, i.e.,

$$\begin{cases} \ddot{\boldsymbol{p}} = \boldsymbol{\sigma_p} \\ \ddot{\boldsymbol{\Omega}} = \boldsymbol{\sigma_\Omega} \\ \ddot{\boldsymbol{\eta}} = \boldsymbol{\sigma_\eta} \end{cases}. \tag{5.23}$$

Because $\boldsymbol{p}$, $\boldsymbol{\Omega}$, and $\boldsymbol{\eta}$ are vectors including more than one variable, a parallel PID controller structure [29] designed for each specific variable can be considered. Take the translational position x, which is an element in p, for example. Its corresponding PID controller is

$$u_{p_x} = \ddot{x}_{des} = k_{P_x}(x - x_{des}) + k_{I\_x}\int_0^t (x - x_{des})dt + k_{D\_x}(\dot{x} - \dot{x}_{des}), \tag{5.24}$$

where x_{des} is the desired quadrotor's x position, $\dot{x}_{des}$ is the rate of desired x position change, k_{P_x} is the proportional gain, k_{I_x} is integral gain, and k_{D_x} is the derivative gain. Other elements in $\boldsymbol{p}$, $\boldsymbol{\Omega}$, and $\boldsymbol{\eta}$ are not explained in detail here because they can be achieved in a similar way.

5.4.2 MOO-Based PID Control Parameters Auto-Tuning

As shown in Figure 5.2, the multi-objective evolutionary algorithm (MOEA) is used to automatically tune the PID control parameters. The integration of parameters auto-tuning process and the PID controller results in the MOO based PID controller for aerial manipulators. This controller includes two PID control loops. One is for the quadrotor and another one is for the robotic arm. Each PID controller has a set of control parameters. Because the quadrotor has 6 specific variables, so the quadrotor PID controller includes a set of (6*3) parameters. Similarly, the robotic arm PID controller contains a set of (3*3) parameters. Based on the difference between the desired and actual motions as well as the difference between current and previous control inputs, the MOEA finds optimal parameters to meet different criteria.

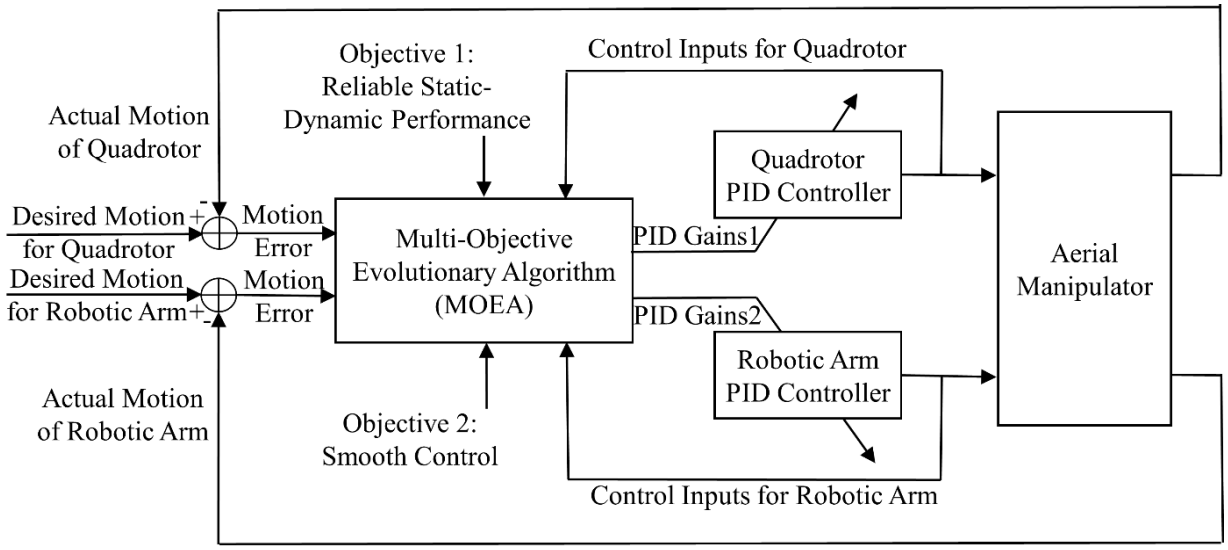

Figure 5.2 Multi-objective-optimization-based PID control structure for the quadrotor-arm system

Among all the MOEAs, we select the improved Non-Dominated Sorting in Genetic Algorithm (NSGA-II) [30] as the specific optimization algorithm due to its efficiency and the ease to be implemented in practical applications. While traditional optimization methods convert the multi-objective optimization to single-objective problems by emphasizing one particular Pareto-optimal solution at a time and running many times, the NSGA-II algorithm is more efficient because its evolutionary mechanism can find multiple Pareto-optimal solutions in one single run. Compared with other popular evolutionary algorithms developed in the same time period such as Pareto-archived evolution strategy (PAES) [31] and strength-Pareto evolutionary algorithm (SPEA) [32] and the original NSGA [33], NSGA-II has an outperforming performance in terms of finding a diverse set of solutions and in converging near the true Pareto-optimal set because it has a better sorting algorithm and incorporates elitism. It has also been proved to be easily implemented in practical problems like generation expansion planning [34], water resources management [35], and distributed hydrologic modeling [36] with good results. Due to the complexity of the aerial manipulator system, the auto-tuning process based on NSGA-II is off-line as the inside

evolutionary scheme costs considerable time like minutes or hours to find optima. In this work, it takes 30 minutes to find the optimal PID parameters.

Two objectives are considered for the NSGA-II algorithm. One is the reliable static-dynamic performance, and the other is the smooth control. Both these two objectives are important for the aerial manipulator system to keep stability, but they cannot be achieved simultaneously because they conflict with each other. In general, seeking for some dynamic specifications usually causes the large variance of control law and significant oscillating of the actuators. The performance balance between these two objectives will be discussed in the next section. The reliable static-dynamic performance is represented by $G_1 = \sum_{k=1}^{T} \sum_{i=1}^{9} e_i^2(k)$. Minimizing G_1 aims to provide good reference tracking and better disturbance rejection. The objective of smooth control is represented by $G_2 = \sum_{k=1}^{T} \sum_{j=1}^{9} \frac{1}{2} \big(u_j(k) - u_j(k-1)\big)^2$ so that minimizing G_2 aims to smooth the control signals and avoid significant oscillation of the actuators. Hence, the MOO based PID control problem can be mathematically formulized as

$$\begin{aligned} \min_{\{K_P, K_I, K_D\}} \; & G_1(\{K_P, K_I, K_D\}), G_2(\{K_P, K_I, K_D\}) \\ \text{s.t.} \quad & LB \le \{K_P, K_I, K_D\} \le UB \end{aligned}, \tag{5.25}$$

where $e_i(k)$ is the error between the desired and actual output, T is the number of total indices of the time discretization. $\{K_P, K_I, K_D\}$ groups all the PID control parameters together with a vector-shaped lower bound LB and upper bound UB.

5.5 Validation Simulations

In this work, all simulations are completed in MATLAB/Simulink. The parameters of the quadrotor-arm system [12] are listed in Table 5.1.

The parameters of NSGA-II algorithm are listed in Table 5.2. As mentioned before, the lower bound LB and upper bound UB should be vectors corresponding to each control variable. Because

we consider all the boundary values to be the same, there is only one value in Table 5.2, for simplicity. In fact, the bounds of the gain parameters may be difficult to be determined beforehand in practical situations. They are assigned values here only for a convenient calculation and can be practically expanded to a larger value to include more possible solutions. Other NSGA-II related parameters are empirically chosen.

Table 5.1 Quadrotor simulation parameters

Parameters	Values	Descriptions
I_{bx}, I_{by}	1.24 kg · m^2	Moment of inertia around quadrotor's X, Y axis
I_{bz}	2.48 kg · m^2	Moment of inertia around quadrotor's Z axis
m_b	2 kg	Mass of quadrotor
γ_t	3.13*10^(-5)	Thrust factor
γ_d	7.5*10^(-7)	Drag factor
l	0.1 m	Distance between the rotor center and quadrotor center
m_1, m_2, m_3	0.1 kg	Mass of the first, second, and third link of the arm
L_1, L_2, L_3	0.1 m	Length of the arm's first, second, and third link
I_{x1}, I_{y1}	0.0025 kg · m^2	Moment of inertia around first link's X, Y axis
I_{z1}	0.0075 kg · m^2	Moment of inertia around first link's Z axis
I_{x2}, I_{y2}	0.0025 kg · m^2	Moment of inertia around second link's X, Y axis
I_{z2}	0.0075 kg · m^2	Moment of inertia around second link's Z axis
I_{x3}, I_{y3}	0.0025 kg · m^2	Moment of inertia around third link's X, Y axis
I_{z3}	0.0075 kg · m^2	Moment of inertia around third link's Z axis

Table 5.2 Parameters used for NSGA-II

Descriptions	Values
Population size	60
Number of generations	50
Crossover probability	0.9
Mutation probability	0.33
Lower bounds of the gain parameters	[0 0 0]
Upper bounds of the gain parameters	[50 50 50]

In order to compare the performance of different PID parameters, the quadrotor performs a basic but important action within the aerial manipulation. Specifically, the quadrotor is supposed to follow a planar and circular trajectory: $x = \sin(0.2t)$, $y = \cos(0.2t)$, $z = 1$. The initial configuration of the aerial manipulator system is $\boldsymbol{p} = [0, 0, 3]$ (m), $\boldsymbol{\Omega} = [0, 0, 0]$ (degrees), and $\boldsymbol{\eta} = [-45, 45, 45]$ (degrees). The arm moves from configuration A to configuration B ($\boldsymbol{\eta} = [0, 45, 45]$). This movement is intentionally designed to include a stable tracking process in the whole simulation. The total simulation time is 32s, corresponding to a complete circle, and the sampling time is 0.02s. A visualization of the simulated process can be seen in Figure 5.3. The arm should move from initial configuration A (red solid lines) to desired configuration B (green dotted lines) with the quadrotor following the circular trajectory (purple dashed lines).

In the next section, we will show a comparison between results using manual-tuned parameters (such as using Ziegler-Nichols method [37]) and auto-tuned parameters. In order to further discuss the performance of the automatic tuning method, we will also provide the simulation results of a state-of-the-art control method, model predictive control (MPC). MPC is an advanced process control method which calculates the future state of the system and organizes the control action accordingly. It predicts the future of the system and sends control signals in such a manner that it reduces a cost function defined as the error between the output and the desired tracking point over a particular prediction horizon [38]. It has been proven in many complex systems [38-41] to provide better performances, especially stronger robustness against uncertainties, than a majority of current control methods like PID control and sliding mode control. Two key parameters, the control horizon (M) and the prediction horizon (N), are both chosen to be 10 in this paper.

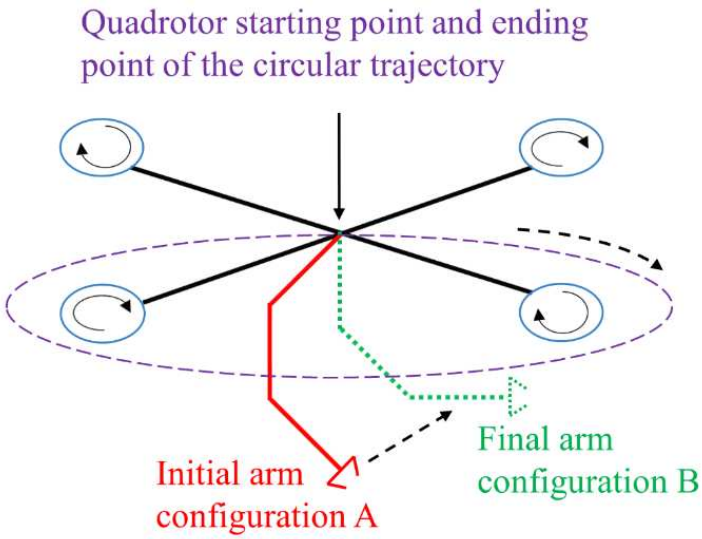

Figure 5.3 Visualization of simulated movements. The quadrotor is designed to follow a circular trajectory. The arm is designed to move from configuration A to configuration B when the quadrotor starts flying.

5.6 Results

(1) The Necessity of Automatic Tuning

The performances of several different sets of manual-tuned PID parameters are shown in Table 5.3. Note that the dynamic responses for different PID parameter sets are totally different. For example, the overshoot of link 1 for parameter set 1 is 3.4 degrees less than set 2, but the stabilization time of link 1 is 7.3 seconds longer than for set 2. This makes sense because the overshoot usually behaves inversely with respect to the stabilization time. If we want the aerial manipulator system to stabilize quickly, we need to have a larger control output during the beginning phase, which leads to a larger overshoot. This makes it difficult to judge which parameter set is better because each application has a specific requirement. Just for an illustration, we will consider the less overshoot of links as a better performance in the remaining paper. Due to a careful manual tuning by experts, the performances of set 3 and set 4 are much better than set 1 and set 2. Considering the link's overshoot, set 4 is the best in all the four sets.

Table 5.3 Manually tuned PID parameter values and corresponding performances (each link length L=0.1m, each link mass m=0.1kg)

		Set 1	Set 2	Set 3	Set 4
Parameters	$[K_{P\_x}, K_{I\_x}, K_{D\_x}]$	[8, 0, 2]	[8, 0, 2]	[2, 0, 2]	[2, 0, 2]
	$[K_{P\_y}, K_{I\_y}, K_{D\_y}]$	[8, 0, 2]	[8, 0, 2]	[2, 0, 2]	[2, 0, 2]
	$[K_{P\_z}, K_{I\_z}, K_{D\_z}]$	[8, 0, 2]	[8, 0, 2]	[2, 0, 2]	[2, 0, 2]
	$[K_{P\_\phi}, K_{I\_\phi}, K_{D\_\phi}]$	[10, 0, 2]	[10, 0, 2]	[10, 0, 2]	[10, 0, 2]
	$[K_{P\_\theta}, K_{I\_\theta}, K_{D\_\theta}]$	[10, 0, 2]	[10, 0, 2]	[10, 0, 2]	[10, 0, 2]
	$[K_{P\_\psi}, K_{I\_\psi}, K_{D\_\psi}]$	[10, 0, 2]	[10, 0, 2]	[10, 0, 2]	[10, 0, 2]
	$[K_{P\_\eta_1}, K_{I\_\eta_1}, K_{D\_\eta_1}]$	[1, 0, 2]	[15, 0, 3]	[1, 0, 2]	[15, 0, 3]
	$[K_{P\_\eta_2}, K_{I\_\eta_2}, K_{D\_\eta_2}]$	[1, 0, 2]	[15, 0, 3]	[1, 0, 2]	[15, 0, 3]
	$[K_{P\_\eta_3}, K_{I\_\eta_3}, K_{D\_\eta_3}]$	[1, 0, 2]	[15, 0, 3]	[1, 0, 2]	[15, 0, 3]
Performances	Overshoot of link 1 (degrees)	20.1	23.5	16.2	23.1
	Stabilization time of link 1 (seconds)	10.6	3.3	7.4	2.8
	Overshoot of link 2 (degrees)	11.6	1.6	5.1	0.6
	Stabilization time of link 2 (seconds)	10.2	2.8	7.6	1.4
	Overshoot of link 3 (degrees)	15.2	2.1	9.0	1.1
	Stabilization time of link 3 (seconds)	14.1	3.5	11.5	2.4
	Maximum absolute deviation of quadrotor's positions (meters)	0.14	0.08	0.12	0.08
	Maximum absolute deviation of quadrotor's angles (degrees)	9.1	6.2	11.6	6.3

If we change the values of arm length and mass to be $L_1 = L_2 = L_3 = 0.05$m, $m_1 = m_2 = m_3 = 0.05$kg, the results, obtained by using the same parameter sets as in Table 5.3, are shown in Table 5.4. Although employing the same control parameters, the dynamic responses of the new aerial manipulator system have changed. For instance, the overshoot of link 1 for the new system is 8.9 degrees when using parameter set 4. It is 4.3 degrees larger than the corresponding value in Table 5.3. If there is a strict limitation on the link overshoot in practice like flying through a narrow

hallway, set 4's performance can be not acceptable. If considering the link's overshoot of all the four sets for the new arm, the best set changes to set 3. This change indicates that the control parameters may need to be tuned each time because the quadrotor may carry different robotic arms to execute different tasks. Unfortunately, this frequent parameter tuning is not always guaranteed with the manual tuning method due to the lack of experts and the high costs of time.

Table 5.4 PID parameter values and corresponding performances for different arm settings (each link length L=0.05m, each link mass m=0.05kg)

		Set 1	Set 2	Set 3	Set 4
Parameters	$[K_{P\_x}, K_{I\_x}, K_{D\_x}]$	[8, 0, 2]	[8, 0, 2]	[2, 0, 2]	[2, 0, 2]
	$[K_{P\_y}, K_{I\_y}, K_{D\_y}]$	[8, 0, 2]	[8, 0, 2]	[2, 0, 2]	[2, 0, 2]
	$[K_{P\_z}, K_{I\_z}, K_{D\_z}]$	[8, 0, 2]	[8, 0, 2]	[2, 0, 2]	[2, 0, 2]
	$[K_{P\_\phi}, K_{I\_\phi}, K_{D\_\phi}]$	[10, 0, 2]	[10, 0, 2]	[10, 0, 2]	[10, 0, 2]
	$[K_{P\_\theta}, K_{I\_\theta}, K_{D\_\theta}]$	[10, 0, 2]	[10, 0, 2]	[10, 0, 2]	[10, 0, 2]
	$[K_{P\_\psi}, K_{I\_\psi}, K_{D\_\psi}]$	[10, 0, 2]	[10, 0, 2]	[10, 0, 2]	[10, 0, 2]
	$[K_{P\_\eta_1}, K_{I\_\eta_1}, K_{D\_\eta_1}]$	[1, 0, 2]	[15, 0, 3]	[1, 0, 2]	[15, 0, 3]
	$[K_{P\_\eta_2}, K_{I\_\eta_2}, K_{D\_\eta_2}]$	[1, 0, 2]	[15, 0, 3]	[1, 0, 2]	[15, 0, 3]
	$[K_{P\_\eta_3}, K_{I\_\eta_3}, K_{D\_\eta_3}]$	[1, 0, 2]	[15, 0, 3]	[1, 0, 2]	[15, 0, 3]
Performances	Overshoot of link 1 (degrees)	24.8	23.6	18.1	23.7
	Stabilization time of link 1 (seconds)	6.8	3.4	9.2	3.5
	Overshoot of link 2 (degrees)	14.2	2.2	7.9	2.1
	Stabilization time of link 2 (seconds)	7.1	3.8	4.5	3.4
	Overshoot of link 3 (degrees)	18.4	3.3	19.3	3.6
	Stabilization time of link 3 (seconds)	7.8	4.4	8.2	2.3
	Maximum absolute deviation of quadrotor's positions (meters)	0.19	0.06	0.013	0.07
	Maximum absolute deviation of quadrotor's angles (degrees)	14.2	8.2	13.8	8.1

The results from both Table 5.3 and Table 5.4 show the importance and necessity of good tuning of PID control parameters. The well-tuned parameters lead to better control performances, generally satisfying the goals of each task. The major drawback of manual tuning is the time-consuming involvement of humans, which could be unacceptable in practical applications. Hence, it is necessary and convenient for the controller to have a set of automatically tuned parameters, adapting to different environments and systematical uncertainties.

(2) Performance Evaluation of MOO Based PID Control

In this paper, auto-tuning is achieved using multi-objective optimization, NSGA-II. In fact, there are multiple sets of PID parameters that are obtained by NSGA-II. The set of these solutions, called the Pareto front [42], can be seen in Figure 5.4. All these solutions are regarded as optimal because all of them satisfy the two objectives simultaneously and no objective can be improved without sacrificing the other objective. The differences between solutions indicate various trade-offs between the objectives.

The two axes of Figure 5.4 are two objectives, G_1 and G_2. As shown in Figure 5.4, there are two significant points, A and B. The values of the two objectives G_1 and G_2 at point A are 90000 and 0.0520 while the two values at point B are 113000 and 0.0031. This difference is large enough to show a trade-off between objectives G_1 and G_2 at points A and B. The solution at point A focuses more on minimizing the integrated time square error (G_1) while the solution at point B makes more efforts to keep the control rate (G_2) minimal. In fact, the trade-offs can also be indicated by other solutions. The values of objective G_2 decreases quickly as objective G_1 increases to the left of point A. To the right of point B, the value of objective G_2 almost remains the same even while objective G_1 increases. Between points A and B, we have a smooth change

of objective G_2 with respect to objective G_1. The solutions between points A and B can be regarded as considering both objectives to be approximately equal, which is optimal for most situations.

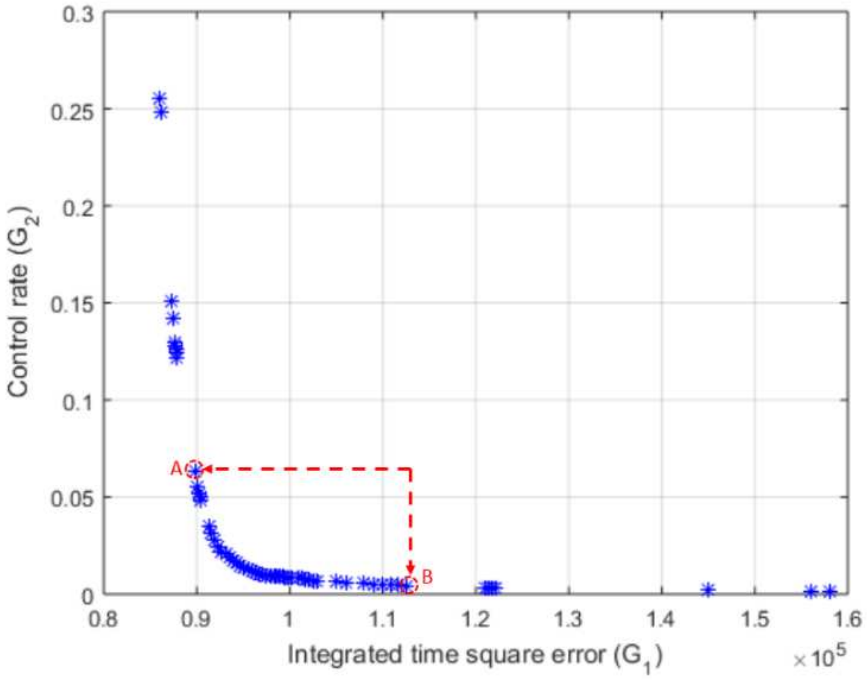

Figure 5.4 Pareto front for multi-objective optimization in terms of integrated time square error and control rate. The solutions to the left of point A focus more on minimizing the integrated time square error while the solutions to the right of point B focus more on minimizing the control rate. The solutions between point A and B are more balanced between the integrated time square error and control rate.

Due to the trade-offs between two objectives in the Pareto front, the dynamic responses of each solution are totally different. For simplicity, we randomly pick two solutions between points A and B out of all the solutions to analyze their performances. The performance comparison can be seen in Table 5.5. Link 1's overshoot using parameter set 6 is zero, which is better than set 5's 1.6 degrees. However, the stabilization time for link 1 using set 6 is 1.3 seconds longer than set 5. The conflicts and trade-offs can give users some suggestions on selecting the most suitable parameters. If users focus more on the control smoothness, then parameter sets close to point B in

Figure 5.4 are recommended. If the integrated time square error is more important, then parameter sets close to point A are suggested.

Table 5.5 Performances of auto-tuned PID parameter sets 5 and 6

		Set 5	Set 6
Parameters	$[K_{P\_x}, K_{I\_x}, K_{D\_x}]$	[13.15, 0.07, 2.94]	[15.84, 0.06, 3.02]
	$[K_{P\_y}, K_{I\_y}, K_{D\_y}]$	[11.38, 0.03, 3.25]	[13.62, 0.04, 4.01]
	$[K_{P\_z}, K_{I\_z}, K_{D\_z}]$	[18.46, 0.06, 3.99]	[20.06, 0.03, 4.49]
	$[K_{P\_\phi}, K_{I\_\phi}, K_{D\_\phi}]$	[26.82, 0.09, 5.01]	[25.53, 0.07, 3.41]
	$[K_{P\_\theta}, K_{I\_\theta}, K_{D\_\theta}]$	[29.10, 0.11, 5.18]	[28.61, 0.06, 3.93]
	$[K_{P\_\psi}, K_{I\_\psi}, K_{D\_\psi}]$	[32.86, 0.09, 6.20]	[29.84, 0.05, 4.07]
	$[K_{P\_\eta_1}, K_{I\_\eta_1}, K_{D\_\eta_1}]$	[20.34, 0.01, 9.54]	[24.58, 0.01, 15.54]
	$[K_{P\_\eta_2}, K_{I\_\eta_2}, K_{D\_\eta_2}]$	[19.42, 0.01, 8.98]	[21.67, 0.01, 13.45]
	$[K_{P\_\eta_3}, K_{I\_\eta_3}, K_{D\_\eta_3}]$	[19.98, 0.01, 9.06]	[23.12, 0.01, 13.33]
Performances	Overshoot of link 1 (degrees)	1.6	0
	Stabilization time of link 1 (seconds)	2.1	3.4
	Overshoot of link 2 (degrees)	0.32	0.56
	Stabilization time of link 2 (seconds)	5.8	8.2
	Overshoot of link 3 (degrees)	0.3	0.44
	Stabilization time of link 3 (seconds)	5.8	7.0
	Maximum absolute deviation of quadrotor's positions (meters)	0.064	0.064
	Maximum absolute deviation of quadrotor's angles (degrees)	2.5	2.6

(3) Performance Comparison with Manual Tuning and MPC

For convenience, we directly use the auto-tuned PID parameter set 5 in Table 5.5 to compare its performances with those of other different control methods. The performance comparison between the auto-tuned parameter set 5 and manual-tuned parameter set (e.g. set 1 and set 4 in Table 5.3) is shown in Table 5.6. Also, as mentioned at the end of simulation setup, we add the MPC simulation results in Table 5.6 for comparison. Additionally, Figure 5.5 is presented to show

the change of joint angles using PID parameter set 1, set 4, set 5, and MPC. Correspondingly, Figure 5.6 shows the quadrotor trajectories. Figure 5.7 uses a bar plot to present a more intuitive comparison of various system responses in Table 5.6. This case uses the original aerial manipulator system parameters in Table 5.1.

Table 5.6 Performance comparison of PID parameter set 1, 5 (auto-tuned) and MPC

		Set 1	Set 5	MPC
Parameters	$[K_{P\_x}, K_{I\_x}, K_{D\_x}]$	[8, 0, 2]	[13.15, 0.07, 2.94]	M=10 N=10
	$[K_{P\_y}, K_{I\_y}, K_{D\_y}]$	[8, 0, 2]	[11.38, 0.03, 3.25]	
	$[K_{P\_z}, K_{I\_z}, K_{D\_z}]$	[8, 0, 2]	[18.46, 0.06, 3.99]	
	$[K_{P\_\phi}, K_{I\_\phi}, K_{D\_\phi}]$	[10, 0, 2]	[26.82, 0.09, 5.01]	
	$[K_{P\_\theta}, K_{I\_\theta}, K_{D\_\theta}]$	[10, 0, 2]	[29.10, 0.11, 5.18]	
	$[K_{P\_\psi}, K_{I\_\psi}, K_{D\_\psi}]$	[10, 0, 2]	[32.86, 0.09, 6.20]	
	$[K_{P\_\eta_1}, K_{I\_\eta_1}, K_{D\_\eta_1}]$	[1, 0, 2]	[20.34, 0.01, 9.54]	
	$[K_{P\_\eta_2}, K_{I\_\eta_2}, K_{D\_\eta_2}]$	[1, 0, 2]	[19.42, 0.01, 8.98]	
	$[K_{P\_\eta_3}, K_{I\_\eta_3}, K_{D\_\eta_3}]$	[1, 0, 2]	[19.98, 0.01, 9.06]	
Performances	Overshoot of link 1 (degrees)	20.1	1.6	0
	Stabilization time of link 1 (seconds)	10.6	2.1	3.5
	Overshoot of link 2 (degrees)	11.6	0.32	0.18
	Stabilization time of link 2 (seconds)	10.2	5.8	5.7
	Overshoot of link 3 (degrees)	15.2	0.3	0.19
	Stabilization time of link 3 (seconds)	14.1	5.8	5.6
	Maximum absolute deviation of quadrotor's positions (meters)	0.14	0.064	0.060
	Maximum absolute deviation of quadrotor's angles (degrees)	9.1	2.5	3.2

It is worthy to notice that all the performances of parameter set 5 are better than set 1, as the optimization method is used. It can be seen in Figure 5.5 that there are less oscillations of parameter set 5 than set 1. This means the changes of control inputs are less and smoother, meeting objective G_2. The integrated time square error of set 5 can also be clearly seen to be much less than set 1,

satisfying objective G_1. Although there is a relatively larger overshoot for link 1, the stabilization time of parameter set 5 is shorter than MPC. The performance of auto-tuned set 5 is also better than best manual-tuned set 4 although they are close. This indicates expert-tuned results may still be not optimal and can be improved.

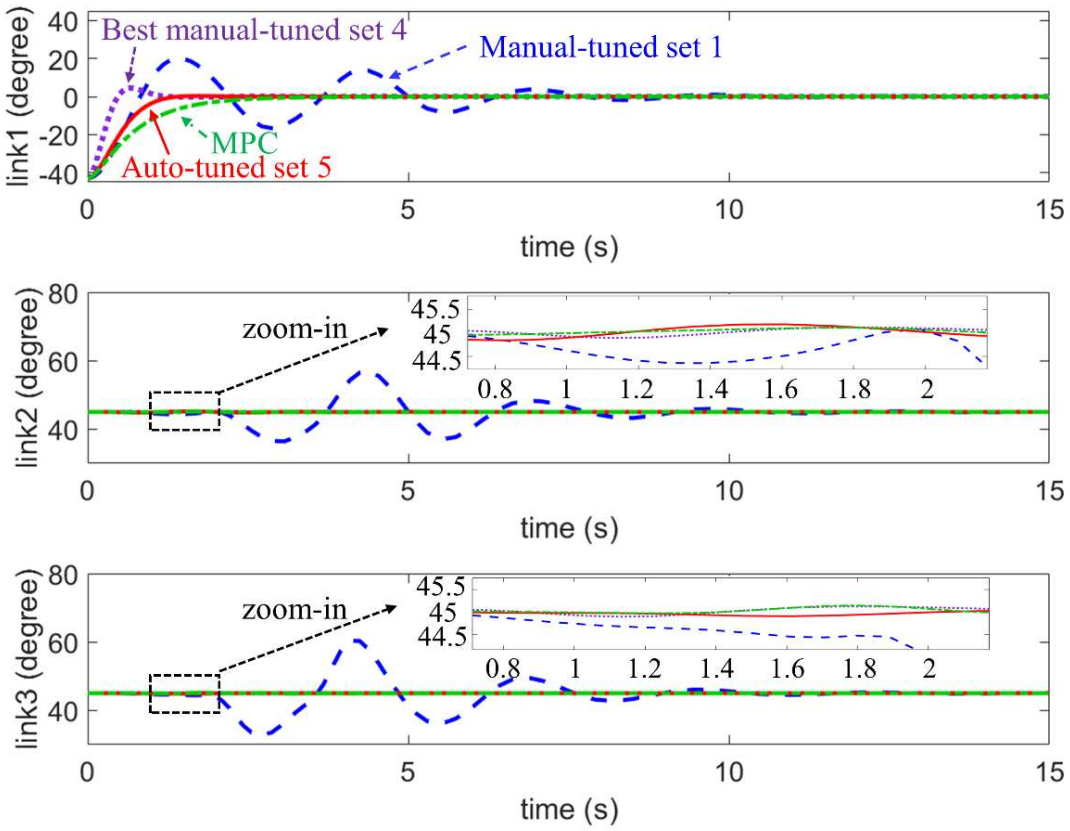

Figure 5.5 Link performance comparison using PID parameter set 1 (manual-tuned), 4 (best manual-tuned), 5 (auto-tuned), and MPC. The auto-tuned performances are comparable to MPC because the differences between the red solid lines and green dash-dot lines are small. Although the best manual-tuned performances (purple dotted lines) are close to auto-tuned and MPC performances for links 2 and 3, the best manual-tuned link 1 performance is notably worse than the auto-tuned and MPC performance. The manual-tuned performances (blue dashed lines) are worst with significant oscillations.

From Figure 5.6, all the quadrotor trajectories have an obvious deviation at the first several seconds. However, the trajectory using PID parameter set 5 and MPC quickly follows the desired trajectory around 6s while the one using PID parameter set 1 exactly tracks around 11s. Hence, the

performance of parameter set 5 can be regarded as comparable to MPC. There is no doubt that the performance of MPC will be better, possibly outperforming our MOO based PID method, if increasing the value of the control horizon (M) and the prediction horizon (N). However, it multiples the time used to find solutions, which is 15 minutes longer than the MOO based PID method (the offline optimization takes 30 minutes) in current MPC setting. In terms of both time cost and performance, the MOO based PID method is preferred.

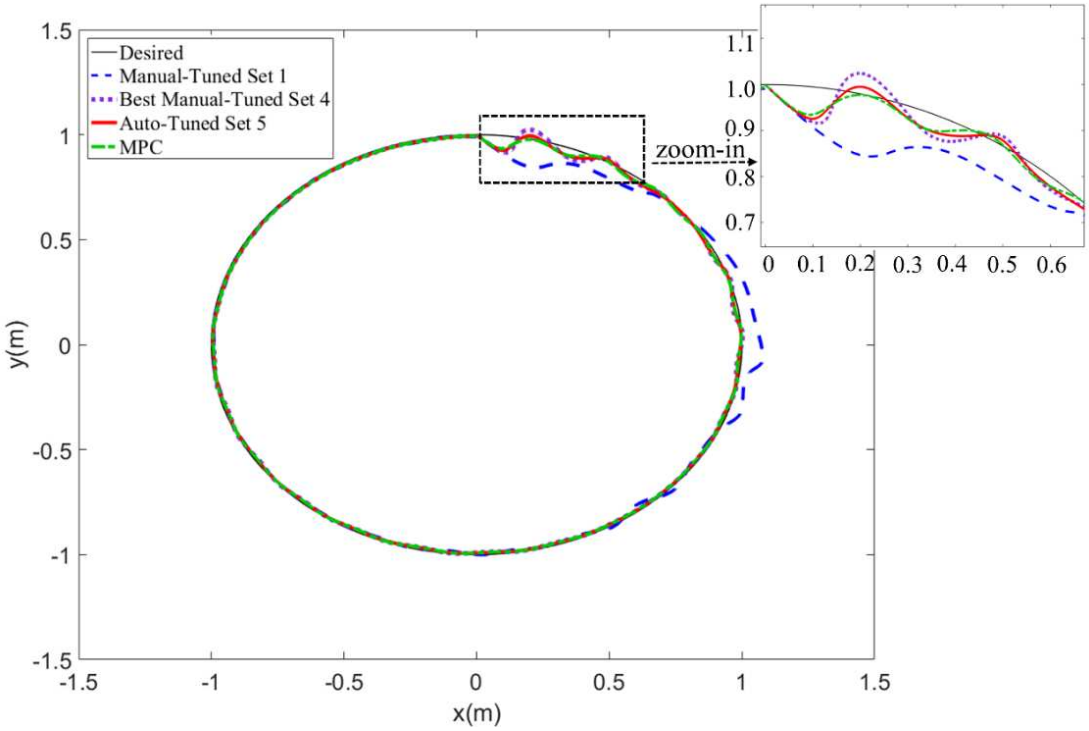

Figure 5.6 Quadrotor trajectories using PID parameter set 1 (manual-tuned), 4 (best manual-tuned), 5 (auto-tuned), and MPC. The auto-tuned parameter set 5 has comparable performances with MPC, but better performances than manual tuned parameter sets in terms of less deviations and faster stabilization time.

Since there may be more than two objectives in practical applications, we also simulate a case with three objectives by adding a new one specifying no overshoot for the link 1 joint angle. Similar to the two-objective case, there is also a Pareto front with numerous solutions. For convenience, we picked out a solution which happens to be the same with parameter set 6 in Table

5.5. It does not have an overshoot for link 1's joint angle, which is verified to meet the newly added objective. However, the three-objective optimization needs a longer tuning time for the whole simulation, which is about 3 times than the time used for the two-objective case. This indicates that we can add more and new objectives into the MOO based PID control formulation, i.e., equation (5.25), but the feasible number of objectives may have an upper limit due to the additional computational complexity. MPC needs much more time than MOO based PID if adding new objectives due to its own high complexity, which sometimes limits its practical application. Also, the PID controller is most widely used controller in the industry, it may be easier and more convenient to keep the PID structure unchanged rather than developing a new and complex MPC controller.

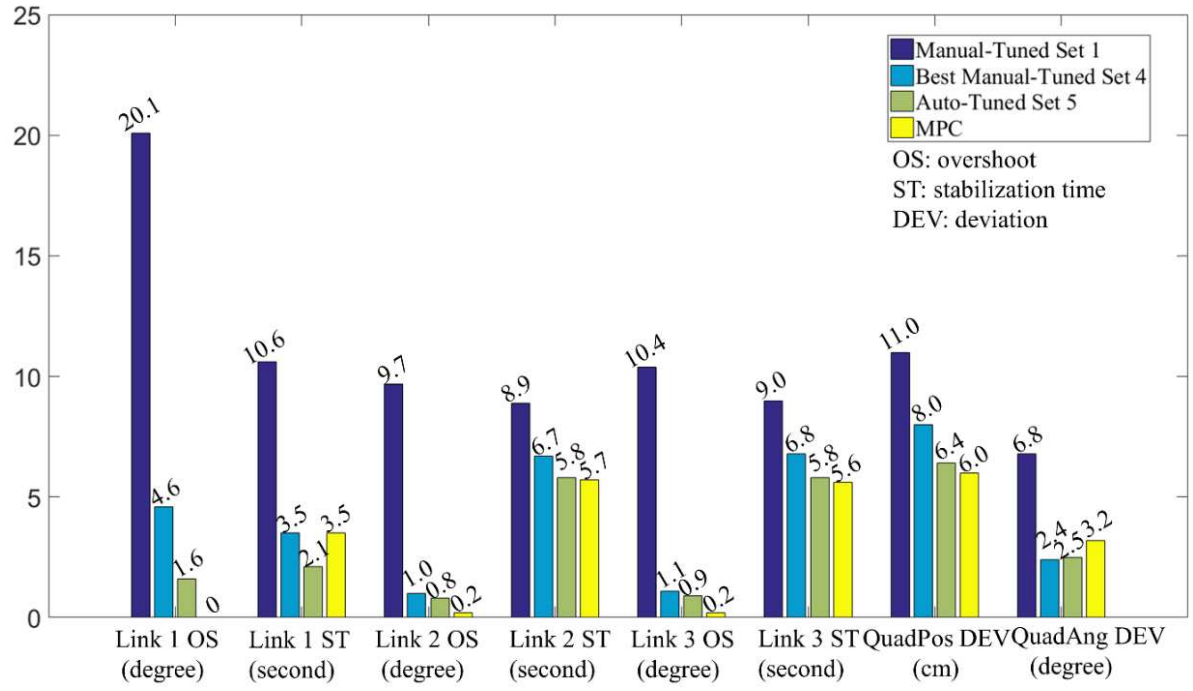

Figure 5.7 Performance comparison of PID parameter set 1 (manual-tuned), 4 (best manual-tuned), 5 (auto-tuned), and MPC. All performances including the overshoot, stabilization time, quadrotor position deviation, and quadrotor angle deviation are considered better with smaller values. The auto-tuned and MPC performances are comparable while they are better than the manual-tuned performances.

5.7 Conclusion

In this paper, we present a multi-objective-optimization based control parameters auto-tuning method for the aerial manipulator system to be better applied in practical applications. This method uses NSGA-II, a multi-objective evolutionary algorithm, to automatically find optimal parameters for different control goals which are related to the desired and actual measurements of the aerial manipulator system. The auto-tuning performances are comparable to a well-designed state-of-the-art model predictive controller while both are better than manual-tuned performances.

The reasons about the better performances of auto-tuned controllers can be concluded as (1) the flexibility to changing conditions and (2) inherent satisfaction of multiple objectives. The practical changing conditions (including both robot dynamics and task environments) easily make current manual-tuned controllers which are designed for specific configurations or tasks not suitable. In contrast, the auto-tuning method has more flexibilities to these changes due to the inside optimization mechanism. Even for the case that humans can tune parameters for each situation, although this is nearly impossible, it costs significantly more time than the auto-tuning method. Furthermore, it is sometimes extremely difficult for even experts to tune appropriate parameters for multiple control objectives. Humans may not be able to quantitatively and precisely balance between different control objectives due to human's fuzzy logic and approximate reasoning. Instead, the auto-tuning method can use an exact mathematical form to accurately evaluate trade-offs between different objectives. Compared with the advanced MPC, the MOO based auto-tuning method is easier to be implemented and extended due to less computational complexity and more flexibility to add additional objectives.

Although the proposed MOO based auto-tuning method is an off-line method, it can still be applied to many practical applications that require relatively less time sensitivity. If the aerial

manipulator is used to transport different loads in different tasks, for example, then this auto-tuning method could be more efficient than a manual-tuned aerial manipulator to adapt to a specific load. Moreover, there could be different combinations of UAVs and robotic arms to generate an aerial manipulator system. It will be easier to use the auto-tuning method to handle such a large variety.

Despite the advantages, the MOO based auto-tuning method can still be improved. For example, it is worth investigating how to select the best solution from the Pareto-front. This may be solved by using graphic methods to visualize the distribution of solutions in Pareto-front. It could be especially useful when the solution dimension is high. When the number of objectives becomes more than three, the complexity of multi-objective-optimization greatly increases and hence the efficiency correspondingly decreases. It may be necessary to find another efficient optimization algorithm for this problem. We also plan to conduct experiments on real aerial manipulators to see how the presented method works with unavoidable noises.

5.8 References

[1] K. Mathe, L. Busoniu, L. Barabas, C. Iuga, L. Miclea, and J. Braband, "Vision-based control of a quadrotor for an object inspection scenario," in *International Conference on Unmanned Aircraft Systems (ICUAS)*, Arlington, VA, USA, June 7-10, 2016.

[2] L. V. C. Arcos, J. C. C. Muñoz, and A. L. Espino, "Remote sensing for agricultural crops based on a low cost quadcopter," *Sistemas and Telemática*, vol. 13, no. 34, pp. 49-63, 2015.

[3] H. C. Yang, R. Abousleiman, B. Sababha, E. Gjioni, D. Korff, and O. Rawashdeh, "Implementation of an autonomous surveillance quadrotor system," in *AIAA Infotech Aerospace Conference*, Seattle, WA, USA, April 6-9, 2009.

[4] Q. Lindsey, D. Mellinger, and V. Kumar, "Construction of cubic structures with quadrotor teams," in *Proceedings of Robotics: Science and Systems (RSS)*, Los Angeles, CA, June 27 – July 1, 2011.

[5] J. Willmann, F. Augugliaro, T. Cadalbert, R. DAndrea, F. Gramazio, and M. Kohler, "Aerial robotic construction towards a new field of architectural research," *International Journal of Architectural Computing*, vol. 10, no. 3, pp. 439-460, 2012.

[6] M. Bisgaard, A. la Cour-Harbo, and J. Bendtsen, “Adaptive control system for autonomous helicopter slung load operations,” *Control Engineering Practice*, vol. 18, no. 7, pp. 800811, 2010.

[7] I. Palunko, R. Fierro, and P. Cruz, “Trajectory generation for swing-free maneuvers of a quadrotor with suspended payload: A dynamic programming approach,” in *IEEE International Conference on Robotics and Automation (ICRA)*, St Paul, MN, USA, May 14-18, 2012.

[8] N. Michael, J. Fink, and V. Kumar, “Cooperative manipulation and transportation with aerial robots,” *Autonomous Robots*, vol. 30, no. 1, pp. 73-86, 2011.

[9] M. Bernard, K. Kondak, I. Maza, and A. Ollero, “Autonomous transportation and deployment with aerial robots for search and rescue missions,” *Journal of Field Robotics*, vol. 28, no. 6, pp. 914931, 2011.

[10] D. Lee and C. Ha, “Mechanics and control of quadrotors for tool operation,” in *ASME 2012 5th Annual Dynamic Systems and Control Conference joint with the JSME 2012 11th Motion and Vibration Conference*, Fort Lauderdale, FL, USA, October 17-19, 2012.

[11] A. Albers, S. Tautmann, T. Joward, T. A. Nguyen, M. Frietsch, and C. Sauter, “Semi-autonomous flying robot for physical interaction with environment,” in *IEEE conference on Robotics, Automation and Mechatronics (RAM),* Singapore, June 28-30, 2010.

[12] V. Lippiello and F. Ruggiero, “Cartesian impedance control of a UAV with a robotic arm,” in *10th International IFAC Symposium on Robot Control*, Dubrovnik, Croatia, September 5-7, 2012.

[13] G. Arleo, F. Caccavale, G. Muscio, and F. Pierri, “Control of quadrotor aerial vehicles equipped with a robotic arm,” in *21st Mediterranean Conference on Control and Automation*, Crete, Greece, June 25-28, 2013.

[14] S. Kim, S. Choi, and H. J. Kim, “Aerial manipulation using a quadrotor with a two dof robotic arm,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Tokyo, Japan, November 3-7, 2013.

[15] F. Ruggiero, et al., "A multilayer control for multirotor UAVs equipped with a servo robot arm," in *IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, USA, May 26-30, 2015.

[16] G. Heredia, A. E. Jimenez-Cano, I. Sanchez, et al., “Control of a multirotor outdoor aerial manipulator,” in: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Chicago, IL, USA, September 14-18, 2014.

[17] H. Lee, S. Kim, and H. J. Kim, “Control of an aerial manipulator using on-line parameter estimator for an unknown payload,” in: *IEEE International Conference on Automation Science and Engineering (CASE)*, Gothenburg, Sweden, August 24-28, 2015.

[18] S. Di Lucia, G. D. Tipaldi, and W. Burgard, "Attitude stabilization control of an aerial manipulator using a quaternion-based backstepping approach," in: *IEEE European Conference on Mobile Robots (ECMR)*, Troia Portugal, May 24-29, 2015.

[19] A. E. Jimenez-Cano, G. Heredia, M. Bejar, et al., "Modelling and control of an aerial manipulator consisting of an autonomous helicopter equipped with a multi-link robotic arm," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 230, no. 10, pp. 1860-1870, 2016.

[20] R. M. Jones, D. Sun, G. B. Haberfeld, et al., "Design and control of a small aerial manipulator for indoor environments," in: *AIAA Information Systems-AIAA Infotech@ Aerospace*, Grapevine, TX, USA, January 9-13, 2017.

[21] G. Gigerenzer and R. Selten. *Bounded rationality: The adaptive toolbox*, Cambridge: MIT press, 2002.

[22] G. S. Halford, R. Baker, J. E. McCredden, et al., "How many variables can humans process?," *Psychological Science*, vol. 16, no.1, pp. 70-76, 2005.

[23] M. A. Sahib and B. S. Ahmed, "A new multiobjective performance criterion used in PID tuning optimization algorithms," *Journal of advanced research*, vol. 7, no. 1, pp. 125-134, 2016.

[24] A. Popov, A. Farag, and H. Werner, "Tuning of a PID controller using a multi-objective optimization technique applied to a neutralization plant," in: *European Control Conference on Decision and Control (CDC-ECC'05)*, Seville, Spain, December 12-15, 2005.

[25] C. L. Lin, H. Y. Jan, and N. C. Shieh, "GA-based multiobjective PID control for a linear brushless DC motor," *IEEE/ASME transactions on mechatronics*, vol. 8, no. 1, pp. 56-65, 2003.

[26] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. *Robotics: Modelling, planning and control*. Springer-Verlag, London, UK, 2009.

[27] R. M. Murray, Z. Li, and S. S. Sastry. *A mathematical introduction to robotic manipulation.* Boca Raton, FL: CRC, 1994.

[28] K. Nonami, F. Kendoul, S. Suzuki, W. Wang. *Atonomous flying robots, Unmanned aerial vehicles and micro aerial vehicles*. Springer, 2010.

[29] X. Zhou, R. Liu, J. Zhang, X. Zhang, "Stabilization of a quadrotor with uncertain suspended load using sliding mode control," in *Proceedings of the ASME International Design Engineering Technical Conferences & Computers & Information in Engineering Conference*, Charlotte, NC, USA, August 21-24, 2015.

[30] K. Deb, A. Pratap, S. Agarwal, et al., "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182-197, 2002.

[31] J. Knowles and D. Corne, "The Pareto archived evolution strategy: A new baseline algorithm for multiobjective optimization," in: *Proceedings of the 1999 Congress on Evolutionary Computation*, Washington DC, USA, July 6-9, 1999.

[32] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach," *IEEE transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257-271, 1999.

[33] N. Srinivas and K. Deb, "Multiobjective function optimization using nondominated sorting genetic algorithms," *Evolutionary computation*, vol. 2, no. 3, pp. 221–248, 1995.

[34] S. Kannan, S. Baskar, and J. McCalley, "Application of NSGA-II algorithm to generation expansion planning," *IEEE Transactions on Power systems*, vol. 24, no. 1, pp. 454-461, 2009.

[35] M. Atiquzzaman, S. Liong, and X. Yu, "Alternative decision making in water distribution network with NSGA-II," *Journal of water resources planning and management*, vol. 132, no. 2, pp. 122-126, 2006.

[36] E. G. Bekele and J. W. Nicklow, "Multi-objective automatic calibration of SWAT using NSGA-II," *Journal of Hydrology*, vol. 341, nos. 3-4, pp. 165-176, 2007.

[37] C. C. Hang, J. K. Astrom, and W. K. Ho, "Refinements of Ziegler Nichols tuning formula," *IEEE Proceedings*, vol. 138, no. 2, pp. 111, 1991.

[38] X. Zhou, X. Zhang, J. Zhang, and R. Liu, "Stabilization and trajectory control of a quadrotor with uncertain suspended load," *arXiv:1612.04324*, 2016.

[39] M. Chipofya, D. J. Lee, and K. T. Chong, 2015, "Trajectory tracking and stabilization of a quadrotor using model predictive control of Laguerre functions," *Abstract and Applied Analysis*, vol. 25, Article ID 916864, 11 pages.

[40] K. Alexis, G. Nikolakopoulos, and A. Tzes, "Experimental model predictive attitude tracking control of a quadrotor helicopter subject to wind-gusts," in *Proceedings of the 18th Mediterranean Conference on Control and Automation*, pp. 1461-1466, 2010.

[41] G. V. Raffo, M. G. Ortega, and F. R. Rubio, "An integral predictive/nonlinear H_∞ control structure for a quadrotor helicopter," *Automatic*, vol. 46, pp. 29-39, 2008.

[42] C. von Lucken, B. Baran, and C. Brizuela, "A survey on multi-objective evolutionary algorithms for many-objective problems," *Computational Optimization and Applications*, vol. 58, no. 3, pp. 707-756, 2014.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Contributions

In this dissertation, new self-reflective experiential learning algorithms have been proposed to improve the learning safety so that robot learning can be used as a promising way to achieve the true persistent autonomy. The critical contribution of this dissertation can be considered as endowing robots with the capability of human's self-reflection. This capability makes a robot think like a human, not only act like a human. By learning from the robot's own experiences, it changes the robot's adaptation pattern from passive to active with thinking about if it should change, when it should change, and how it should change. All these changes significantly reduce learning failures and require minimum human interventions. This dissertation lays a solid foundation for robot learning, specifically reinforcement learning, to be deployed from controlled simulated environments to open practical environments. To be more specific, this dissertation introduces three new ways to achieve safe adaptations to different environments.

Firstly, two high-level and sophisticated mechanisms, including adding heuristic safety concerns (Chapter 2) and comprehensively evaluating learning gains and costs (Chapter 3) have been proposed for reinforcement learning to decide whether and when it should learn if the environment changes. This is important because it is the first time to argue and prove that learning is not always necessary and beneficial in a safe adaptation to different environments. The proposed mechanisms can both improve the safety and reduce human interventions.

Secondly, the capabilities of self-recovery from failures and preventing similar failures in the future (Chapter 4) have been added into reinforcement learning to decide how it should change when environment changes. This method totally learns from its own interactions with the

environment, so it requires minimum human interventions when using self-recoverable reinforcement learning.

Thirdly, the multi-objective optimization has been used to automatically tune the control parameters for different scenarios (Chapter 5). It can be regarded as how the robot should change, but the change is with respect to the low-level control part. As a perfect plan cannot be successfully implemented without a reliable executor, this method enhances the low-level controller's robustness to provide a solid basis for persistent autonomy.

6.2 Future Work

Although the learning safety has been significantly improved in this dissertation, there are still some open problems when using reinforcement learning to achieve persistent autonomy.

6.2.1 An Extension from Deterministic Environment to Stochastic, Noisy Environment

In this dissertation, we mainly focus on the deterministic environment with assuming the influences of noises can be neglected. For example, we assume that the sensor can provide accurate information feedback and the actuator can precisely follow a command. In practical applications, however, it may be impossible to ignore the noises. Thus, the transitions in the Markov Decision Process can be stochastic and the observed rewards can be inaccurate. In addition, the recovery plan may not be exactly executed either. All these make the learning more complicated with more chances to encounter failures. In the future research, it is important to consider the uncertainties in new reinforcement learning algorithms so that the new algorithms can be more robust.

6.2.2 An Extension from discrete MDP to continuous MDP

In this dissertation, we mainly use discrete MDP problems to validate our new reinforcement learning algorithms. In practical situations, however, there are also many continuous MDP

problems. One way to solve the continuous MDP problems is to discretize them back to the discrete MDP problems. However, this usually leads to a great increase in the dimensions of the state space and action space. It is generally difficult to solve a high-dimensional MDP problem, especially considering the limited resources in real-world robots. Another way to solve the continuous MDP problems is to approximate the value function in MDPs. However, if the approximation lacks accuracy, the final learned policy will be meaningless. Thus, in the future, it is worth investigating how to design an accurate value function approximation or an efficient discretization rule for the continuous MDP.