

University of Nevada, Reno

**Deep Reinforcement Learning for Robotic Tasks:  
Manipulation and Sensor Odometry**

A dissertation submitted in partial fulfillment of the  
requirements for the degree of Doctor of Philosophy in  
Computer Science and Engineering

by  
Adarsh Sehgal

Hung Manh La/Dissertation Advisor

December 2022



THE GRADUATE SCHOOL

We recommend that the dissertation  
prepared under our supervision by

**ADARSH SEHGAL**

entitled

**Deep Reinforcement Learning for Robotic Tasks: Manipulation and  
Sensor Odometry**

be accepted in partial fulfillment of the  
requirements for the degree of

**Doctor of Philosophy**

Hung La  
*Advisor*

Sushil Louis  
*Committee Member*

Christos Papachristos  
*Committee Member*

Alireza Tavakkoli  
*Committee Member*

Hao Xu  
*Graduate School Representative*

Markus Kemmelmeier, Ph.D., Dean  
*Graduate School*

December, 2022

# *Abstract*

by Adarsh Sehgal

Research in robotics has frequently focused on artificial intelligence (AI). To increase the effectiveness of the learning process for the robot, numerous studies have been carried out. To be more effective, robots must be able to learn effectively in a shorter amount of time and with fewer resources. It has been established that reinforcement learning (RL) is efficient for aiding a robot's learning. In this dissertation, we proposed and optimized RL algorithms to ensure that our robots learn well. Research into driverless or self-driving automobiles has exploded in the last few years. A precise estimation of the vehicle's motion is crucial for higher levels of autonomous driving functionality. Recent research has been done on the development of sensors to improve the localization accuracy of these vehicles. Recent sensor odometry research suggests that Lidar Monocular Visual Odometry (LIMO) can be beneficial for determining odometry. However, the LIMO algorithm has a considerable number of errors when compared to ground truth, which motivates us to investigate ways to make it far more accurate. We intend to use a Genetic Algorithm (GA) in our dissertation to improve LIMO's performance. Robotic manipulator research has also been popular and has room for development, which piqued our interest. As a result, we researched robotic manipulators and applied GA to Deep Deterministic Policy Gradient (DDPG) and Hindsight Experience Replay (HER) (GA+DDPG+HER). Finally, we kept researching DDPG and created an algorithm named AACHER. AACHER

uses HER and many independent instances of actors and critics from the DDPG to increase a robot’s learning effectiveness. AACHER is used to evaluate the results in both custom and existing robot environments.

In the first part of our research, we discuss the LIMO algorithm, an odometry estimation technique that employs a camera and a Lidar for visual localization by tracking features from their measurements. LIMO can estimate sensor motion via Bundle Adjustment based on reliable keyframes. LIMO employs weights of the vegetative landmarks and semantic labeling to reject outliers. LIMO, like many other odometry estimating methods, has the issue of having a lot of hyperparameters that need to be manually modified in response to dynamic changes in the environment to reduce translational errors. The GA has been proven to be useful in determining near-optimal values of learning hyperparameters. In our study, we present and propose the application of the GA to maximize the performance of LIMO’s localization and motion estimates by optimizing its hyperparameters. We test our approach using the well-known KITTI dataset and demonstrate how the GA supports LIMO to lower translation errors in various datasets. Our second contribution includes the use of RL. Robots using RL can select actions based on a reward function. On the other hand, the choice of values for the learning algorithm’s hyperparameters could have a big impact on the entire learning process. We used GA to find the hyperparameters for the Deep Deterministic Policy Gradient (DDPG) and Hindsight Experience Replay (HER). We proposed the algorithm GA+DDPG+HER to optimize learning hyperparameters and applied it to the robotic manipulation tasks of

FetchReach, FetchSlide, FetchPush, FetchPick&Place, and DoorOpening. With only a few modifications, our proposed GA+DDPG+HER was also used in the AuboReach environment. Compared to the original algorithm (DDPG+HER), our experiments show that our approach (GA+DDPG+HER) yields noticeably better results and is substantially faster. In the final part of our dissertation, we were motivated to use and improve DDPG. Many simulated continuous control problems have shown promising results for the DDPG, a unique Deep Reinforcement Learning (DRL) technique. DDPG has two parts: Actor learning and Critic learning. The performance of the DDPG technique is therefore relatively sensitive and unstable because actor and critic learning is a considerable contributor to the robot's total learning. Our dissertation suggests a multi-actor-critic DDPG for reliable actor-critic learning as an improved DDPG to further enhance the performance and stability of DDPG. This multi-actor-critic DDPG is further combined with HER, called AACHER. The average value of numerous actors/critics is used to replace the single actor/critic in the traditional DDPG approach for improved resistance when one actor/critic performs poorly. Numerous independent actors and critics can also learn from the environment in general. In all the actor/critic number combinations that are evaluated, AACHER performs better than DDPG+HER.

*Dedicated to Pushpa Sehgal*

# *Acknowledgements*

I would like to extend sincere thanks to my advisor, Dr. Hung La, who was a constant source of inspiration and enlightenment required for all the phases of my research and for helping me complete this research in the Advanced Robotics and Automation Lab (ARA).

I also thank Dr. Sushil Louis, Dr. Christos Papachristos, Dr. Alireza Tavakkoli, and Dr. Hao Xu for being on my committee, and for taking the time to review this dissertation and provide guidance on my research.

Ultimately, I would like to express my gratitude to my family, especially my father, Naveen Kumar Sehgal, my mother, Vijay Kumari, and my wife, Muskan Sehgal, for supporting me through this hectic time in my life, and for being pillars of motivation and emotional support throughout my time in this school.

This work is supported by the U.S. National Science Foundation (NSF) under grants NSF-CAREER: 1846513 and NSF-PFI-TT: 1919127, and the U.S. Department of Transportation, Office of the Assistant Secretary for Research and Technology (USDOT/ OST-R) under Grant No. 69A3551747126 through INSPIRE University Transportation Center.

# Contents

<b>Abstract</b>	<b>i</b>
<b>Dedication</b>	<b>iv</b>
<b>Acknowledgements</b>	<b>v</b>
<b>List of Tables</b>	<b>viii</b>
<b>List of Figures</b>	<b>ix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Background on Genetic Algorithm (GA) . . . . .	2
1.3 Background on Lidar-Monocular Visual Odometry (LIMO) . . . . .	3
1.4 Background on Deep Reinforcement Learning (DRL) . . . . .	7
1.5 Background on Proximal Policy Optimization (PPO) . . . . .	8
1.6 Background on Advantage Actor Critic (A2C) . . . . .	9
1.7 GA on DRL and LIMO . . . . .	10
1.8 Improved DRL . . . . .	11
1.9 Content . . . . .	12
<b>2 GA-LIMO: Genetic Algorithm-Based Function Optimizer in Lidar-Monocular Visual Odometry</b>	<b>13</b>
2.1 LIMO . . . . .	13
2.1.1 Feature extraction and pre-processing . . . . .	14
2.1.2 Scale Estimation . . . . .	14
2.1.3 Frame to Frame Odometry . . . . .	16
2.1.4 Backend . . . . .	17
2.2 Open Problem Discussion . . . . .	19
2.3 GA-LIMO . . . . .	19
2.4 Experimental Results . . . . .	23
2.5 Summary . . . . .	28

<b>3</b>	<b>GA+DDPG+HER: Genetic Algorithm as Function Optimizer in Reinforcement Learning</b>	<b>31</b>
3.1	Reinforcement Learning . . . . .	31
3.2	Deep Deterministic Policy Gradients (DDPG) . . . . .	32
3.3	Hindsight Experience Replay (HER) . . . . .	33
3.4	Open Problem Discussion . . . . .	34
3.5	DDPG+HER and GA . . . . .	35
3.6	Experimental Results . . . . .	40
3.6.1	Experimental setup . . . . .	40
3.6.2	Running GA . . . . .	45
3.6.3	Modifications required for AuboReach . . . . .	49
3.6.4	Training evaluation . . . . .	53
3.6.5	Efficiency evaluation . . . . .	54
3.6.6	Analysis . . . . .	60
3.7	Summary . . . . .	63
<b>4</b>	<b>AACHER: Assorted Actor-Critic Deep Reinforcement Learning with Hindsight Experience Replay</b>	<b>65</b>
4.1	Introduction . . . . .	65
4.2	Reinforcement Learning . . . . .	69
4.3	Deep Reinforcement Learning (DRL) . . . . .	71
4.4	Deep Deterministic Policy Gradients (DDPG) . . . . .	72
4.5	Hindsight Experience Replay (HER) . . . . .	74
4.6	Open Problem Discussion . . . . .	76
4.7	AACHER . . . . .	76
4.8	Experimental Results . . . . .	80
4.8.1	Simulated environments . . . . .	80
4.8.2	Experimental setup . . . . .	82
4.8.3	Experimental results . . . . .	84
4.8.4	Analysis . . . . .	88
4.9	Summary . . . . .	90
<b>5</b>	<b>Conclusion and Future Work</b>	<b>93</b>
5.1	Conclusion . . . . .	93
5.2	Future Work . . . . .	95
5.3	Publications . . . . .	95
	<b>Bibliography</b>	<b>98</b>

# List of Tables

2.1	When GA was run on LIMO with sequence 04 separately, the values of hyperparameters were compared between LIMO and GA-LIMO. . .	26
2.2	When GA was run on LIMO with sequence 01 separately, the values of hyperparameters were compared between LIMO and GA-LIMO. . .	27
2.3	When GA is run on LIMO with combined sequence 01 and 04, the values of hyperparameters are compared between LIMO and GA-LIMO.	27
3.1	DDPG+HER vs. GA+DDPG+HER values of hyperparameters. . . .	48
3.2	Efficiency evaluation: For all activities, compare average (over ten runs) episodes to accomplish the target. . . . .	58
3.3	Efficiency evaluation: For all activities, compare the average (over ten runs) running time (s) to attain the target. . . . .	59
3.4	Efficiency evaluation: For all activities, compare the average (over ten runs) steps taken to attain the target. . . . .	59
3.5	Efficiency evaluation: Average (over ten runs) epochs comparison to reach the goal, for all the tasks. . . . .	60
4.1	The table displays the success rate, reward, and average Q value for each of the five environments. For the 25 <sup>th</sup> epoch, the average of all the values over 20 runs is used. . . . .	87

# List of Figures

1.1	VSLAM pipeline. The input is a temporal sequence of images, and the system outputs a sparse reconstruction of the observed environment and the camera poses [1–4]. In this work, LIMO does not perform loop closure [5]. . . . .	5
2.1	Camera data while GA-LIMO is in action. . . . .	23
2.2	GA-LIMO estimating the pose. The video for this visualization can be found on: <a href="https://www.youtube.com/watch?v=_ApeUcYy6_g">https://www.youtube.com/watch?v=_ApeUcYy6_g</a> . . .	24
2.3	Results comparison for sequence 04 (algorithm 2). LIMO has a 1.01% translation error, while GA-LIMO has about half this error with 0.56%. . . . .	25
2.4	Results comparison for sequence 01 (algorithm 2). LIMO has a 3.71% translation error, while GA-LIMO has 3.8%. . . . .	26
2.5	GA-LIMO is used to find the hyperparameters, which are a mixture of sequences 01 and 04 (Algorithm 1). These factors are subsequently put to the test in three different ways. GA-LIMO outperforms LIMO in each of the three sequences. . . . .	29
2.6	When GA-LIMO was run in Algorithm 1, the trajectory was compared. GA-LIMO outperforms LIMO in each of the three sequences. . . . .	30
3.1	Success rate vs. epochs for various $\gamma$ for <i>FetchPick&amp;Place-v1</i> task. . . . .	38
3.2	Chromosome representation for the GA. . . . .	41
3.3	The matching DDPG+HER versus GA+DDPG+HER charts are produced once GA has identified all six hyperparameters. All graphs are averaged across ten runs. In this figure, DDPG+HER is referred to as DRL. . . . .	42
3.4	The corresponding DDPG+HER versus GA+DDPG+HER charts are produced when all six hyperparameters have been discovered by GA. Each graph is averaged across ten runs. In this picture, DRL stands for DDPG+HER. . . . .	43
3.5	Using the most accurate policy learned via GA+DDPG+HER, the AuroReach environment performs a task in a real experiment. . . . .	44
3.6	In a simulated experiment, the AuroReach environment performs a task using the best policy learned via GA+DDPG+HER. . . . .	44

3.7	Success rate vs. epochs for <i>FetchPush-v1</i> task when $\gamma$ and $\tau$ are found using the GA. . . . .	45
3.8	Success rate vs. epochs for <i>FetchSlide-v1</i> task when $\gamma$ and $\tau$ are found using the GA. . . . .	47
3.9	The success rate of the <i>AuboReach</i> task in comparison to epochs. The average of ten runs is shown in this graph. In this figure, "DRL" stands for DDPG+HER. . . . .	51
3.10	The GA+DDPG+HER training evaluation charts are produced after GA discovers all six hyperparameters. This was the result of just one GA run. . . . .	54
3.11	The GA+DDPG+HER training evaluation plots when GA identified all six hyperparameters. This outcome came from a single GA run. . . .	55
3.12	The GA+DDPG+HER training evaluation charts are produced after GA learns all six hyperparameters. This was the result of just one GA run. . . . .	56
3.13	The DDPG+HER vs. GA+DDPG+HER efficiency evaluation charts are produced when all six hyperparameters are determined using GA (Total reward vs episodes). All graphs are averaged across ten runs. In this figure, DDPG+HER is referred to as DRL. . . . .	57
3.14	GA+DDPG+HER comparison with PPO [6], A2C [7], DDPG+HER (DDPG[8] + HER[9]) and DDPG [8]. All plots are averaged over ten runs. The term "DRL" in this figure refers to DDPG+HER. . . . .	61
4.1	Interaction between a robot and its environment in reinforcement learning (RL). . . . .	69
4.2	Taxonomy of Deep Reinforcement Learning (DRL). . . . .	71
4.3	DDPG+HER explanation . . . . .	75
4.4	Use of multiple actors and critics in AACHER . . . . .	79
4.5	Naming convention for AACHER-based experimentation. . . . .	80
4.6	The <i>AuboReach</i> environment executes a task in a real experiment using the most accurate policy learned using AACHER. . . . .	81
4.7	The <i>AuboReach</i> environment performs a task in a simulated experiment using the best policy discovered via AACHER. . . . .	81
4.8	Plots comparing several experiments run in the <i>AuboReach</i> environment while using AACHER and DDPG+HER. Each experiment in the AACHER is marked following the naming convention. Comparisons have been made between success rate, reward, and average Q value. These are all averaged over 20 runs and plotted against epochs. Each plot is also shown in a zoomed-in view for clarification. . . . .	85

- 4.9 Plots versus DDPG+HER are shown for the two best-performing experiments, A10C10 and A20C20, when they are applied to the *Au-  
boReach* environment. Plots showing success rate, reward, and average Q values are shown using epochs and an average of 20 runs. The range of values for each plot for 20 runs is shown in the shaded area. For clarity, each plot has a zoomed-in version. . . . . 86
- 4.10 The two top-performing trials, A10C10 and A20C20, when applied to the various settings, are plotted against DDPG+HER for comparison. Underneath them are the plots for each environment. Using epochs and an average of 20 runs, plots for success rate, reward, and average Q values are displayed. The shaded region displays the range of values for each plot during the 20 runs. . . . . 92

# Chapter 1

## Introduction

### 1.1 Motivation

Accuracy and efficiency are important in both emerging and current technology. When it comes to accuracy, it matters in the case of self-driving cars [10] because human lives are at stake. Because of a glitch in Tesla's semi-autonomous driving technology, a Model X smashed into a barrier in California while on adaptive cruise control. Because high-accuracy GPS is pricey and difficult in areas where there is no GPS signal, the self-driving car must rely on additional sensors to detect its exact location on the road. To estimate the motion and odometry of self-driving automobiles, various types of sensors have been used. While some studies [11] employed generalist cameras, others [12] demonstrated how diverse sensors are used. Furthermore, it has been determined whether these types of vehicles can be trusted, as there are

numerous types of attacks that can compromise the sensor's reliability. Recent sensor odometry research has revealed that LIDAR combined with a monocular camera can be utilized to estimate odometry. Lidar-Monocular Visual Odometry (LIMO) [4] is one such approach. This algorithm has a significant amount of error when compared to ground truth; thus there is room for improvement.

Intelligent robots [13] are one of the many fields of robotics where efficiency is important. Teleoperated robots have been around for a while. Artificial intelligence (AI) is already being implemented in robots around the world. Many technologies aid robot decision-making. It is an issue of efficiency to determine how well the robot learns. Let's imagine it takes a robot a few months to learn how to open a door, which is inefficient due to the length of time it would take to learn. As a result, self-learning robots must be efficient. Robots should be able to learn more quickly, saving both resources and time. In this research, we focus on Reinforcement Learning (RL) [14] to improve the efficiency of learning robots (agents).

## 1.2 Background on Genetic Algorithm (GA)

Genetic algorithms (GAs) [15–17] were developed to investigate poorly known fields, where it is impossible to conduct an exhaustive search and other search techniques are ineffective. When GAs are used to optimize functions, they try to maximize fitness that is related to the optimization goal. Evolutionary computing approaches

in general, and GAs in particular, have achieved significant empirical success on a variety of challenging design and optimization problems. They start with an initial population of randomly initialized candidate solutions, which are frequently encoded as a string (chromosome). While crossover and mutation operators offer fresh possible answers, selection operators focus the search space on the most promising regions.

We used ranking selection [18] to pick parents for crossover and mutation. Through rank selection, higher-ranked (fitter) individuals are probabilistically chosen. Ranking selection, in contrast to fitness proportionate selection, is more concerned with the existence of a fitness difference than with its magnitude. Children are generated through uniform crossover [19], who then undergo flip mutation [17] to change them. Chromosomes are encoded via concatenated hyperparameter binary coding. One such instance of GA combined with Lidar-monocular visual odometry (LIMO) is seen in [20].

### **1.3 Background on Lidar-Monocular Visual Odometry (LIMO)**

Motion estimation has long been a prominent study topic, with a plethora of techniques produced over time [21]. Visual Simultaneous Localization and Mapping (VSLAM), also known as Visual Odometry [22], is a technique that estimates the camera's velocity as well as the 3D structure of the viewed environment at the same time. [23]

provides a recent evaluation of SLAM techniques for autonomous car driving. Bundle Adjustment is the most popular VSLAM approach. Bundle Adjustment is a process that reduces the re-projection error between the observed and forecast locations (landmarks concerning LIMO). Offline VSLAM is being used for mapping and localization in recent advances [24–26].

The structure of the VSLAM pipeline is depicted in Figure 1.1 [4]. Pre-processing and feature extraction are used in algorithms like Robust Outlier Criterion for Camera-based Odometry (ROCC) [27] and Stereo Visual Odometry based on Feature Selection and Tracking (SOFT) [28], in contrast to most of the methods that obtain scale information from a camera placed at a different viewpoint [28]. SOFT and ROCC have achieved good performance on the KITTI Benchmark [29], even without Bundle Adjustment, by extracting robust and precise features and selecting them using specific algorithms.

The reliance on external camera calibration is a key disadvantage of a stereo camera. It was eventually shown that by learning a deformation field to compensate for calibration bias, performance might be improved [30]. LIDAR-camera calibration using a Light Detection and Ranging sensor is also a growing area of research [31, 32]. VSLAM and LIDAR have been used in the past [33–36]. LIMO [4] combines the camera’s feature tracking capability with depth readings from a LIDAR sensor, although it suffers from translation and rotation inaccuracies. We’ll go over our strategy for making LIMO more resilient to translation problems later on.

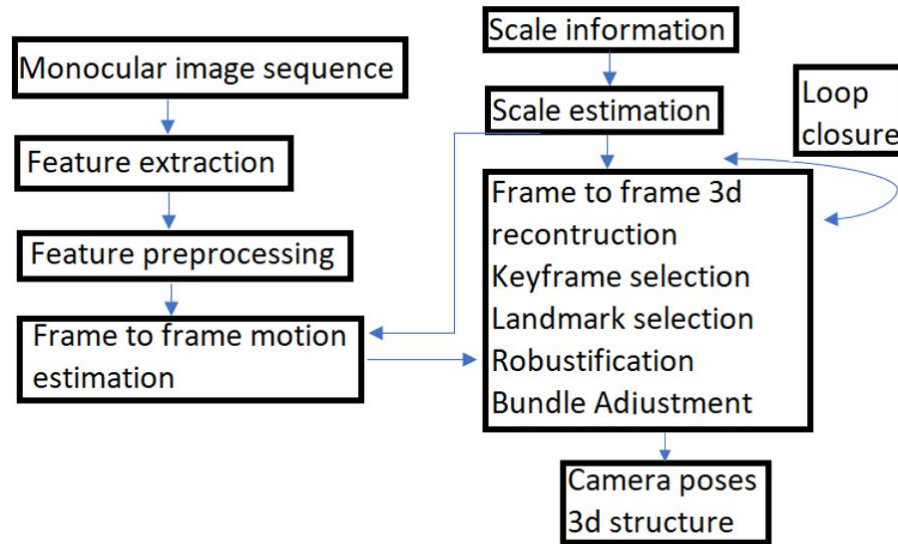


Figure 1.1: VSLAM pipeline. The input is a temporal sequence of images, and the system outputs a sparse reconstruction of the observed environment and the camera poses [1]{4}. In this work, LIMO does not perform loop closure [5].

LIMO makes use of LIDAR depth information for feature detection in the image. If outliers do not meet local plane assumptions, they are eliminated, and points on the ground plane are tested for robustness. In the VSLAM pipeline, depth information is combined with monocular feature identification algorithms, as seen in figure 1.1. To meet real-time constraints, a different approach is used for prior estimation, landmark selection, and keyframe selection. LIMO, unlike, [34], does not employ any LIDAR-SLAM techniques such as Iterative Closest Point (ICP). The main disadvantage of LIMO is that it has a large number of settings that must be manually adjusted. Existing techniques such as Lidar Odometry and Mapping (LOAM) [36] and Vision-Lidar Odometry and Mapping (V-LOAM) [37] suffer from more translation and rotation errors than LIMO. Researchers typically modify settings (including those in LIMO)

to reduce these mistakes. But there is always the possibility of finding better parameter sets that are suited for certain cameras and LIDAR technology for specific scenarios. As a result, optimization procedures are required to improve LIMO’s performance. We propose utilizing a genetic algorithm (GA) to efficiently search the space of possible LIMO parameter values to identify precise settings that maximize performance in this dissertation. Our tests with the novel GA-LIMO algorithm reveal that GA-LIMO outperforms stock LIMO statistically considerably.

Many empirical studies suggest that evolutionary computing approaches like Genetic Algorithms (GAs) operate well as function optimizers in non-linear, discontinuous spaces [17, 38–41]. On a variety of challenges, GAs [15, 42] and the GA operators of crossover and mutation [43] have been evaluated. GAs have been used in early SLAM optimization issues [44], mobile localization utilizing ultrasonic sensors [45] [46], deep reinforcement learning [47], and breast mass detection in mammograms [48], all of which are relevant to our research. This provides strong support for the effectiveness of GAs in solving localization difficulties, and our key contribution in this dissertation is a demonstration of much lower translation error when using a GA to tweak LIMO parameter values over the stock LIMO algorithm [4]. Our findings reveal that translation errors are non-linearly related to LIMO parameters; i.e., a translation error can change non-linearly depending on LIMO parameter values. The LIMO, GA, and GA-LIMO algorithms are described in the following sections. The outcomes of running LIMO on the KITTI odometry data sequences with GA-adjusted parameters are then shown [49].

## 1.4 Background on Deep Reinforcement Learning (DRL)

Since its inception, significant research has been done in the field of Q-learning [50], with some work focusing on continuous action spaces [51–54] and others on discrete action spaces [55]. Autonomous robots use Q-learning approaches to perform a variety of tasks [56]. Reinforcement Learning (RL) [14] has been utilized to enhance manipulation [57, 58] and locomotion [59, 60]. Deep reinforcement learning (DRL) has emerged as a powerful control technique in the realm of robotic research [61]. The DRL’s thorough analysis of the environment is stronger than its understanding of control theory. This DRL capability results in more intelligent and human-like behavior in robots when applied. When DRL approaches are combined with proper training and reinforcement learning, robots can extensively evaluate their surroundings and find solutions [62].

Off-policy and on-policy RL approaches are the two categories that exist. On-policy methods, like SARSA learning [63], aim to evaluate or enhance the policy upon which choices are based. Off-policy techniques [64] like the Deep Deterministic Policy Gradient algorithm (DDPG) [8], Proximal Policy Optimization [6], Advantage Actor-Critic (A2C) [7], and Normalized Advantage Function algorithm (NAF) [65] are helpful for real robot systems. Robotic manipulators have also received a lot of attention [66, 67]. To achieve its objectives, some of this work depended on fuzzy wavelet networks [68],

while others relied on neural networks [69, 70]. A thorough summary of current deep reinforcement learning (DRL) techniques for robot handling can be found in [71]. In theory, goal-conditioned reinforcement learning (RL) can teach a range of skills [72] because it frames each activity in terms of the desired outcome. The use of hindsight experience replay (HER) [9] is frequently used to increase the robustness and sample efficacy of goal-achieving techniques. We are using DDPG in conjunction with HER for our experiments. Recent work on using experience ranking to quicken DDPG+HER’s learning is described in [73].

A single robot [74, 75] as well as a group of robots [76–80] have both undergone intensive RL training. Model-based and model-free learning algorithms have both been researched in the past. Deep network policies are trained in real-world scenarios using model-based learning algorithms, which largely rely on a model-based teacher.

In a similar vein, a lot of work has been done on GA’s [15] [81] and the GA operators of crossover and mutation [43], which have been used to solve a wide range of issues. Numerous RL issues have been resolved using GA [43, 82–84].

## 1.5 Background on Proximal Policy Optimization (PPO)

One of the RL strategies used to optimize the policy is the Policy Gradient (PG) [85] approach. In these techniques, the incentives are used to generate an estimation of

the policy gradient and insert it into a stochastic gradient ascending algorithm [86]. The Proximal Policy Optimization (PPO) [6] technique can be used in situations with continuous or discrete action spaces [87]. PPO is a family of Policy Gradient algorithms for reinforcement learning [88]. It is a model-free reinforcement learning with no prior knowledge. A DRL agent is referred to as model-free if it hasn't developed an explicit model of its environment. The core of the PPO algorithm is a surrogate objective for computing policy updates. The surrogate objective controls substantial policy updates in the spirit of a trust region approach so that each step remains within proximity to the previous-iteration policy [89].

## 1.6 Background on Advantage Actor Critic (A2C)

Advantage actor-critic also referred to as A2C, is a synchronous version of the asynchronous advantage actor-critic (A3C) models that are as effective as or more effective than the asynchronous version [6]. Policy gradients are the foundation of the A2C [7] method. It separates value estimates from action choices by predicting what activities would be beneficial in the future. Using value estimate, the policy is only modified in a useful way. To put it simply, it immediately increases the likelihood of positive behaviors and decreases the likelihood of negative ones [90]. [91] An advantage function can be estimated using the same methods used for policy estimation and value function estimation in actor-critic-based DRL models. Instead of utilizing a single learner neural network to estimate the Q-value function, the A2C approach

estimates the policy function using a critic network and the Q-value function with an actor network. Using A2C actor-critic models, actor and critic networks are updated simultaneously. A3C models require more time to calculate since they simultaneously and asynchronously estimate actor and critic networks. Since there is little performance difference between synchronous and asynchronous actor-critic models, we will be employing A2C in our studies.

## 1.7 GA on DRL and LIMO

As a function optimizer, GA can be employed to address a range of optimization issues. This dissertation is focused on the DDPG+HER and LIMO, with background information given earlier in this chapter. Based on their fitness levels, GAs can be used to optimize the system's parameters. GA aims to maximize the benefits of fitness. An objective function can be changed into a fitness function using a variety of mathematical techniques.

LIMO estimates sensor odometry using a fixed set of parameters. In this circumstance, our trials reveal that there is room to improve the system's accuracy. When GA is used in conjunction with LIMO, a better set of parameters is discovered, increasing the accuracy of odometry estimates. In this GA implementation, the fitness value is the inverse of a translation error. The GA-LIMO system has a better level of precision.

The parameters of the current DRL algorithms, on the other hand, are fixed. A better set of parameters is discovered when GA is applied to DRL, which speeds up learning for the learning agent. The inverse of the number of epochs is the fitness value for this task. GA seems to be a method that might be used to increase the effectiveness of the system.

## 1.8 Improved DRL

Two elements of the renowned and widely applied Deep Deterministic Policy Gradient (DDPG) reinforcement learning method are actor learning and critic learning. As a result of actor and critic learning is so important to the robot’s total learning, the DDPG approach’s performance is quite sensitive and unstable. For trustworthy actor-critic learning, we suggest a multi-actor-critic DDPG to further improve the effectiveness and stability of DDPG. Then, we combined this multi-actor-critic DDPG with Hindsight Experience Replay (HER) to create a new deep learning framework, known as AACHER. When a single actor or critic performs poorly, AACHER replaces them with the average value of several actors or critics to boost resistance. The environment as a whole can also educate a lot of independent actors and critics.

We put our proposed AACHER into practice on goal-based settings, including *Au-  
boReach*, *FetchReach-v1*, *FetchPush-v1*, *FetchSlide-v1*, and *FetchPick&Place-v1*. We

used a variety of actor/critic pairings for our trials, with A10C10 and A20C20 demonstrating the highest levels of performance. Overall results demonstrate that in all actor/critic number combinations used for assessment, AACHER outperforms the conventional approach (DDPG+HER). The performance improvement for A20C20 on *FetchPick&Place-v1* is as much as about 3.8 times the success rate in DDPG+HER.

## 1.9 Content

The following are the chapters of this dissertation: The GA-LIMO algorithm, which helps to reduce translation errors in sensor tracking when compared to ground truth, is explored at length in Chapter 2. This chapter includes the corresponding experimental results. Chapter 3 explains DRL algorithms, discusses the open problem, provides a solution, and displays the experimental results. The details of the recently developed algorithm AACHER, its experiments, and analysis are covered in Chapter 4. Finally, in the final chapter of this dissertation, the conclusion and future work are discussed.

## Chapter 2

### GA-LIMO: Genetic

### Algorithm-Based Function

### Optimizer in Lidar-Monocular

### Visual Odometry

#### 2.1 LIMO

We present previous work on our GA-LIMO algorithm in this part. The VSLAM pipeline is described first, followed by the LIMO algorithm.

### 2.1.1 Feature extraction and pre-processing

The pipeline’s feature extraction technique is depicted in Figure 1.1. Using the Viso2 library [92], feature extraction entails tracking features and associating them. It is also utilized to implement feature tracking, which includes non-maximum suppression, sub-pixel refining, and flow-based outlier rejection. To reject landmarks that are moving objects, deep learning is applied. In a semantic image [93], the neighborhood of the feature point is scanned, and if the majority of nearby pixels belong to a dynamic class, such as a vehicle or pedestrian, the landmark is excluded.

### 2.1.2 Scale Estimation

The camera’s identified feature points are transferred to depth derived from LIDAR for scale estimation. LIMO employs a single-shot depth estimate method. The LIDAR point cloud is first projected onto the image plane before being translated into the camera frame. For each feature point  $f$ , the following processes are carried out in detail:

1. Around  $f$ , which is a set  $F$  of anticipated LIDAR points, a zone of interest is chosen.
2. By segmenting the elements of  $F$ , a new set called the foreground set  $F_{seg}$  is generated.

3. A plane  $\rho$  is fitted to the  $F_{seg}$  components. If  $f$  belongs to the ground plane, a particular fitting algorithm is performed.
4. To calculate depth, the line of sight corresponding to  $f$  is intersected with  $\rho$ .
5. A test is carried out for the previously calculated depth. Estimates of depths of more than 30 meters are ignored because they are prone to error. In addition, the angle between the feature point's line of sight and the plane's normal must be less than a threshold.

Neighborhoods for ordered point clouds can be selected straight from the point clouds. If the point clouds are not ordered, projections of the LIDAR points on the picture are employed, and the points within a rectangle in the image plane around  $f$  are selected. The foreground  $F_{seg}$  is segmented before the plane estimation is done. The next step is to generate a depth histogram with a set bin width of  $h = 0.3\text{m}$  and interpolate it with entries in  $F$ . The adjacent bin's LIDAR points are used to execute segmentation utilizing all detected feature points. Fitting the plane to  $F_{seg}$  can aid in correctly calculating the local surface near  $f$ . Three points are picked from the  $F_{seg}$  points that traverse the triangle  $F$  with the greatest area. To avoid improperly estimated depth, depth estimation is avoided if the area of  $F$  is too small.

However, because LIDAR has a worse resolution in a perpendicular direction than in a level direction, the above approach cannot be utilized to estimate the depth of points on the ground plane. To enable depth estimates for a relevant ground plane, a

separate approach is used. RANSAC with refinement is applied to the LIDAR point cloud to extract the ground plane [25]. Points that correspond to the ground plane are subdivided to estimate feature points along the route. Only local planes that are near the ground plane are allowed; therefore outliers are eliminated.

### 2.1.3 Frame to Frame Odometry

The starting point for frame-to-frame motion estimate is the Perspective-n-Point-Problem [25].

$$\underset{x; y; z; \dots}{\operatorname{argmin}} \sum_i k'_{i; 3d; 2d} k_2^2 \quad (2.1)$$

$${}_{3d; 2d} \bar{p}_i = (p_i; P(x; y; z; \dots; \dots)); \quad (2.2)$$

where  $\bar{p}_i$  is the observed feature point in the current frame,  $p_i$  is the 3D-point corresponding to  $\bar{p}_i$ , and freedom  $P(x; y; z; \dots; \dots)$  denotes the transform from the previous to the current frame, which has three translation and three rotation degrees of freedom. The projection function from the 3D to a 2D domain is  $(\dots)$ . In situations with poor structure and high optical flux, recovered features with valid estimated depth may be quite small. Epipolar error is introduced by LIMO as  ${}_{2d; 2d}$  [24].

$${}_{2d; 2d} = p_i F\left(\frac{x}{z}; \frac{y}{z}; \dots; \dots\right) \bar{p}_i; \quad (2.3)$$

where  $F$  is the fundamental matrix, which may be determined using the camera's inherent calibration and frame-to-frame motion. The loss function, according to LIMO, is a Cauchy function [24]: Where  $a(s)$  is the fixed outlier threshold,  $s(x) = a(s)^2 \cdot \log(1 + \frac{x}{a(s)^2})$ . The optimization issue for frame-to-frame motion estimation can be written as:

$$\underset{x,y,z; i; j}{\operatorname{argmin}} \sum_{i,j} \left( \frac{1}{3d} \|k'_i - k'_j\|_2^2 + \frac{1}{2d} \|k'_i - k'_j\|_2^2 \right) \quad (2.4)$$

### 2.1.4 Backend

LIMO provides a keyframe-based Bundle Adjustment system, including essential components such as keyframe selection, landmark selection, cost functions, and robustification measures. The advantage of this method is that it keeps the set of data that is needed for accurate posture estimation while removing the measurements that aren't needed. Required, rejected, and sparsified keyframes are the three types of keyframes. The dimensions of the required frames are extremely important. When the vehicle does not move, the frame is rejected. The technique collects the remaining frames and picks frames every 0.3s. Finally, the length of the optimization window is chosen during keyframe selection.

A good group of landmarks is easily visible, small, free of outliers, and uniformly distributed. Landmark selection separates all landmarks into three bins: near, middle, and far, each with a predetermined number of landmarks for Bundle Adjustment.

The semantic information is then used to determine the weights of the landmarks.

An extra cost function takes into account predicted landmark depth,

$$c_{ij}(l_i; P_j) = \begin{cases} \infty & \\ \infty & 0; \text{ if } l_i \text{ has no depth estimate} \\ \infty & \text{" \#} \\ \infty & \hat{d}_{ij} \quad 0 \quad 0 \quad 1 \quad (l_i; P_j); \text{ else;} \end{cases} \quad (2.5)$$

where  $l_i$  stands for landmark;  $\mathcal{T}$  stands for mapping from world to camera frame; and  $\hat{d}$  stands for depth estimate. The landmark-pose is denoted by the combination of indices  $i;j$ . Deviations from the length of the translation vector are penalized by a cost function  $c_{ij}$ ,

$$c_{ij}(P_1; P_0) = \hat{s}(P_1; P_0) \quad s; \quad (2.6)$$

where  $P_0, P_1$  are the final two poses in the optimization window, and  $\hat{s}(P_0; P_1) = k \text{translation}(P^{-1}P_1)_2^2$ , where  $s$  is a constant having the value  $\hat{s}(P_1; P_0)$  before optimization.

Outliers must be deleted since they prevent Least-Square methods from converging [94, 95], although semantics and chirality only do early outlier rejection. The challenge

of LIMO optimization can now be stated as follows:

$$\begin{aligned} & \underset{P_j \in \mathbb{R}^{2 \times 2}; l_i \in \mathbb{R}^2; d_i \in \mathbb{R}^2}{\operatorname{argmin}} w_0 k(P_1; P_0 k_2^2) + \\ & \times \times_{i,j} w_1 (k_{ij}(l_i; P_i) k_2^2) + w_2 (k_{ij}(l_i; P_j) k_2^2); \end{aligned} \quad (2.7)$$

where the re-projection error is  $k_{ij}(l_i; P_j) = \bar{l}_{ij} - (l_i; P_j)$ , and weights  $w_0$ ,  $w_1$ , and  $w_2$  are used to scale the cost functions to the same order of magnitude.

## 2.2 Open Problem Discussion

LIMO has a problem with accuracy. When LIMO is used in a variety of situations, it encounters a problem with ground truth. This research introduces the GA-LIMO algorithm to address this issue, which attempts to reduce translation errors in sensor odometry estimates. This chapter goes on to show the proposed algorithm as well as the experimental findings.

## 2.3 GA-LIMO

This section highlights one of the chapter's most important contributions. Open source code is available at <https://github.com/aralab-unr/LIMOWithGA>. [20] provides a more detailed description. The specific GA searches the space of LIMO hyperparameter values for the ones that maximize performance while minimizing

translation error due to pose estimation. Outlier rejection quantile  $q$ ; the maximum number of landmarks for a near bin  $n_{near}$ ; the maximum number of landmarks for middle bin  $n_{middle}$ ; the maximum number of landmarks for far bin  $n_{far}$ ; and weight for the vegetation landmarks  $w_{veg}$  are among the hyperparameters we’re aiming for. As mentioned in the background section, rejecting outliers,  $q$ , plays a crucial role in reducing translation errors to a minimum. The weight of outlier rejection has a significant impact on translation errors. The landmarks are divided into three groups, each of which has a significant impact on the calculation of translation errors. Trees with a complex structure produce feature points that are easy to follow but can shift. As a result, determining the optimum weight for vegetation can reduce translation errors dramatically.  $n_{near}$ ,  $n_{middle}$ , and  $n_{far}$  range from 0 to 999, whereas,  $w_{veg}$  and  $q$  range from 0 to 1. These ranges were established based on preliminary test findings.

---

**Algorithm 1** GA-LIMO

---

- 1: Select an  $n$ -chromosome population.
  - 2: Enter the hyperparameter values into the chromosome.
  - 3: **for** all chromosome values **do**
  - 4:     Run LIMO on KITTI odometry data set sequence 01
  - 5:     Compare LIMO estimated poses with ground truth
  - 6:     Translation error  $e_1$  is found
  - 7:     Run LIMO on KITTI odometry data set sequence 04
  - 8:     Compare LIMO estimated poses with ground truth
  - 9:     Translation error  $e_4$  is found
  - 10:    Average error  $e_{avg} = \frac{e_1 + e_4}{2}$
  - 11:    **return**  $e_{avg}$
  - 12: **end for**
  - 13: Perform Uniform Crossover
  - 14: Perform Flip Mutation at rate 0.1
  - 15: Repeat for the required number of generations for the optimal solution
- 

Our results reveal that altering hyperparameter values did not result in a linear

or immediately discernible decrease or increase in translation error. As a result, a basic hill climber is unlikely to identify optimal hyperparameters. To improve these settings, we use a GA.

The combination of LIMO with the GA, which utilizes a population size of 50 runs for 50 generations, is explained in Algorithm 1. To choose the parents for crossover and mutation, we employed ranking selection [18]. Higher ranking (fitter) individuals are probabilistically selected through rank selection. Unlike fitness proportionate selection, ranking selection is concerned with the existence of a fitness difference rather than its magnitude. Uniform crossover [19] is used to create children, who are subsequently altered by flip mutation [17]. Binary coding with concatenated hyperparameters is used to encode chromosomes. Because changes in hyperparameter values produce significant changes in translation error,  $\alpha$  and  $\beta$  are considered up to three decimal places, which is a step size of 0.001. Because all of the hyperparameters require 11 bits to describe their range of values, we have a 55-bit chromosome with hyperparameters organized in the following order:  $\alpha$ ,  $\beta$ ,  $\gamma$ ,  $\delta$ ,  $\epsilon$ ,  $\zeta$ ,  $\eta$ ,  $\theta$ ,  $\iota$ ,  $\kappa$ ,  $\lambda$ ,  $\mu$ ,  $\nu$ ,  $\xi$ ,  $\omicron$ ,  $\pi$ ,  $\rho$ ,  $\sigma$ ,  $\tau$ ,  $\upsilon$ ,  $\phi$ ,  $\chi$ ,  $\psi$ ,  $\omega$ ,  $\delta$ ,  $\epsilon$ ,  $\zeta$ ,  $\eta$ ,  $\theta$ ,  $\iota$ ,  $\kappa$ ,  $\lambda$ ,  $\mu$ ,  $\nu$ ,  $\xi$ ,  $\omicron$ ,  $\pi$ ,  $\rho$ ,  $\sigma$ ,  $\tau$ ,  $\upsilon$ ,  $\phi$ ,  $\chi$ ,  $\psi$ ,  $\omega$ .

The procedure begins by producing a population of  $n$  individuals at random. LIMO is tasked with evaluating each chromosome. LIMO assesses each individual’s hyperparameter configuration by using those hyperparameters to run on the KITTI dataset [29]. The KITTI benchmarks are well-known and are the most widely used Visual Odometry and VSLAM benchmarks. This dataset includes grayscale photos, color photographs, LIDAR point clouds, and their calibration, as well as rural and urban

scenes and highway sequences. The majority of LIMO configurations are as shown in [4]. For finding the values of hyperparameters, we concentrate our efforts on two sequences in particular: sequence 01 and sequence 04. Sequence 01 is a highway scenario, which is difficult because depth measurements can only be done on a road. Sequence 04 is a city scenario with a huge number of landmarks to estimate depth. For each GA evaluation, we consider both sequences since we want a common set of hyperparameters that will operate with a variety of scenes.

The inverse of translation error is used to calculate the fitness of each chromosome. As required for GA optimization, this turns the minimization of translation error into the maximization of fitness. We use the GA because an exhaustive search of the  $2^{55}$  size search area is not practicable. After all, each fitness evaluation takes a significant amount of time. During a fitness examination, the GA starts with sequence 01 of the LIMO. It then compares the LIMO estimated poses to ground truth (also found in [29]) and uses the official KITTI measure to determine the translation error. Sequence 04 follows the same stages as sequence 03. The average of the inverse translation mistakes from the two sequences is the fitness value of each chromosome.

$$avg = \frac{1 + 4}{2}. \quad (2.8)$$

Selected chromosomes are then crossed over and altered to make new chromosomes, resulting in the next population. The next phase of GA evaluation, selection, crossover,



Figure 2.1: Camera data while GA-LIMO is in action.

and mutation begins now. Because we are conducting  $50 \times 50 = 2500$  LIMO assessments to establish the ideal hyperparameters, the entire system takes a long time. Our tests with individual and combined sequences, with and without the GA, are shown in the next section. Our findings suggest that GA-LIMO outperforms LIMO [4] outcomes.

## 2.4 Experimental Results

We show our studies with single KITTI sequences, a combination of sequences, and overall outcomes in this section. First, we run sequences 01 and 04 of the GA-LIMO independently. In comparison to ground truth (reference), we display the translation error and the error projected onto the trajectory [49]. The results of the GA-LIMO analyses on both sequences 01 and 04 are then shown. Finally, we compare the hyperparameters found by GA-LIMO to those found by LIMO.

Figure 2.1 depicts camera data, while figure 2.2 depicts GA-LIMO calculating the pose from that data. In sequence 04, Figure 2.3 compares LIMO and GA-LIMO's

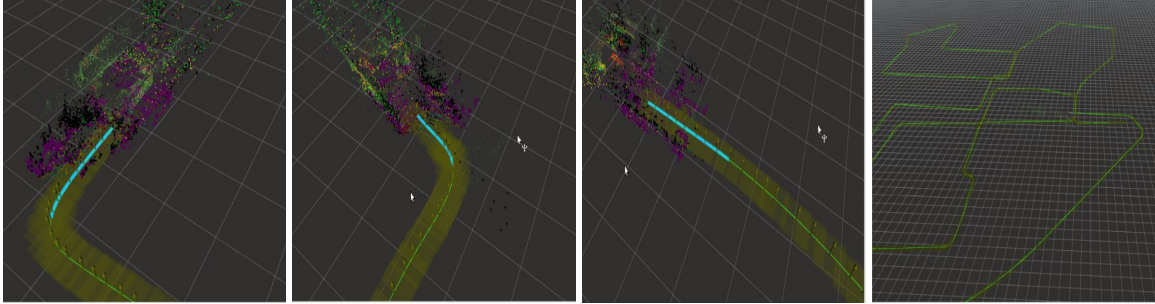


Figure 2.2: GA-LIMO estimating the pose. The video for this visualization can be found on: [https://www.youtube.com/watch?v=\\_4peUcYy6\\_g](https://www.youtube.com/watch?v=_4peUcYy6_g)

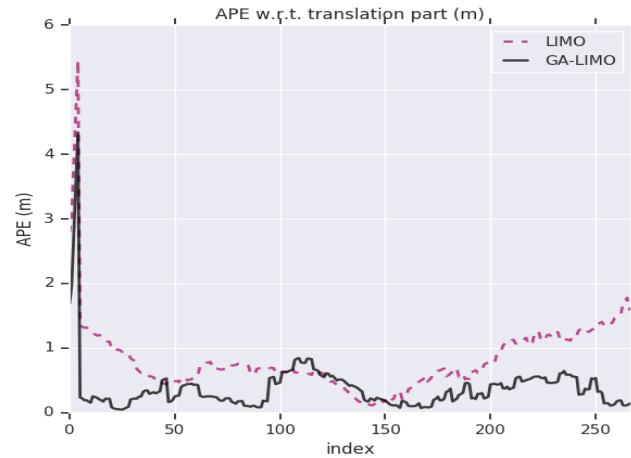
---

**Algorithm 2** GA-LIMO individual

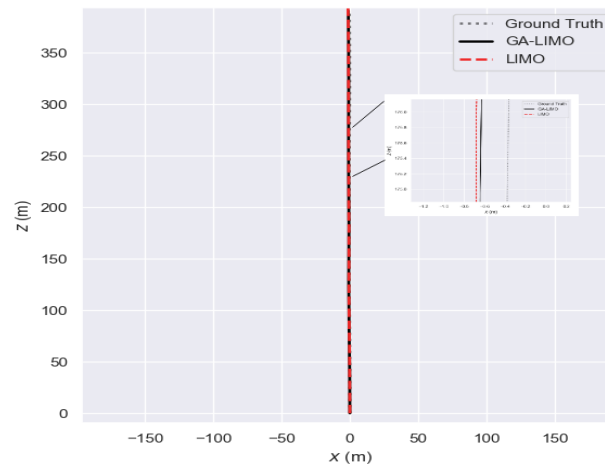
---

- 1: Choose a population of  $n$  chromosomes
  - 2: Set the values of hyperparameters into the chromosome
  - 3: **for** all chromosome values **do**
  - 4:     Run LIMO on individual KITTI odometry dataset sequence
  - 5:     Compare LIMO estimated poses with ground truth
  - 6:     Translation error  $\epsilon$  is found
  - 7:     **return**  $\epsilon$
  - 8: **end for**
  - 9: Perform Uniform Crossover
  - 10: Perform Flip Mutation at rate 0.1
  - 11: Repeat for the required number of generations to find the optimal solution
- 

performance. As in algorithm 2, GA was performed on each sequence individually to identify the optimal hyperparameter values. One of the main measurements is Absolute Pose Error (APE) and Root Mean Squared Error (RMSE) [96]. The RMSE determined concerning ground truth is the translation error for each sequence. Figure 2.3a compares the translation error across poses, while figure 2.3b compares the error mapped onto the trajectory to the zoomed-in trajectory. The values of hyperparameters for LIMO and GA-LIMO are compared in Table 2.1. The values of LIMO and GA-LIMO are significantly different for  $\epsilon$ . In comparison to LIMO, our findings demonstrate that the GA-LIMO trajectory is closer to ground reality. In



(a) Translation error comparison over the poses.

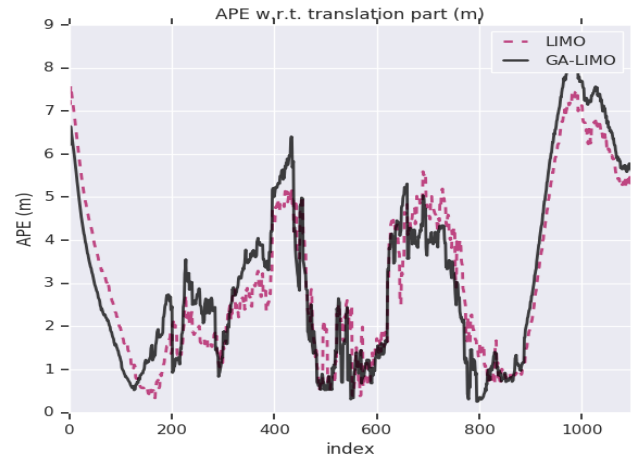


(b) Trajectory comparison for sequence 04, when GA-LIMO was run on this sequence individually (algorithm 2).

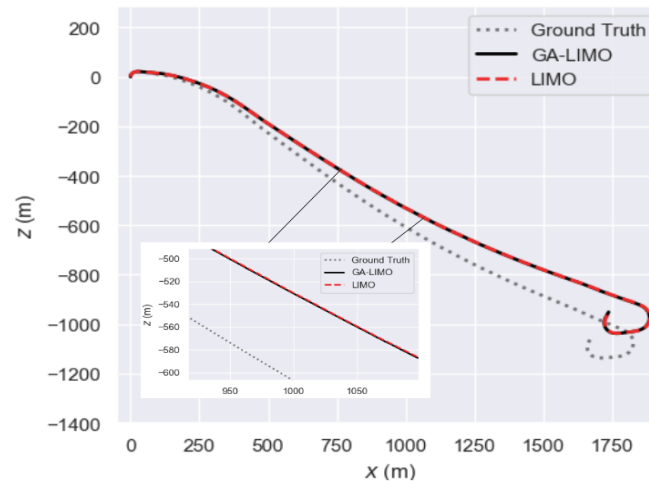
Figure 2.3: Results comparison for sequence 04 (algorithm 2). LIMO has a 1.01% translation error, while GA-LIMO has about half this error with 0.56%.

comparison to LIMO, GA-LIMO had a translation error of 0.56%, whereas LIMO had a translation error of 1.01%.

When the system is run on only sequence 01, Figure 2.4 compares the performance of LIMO with GA-LIMO. In this scenario, the GA was first applied to sequence 01 (Algorithm 2) before the GA-LIMO hyperparameters were applied to the same



(a) Translation error comparison over the poses.



(b) Trajectory comparison.

Figure 2.4: Results comparison for sequence 01 (algorithm 2). LIMO has a 3.71% translation error, while GA-LIMO has 3.8%.

Hyperparameters	LIMO	GA-LIMO
	0.95	0.986
<i>near</i>	400	999
<i>middle</i>	400	960
<i>far</i>	400	859
	0.9	0.128

Table 2.1: When GA was run on LIMO with sequence 04 separately, the values of hyperparameters were compared between LIMO and GA-LIMO.

sequence. The original and GA-discovered hyperparameter values are compared in Table 2.2. Figure 2.4a compares translation errors, while figure 2.4b displays the error translated onto the LIMO and GA-LIMO trajectory. GA-LIMO is closer to ground reality, as illustrated in the zoomed-in picture 2.4b. With sequence 01., the translation error for LIMO is found to be around 3.71% and 3.8% percent for GA-LIMO. When GA-LIMO was run on simple sequence 01, it discovered hyperparameters that did not outperform the original values.

Hyperparameters	LIMO	GA-LIMO
	0.95	0.958
<i>near</i>	400	999
<i>middle</i>	400	593
<i>far</i>	400	877
	0.9	0.813

Table 2.2: When GA was run on LIMO with sequence 01 separately, the values of hyperparameters were compared between LIMO and GA-LIMO.

Hyperparameters	LIMO	GA-LIMO
	0.95	0.963
<i>near</i>	400	999
<i>middle</i>	400	554
<i>far</i>	400	992
	0.9	0.971

Table 2.3: When GA is run on LIMO with combined sequence 01 and 04, the values of hyperparameters are compared between LIMO and GA-LIMO.

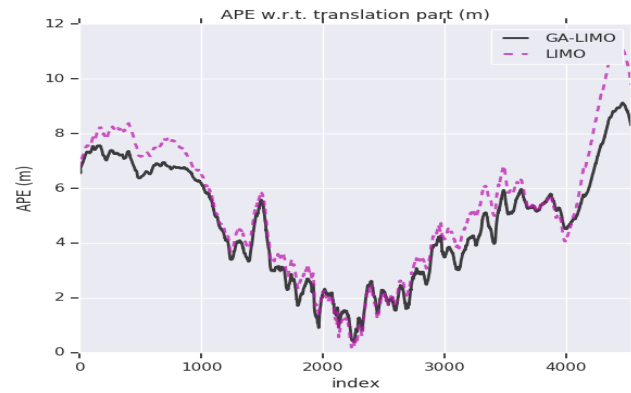
Finally, as mentioned in Algorithm 1, we operated the system with both sequences 01 and 04 at the same time. The average of the sequences' translation errors, when executed with the input hyperparameters, is the fitness of each evaluation. The hyperparameters discovered in GA-LIMO, as indicated in Table 2.3, were then evaluated

on sequences 00, 01, and 04, as depicted in figures 2.5 and 2.6. In each of the three sequences, GA-LIMO outperformed LIMO. The zoomed-in figures provide a closer look at a portion of the trajectories. GA-LIMO trajectories are more accurate and have fewer translation errors than conventional trajectories. When utilizing original hyperparameters, GA-LIMO has a translation error of 5.13% with sequence 00; 3.59% with sequence 01, and 0.65% with sequence 04, compared to 5.77% with sequence 00; 3.71% with sequence 01, and 1.01% with sequence 04 when using LIMO.

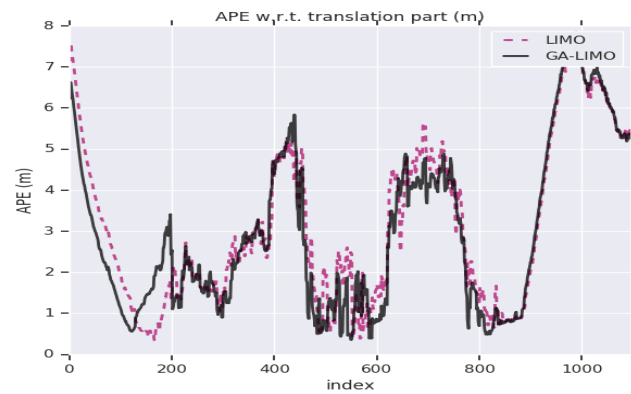
Our strategy assisted in the discovery of a common set of GA-LIMO hyperparameters that performs better and so leads to improved performance in a variety of settings.

## 2.5 Summary

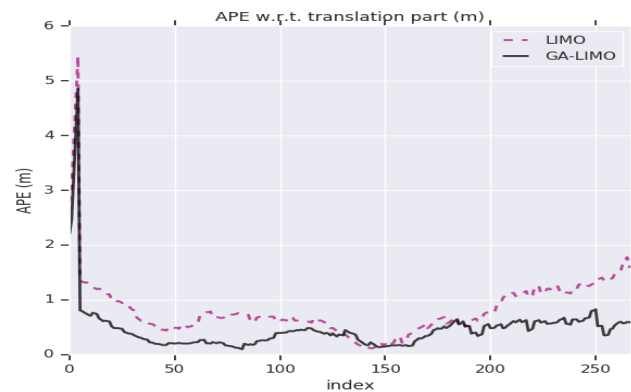
The LIMO algorithm is presented in this chapter. The chapter also introduces the GA-LIMO algorithm, analyzes the performance of LIMO [4] and GA-LIMO [20], and concludes that GA-LIMO outperforms LIMO.



- (a) For sequence 00, a comparison of translation errors across poses was made. LIMO has a translation error of 5.77%, whereas GA-LIMO has a translation error of 5.13%.

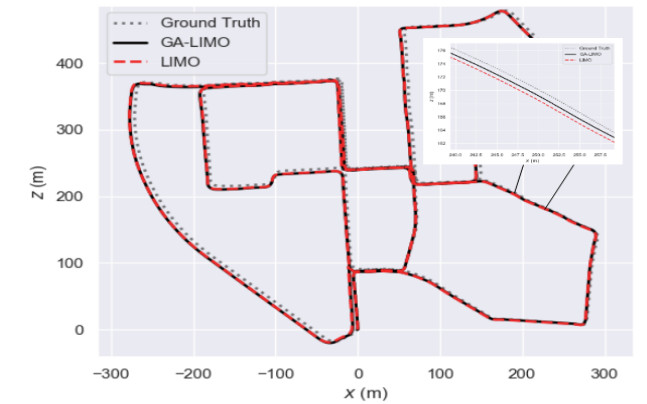


- (b) For sequence 01, a comparison of translation errors across the poses was made. LIMO has a translation error of 3.71%, while GA-LIMO has a translation error of 3.59%.

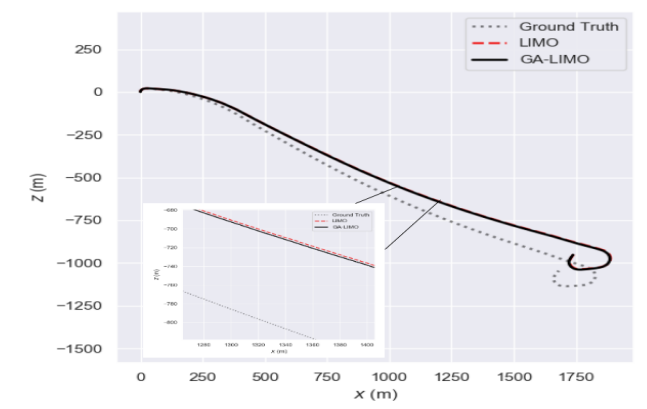


- (c) For sequence 04, a comparison of translation errors across the poses was made. LIMO has a translation error of 1.01%, while GA-LIMO has a translation error of 0.65%.

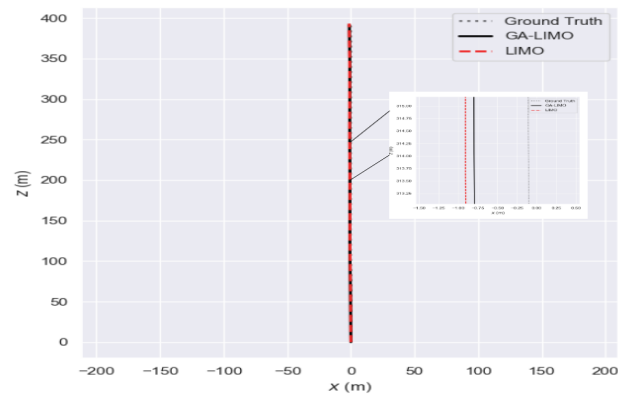
Figure 2.5: GA-LIMO is used to find the hyperparameters, which are a mixture of sequences 01 and 04 (Algorithm 1). These factors are subsequently put to the test in three different ways. GA-LIMO outperforms LIMO in each of the three sequences.



(a) The trajectories in Sequence 00 show GA-LIMO getting closer to the ground truth.



(b) The trajectories in Sequence 01 show GA-LIMO getting closer to the ground truth.



(c) The trajectories in Sequence 04 show GA-LIMO getting closer to the ground truth.

Figure 2.6: When GA-LIMO was run in Algorithm 1, the trajectory was compared. GA-LIMO outperforms LIMO in each of the three sequences.

## Chapter 3

# GA+DDPG+HER: Genetic Algorithm as Function Optimizer in Reinforcement Learning

### 3.1 Reinforcement Learning

In typical reinforcement learning, a robot engages in discrete interactions with the environment to gather as many rewards from the environment as it can. A Markov Decision Process (MDP) [97] can be used to model the issue as a tuple of  $(S, A, P, R, \rho)$  [98], [99].  $S$  stands for the set of states,  $A$  for the set of actions,  $\rho(s_0)$  for the distribution of initial states,  $r : S \times A \rightarrow R$  for the reward function,  $p(s_{t+1}|s_t; a_t)$  for the

transition probabilities, and  $\gamma \in [0; 1]$  for a discount factor that can be utilized to represent the environment [9].

The relationship between states and actions is shown by a deterministic policy as follows:  $\pi: S \rightarrow A$ . The sampling of the initial state,  $s_0$ , occurs at the start of every episode. Based on the current state  $s_t$ :  $a_t = \pi(s_t)$ , the robot acts at each timestep  $t$ . With  $r_t = r(s_t; a_t)$ , the action is deemed successful, and the distribution  $p(\cdot | s_t; a_t)$  helps sample the environment's new state. The discounted sum of future rewards is  $R_t = \sum_{i=t}^{\infty} \gamma^i r_i$ . The robot aims to increase its expected return  $E[R_t | s_t; a_t]$ . Any policy  $\pi$  can be referred to as an ideal policy,  $Q^{\pi}(s; a) = E[R_t | s; a]$  for each  $s \in S; a \in A$ , and any policy  $\pi^*$ . A policy that satisfies the *Bellman* equation and has the same Q-function as the optimum policy is said to have an optimal Q-function or  $Q^*$ .

$$Q^*(s; a) = E_{s' \sim p(\cdot | s; a)}[r(s; a) + \max_{a' \in A} Q^*(s'; a')]; \quad (3.1)$$

## 3.2 Deep Deterministic Policy Gradients (DDPG)

*Deep Deterministic Policy Gradients* [8] (DDPG) has two neural networks: an Actor and a Critic. The actor neural network is a target policy  $\pi: S \rightarrow A$  and the critic neural network is an action-value function approximator  $Q: S \times A \rightarrow R$ . The critic

network  $Q(s; a; \theta^Q)$  and actor network  $\pi(s; \theta^\pi)$  are randomly initialized with weights  $\theta^Q$  and  $\theta^\pi$ .

The target network, which is normally built as a replicate of the critic network, is also employed to update the critic network by providing a reliable target for the training process. A behavioral policy, which is a noisy variant of the target policy  $\pi_b(s) = \pi(s) + \mathcal{N}(0; 1)$ , is used to create episodes. The training process for a critic neural network is similar to that for a DQN, with the exception that the target  $y_t$  is calculated as  $y_t = r_t + \gamma Q(s_{t+1}; \pi_b(s_{t+1}))$ , where  $\gamma$  is the discounting factor. The loss  $L_a = E_a Q(s; \pi(s))$  is used in the actor network's training. For the DDPG approach to maintain its training and enable deep neural networks, the experience replay, and target network are both essential.

### 3.3 Hindsight Experience Replay (HER)

Through the imitation of human behavior, Hindsight Experience Reply (HER) [9] tries to learn from failures. Even if the robot doesn't succeed in achieving the desired result, it learns from every experience. Whatever state the robot achieves is what HER regards as the changed goal. In a standard experience replay, only the transition  $(s_t; a_t; r_t; s_{t+1}; g)$  with the original goal  $g$  is saved. HER also stores the transition  $(s_t; a_t; r_t; s_{t+1}; g')$  from the original goal  $g$  to the updated goal  $g'$ . With really few

incentives, HER functions wonderfully and is considered superior to shaped rewards in this aspect.

### 3.4 Open Problem Discussion

DDPG+HER's effectiveness is a problem. Existing DDPG+HER algorithms are limited in the number of hyperparameters that can be changed. The bulk of robotic tasks can be performed more efficiently by using a better set of hyperparameters in the algorithm. Performance can be evaluated based on how many epochs the learning agent needs to learn a given robotic task. A system's hyperparameters can be optimized using GAs based on how fit they are. GA strives to get the best level of fitness. With the help of several mathematical techniques, an objective function can be converted into a fitness function. The learning agent (robot) can absorb new information more quickly when GA is applied to DDPG+HER because a better set of hyperparameters is discovered. It seems that a strategy that could improve the system's efficiency is GA. The subsequent sections of this chapter show how altering the settings of certain hyperparameters significantly affects the robot's proficiency. The solution to this issue is discussed further in this chapter, and the experimental results that support them demonstrate that the suggested system performs better than the current reinforcement learning method.

### 3.5 DDPG+HER and GA

The genetic algorithm searches the space of hyperparameter values used in DDPG+HER for values that maximize task performance while minimizing the number of training epochs. This is the main contribution of this chapter. Open source code is available at <https://github.com/aralab-unr/ga-drl-aubo-ara-lab>. We focus on the following hyperparameters: Discounting factor  $\gamma$ , Polyak-averaging coefficient  $\tau$  [100], learning rates for critic and actor networks ( $\alpha_{critic}$ ,  $\alpha_{actor}$ ), the percentage of times a random action is taken  $\epsilon$ , and the standard deviation of Gaussian noise added to not entirely random actions as a percentage of the maximum absolute value of actions on different coordinates  $\sigma$ . The equations in this section can be used to explain why all of the hyperparameters have a range of 0–1.

Our investigations showed that changing the settings of the hyperparameters neither increased nor decreased the robot’s learning in a linear or immediately obvious fashion. Therefore, it is doubtful that a simple hill climber will find the best hyperparameters. Because GAs were developed for such poorly understood problems, we employed our GA to maximize these hyperparameter values.

The Polyak-averaging coefficient,  $\tau$ , is used to demonstrate the performance non-linearity for various values of  $\tau$ . Equation 3.2 illustrates how the algorithm uses

:

$$Q^o = Q + (1 - \alpha) Q^o; \quad (3.2)$$

Equation 3.4 describes the Q-Learning update, whereas Equation 3.3 illustrates how is used in the DDPG+HER approach. The learning rate is shown by the variable . To train networks, use this update equation.

$$y_i = r_i + \alpha (Q(s_{i+1}; a_{i+1}) - Q(s_i; a_i)); \quad (3.3)$$

$$Q(s_t; a_t) = Q(s_t; a_t) + \alpha [r_{t+1} + Q(s_{t+1}; a_{t+1}) - Q(s_t; a_t)]; \quad (3.4)$$

Since we have two different types of networks, we will require two learning rates: one for the actor-network  $\alpha_{actor}$  and the other for the critic network  $\alpha_{critic}$ . Equation 3.5 explains the use of the  $\tau$  value, which represents the frequency with which a random action occurs.

$$a_t = \begin{cases} \infty & \text{with probability } 1 \\ \text{random action} & \text{with probability } \end{cases} \quad (3.5)$$

The significance of adopting a GA is highlighted by the fact that Figure 3.1 shows that altering the value of  $\epsilon$  results in a change in the robot’s learning. The first plot displays a comparison of 11  $\epsilon$  values. Two more charts that compare three  $\epsilon$  values have been added for clarification. The initial (untuned) value of  $\epsilon$  in DDPG was 0.95, and we are using four CPUs. To investigate how the success rate alters as the hyperparameter values alter, all values are taken into consideration to two decimal places. The plots demonstrate that there is much space for raising the original success rate.

Algorithm 3 describes how to combine DDPG+HER and a GA, which employs a population size of 30 across 30 generations. To calculate the population size and the number of generations, a number of tests were run with varied variables. For a large enough sample size of chromosomes, these quantities are adequate to calculate the success rate. *Ranking selection* [18] was used to pick parents. Based on rank, which is determined by relative fitness, the parents are determined probabilistically (performance). Children are then produced using *uniform crossover* [19]. We also make use of the 0.1 mutation frequency *ip mutation* [17]. In order to create a chromosome for the GA, a binary chromosome is used to encode each hyperparameter.

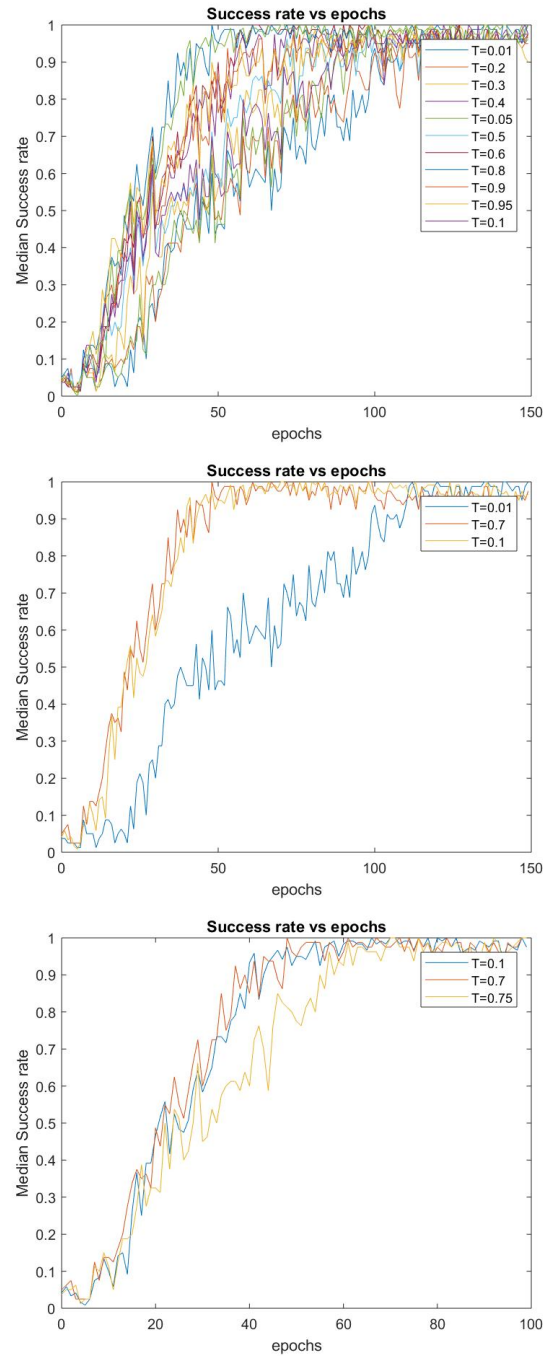


Figure 3.1: Success rate vs. epochs for various hyperparameters for *FetchPick&Place-v1* task.

The six hyperparameters are, in order, Polyak-averaging coefficient, discounting factor, learning rate for critic network, learning rate for actor network, percent of times a random action is taken, and standard deviation of Gaussian noise added to not

---

**Algorithm 3** Proposed GA+DDPG+HER Algorithm
 

---

```

1: Choose a population of  $n$  chromosomes
2: Set the values of hyperparameters into the chromosome
3: Run the DDPG + HER to get the number of epochs for which the algorithm first
   reaches success rate 0.85
4: for all chromosome values do
5:   Initialize DDPG
6:   Initialize replay buffer  $R$ 
7:   for episode=1, M do
8:     Sample a goal  $g$  and initial state  $s_0$ 
9:     for t=0, T-1 do
10:      Sample an action  $a_t$  using DDPG behavioral policy
11:      Execute the action  $a_t$  and observe a new state  $s_{t+1}$ 
12:    end for
13:    for t=0, T-1 do
14:       $r_t := r(s_t; a_t; g)$ 
15:      Store the transition  $(s_t; a_t; r_t; s_{t+1}; g)$  in  $R$ 
16:      Sample a set of additional goals for replay  $G := S(\text{current episode})$ 
17:      for  $g^j \in G$  do
18:         $r^j := r(s_t; a_t; g^j)$ 
19:        Store the transition  $(s_t; a_t; r^j; s_{t+1}; g^j)$  in  $R$ 
20:      end for
21:    end for
22:    for t=1, N do
23:      Sample a minibatch  $B$  from the replay buffer  $R$ 
24:      Perform one step of optimization using  $A$  and minibatch  $B$ 
25:    end for
26:  end for
27:  return 1=epochs
28: end for
29: Perform Uniform Crossover
30: Perform Flip Mutation at a rate of 0.1
31: Repeat for the required number of generations to find an optimal solution

```

---

completely random actions as a percentage of the maximum absolute value of actions on various coordinates. Since each hyperparameter requires 11 bits to be represented to three decimal places, we need 66 bits for the six hyperparameters. Then, using these string chromosomes, domain independent crossover and mutation operators can

produce new hyperparameter values. We looked at hyperparameter values up to three decimal places because even little changes in values can have a big impact on success rates. The best fit for our issue, for instance, is thought to be a step size of 0.001. It could be significant to note that each chromosome was precisely inspected at one time. We believe it is safe to assume that the fitness function has a low variation, which keeps the calculation cost cheap, even though identical chromosomal runs may have somewhat different results.

The fitness of each chromosome is determined by the inverse of the number of epochs required for the learning agent to reach close to the maximal success rate 0.85 for the first time (set of hyperparameter values). Our minimizing problem becomes a maximizing problem since GA always maximizes the objective function and fitness is inversely proportional to the number of epochs. Due to the time of each fitness evaluation, it is not practical to conduct an exhaustive search of the  $2^{66}$ -size search region. Therefore, we use a GA search.

## **3.6 Experimental Results**

### **3.6.1 Experimental setup**

As was said earlier, a chromosome is binary encoded. The result of merging all of the GA's arguments is each chromosomal string. The sample chromosome in Figure 3.2 has four binary-encoded hyperparameters.

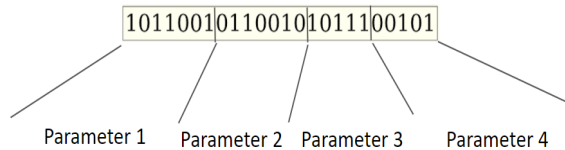


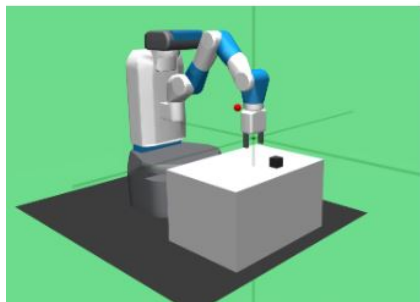
Figure 3.2: Chromosome representation for the GA.

The environments used to test robot learning on five distinct simulation tasks are *FetchPick&Place-v1*, *FetchPush-v1*, *FetchReach-v1*, *FetchSlide-v1*, and *DoorOpening* (see Figures 3.3 and 3.4).

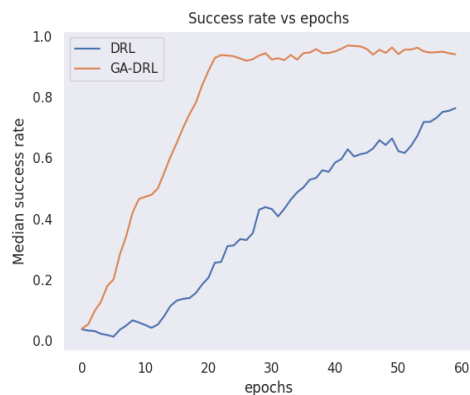
The *AuboReach* habitats shown in figures 3.5 and 3.6 are utilized in both simulated and real research.

We test our algorithm using these six gym environments. [101] contains the fetch environments *FetchPick&Place*, *FetchPush*, *FetchReach*, and *FetchSlide*. We developed two unique gym environments, *DoorOpening* and *AuboReach*. The six tasks are described in full below:

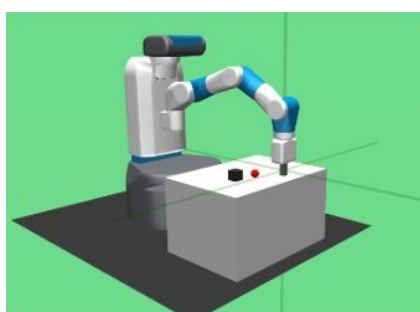
- **FetchPick&Place:** The robot proceeds to the objective location, which might be anywhere on the table or the space above it, after picking up the box from the table.
- **FetchPush:** There is a box kept in front of the robot. It rolls or pushes the box toward the desired spot on the table. It is forbidden for the robot to grab hold of the box.
- **FetchReach:** The robot must move it to the desired position in the vicinity of the end-effector.



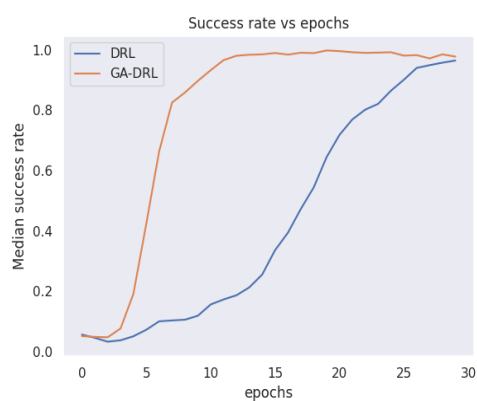
(a) FetchPick&amp;Place environment



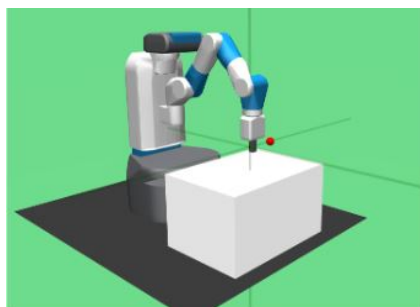
(d) FetchPick&amp;Place plot



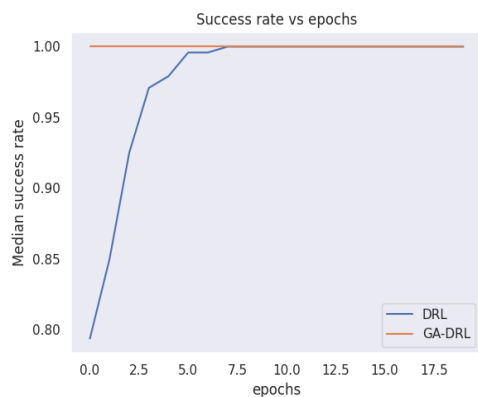
(b) FetchPush environment



(e) FetchPush plot



(c) FetchReach environment



(f) FetchReach plot

Figure 3.3: The matching DDPG+HER versus GA+DDPG+HER charts are produced once GA has identified all six hyperparameters. All graphs are averaged across ten runs. In this figure, DDPG+HER is referred to as DRL.

(a) Door Opening environment

(c) DoorOpening plot

(b) FetchSlide environment

(d) FetchSlide plot

Figure 3.4: The corresponding DDPG+HER versus GA+DDPG+HER charts are produced when all six hyperparameters have been discovered by GA. Each graph is averaged across ten runs. In this picture, DRL stands for DDPG+HER.

- ^ FetchSlide : The puck is put within the robot's reach on a slick surface. It must make contact with the puck hard enough for friction to cause it to stop in front of the goal.
- ^ DoorOpening : With the door handle pointed in the robot's direction, a simulated Aubo i5 manipulator is placed close to a door. By exerting pressure around the door handle, it is intended to force the door open.

Figure 3.5: Using the most accurate policy learned via GA+DDPG+HER, the AuboReach environment performs a task in a real experiment.

Figure 3.6: In a simulated experiment, the AuboReach environment performs a task using the best policy learned via GA+DDPG+HER.

^ AuboReach : Real or virtual Aubo i5 manipulators can learn to attain a particular joint configuration and use a gripper to pick up an object.

(a) GA+DDPG+HER over ten runs, vs. Original

(b) GA+DDPG+HER averaged over ten runs, vs. Original

Figure 3.7: Success rate vs. epochs for FetchPush-v1 task when  $\gamma$  and  $\beta$  are found using the GA.

### 3.6.2 Running GA

To evaluate the efficacy of our approach, we independently ran the GA on each of these scenarios and compared the outcomes to the original values of the hyperparameters. When contrasting GA+DDPG+HER with DDPG+HER, PPO, A2C, and DDPG, we would assume that the algorithm with the fewest episodes, running time, steps,

and average epochs will perform the best. This would serve as a demonstration of the need for optimizing hyperparameters rather than using the built-in default hyperparameters of the algorithms. The results of our FetchPush-v1 experiment are shown in Figure 3.7a. The GA was used to run the system and find the optimal values for the hyperparameters and  $\epsilon$ . Because the GA is probabilistic, the findings demonstrate that optimized hyperparameters determined by the GA can improve performance. We present the results from 10 GA runs. Faster learning and a better success rate are both advantages of the learning agent.

Figure 3.7b shows the average learning for the ten GA iterations as well as one learning run for the initial hyperparameter set. The outcomes shown in Figure 3.7 show changes when only two hyperparameters are modified as we assessed the genetic algorithm. We can identify where performance could be enhanced. Our conclusions from optimizing all five hyperparameters are listed below and support our optimism.

The comparison of one original experiment with two averaged runs for adjusting the hyperparameters  $\epsilon$  and  $\delta$  is shown in Figure 3.8 (b). We only ran this procedure twice because it can take several hours to complete in a single run and because it was a part of one of our initial testing. The outcomes shown in Figures 3.7 and 3.8 show changes when only two hyperparameters are changed as we assessed the genetic algorithm. We can identify where performance could be enhanced. Our results from optimizing each of the five hyperparameters provide evidence in support of our optimism.

Then, all hyperparameters were optimized using GA, with the outcomes for each task

(a) GA+DDPG+HER over 2 runs, vs. Original

(b) GA+DDPG+HER averaged over 2 runs, vs. Original

Figure 3.8: Success rate vs. epochs for FetchSlide-v1 task when  $\gamma$  and  $\beta$  are found using the GA.

being shown in Figures 3.3 and 3.4. The GA-discovered hyperparameters are contrasted with the initial hyperparameters of the RL algorithm in Table 3.1. Except for AuroReach, every simulation environment made use of the identical set of hyperparameters that GA+DDPG+HER found. Another set of hyperparameters was produced for AuroReach, as can be seen in table 3.1. The learning rates  $\alpha_{actor}$ , and

critic remain unchanged from the beginning, but the values of the other four hyperparameters have changed. The hyperparameters that the GA identified outperformed the initial hyperparameters, as shown in Figures 3.3 and 3.4 indicating that the learning agent was able to learn more quickly. Averaging across ten runs produced each of the plots in the previous figure.

		All environments except Aubo-i5	Aubo-i5 - Fixed Initial and Target state	Aubo-i5 - Random Initial and Target state
hyper-parameters	DDPG+HER	GA+DDPG+HER	GA+DDPG+HER	GA+DDPG+HER
	0.98	0.928	0.949	0.988
	0.95	0.484	0.924	0.924
actor	0.001	0.001	0.001	0.001
critic	0.001	0.001	0.001	0.001
	0.3	0.1	0.584	0.912
	0.2	0.597	0.232	0.748

Table 3.1: DDPG+HER vs. GA+DDPG+HER values of hyperparameters.

The Aubo-i5 robotic manipulator's custom-built gym environment was subjected to GA+DDPG+HER settings, which are listed in table 3.1 as shown in figures 3.5 and 3.6. The MOVEit package [102] controls the motors in this environment, but the DDPG+HER acts as the movement's brain. DDPG+HER decides the actions (combinations of joint states) the robot should take. At first, the outcomes were unexpected. It took several hours (10-15 hours) to finish each epoch. Since it might take several weeks, we did not finish the entire program. The same is true for settings for DDPG+HER. This is due to the Aubo i5 robotic manipulator's movements being

kept slow in both simulation and real-world investigations to prevent any unforeseen sudden movements that can be harmful. The successful completion of each action in the AuboReach environment also involves planning and execution procedures. Unlike the other gym settings examined in this study, AuboReach could only be run on a single CPU. This is because various settings were added to MuJoCo and were easily able to utilize the entire pool of CPUs. Faster learning is made possible by MuJoCo's ability to spawn several instances for training. Like a genuine robot, AuboReach can only carry out one action at a time. These qualities make training in this environment time-consuming.

### 3.6.3 Modifications required for AuboReach

Then, only action values in the AuboReach environment were subjected to the GA+DDPG+HER settings. This indicates that the robot did not run to perform the activity in both simulated and real studies. This also implies that the robot is deemed to have made whatever choice the DDPG+HER makes. This is further backed up by the fact that MOVEit takes care of the trouble of sending move signals to various joints. This is true since the robot is capable of carrying out every action suggested by the DDPG+HER algorithm. This is because internal joint movement in the robot requires planning and execution [103]. Additionally, by following these steps, the chance of a collision is avoided, which might have happened otherwise. With a significant decrease in training time that makes it possible to use this environment for

training, each epoch now took less than a minute to complete.

Now that some of the environmental obstacles had been removed, the GA+DDPG+HER settings (table 3.1) were re-applied. These hyperparameters did not perform any better than the original hyperparameters. We think this is because this environment is so much more complex and one-of-a-kind than others. To make sure that the environment is trainable, we took into account even additional factors. Four joints are used in this environment for training and testing (instead of six). The joints used are shoulder, forearm, upper-arm, and wrist1. This was done to make sure that the learning could be completed on time. The range of each joint is  $-1.7$  to  $1.7$  radians. The robot was in the upright position in both its initial and reset states, which are  $[0, 0, 0, 0]$ .

The GA+DDPG+HER method was improved for faster hyperparameter search and better learning with minor environment modifications. The ten successful epochs were included in the definition of success. Accordingly, the GA was deemed successful if it experienced a success rate of 100% for ten consecutive epochs. Experiments show that learning never converges when  $\alpha_{actor}$  and  $\alpha_{critic}$  are both bigger than 0.001. Algorithm's  $\alpha_{actor}$  and  $\alpha_{critic}$  were set to 0.001 as a consequence. When only action values are used, multi-threading can happen, which means four CPUs could be used. AutoReacher deems the DDPG+HER-determined joint states to be successful if the total difference between the target and the achieved joint states is less than 0.1 radians. The chosen objective joint states were  $[-0.503, 0.605, -1.676, 1.391]$ . These modifications

(a) Trained initial and target state

(b) Training is done with random initial and target joint states with 1 CPU

Figure 3.9: The success rate of theAuboReach task in comparison to epochs. The average of ten runs is shown in this graph. In this gure, "DRL" stands for DDPG+HER.

to the algorithm allowed us to discover a new set of hyperparameters, as illustrated in Table 3.1. The difference in training success rates between GA+DDPG+HER and DDPG+HER is shown in Figure 3.9a. The GA+DDPG+HER performs significantly better than the DDPG+HER.

The training was repeated using four CPUs to identify the optimum policy once the

GA+DDPG+HER had found the best hyperparameters for the AuboReach environment. Then, in both simulated, and real testing, the robots were put under this policy. The important thing to keep in mind is that the CPU utilization was set to one for testing purposes. The robot was able to go from the starting joint space of the training to the objective joint space in both studies. Because there was no new unpredictable element during training, there was just one viable course for the environment. There was no noticeable difference throughout testing because both DDPG+HER and GA+DDPG+HER ultimately had a 100% success rate. The primary difference is how quickly the environment can learn given a particular set of hyperparameters.

In a different experiment, AuboReach was modified to learn about random joint states. The update allowed the robot to start and complete tasks in a variety of joint states during testing. The GA was applied to this setting, and the hyperparameters it identified are displayed in table 3.1. According to Figure 3.9b, the plot of GA+DDPG+HER is still superior to DDPG+HER. In the figures 3.5 and 3.6, the robot can be seen picking up an object in both real-world and simulated tests.

The performance of the algorithm was enhanced by the automatic DDPG+HER hyperparameter adjustment brought about by the usage of GA+DDPG+HER in the AuboReach environment.

### 3.6.4 Training evaluation

A GA's operational state must be closely monitored to ensure that the system's performance is maximized. Certain chromosomes will perform better than others due to the way GAs function. The performance graph is therefore unlikely to have a smooth increase curve. When a fitness function test yields a zero, it indicates that the chromosome is not suitable for use. Although the slope is not smooth, it is anticipated that as GA develops, the system's overall performance will get better.

To monitor GA's development as we searched for the optimum hyperparameters, we generated a ton of charts. The figures 3.10, 3.11, and 3.12 demonstrate how the system's performance increases as GA advances. When assessing training effectiveness, hyperparameters, such as the median success rate across fitness function evaluations, total reward across episodes, and epochs to achieve the target are all taken into consideration. The system's overall performance is improving, as can be seen. When fitness function evaluations are used, the overall reward rises as the number of episodes and epochs the robot needs to complete the task falls. This shows that the GA is making progress in identifying the ideal hyperparameter values. Because each GA run requires many hours to many days of run time, we only plotted data for one GA run and limited the length of a GA run. We chose to end the GA once we began to notice results.

We will use the GA's identified hyperparameters to assess the system's effectiveness now that it has performed as expected.

- (a) FetchPick&Place - Median reward vs Times GA fitness function evaluated
- (b) FetchPick&Place - Epochs vs Times GA fitness function evaluated
- (c) FetchPick&Place - Median success rate vs Times GA fitness function evaluated
- (d) FetchPush - Median reward vs Times GA fitness function evaluated
- (e) FetchPush - Epochs vs Times GA fitness function evaluated
- (f) FetchPush - Median success rate vs Times GA fitness function evaluated

Figure 3.10: The GA+DDPG+HER training evaluation charts are produced after GA discovers all six hyperparameters. This was the result of just one GA run.

### 3.6.5 Efficiency evaluation

To assess and compare the effectiveness of the GA+DDPG+HER algorithm for teaching the robot to complete a task, we generated data for a variety of hyperparameters.

These hyperparameters serve as reliable gauges of algorithm performance. The overall

- (a) FetchSlide - Median reward vs Times GA fitness function evaluated (d) DoorOpening - Median reward vs Times GA fitness function evaluated
- (b) FetchSlide - Epochs vs Times GA fitness function evaluated (e) DoorOpening - Epochs vs Times GA fitness function evaluated
- (c) FetchSlide - Median success rate vs Times GA fitness function evaluated (f) DoorOpening - Median success rate vs Times GA fitness function evaluated

Figure 3.11: The GA+DDPG+HER training evaluation plots when GA identified all six hyperparameters. This outcome came from a single GA run.

reward has greatly increased for the bulk of the training tasks, as illustrated in Figure 3.13. When rewards were increased, the DDPG+HER algorithm's efficiency greatly increased. The robot can learn a great deal faster since it is guided much more quickly toward the intended job. We averaged these plots over ten runs to provide a fair assessment. The FetchSlide environment performed poorly with GA+DDPG+HER.

- (a) AuboReach - Median reward vs Times GA fitness function evaluated (d) FetchReach - Median reward vs Times GA fitness function evaluated
- (b) AuboReach - Epochs vs Times GA fitness function evaluated (e) FetchReach - Epochs vs Times GA fitness function evaluated
- (c) AuboReach - Median success rate vs Times GA fitness function evaluated (f) FetchReach - Median success rate vs Times GA fitness function evaluated

Figure 3.12: The GA+DDPG+HER training evaluation charts are produced after GA learns all six hyperparameters. This was the result of just one GA run.

The difficulty of the undertaking, in our opinion, is to blame. Jobs that failed to reach the target during training were represented in the tables by the hyperparameter's maximum value.

We also produced more data to assess how many episodes, running times (s), steps,

(a) FetchPick&Place

(d) FetchSlide

(b) FetchPush

(e) DoorOpening

(c) FetchReach

(f) AuboReach

Figure 3.13: The DDPG+HER vs. GA+DDPG+HER efficiency evaluation charts are produced when all six hyperparameters are determined using GA (Total reward vs episodes). All graphs are averaged across ten runs. In this figure, DDPG+HER is referred to as DRL.

and epochs a robot would need to learn to accomplish the desired goal. This information is presented in the tables 3.2-3.5. An average of 10 runs is used to generate the data in the tables. Table 3.2 compares the number of episodes required for a robot to accomplish a task. The performance of the bolded numbers suggests greater performance, and most environments exceed the other algorithms. In the Fetch-Push environment, learning the task takes 54% fewer episodes than it does in the DDPG+HER environment.

Method	Fetch Pick & Place	Fetch Push	Fetch Reach	Fetch Slide	Door Opening	Aubo Reach
DDPG+HER	6,000	2,760	100	4,380	960	320
GA+DDPG+HER	2,270	1,260	60	6,000	180	228
PPO	2,900	2,900	1,711	4,880	1,500	2,900
A2C	119,999	119,999	119,999	119,999	119,999	32,512.3
DDPG	10,000	2,000	423	10,000	706	1,000

Table 3.2: Efficiency evaluation: For all activities, compare average (over ten runs) episodes to accomplish the target.

Running time is another aspect to take into account when assessing an algorithm's effectiveness. Time is measured in seconds. If learning the task takes less time, the algorithm is superior. According to Table 3.3, the GA+DDPG+HER algorithm has the shortest running time in the majority of the situations. For instance, FetchPush using GA+DDPG+HER takes roughly 57:004% less time than the DDPG+HER method.

Method	Fetch Pick & Place	Fetch Push	Fetch Reach	Fetch Slide	Door Opening	Aubo Reach
DDPG+HER	3,069.981	1,314.477	47.223	2,012.645	897.816	93.258
GA+DDPG+HER	1,224.697	565.178	28.028	3,063.599	167.883	66.818
PPO	1,964.411	2,154.052	776.512	2,379.393	997.779	710.576
A2C	2,025.344	2,082.807	2,061.807	2,268.114	2,718.769	214.075
DDPG	5,294.984	1,000.586	236.4	5,346.516	438.7	1,721.992

Table 3.3: Efficiency evaluation: For all activities, compare the average (over ten runs) running time (s) to attain the target.

When examining and evaluating the performance of the GA+DDPG+HER algorithm, another factor to take into account is the average number of steps required to reach the goal. In Table 3.4, the typical number of steps taken by a robot in each environment are displayed. The majority of environments outperform all other algorithms when used with GA+DDPG+HER, except for the FetchSlide environment. FetchPush with GA+DDPG+HER requires roughly 54 :35% fewer steps than the DDPG+HER method.

Method	Fetch Pick & Place	Fetch Push	Fetch Reach	Fetch Slide	Door Opening	Aubo Reach
DDPG+HER	300,000	138,000	5,000	219,000	48,000	65,600
GA+DDPG+HER	113,000	63,000	3,000	300,000	9,000	46,000
PPO	595,968	595,968	324,961	1,000,000	301,056	595,968
A2C	600,000	600,000	600,000	600,000	600,000	162,566.5
DDPG	500,000	100,000	21,150	500,000	35,300	200,000

Table 3.4: Efficiency evaluation: For all activities, compare the average (over ten runs) steps taken to attain the target.

The last hyperparameter utilized to contrast the performance of GA+DDPG+HER with four other algorithms is the number of epochs required for the robot to reach the target. Table 3.5 displays the average epochs for each environment. In terms of efficiency, almost all environments do better than GA+DDPG+HER. For instance, Fetch-Push requires 5435% fewer epochs with GA+DDPG+HER than with DDPG+HER.

Method	Fetch Pick & Place	Fetch Push	Fetch Reach	Fetch Slide	Door Opening	Aubo Reach
DDPG+HER	60	27.6	5	43.8	47	16
GA+DDPG+HER	22.6	12.6	3	60	8	11.4
PPO	290	290	171.1	488	150	290
A2C	1,200	1,200	1,200	1,200	1,200	325.2
DDPG	1,000	100	42.3	1,000	70.6	100

Table 3.5: Efficiency evaluation: Average (over ten runs) epochs comparison to reach the goal, for all the tasks.

The GA+DDPG+HER algorithm's overall comparison to the other algorithms is then presented.

### 3.6.6 Analysis

In the preceding subsections, we presented numerous findings and the methodology for comparing the performance of the GA+DDPG+HER algorithm to that of the DDPG+HER, PPO, A2C, and DDPG algorithms. Overall, GA+DDPG+HER performs best, with the FetchSlide environment being the only exception. The average comparison tables show how each environment can assume a different value

(a) FetchPick&Place

(d) FetchSlide

(b) FetchPush

(e) DoorOpening

(c) FetchReach

(f) AuboReach

Figure 3.14: GA+DDPG+HER comparison with PPO [6], A2C [7], DDPG+HER (DDPG[8] + HER[9]) and DDPG [8]. All plots are averaged over ten runs. The term "DRL" in this figure refers to DDPG+HER.

for the assessment hyperparameters. The kind of task the robot is attempting to learn will determine this. The majority of the tasks outperformed DDPG+HER with an improvement in the efficiency of more than 50%, although FetchSlide performed worse than DDPG+HER. This performance is also attributable to the task's objective. This operation is distinct since the end-effector does not physically move to the location where the box should be placed. A mean of more than ten runs and a range of hyperparameters were used to test GA+DDPG+HER. This alone demonstrates that GA+DDPG+HER outperformed several other algorithms. Figures 3.3, 3.4 and 3.9b corroborate our claim by showing that the task can be learned significantly more quickly in most situations when GA+DDPG+HER is used. Finally, we compare the outcomes of five distinct methods (PPO, A2C, DDPG, DDPG+HER, GA+DDPG+HER) in Figure 3.14. As can be seen, in every scenario, GA+DDPG+HER outperforms all other algorithms; the only exception is FetchSlide where DDPG+HER surpasses GA+DDPG+HER. We believe the total success percentage on this task (FetchSlide) is minimal, even for the winning DDPG+HER (the highest success rate is probably around 0.6 or 60%). We believe that in terms of problem-solving time, GA+DDPG+HER performs similarly to its counterpart DDPG+HER, but differs when a problem may have fewer genetic solutions. DDPG+HER surpasses other algorithms in every situation (such as PPO, DDPG, and A2C). This might be a result of HER's better performance after being added to DDPG. Additionally, DDPG+HER performs better in situations with sparse rewards [104]. Except for the FetchReach

and DoorOpening environments, where it has a zero success rate, DDPG alone (without HER) cannot operate in any environment. We believe this is the case because, for DDPG, dense rewards are frequently easier to learn from than sparse ones. FetchReach environment is unquestionably quite straightforward, and all configurations may properly solve it. In a DoorOpening environment, DDPG can function, but not as well as DDPG+HER, and GA+DDPG+HER. All environments for PPO have a zero success rate, except for FetchReach. This might be the case since PPO performs better in discrete environments than in continuous environments. This might also be the case for A2C, which has a success rate of 0 in all environments except for FetchReach and AutoReach. In conclusion, GA+DDPG+HER outperforms all of these methods. This illustrates how hyperparameter adjustments with GA can enhance performance.

### 3.7 Summary

Early results from this study indicated how a genetic algorithm may improve the hyperparameters of a reinforcement learning system to improve performance, as shown by quicker competence on six manipulation tasks. We reviewed earlier work on reinforcement learning in robotics, presented the GA+DDPG+HER algorithm for reducing the time to reach peak performance, and discussed why a GA would be suitable for such optimization. This chapter discusses RL, DDPG, and HER. The main contribution, as described in [47], [105], [106] and [107], is also presented in this chapter. Experimental results were compared between GA+DDPG+HER [47]

and existing scenarios. Our findings on the six manipulation tasks demonstrate that the GA is capable of determining the hyperparameter values that result in quicker learning and better (or equal) performance on the tasks we have selected, supporting our hypothesis that GAs are an appropriate fit for such hyperparameter optimization. We evaluated GA+DDPG+HER with alternative strategies and concluded that ours was the most successful. In conclusion, the implementation validates the hypothesis that when hyperparameters for DDPG+HER are found via GA+DDPG+HER, real robots learn more fast. The actual robot environment, `AuboReach`, allowed for quick changes between the initial and destination states. It is also possible to argue that GA+DDPG+HER, has the same limitations when a problem may only have a small number of genetic solutions, such as in the `FetchSlide` environment, but behaves similarly to its equivalent of DDPG+HER in terms of problem-solving speed. It was found that GA-found parameters outperformed all the methods.

We also demonstrated that heuristic search, as used by genetic and other evolutionary computing techniques, is a feasible computational tool for enhancing the performance of sensor odometry and reinforcement learning. During system execution, adaptive genetic algorithms can be used to adapt to various sets of hyperparameters. This might be a sign of online hyperparameter change, which can enhance the performance of any system, irrespective of the testing environment or domain.

## Chapter 4

### AACHER: Assorted Actor-Critic

### Deep Reinforcement Learning with

### Hindsight Experience Replay

#### 4.1 Introduction

Deep Learning, a branch of machine learning, uses hierarchical structures to extract high-level abstractions from data. It is a developing strategy that has been used in a range of industries, including transfer learning [108], the medical industry [109], the genetics industry [48], and many more [20, 110{116}. Robotics breakthroughs are shown by Deep Reinforcement Learning (DRL) [117] methods. The enormous number of interaction samples that are frequently needed for training in both simulated

and real environments is one of the restrictions [118]. Numerous studies have been conducted in this area since many years ago, with some focused on continuous action spaces and others on discrete action spaces [47]. Autonomous robots use Q-learning techniques for a variety of tasks. [119] Several robotic tasks, including locomotion [59], manipulation [120], [121], [57], and autonomous vehicle control [122] have been controlled via reinforcement learning (RL) techniques [14]. Robotic hands that passively adjust to environmental uncertainty are one of the most effective implementations of reinforcement learning. Soft body robots can boost movement over soft materials by performing simple tasks like grabbing, according to [123]. [105], [47], [106], [124] also use robotic hand manipulators in similar ways.

Reinforcement learning uses three different types of algorithms: actor-only, critic-only, and actor-critic technique [125]. Deep Deterministic Policy Gradient (DDPG) [8], a new deep reinforcement learning method, has recently demonstrated strong performance in a variety of simulated continuous control tasks. In actor-critic techniques, the deep deterministic policy gradient (DDPG) algorithm is crucial. In DDPG, the experience replay (ER) [126] approach is important. HER (Hindsight Experience Replay) [9], which enables sample-efficient learning from sparse and binary incentives and avoids the need for complicated reward engineering, is one of the common experience replay techniques.

Utilizing Hindsight Experience Replay (HER), we are comparing our work against that of DDPG as part of our research. The most current instance of DDPG+HER

usage was found in [124].

Actor and critic learning make-up DDPG. Both actors and critics have a big impact on how well the algorithm performs. One of the research [127] recommended using double experience replay (DER), an extension of the experience replay mechanism, to further enhance the stability and performance of DDPG. This resulted in a more stable training process and preferred performance in practice. In contrast to other studies, our approach employs [128] to develop a unique algorithm that can interact with more actors and critics in addition to HER. We call this system Actor-Critic Deep Reinforcement Learning with Hindsight Experience Replay or AACHER. The state-of-the-art DDPG algorithm is used in AACHER together with several actor-critic schemes, and its advances are evaluated in a variety of scenarios under diverse conditions. The technique is used on five goal-based gym environments constructed using specially designed robotic manipulators: AuroReach, FetchReach-v1, FetchPush-v1, FetchSlide-v1 and FetchPick&Place-v1. To establish whether the method is beneficial in boosting the overall effectiveness of the learning process, the full algorithm is also looked at. The final results support our claim and provide resounding evidence that adding more actors and critiques improves the robot's overall learning process. Open source code is available at <https://github.com/aralab-unr/multi-actor-critic-ddpg-with-auro>.

The main contributions of this chapter are listed below:

- ^ AACHER is a unique algorithm developed by combining advanced DDPG with

HER (Actor-Critic Deep Reinforcement Learning with Hindsight Experience Replay).

- ^ AACHER creates independent instances for a large number of actors, critics, or both, using various combinations of actors and critics.
- ^ We used the average of the actor and critic networks to further calculate the loss and actions. The term "average" in this context refers to both the average of the critic and actor networks. The calculated average for both actors and critics is used by the target networks, which apply the updated parameters.
- ^ To analyze the algorithm, we created Aubo-i5 custom environments (referred to as AuboReach).
- ^ AACHER is also used in four of OpenAI's gym settings FetchReach-v1, FetchPush-v1, FetchSlide-v1 and FetchPick&Place-v1
- ^ The usefulness of AACHER was investigated using various combinations of actors and critics in both simulated and real manipulation tasks.
- ^ AACHER's outcomes were contrasted with those of DDPG+HER.
- ^ Our tests demonstrate how AACHER performs better than DDPG+HER, demonstrating the value of using several actors and critics in DDPG.

## 4.2 Reinforcement Learning

In a typical reinforcement learning (RL) scenario, a robot interacts discretely with the environment to reap as many rewards as it can. How a robot interacts with the environment in reinforcement learning is depicted in Figure 4.1. The problem may be modeled using a Markov Decision Process (MDP) as a tuple of  $S; A; P; R; \gamma$  [98], [99].  $S$  is the set of states, where  $A$  denotes the collection of actions.  $P : S \times A \times S \rightarrow [0; 1]$  is the transition probability function.  $R : S \times A \rightarrow \mathbb{R}$  is the reward function, and  $\gamma$  is a scalar discount factor that is equivalent to the idea in Bellman Equation [129]. The robot wants to maximize the expected value of discounted total rewards:  $R_t = \sum_{i=t}^T \gamma^{i-t} r(s_i; a_i)$  by learning a perfect policy, where  $t$  is the time step that terminates at time  $T$  and  $r(s_t; a_t)$  denotes the reward taking action  $a_t$  in state  $s_t$ .

Figure 4.1: Interaction between a robot and its environment in reinforcement learning (RL).

More specifically, the long-term reward is represented by the state-action value function with policy:

$$\begin{aligned}
 Q(s_t; a_t) &= E[R_t | s = s_t; a = a_t] \\
 &= E \left[ \sum_{i=t}^{\infty} \gamma^{i-t} r(s_i; a_i) \right]; \quad (4.1)
 \end{aligned}$$

where the expected reward for a state if a robot obeys a policy is represented by the state-value function:

$$V(s) = E[R_t | S_t = s]; \quad (4.2)$$

In an actor-critic reinforcement learning [130], actor and critic are specified as state-action value function  $Q(s_t; a_t)$  and actor function  $\pi(s_t; a_t)$  with parameters  $\theta$  and  $\phi$ , respectively. The actor adopts the guidelines after consulting the critic, who also offers an estimation of the entire compensation. It is well known that the critic reduces the squared TD error loss function [131] to grasp the state-action value function:

$$L(\theta, \phi) = (r(s_t; a_t) + \gamma Q(s_{t+1}; a_{t+1}; \phi) - Q(s_t; a_t; \theta))^2; \quad (4.3)$$

However, the critic tends to diverge when deep neural networks are used to approximate the state-action value function, leading to the invalidity of an actor as well.

In particular, recent successful RL research on experience replay and target networks [132] helps resolve the issue. Since both of these are relevant to the DDPG strategy,

in-depth explanations of each are given in the next section.

### 4.3 Deep Reinforcement Learning (DRL)

Since its introduction, extensive research has been conducted in this field [50], with some work focused on continuous action spaces [51{54] and others on discrete action spaces [55]. Q-learning approaches [50] is used by autonomous robots to carry out a variety of tasks [56]. [14] The use of Reinforcement Learning (RL) has been beneficial for both locomotion [59, 60] and manipulation [57, 58]. The comprehensive DRL taxonomy is explained in Figure 4.2.

Figure 4.2: Taxonomy of Deep Reinforcement Learning (DRL).

It has been established that model-free deep reinforcement learning algorithms are capable of learning a wide range of tasks, from mastering complex locomotion strategies [133] to playing video games from images [134]. Model-based algorithms often perform asymptotically worse than model-free learners due to model bias. We chose DDPG for our experiments since it is a well-known model-free DRL algorithm that is based on policies. Recent successful RL experiments, like Experience replay [126] and Target networks [132], have significantly improved DDPG performance. When adopting hindsight experience replay (HER), a technique for experience replay, the robustness and sample efficiency of goal-achieving procedures are typically increased. [9].

In this chapter, our testing will combine DDPG with Hindsight Experience Replay (HER). A summary of recent findings on how experience ranking might accelerate the learning rate of DDPG+HER can be found in [73]. A detailed explanation of DDPG and HER is given in the following subsection.

## 4.4 Deep Deterministic Policy Gradients (DDPG)

The Deep Deterministic Policy Gradients (DDPG) method [8] integrates a variety of approaches to deal with continuous control problems. The algorithm known as Deterministic Policy Gradients (DDPG) is an extension of the DPG algorithm. The

DDPG method, initially proposed in [135], consists of the actor-critic technique, experience replay, target network, and deterministic policy gradient theorem. The main contribution is the demonstration of deterministic policy  $\pi : S \rightarrow A$ , which generates the precise action for the robot by providing the state instead of a probability distribution across all the actions. According to the DDPG approach, the objective is:

$$\begin{aligned} J(\pi) &= \int_S \int_A p(s) \pi(a|s) r(s; a) da ds \\ &= E_{s \sim p; a \sim \pi} [r(s; a)]; \end{aligned} \quad (4.4)$$

where  $p(s)$  represents the state distribution. According to [98] and [130], the deterministic policy's objective is:

$$\begin{aligned} J(\pi) &= \int_S p(s) r(s; \pi(s)) ds \\ &= E_s [r(s; \pi(s))]; \end{aligned} \quad (4.5)$$

The state-action value (or critic) network  $Q(s_t; a_t; \theta)$  and the actor network  $\pi(s_t; \mu)$  are both designed to replicate the state-action value function and actor function, respectively, in the DDPG approach. The neural networks' parameters are  $\theta$  and  $\mu$ . When the networks are updated, the experiences utilized for training are taken from the experience replay. Typically, the 4-element tuples  $(s_t; a_t; r_t; s_{t+1})$  is kept in a buffer that provides a batch of them for updating actor and critic networks. Since

the current experience will replace the older one once the buffer is full, just a small percentage of the most recent experiences are kept. By offering a consistent target for the training procedure, the target network, which is normally constructed as a replica of the main network, is also utilized to update the critic/actor network. The target network, denoted as  $Q^{\text{tar}}$ , is utilized in place of  $Q^0(s_{t+1}; a_t)$  in 4.3:

$$L_{\text{tar}}(\theta) = \mathbb{E} \left[ r(s_t; a_t) + Q^{\text{tar}}(s_{t+1}; a_{t+1}; \theta) - Q(s_t; a_t; \theta) \right]^2; \quad (4.6)$$

where  $\theta$  is the parameter from the previous iteration. The experience replay and target networks are equally important for enabling deep neural networks and sustaining the DDPG method's training.

## 4.5 Hindsight Experience Replay (HER)

Hindsight Experience Replay (HER) [9] is an efficient technique for Experience Replay that can be combined with any off-policy RL algorithm. When there are numerous goals, HER might be used. To train robots more quickly in expansive state and action settings, HER uses a sophisticated technique. HER aims to learn from failures by modeling human behavior. Robots always learn from experience, regardless of whether they ultimately succeed in producing the desired outcome. HER considers the robot's fulfillment of any requirement to be the modified aim. Only the transition  $(s_t; g; a; r_t; s_{t+1}; g)$  with the initial objective  $g$  is captured in a typical experience

Figure 4.3: DDPG+HER explanation

replay. A transition from the original goal  $g$  to the modified goal  $g^0$  is also stored in HER as  $(s_t, j, g^0; a_t; r_t^0; s_{t+1}, j, g^0)$ . HER operates admirably with very few incentives and is preferred to tailored rewards in this respect. The operation of DDPG+HER is shown in Figure 4.3. New goals  $g^0$  are chosen at random by the HER algorithm from the batch of episodes [136]. These goals describe random states that a robot might reach during the episode. To create new transitions connected to new goals, rewards for these new additional goals will then be calculated and saved in the replay buffer for policy training. A future mechanism is used by the HER to choose new goals, and the additional goals selected are random states from the same episode as the transition being repeated.

## 4.6 Open Problem Discussion

The effectiveness of DDPG+HER raises questions. If an actor or critic has a poor performance, using a single actor or critic network is problematic. Utilizing several actor/critic networks can help to prevent situations like this. When the average of all actors and critics is used in place of utilizing a single actor and critic, this works well within the conventional DDPG technique. This can help a robot learn a task considerably more quickly by producing a greater reward and Q-value. The next sections will show how using more than one actor or critic has a significant impact on the robot's competence. In a later section of this chapter, the solution to this issue is discussed, and the results of the experiments that go along with it demonstrate that the proposed solution performs superior to the existing reinforcement learning method.

## 4.7 AACHER

Although there are still areas for performance improvement and stability issues, DDPG has shown good performance in several areas [137]. The effectiveness of the actor/critic learning process is particularly important to the training of the DDPG approach since the actor's learning process heavily relies on the critic. It is possible to get a significant improvement in stability and performance using the DDPG method

and an average actor/critic. This section presents the details of the multi-actor/critic DDPG+HER technique (AACHER).

---

Algorithm 4 AACHER

---

```

1: Create D Actor Neural networks
2: Create P Critic Neural networks
3: Initialize losses as an average of Actor and Critic neural networks
4: Initialize replay buffer R
5: for episode=1, M do
6:   Sample a goal and initial state  $s_0$ 
7:   for  $t=0, T-1$  do
8:     Sample an action  $a_t$  using behavioral policy generated by taking an average
       of policy neural networks
9:     Execute the action  $a_t$  and observe a new state  $s_{t+1}$ 
10:   end for
11:   for  $t=0, T-1$  do
12:      $r_t := r(s_t; a_t; g)$ 
13:     Store the transition  $(s_t, a_t, r_t, s_{t+1}, g)$  in R
14:     Sample a set of additional goals for replay  $G := S(\text{current episode})$ 
15:     for  $g^0 \in G$  do
16:        $r^0 := r(s_t; a_t; g^0)$ 
17:       Store the transition  $(s_t, a_t, r^0, s_{t+1}, g^0)$  in R
18:     end for
19:   end for
20:   for  $t=1, N$  do
21:     Sample a minibatch B from the replay buffer R
22:     Perform one step of optimization using A and minibatch B
23:   end for
24: end for

```

---

The actor network is represented by the average of actor values, and the state-action value function is approximated by the average of critic values since AACHER uses both D actors and P critics in its actor-critic architecture:

$$\begin{aligned}
 \mu_{\text{avg}}(s; \theta) &= \frac{1}{D} \sum_{i=1}^D \mu_i(s; \theta_i); \\
 Q_{\text{avg}}(s; a; \phi) &= \frac{1}{P} \sum_{i=1}^P Q_i(s; a; \phi_i);
 \end{aligned} \tag{4.7}$$

where the  $i$ -th actor and critic parameters, respectively, are represented by  $\theta_i$  and  $\phi_i$ . The AACHER method creates D/P independent actor/critic networks as opposed to the actions/Q-values that were previously learned. The average of all actors/critics will therefore considerably reduce the adverse effects when one actor or critic gives a bad performance. Additionally, a wide range of independent actors and critics can learn more about the environment. The critic networks are updated by TD errors, whereas the actor networks are composed of a parameterized collection of policies that are typically updated by a policy gradient.

TD errors between the average critic and the target critic are used in critic training:

$$L_{\text{avg}}(\phi) = \mathbb{E} \left[ r(s; a) + Q_{\text{avg}}^{\text{tar}}(s; a; \phi) - Q_{\text{avg}}(s; a; \phi) \right]^2: \tag{4.8}$$

The AACHER approach is explicitly stated in Algorithm 4, and DDPG and HER's basic conception is unchanged. Additional goals that are sampled from  $G$  are stated in line 15 of the algorithm. The hyperparameter  $k$  regulates the ratio of the total number of elements in  $G$  and  $G^0$ . Here,  $k$  random states from the same episode as

the transition that was replayed will be used to select additional goals. The diagram in Figure 4.4 provides a visual representation of AACHER. It exhibits average actors and critics across both the main and the target networks.

Figure 4.4: Use of multiple actors and critics in AACHER

We created a naming convention so that we could easily distinguish between the several tests we ran with AACHER. A diagrammatic representation of the naming convention can be seen in Figure 4.5. The abbreviation AD<sub>D</sub>CP is used to refer to the experiments, where A stands for an actor, D for the total number of actors, C for the critic, and P for the total number of critics. As an example, the notation A2C3 indicates the use of two actors and three critics in the main and target networks, which are then averaged for training.

Figure 4.5: Naming convention for AACHER-based experimentation.

## 4.8 Experimental Results

### 4.8.1 Simulated environments

We are utilizing four Open AI gym environments in this chapter: FetchReach-v1, FetchPush-v1, FetchSlide-v1 and FetchPick&Place-v1 [101]. In addition to these four environments, we are also using the AuroReach environment, a customized gym environment we developed for this research. In AuroReach, the environment includes an Auro i5 manipulator that follows the operations to transition from the starting joint state configuration to the destination joint state configuration (robot joint states). The initial and objective states' configurations are random. In this environment, four joints are employed for training and testing (instead of six). The joints that are utilized are the shoulder, forearm, upper arm, and wrist1. This was done to ensure that the learning could be completed quickly. Each joint can move in a range of -1.7 to 1.7 radians. Real and simulated AuroReach settings are shown in 4.6 and 4.7.

Figure 4.6: The AuboReach environment executes a task in a real experiment using the most accurate policy learned using AACHER.

Figure 4.7: The AuboReach environment performs a task in a simulated experiment using the best policy discovered via AACHER.

## 4.8.2 Experimental setup

In this subsection, the precise conditions for each experiment are described. Each experiment has a set of specified conditions. Learning rates for actors and critics are fixed at 0.001. The discount factor is 0.98, and the update rates for target networks are 0.01. During exploration, a zero-mean Gaussian noise with a variance of 0.2 is added to the action. Each experiment's training approach consists of 25 epochs, each of which has 15 cycles. The robot performs 100 steps in each cycle, followed by 20 rounds of robot training. The batch size is set to 256 by default. The experience replay buffer is built as a circular queue with a length of 10<sup>6</sup>. During each trajectory roll-out, all experience tuples (state, action, reward, and future state) are saved and kept in a replay buffer of finite size. When updating the value and policy networks, we take a random sample of the replay buffer's experience. The replay buffer will automatically keep 20% of the typical transitions with the original goal because there are  $k=4$  additional goals used in the replay. L2 regularization is applied to the loss when the weights of the neural network become extremely large. Layer normalization is used by actor networks and critic networks alike [138]. The observations made by the robot are also normalized. A momentum-based technique called Adam [139] is used to optimize the loss function during training. For our experiments, we are utilizing a GeForce GTX 1080 Ti graphics card and Ubuntu 16.04.

The actor and the critic have three hidden layers, each with 256 units, in every

experiment. AACHER experiments with various actor and critic network combinations. Both the actor and critic networks have comparable shapes. DDPG+HER and AACHER were tested independently 20 times each to confirm that the experiment's results were accurate.

The robot was trained in a simulated AuroReach environment to avoid colliding with nearby objects and to make the training process simpler. Additionally, during simulation training, the robot's motors were turned off. In other words, it is anticipated that the robot can effectively transition to any action (set of joint states) that is chosen by any of the algorithms under test. The implementation of this environment makes use of the MOVEit package [102], which handles the planning and execution necessary for internal joint movement in the robot [103]. Furthermore, by taking these measures, a collision that would have occurred is prevented. Therefore, it is safe to presume that there won't be any collisions between the robot's joints.

If the overall deviation between the target and the obtained joint states is less than 0.1 radians, AuroReach rates the algorithm-determined joint states as successful. The training's target joint states were [-0.503, 0.605, -1.676, 1.391]. Using the policy created by training in AuroReach, the random initial and target state configurations were tested in real-world and computer-generated environments with motors turned on. The user-selected initial and target state settings were successfully transitioned between by the robot.

### 4.8.3 Experimental results

This subsection presents all of the tests for the DDPG+HER and AACHER techniques. They were all completed in the simulated environment that was previously mentioned. Furthermore, these results were subjected to objective evaluations.

We carried out studies utilizing the methodology in the environments of AuroReach, FetchReach-v1, FetchPush-v1, FetchSlide-v1, and FetchPick&Place-v1. To prevent any unexpected manipulator movements, training was done specifically for AuroReach in a virtual environment. Following the training, a real AuroReach manipulator configuration was used to test the policy. Despite the absence of measures for the actual setup, it supports the effectiveness of the trained policy.

The AuroReach, FetchReach-v1, FetchPush-v1, FetchSlide-v1, and FetchPick&Place-v1 environments are used to evaluate the performance of the DDPG+HER and AACHER techniques. These three matrices: average Q values, success rate, and reward are used for evaluation. Each plot is averaged over 20 runs and plotted against epochs. The AuroReach environment served as the initial testing ground for our algorithm. The plots in Figure 4.8 illustrate our findings in the AuroReach environment. For additional comparison, the two top-performing A10C10 and A20C20 were chosen and plotted against DDPG+HER in a separate Figure 4.9. The colored shaded areas in the figures represent the ranges of maximum and minimum values for each experiment's results.

(a) Success rate vs Epochs

(d) Zoomed - Reward vs Epochs

(b) Zoomed - Success rate vs Epochs

(e) Average Q value vs Epochs

(c) Reward vs Epochs

(f) Zoomed - Average Q value vs Epochs

Figure 4.8: Plots comparing several experiments run in theAuboReach environment while using AACHER and DDPG+HER. Each experiment in the AACHER is marked following the naming convention. Comparisons have been made between success rate, reward, and average Q value. These are all averaged over 20 runs and plotted against epochs. Each plot is also shown in a zoomed-in view for clarification.

(a) Success rate vs Epochs

(d) Zoomed - Reward vs Epochs

(b) Zoomed - Success rate vs Epochs

(e) Average Q value vs Epochs

(c) Reward vs Epochs

(f) Zoomed - Average Q value vs Epochs

Figure 4.9: Plots versus DDPG+HER are shown for the two best-performing experiments, A10C10 and A20C20, when they are applied to the AutoReach environment. Plots showing success rate, reward, and average Q values are shown using epochs and an average of 20 runs. The range of values for each plot for 20 runs is shown in the shaded area. For clarity, each plot has a zoomed-in version.

Finally, we tested our method using the following four simulated environments: FetchReach-v1, FetchPush-v1, FetchSlide-v1 and FetchPick&Place-v1. In comparison to DDPG+HER, we only used the top-performing A10C10 and A20C20 from the AuroReach environment for subsequent evaluations in these environments. The average Q values, success rate, and reward plots for each of these environments are shown in Figure 4.10.

Additionally, Table 4.1 contains a summary of the experiment's findings. The average of each metric, such as the reward, success rate, and average Q values, is displayed. Each value in the table represents the average of 20 runs over the 25<sup>th</sup> epoch. The figures that are bolded represent a statistic's best results in that environment.

	Setting	Auro Reach	Fetch Reach-v1	Fetch Push-v1	Fetch Slide-v1	Fetch Pick And Place-v1
Success rate	DDPG+HER	1	1	0.13	0.016	0.08
	A10C10	1	1	0.652	0.023	0.309
	A20C20	1	1	0.647	0.013	0.339
Reward	DDPG+HER	-69.3	-2.5	-89.6	-90.81	-96.9
	A10C10	-63.00	-2.5	-45.00	-90.70	-83.4
	A20C20	-64.60	-2.5	-51.83	-90.8	-81.04
Average Q value	DDPG+HER	-5.8	-0.04	-12.84	-14.16	-12.9
	A10C10	-5.49	-0.02	-5.016	-12.74	-9.19
	A20C20	-5.66	-0.01	-5.19	-13.06	-8.808

Table 4.1: The table displays the success rate, reward, and average Q value for each of the five environments. For the 25<sup>th</sup> epoch, the average of all the values over 20 runs is used.

The next section provides an overall analysis of the AACHER algorithm in comparison to other algorithms.

#### 4.8.4 Analysis

In the previous subsection, we presented several observations as well as a methodology for comparing the effectiveness of the AACHER, and DDPG+HER algorithms. Based on all tests, AACHER outperforms the traditional DDPG+HER. AACHER surpasses other algorithms in terms of success rate, reward, and average Q values.

Each plot in figure 4.8 illustrates how well AACHER performs in the AboReach environment and shows that it outperforms DDPG+HER. Our algorithm makes use of several actor/critic instances for enhanced analysis. All of the plots indicate that the performance of the AACHER instances is superior to that of the conventional DDPG+HER. This is confirmed by the findings that the AACHER algorithm had a greater success rate than DDPG+HER. The scenario for the reward and average Q values is similar. This supports the AACHER technique's stability and effectiveness. It is seen that A10C10 and A20C20 perform reasonably similar to one another in all three matrices. Compared to DDPG+HER, our algorithm shows better performance. This is evident in all of the plots in figure 4.9, which demonstrates how the top-performing instances A10C10 and A20C20 outperform DDPG+HER.

Figure 4.10 depicts the overall performance of our proposed algorithm, AACHER, which, when used in OpenAI's gym environments, outperforms the traditional DDPG+

HER. The performance of A20C20 is observed to be the best in all three metrics in the context of the FetchPick&Place-v1 environment. While A10C10's performance is on par with that of A20C20, the DDPG+HER was unable to perform any better. In all matrices in the FetchPush-v1 environment, A10C10 is observed to perform better than others. A20C20 performs almost as well as A10C10; however, DDPG+HER fell short. There is a noticeable change in the success rate of GA+DDPG+HER at the 25th epoch when FetchPush-v1 is compared to figure 4.10 and figure 3.14. This is because training in AACHER uses 15 cycles per epoch instead of 50 cycles per epoch in GA+DDPG+HER. Additionally, AACHER employs 15 training batches every cycle as opposed to GA+DDPG+HER, which uses 40 batches. The three approaches in FetchReach-v1 have comparable success rates and incentives, but A10C10 and A20C20 have significantly higher average Q values. While all of the approaches perform pretty equally in the FetchSlide-v1 environment, which is considered a challenging task, A10C10 outperforms the others across all matrices.

According to table 4.1, robots in various environments have a success rate of 1. The success rate is still much below the maximum success rate of 1. In FetchPush-v1, FetchSlide-v1 and FetchPick&Place-v1 The robot appears to be still learning these tasks. In FetchPush-v1 and FetchSlide-v1 environments, A10C10 performs best, whereas A20C20 performs best in FetchPick&Place-v1 environment. The increase in the success rate of A10C10 at the 25<sup>th</sup> epoch is over 3.8 times more than that of DDPG+HER in FetchPush-v1 A20C20's success rate improvement

in FetchPick&Place-v1 is roughly 3.3 times greater than DDPG+HER's. The rewards for A10C10 are 50% more than those for DDPG+HER in the FetchPush-v1 environment. The average Q value for FetchReach-v1 for A20C20 is 76% higher than DDPG+HER. This demonstrates that our algorithm, AACHER, outperforms DDPG+HER in terms of efficiency.

## 4.9 Summary

We discussed RL, DDPG, HER, and AACHER in this chapter. This chapter also includes the key contribution, which is discussed in [140]. We proposed the novel algorithm AACHER (Actor-Critic Deep Reinforcement Learning with Hindsight Experience Replay). The development of actors and critics is crucial for DDPG's performance. For DDPG in our algorithm AACHER, we use numerous independent actors and critics, which helps to minimize the impact of a single actor or critic performing poorly. Additionally, we combined HER with our proposed DDPG in AACHER. The AACHER technique uses a more reliable actor-critic learning architecture, which leads to a more stable training environment and higher performance in real-world situations.

We performed various experiments to evaluate our algorithm's effectiveness. We tested AACHER in five simulated environments: AuroReach, FetchReach-v1, FetchPush-v1, FetchSlide-v1 and FetchPick&Place-v1. A10C10 and A20C20 performed the best

out of all the instances created for DDPG. The overall findings demonstrated that AACHER outperforms the conventional DDPG+HER in all of the above-mentioned environments and performs well in all three performance evaluation matrices.

(a) FetchPick&Place environment (e) FetchPush environment (i) FetchReach environment (m) FetchSlide environment

(b) Success rate vs Epochs (f) Success rate vs Epochs (j) Success rate vs Epochs (n) Success rate vs Epochs

(c) Reward vs Epochs (g) Reward vs Epochs (k) Reward vs Epochs (o) Reward vs Epochs

(d) Average Q value vs Epochs (h) Average Q value vs Epochs (l) Average Q value vs Epochs (p) Average Q value vs Epochs

Figure 4.10: The two top-performing trials, A10C10 and A20C20, when applied to the various settings, are plotted against DDPG+HER for comparison. Underneath them are the plots for each environment. Using epochs and an average of 20 runs, plots for success rate, reward, and average Q values are displayed. The shaded region displays the range of values for each plot during the 20 runs.

# Chapter 5

## Conclusion and Future Work

### 5.1 Conclusion

Three novel algorithms, GA-LIMO, GA+DDPG+HER, and AACHER, were presented in this dissertation to address problems in sensor odometry, reinforcement learning, and robotic manipulation.

The preliminary findings of this dissertation describe why a GA might be appropriate for optimization. It demonstrated how a genetic algorithm can optimize the parameters of a reinforcement learning algorithm to provide better results. Additionally, the dissertation's final results emphasize DDPG and describe how using numerous actors and critics can enhance DDPG's performance.

We introduced the GA-LIMO algorithm, which enhances the LIMO's performance using the Genetic Algorithm (GA). The GA-LIMO's results demonstrated that the GA-discovered parameters generate more precise sensor odometry. Furthermore, it was shown how the evolutionary algorithm might be used to adjust sensor odometry algorithm parameters for greater accuracy under various circumstances. It is shown by the efficiency of GA in GA-LIMO, which exemplifies how GA and LIMO collaborate to minimize translation errors across various datasets.

We also provided the GA+DDPG+HER algorithm. To reduce the number of epochs necessary to attain maximum performance, this method combines DDPG+HER with GA. Our findings on the six manipulation tasks demonstrate that the GA is capable of identifying parameter values that result in a quicker learning curve and better (or equal) performance on the tasks we have selected, supporting our hypothesis that GAs are an appropriate fit for such parameter optimization.

In the final part of our dissertation, we proposed a novel algorithm called AACHER that aims to enhance the DDPG algorithm. For DDPG, the algorithm employs several actor/critic networks and then integrates them with HER. According to our findings on the five manipulation tasks, AACHER performs significantly better than conventional DDPG+HER. This supports our hypothesis that using several actors and critics can help DDPG perform better.

## 5.2 Future Work

We demonstrated that heuristic search, as carried out using genetic and other related evolutionary computing techniques, is a workable computational approach for enhancing reinforcement learning and sensor odometry performance. Additionally, adaptive genetic algorithms can be used to run a system with various parameter sets at various times. This might be a sign of online parameter adjustment, which can enhance the performance of any system regardless of the testing environment or domain.

Furthermore, several RL state-action value-related strategies are easy to incorporate with the AACHER methodology as well. It is acknowledged that certain AACHER hyperparameters could be problematic. Therefore, the emphasis can shift in the future to making the loss function's parameters trainable variables. In addition, HER can be looked into to find a better experience replay system.

## 5.3 Publications

1. Sehgal A, La H, Louis S, Nguyen H. Deep reinforcement learning using genetic algorithm for parameter optimization. In 2019 Third IEEE International Conference on Robotic Computing (IRC) 2019 Feb 25-27 (pp. 596-601). IEEE, Naples, Italy.

2. Sehgal A, Singandhupe A, La HM, Tavakkoli A, Louis SJ. Lidar-monocular visual odometry with genetic algorithm for parameter optimization. In International Symposium on Visual Computing 2019 Oct 7-9 (pp. 358-370). Springer, Cham, Lake Tahoe, NV, USA.
3. Sehgal A. Genetic algorithm as function optimizer in reinforcement learning and sensor odometry (Master thesis, University of Nevada, Reno). 2019.
4. Sehgal A, Ward N, La H, Louis S. Automatic Parameter Optimization Using Genetic Algorithm in Deep Reinforcement Learning for Robotic Manipulation Tasks. Submitted to Frontier journal of robotics and AI (Under review). 2022.
5. Sehgal A, Sehgal M, La HM, Bebis G. Deep Learning Hyperparameter Optimization for Breast Mass Detection in Mammograms. In International Symposium on Visual Computing 2022 Oct 3-5. Springer, Cham, San Diego, CA, USA.
6. Sehgal A, Sehgal M, La HM. AACHER: Assorted Actor-Critic Deep Reinforcement Learning with Hindsight Experience Replay. Submitted to International Conference on Autonomous Agents and Multiagent Systems 2023 (Under review).
7. Sehgal A, Ward N, La HM, Papachristos C, Louis S. GA+DDPG+HER: Genetic Algorithm-Based Function Optimizer in Deep Reinforcement Learning for Robotic Manipulation Tasks. In 2022 Sixth IEEE International Conference on Robotic Computing (IRC) 2022 Dec 4. IEEE, Naples, Italy.

8. Sehgal A, Ward N, La HM, Louis S. Deep Reinforcement Learning for Robotic Manipulation Tasks using a Genetic Algorithm-based Function Optimizer. In Encyclopedia with Semantic Computing and Robotic Intelligence. 2022.

## Bibliography

- [1] Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics* 33(5):1255{1262, 2017.
- [2] Johannes Graeter, Tobias Schwarze, and Martin Lauer. Robust scale estimation for monocular visual odometry using structure from motion and vanishing points. In *2015 IEEE Intelligent Vehicles Symposium (IV)* pages 475{480. IEEE, 2015.
- [3] Gabriel Neuzi, Stephan Weiss, Davide Scaramuzza, and Roland Siegwart. Fusion of imu and vision for absolute scale estimation in monocular slam. *Journal of intelligent & robotic systems* 61(1-4):287{299, 2011.
- [4] Johannes Graeter, Alexander Wilczynski, and Martin Lauer. Limo: Lidar-monocular visual odometry. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* pages 7872{7879. IEEE, 2018.

- [5] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2d lidar slam. In 2016 IEEE International Conference on Robotics and Automation (ICRA), pages 1271–1278. IEEE, 2016.
- [6] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.
- [7] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. International conference on machine learning pages 1928–1937. PMLR, 2016.
- [8] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971, 2015.
- [9] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In Advances in Neural Information Processing Systems, pages 5048–5058, 2017.
- [10] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Praveen Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316, 2016.

- [11] Gim Hee Lee, Friedrich Faundorfer, and Marc Pollefeys. Motion estimation for self-driving cars with a generalized camera. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2746–2753, 2013.
- [12] Chen Yan, Wenyuan Xu, and Jianhao Liu. Can you trust autonomous vehicles: Contactless attacks against sensors of self-driving vehicles. DEF CON, 24, 2016.
- [13] Yoseph Bar-Cohen and Cynthia Breazeal. Biologically inspired intelligent robots. In Smart Structures and Materials 2003: Electroactive Polymer Actuators and Devices (EAPAD) volume 5051, pages 14–21. International Society for Optics and Photonics, 2003.
- [14] Richard S Sutton, Andrew G Barto, et al. Introduction to reinforcement learning, volume 135. MIT press, Cambridge, 1998.
- [15] Lawrence Davis. Handbook of genetic algorithms. 1991.
- [16] John H Holland. Genetic algorithms. Scientific American, 267(1):66–73, 1992.
- [17] David E Goldberg and John H Holland. Genetic algorithms and machine learning. Machine learning 3(2):95–99, 1988.
- [18] David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. Foundations of genetic algorithms, volume 1, pages 69–93. Elsevier, Netherlands, 1991.

- [19] Gilbert Syswerda. Uniform crossover in genetic algorithms. *Proceedings of the third international conference on Genetic algorithms* pages 2{9. Morgan Kaufmann Publishers, 1989.
- [20] Adarsh Sehgal, Ashutosh Singandhupe, Hung Manh La, Alireza Tavakkoli, and Sushil J Louis. Lidar-monocular visual odometry with genetic algorithm for parameter optimization. In *International Symposium on Visual Computing* pages 358{370. Springer, 2019.
- [21] Daniel Cremers. Direct methods for 3d reconstruction and visual slam. In *2017 Fifteenth IAPR International Conference on Machine Vision Applications (MVA)* , pages 34{38. IEEE, 2017.
- [22] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. Visual slam algorithms: A survey from 2010 to 2016. *IPSN Transactions on Computer Vision and Applications*, 9(1):16, 2017.
- [23] Ashutosh Singandhupe and Hung La. A review of slam techniques and security in autonomous driving. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 602{607. IEEE, 2019.
- [24] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.
- [25] Richard Szeliski. *Computer vision: algorithms and applications* Springer Science & Business Media, 2010.

- [26] Marc Sons, Henning Lategahn, Christoph G Keller, and Christoph Stiller. Multi trajectory pose adjustment for life-long mapping. In 2015 IEEE Intelligent Vehicles Symposium (IV) pages 901{906. IEEE, 2015.
- [27] Martin Buczko and Volker Willert. Flow-decoupled normalized reprojection error for visual odometry. In 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), pages 1161{1167. IEEE, 2016.
- [28] Igor Cvšić and Ivan Petrović. Stereo odometry based on careful feature selection and tracking. In 2015 European Conference on Mobile Robots (ECMR), pages 1{6. IEEE, 2015.
- [29] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. The International Journal of Robotics Research, 32(11):1231{1237, 2013.
- [30] Ivan Kreso and Sinisa Segvic. Improving the egomotion estimation by correcting the calibration bias. In 10th International Conference on Computer Vision Theory and Applications, 2015.
- [31] Andreas Geiger, Frank Moosmann, Otmar Car, and Bernhard Schuster. Automatic camera and range sensor calibration using a single shot. 2012 IEEE International Conference on Robotics and Automation, pages 3936{3943. IEEE, 2012.

- [32] Johannes Gräter, Tobias Strauss, and Martin Lauer. Photometric laser scanner to camera calibration for low resolution sensors. In 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), pages 1552–1557. IEEE, 2016.
- [33] Yuanhong Xu, Pei Dong, Junyu Dong, and Lin Qi. Combining slam with multi-spectral photometric stereo for real-time dense 3d reconstruction. arXiv preprint arXiv:1807.02294, 2018.
- [34] Ji Zhang and Sanjiv Singh. Visual-lidar odometry and mapping: Low-drift, robust, and fast. In 2015 IEEE International Conference on Robotics and Automation (ICRA), pages 2174–2181. IEEE, 2015.
- [35] Tim Caselitz, Bastian Steder, Michael Ruhnke, and Wolfram Burgard. Monocular camera localization in 3d lidar maps. In 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 1926–1931. IEEE, 2016.
- [36] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems*, volume 2, page 9, 2014.
- [37] Yashar Balazadegan Sarvrood, Siavash Hosseinyalamdary, and Yang Gao. Visual-lidar odometry aided by reduced imu. *ISPRS International Journal of Geo-Information*, 5(1):3, 2016.

- [38] John Henry Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence* MIT press, 1992.
- [39] Melanie Mitchell. *An introduction to genetic algorithms* MIT press, 1998.
- [40] Spencer Gibb, Hung Manh La, and Sushil Louis. A genetic algorithm for convolutional network structure optimization for concrete crack detection. In *2018 IEEE Congress on Evolutionary Computation (CEC)* pages 1–8. IEEE, 2018.
- [41] Alireza Tavakkoli, Amol Ambardekar, Mircea Nicolescu, and Sushil Louis. A genetic approach to training support vector data descriptors for background modeling in video data. In *International Symposium on Visual Computing* pages 318–327. Springer, 2007.
- [42] Deb Kalyanmoy, Amrit Pratap, Sameer Agarwal, and Tamt Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on evolutionary computation*, 6(2):182–197, 2002.
- [43] Pui Wah Poon and Jonathan Neil Carter. Genetic algorithm crossover operators for ordering applications. *Computers & Operations Research* 22(1):135–147, 1995.
- [44] Tom Duckett et al. *A genetic algorithm for simultaneous localization and mapping*. 2003.

- [45] Luis Moreno, Jose M Armingol, Santiago Garrido, Arturo De La Escalera, and Miguel A Salichs. A genetic algorithm for mobile robot localization using ultrasonic sensors. *Journal of Intelligent and Robotic Systems* 34(2):135{154, 2002.
- [46] H. M. La, T. H. Nguyen, C. H. Nguyen, and H. N. Nguyen. Optimal locking control for a mobile sensor network based a moving target tracking. In 2009 IEEE International Conference on Systems, Man and Cybernetics, pages 4801{4806, Oct 2009. doi: 10.1109/ICSMC.2009.5346069.
- [47] Adarsh Sehgal, Hung La, Sushil Louis, and Hai Nguyen. Deep reinforcement learning using genetic algorithm for parameter optimization. In 2019 Third IEEE International Conference on Robotic Computing (IRC) pages 596{601. IEEE, 2019.
- [48] Adarsh Sehgal, Muskan Sehgal, Hung Manh La, and George Bebis. Deep learning hyperparameter optimization for breast mass detection in mammograms. In *International Symposium on Visual Computing* Springer, 2022.
- [49] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.
- [50] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning* 8 (3-4):279{292, 1992.

- [51] Chris Gaskett, David Wettergreen, and Alexander Zelinsky. Q-learning in continuous state and action spaces. In *Australasian Joint Conference on Artificial Intelligence*, pages 417{428. Springer, 1999.
- [52] Kenji Doya. Reinforcement learning in continuous time and space *Neural computation*, 12(1):219{245, 2000.
- [53] Hado Van Hasselt and Marco A Wiering. Reinforcement learning in continuous action spaces. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning* pages 272{279. IEEE, 2007.
- [54] Leemon C Baird. Reinforcement learning in continuous time: Advantage updating. In *Neural Networks, 1994. IEEE World Congress on Computational Intelligence.*, 1994 IEEE International Conference on volume 4, pages 2448{2453. IEEE, 1994.
- [55] Qinglai Wei, Frank L Lewis, Qiuye Sun, Pengfei Yan, and Ruizhuo Song. Discrete-time deterministic q-learning: A novel convergence analysis *IEEE transactions on cybernetics* 47(5):1224{1237, 2017.
- [56] H. M. La, R. Lim, and W. Sheng. Multirobot cooperative learning for predator avoidance. *IEEE Transactions on Control Systems Technology* 23(1):52{63, Jan 2015. ISSN 1063-6536. doi: 10.1109/TCST.2014.2312392.
- [57] Jan Peters, Katharina Mulling, and Yasemin Altun. Relative entropy policy search. In *AAAI* , pages 1607{1612. Atlanta, 2010.

- [58] Mrinal Kalakrishnan, Ludovic Righetti, Peter Pastor, and Stefan Schaal. Learning force control policies for compliant manipulation. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on* pages 4639{4644. IEEE, 2011.
- [59] Nate Kohl and Peter Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on* volume 3, pages 2619{2624. IEEE, 2004.
- [60] Gen Endo, Jun Morimoto, Takamitsu Matsubara, Jun Nakanishi, and Gordon Cheng. Learning cpg-based biped locomotion with a policy gradient method: Application to a humanoid robot. *The International Journal of Robotics Research* 27(2):213{228, 2008.
- [61] Juan Carlos Vargas, Malhar Bhoite, and Amir Barati Farimani. Creativity in robot manipulation with deep reinforcement learning. *arXiv e-prints*, pages arXiv{1910, 2019.
- [62] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research* 32(11): 1238{1274, 2013.
- [63] Alex M Andrew. Reinforcement learning: An introduction by richard s. Sutton and andrew g. Barto, adaptive computation and machine learning series, mit

press (bradford book), cambridge, mass., 1998, xviii+ 322 pp, isbn 0-262-19398-1,(hardback£ 31.95). *Robotica* 17(2):229{235, 1999.

- [64] Remi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems* pages 1054{1062, 2016.
- [65] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. *International Conference on Machine Learning* pages 2829{2838, 2016.
- [66] Marc Peter Deisenroth, Carl Edward Rasmussen, and Dieter Fox. Learning to control a low-cost manipulator using data-efficient reinforcement learning. 2011.
- [67] L. Jin, S. Li, H. M. La, and X. Luo. Manipulability optimization of redundant manipulators using dynamic neural networks *IEEE Transactions on Industrial Electronics*, 64(6):4710{4720, June 2017. ISSN 0278-0046. doi: 10.1109/TIE.2017.2674624.
- [68] Chuan-Kai Lin. H1 reinforcement learning control of robot manipulators using fuzzy wavelet networks. *Fuzzy Sets and Systems* 60(12):1765{1786, 2009.
- [69] Zoran Miljković, Marko Mitić, Mihailo Lazarević, and Bojan Babić. Neural network reinforcement learning for visual control of robot manipulators *Expert Systems with Applications* 40(5):1721{1736, 2013.

- [70] Mihai Duguleana, Florin Grigore Barbuceanu, Ahmed Teirelbar, and Gheorghe Mogan. Obstacle avoidance of redundant manipulators using neural networks based reinforcement learning. *Robotics and Computer-Integrated Manufacturing*, 28(2):132{146, 2012.
- [71] H. Nguyen and H. M. La. Review of deep reinforcement learning for robot manipulation. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 590{595. IEEE, 2019.
- [72] Elliot Chane-Sane, Cordelia Schmid, and Ivan Laptev. Goal-conditioned reinforcement learning with imagined subgoals. In *International Conference on Machine Learning* pages 1430{1440. PMLR, 2021.
- [73] Hai Nguyen, Hung Manh La, and Matthew Deans. Deep learning with experience ranking convolutional neural network for robot manipulator. *arXiv:1809.05819, cs.RQ* 2018.
- [74] Huy X. Pham, Hung M. La, David Feil-Seifer, and Luan V. Nguyen. Autonomous uav navigation using reinforcement learning. *arXiv:1801.05086, cs.RQ*, 2018.
- [75] H. X. Pham, H. M. La, D. Feil-Seifer, and L. Van Nguyen. Reinforcement learning for autonomous uav navigation using function approximation. In *2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)* pages 1{6, Aug 2018. doi: 10.1109/SSRR.2018.8468611.

- [76] H. M. La, R. S. Lim, W. Sheng, and J. Chen. Cooperative tracking and learning in multi-robot systems for predator avoidance. In 2013 IEEE International Conference on Cyber Technology in Automation, Control and Intelligent Systems, pages 337–342, May 2013. doi: 10.1109/CYBER.2013.6705469.
- [77] H. M. La, W. Sheng, and J. Chen. Cooperative and active sensing in mobile sensor networks for scalar field mapping. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 45(1):1–12, Jan 2015. ISSN 2168-2216. doi: 10.1109/TSMC.2014.2318282.
- [78] Huy Xuan Pham, Hung Manh La, David Feil-Seifer, and Aria Neuman. Cooperative and distributed reinforcement learning of drones for field coverage. arXiv:1803.07250, cs.RQ 2018.
- [79] A. D. Dang, H. M. La, and J. Horn. Distributed formation control for autonomous robots following desired shapes in noisy environment. 2016 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI), pages 285–290, Sep. 2016. doi: 10.1109/MFI.2016.7849502.
- [80] Mehdi Rahimi, Spencer Gibb, Yantao Shen, and Hung Manh La. A comparison of various approaches to reinforcement learning algorithms for multi-robot box pushing. In International Conference on Engineering Research and Applications, pages 16–30. Springer, 2018.

- [81] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182{197, 2002.
- [82] Fei Liu and Guangzhou Zeng. Study of genetic algorithm with reinforcement learning to solve the tsp. *Expert Systems with Applications*, 36(3):6995{7001, 2009.
- [83] David E Moriarty, Alan C Schultz, and John J Grefenstette. Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, 11:241{276, 1999.
- [84] Sadayoshi Mikami and Yukinori Kakazu. Genetic reinforcement learning for cooperative traffic signal control. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on* pages 223{228. IEEE, 1994.
- [85] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.
- [86] Ahmed Alagha, Shakti Singh, Rabeb Mizouni, Jamal Bentahar, and Hadi Otrok. Target localization using multi-agent deep reinforcement learning with proximal policy optimization. *Future Generation Computer Systems*, 136:342{357, 2022.

- [87] Ching-An Wu. Investigation of different observation and action spaces for reinforcement learning on reaching tasks. Master's thesis, KTH Royal Institute of Technology, 2019.
- [88] Luckeciano Carvalho Melo and Marcos Ricardo Omena Albuquerque Maximo. Learning humanoid robot running skills through proximal policy optimization. In 2019 Latin american robotics symposium (LARS), 2019 Brazilian symposium on robotics (SBR) and 2019 workshop on robotics in education (WRE), pages 37{42. IEEE, 2019.
- [89] Chloe Ching-Yun Hsu and Celestine Mender-Danner Moritz Hardt. Revisiting design choices in proximal policy optimization. arXiv preprint arXiv:2009.10897, 2020.
- [90] Niels Scholte. Goal, mistake and success learning through resimulation. Master's thesis, 2022.
- [91] Ammar Haydari, Michael Zhang, Chen-Nee Chuah, and Dipak Ghosal. Impact of deep rl-based traffic signal control on air quality. In 2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring), pages 1{6. IEEE, 2021.
- [92] Andreas Geiger, Julius Ziegler, and Christoph Stiller. Stereoscan: Dense 3d reconstruction in real-time. In 2011 IEEE Intelligent Vehicles Symposium (IV), pages 963{968. IEEE, 2011.

- [93] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.
- [94] Philip HS Torr and Andrew Zisserman. Mlesac: A new robust estimator with application to estimating image geometry. *Computer vision and image understanding* 78(1):138–156, 2000.
- [95] Philip HS Torr and Andrew W Fitzgibbon. Invariant fitting of two view geometry. *IEEE transactions on pattern analysis and machine intelligence* 26(5):648–650, 2004.
- [96] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 573–580. IEEE, 2012.
- [97] Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science* 2:331–434, 1990.
- [98] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

- [99] Sander Adam, Lucian Busoniu, and Robert Babuska. Experience replay for real-time reinforcement learning control. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42(2):201{212, 2011.
- [100] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838{855, 1992.
- [101] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540* 2016.
- [102] Sergio Hernandez-Mendez, Carolina Maldonado-Mendez, Antonio Marin-Hernandez, Homero Vladimir Rios-Figueroa, Hector Vazquez-Leal, and Elvia R Palacios-Hernandez. Design and implementation of a robotic arm using ros and moveit! In *2017 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC)*, pages 1{6. IEEE, 2017.
- [103] Glen Field and Yury Stepanenko. Iterative dynamic programming: an approach to minimum energy trajectory planning for robotic manipulators. In *Proceedings of IEEE international conference on robotics and automation*, volume 3, pages 2755{2760. IEEE, 1996.
- [104] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter

- Welinder, et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.
- [105] Adarsh Sehgal. Genetic algorithm as function optimizer in reinforcement learning and sensor odometry. Master's thesis, University of Nevada, Reno, 2019.
- [106] Adarsh Sehgal, Nicholas Ward, Hung Manh La, Christos Papachristos, and Sushil Louis. Ga-drl: Genetic algorithm-based function optimizer in deep reinforcement learning for robotic manipulation tasks. *arXiv preprint arXiv:2203.00141*, 2022.
- [107] Adarsh Sehgal, Nicholas Ward, Hung La, and Sushil Louis. Automatic parameter optimization using genetic algorithm in deep reinforcement learning for robotic manipulation tasks. *arXiv preprint arXiv:2204.03656*, 2022.
- [108] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *International conference on artificial neural networks*, pages 270–279. Springer, 2018.
- [109] Mihalj Bakator and Dragica Radosav. Deep learning and medical diagnosis: A review of literature. *Multimodal Technologies and Interaction*, 2(3):47, 2018.
- [110] Hamed Bozorgi, Xuan Tung Truong, Hung Manh La, and Trung Dung Ngo. 2d laser and 3d camera data integration and filtering for human trajectory tracking. In *2021 IEEE/SICE International Symposium on System Integration (SII)*, pages 634–639. IEEE, 2021.

- [111] Quan Khanh Luu, Hung Manh La, et al. A 3-dimensional printing system using an industrial robotic arm. In *2021 IEEE/SICE International Symposium on System Integration (SII)*, pages 443–448. IEEE, 2021.
- [112] Hoang-Dung Bui, Hai Nguyen, Hung Manh La, and Shuai Li. A deep learning-based autonomous robot manipulator for sorting application. In *2020 Fourth IEEE International Conference on Robotic Computing (IRC)*, pages 298–305. IEEE, 2020.
- [113] Ngo H. M. La V. A. Ho L. Zhou, T-D. Soft robotic hand for sushi grasping and handling. In *15th IEEE/SICE 2023 International Symposium on System Integration (SII)*. IEEE, 2023.
- [114] Praneel Acharya, Kim-Doang Nguyen, Hung M La, Dikai Liu, and I-Ming Chen. Nonprehensile manipulation: a trajectory-planning perspective. *IEEE/ASME Transactions on Mechatronics*, 26(1):527–538, 2020.
- [115] Long Jin, Shuai Li, Hung Manh La, and Xin Luo. Manipulability optimization of redundant manipulators using dynamic neural networks. *IEEE Transactions on Industrial Electronics*, 64(6):4710–4720, 2017.
- [116] Jesse Leaman, Zongming Yang, Yasmine Elglaly, Hung La, and Bing Li. Embodied-ai wheelchair framework with hands-free interface and manipulation.
- [117] Hao Dong, Hao Dong, Zihan Ding, Shanghang Zhang, and Chang. *Deep Reinforcement Learning*. Springer, 2020.

- [118] Andrew Melnik, Luca Lach, Matthias Plappert, Timo Korthals, Robert Haschke, and Helge Ritter. Tactile sensing and deep reinforcement learning for in-hand manipulation tasks. In *IROS Workshop on Autonomous Object Manipulation*, 2019.
- [119] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation. *arXiv preprint arXiv:1610.00633*, 1, 2016.
- [120] Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.
- [121] Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. Reinforcement learning of motor skills in high dimensions: A path integral approach. In *2010 IEEE International Conference on Robotics and Automation*, pages 2397–2403. IEEE, 2010.
- [122] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Ng. An application of reinforcement learning to aerobatic helicopter flight. *Advances in neural information processing systems*, 19, 2006.
- [123] Daniela Rus and Michael T Tolley. Design, fabrication and control of soft robots. *Nature*, 521(7553):467–475, 2015.

- [124] Adarsh Sehgal, Nicholas Ward, Hung La, and Sushil Louis. Automatic parameter optimization using genetic algorithm in deep reinforcement learning for robotic manipulation tasks. *arXiv preprint arXiv:2204.03656*, 2022.
- [125] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.
- [126] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3):293–321, 1992.
- [127] Jiao Wu, Rui Wang, Ruiying Li, Hui Zhang, and Xiaohui Hu. Multi-critic ddpg method and double experience replay. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 165–171. IEEE, 2018.
- [128] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. <https://github.com/openai/baselines>, 2017.
- [129] EN Barron and H Ishii. The bellman equation for minimizing the maximum cost. *Nonlinear Analysis: Theory, Methods & Applications*, 13(9):1067–1090, 1989.
- [130] Ivo Grondman, Lucian Busoniu, Gabriel AD Lopes, and Robert Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307, 2012.

- [131] Justin A Boyan. Technical update: Least-squares temporal difference learning. *Machine learning*, 49(2):233–246, 2002.
- [132] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.
- [133] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.
- [134] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
- [135] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR, 2014.
- [136] Hai Nguyen, Hung Manh La, and Matthew Deans. Hindsight experience replay with experience ranking. In *2019 Joint IEEE 9th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 1–6. IEEE, 2019.

- [137] Yiming Peng, Gang Chen, Mengjie Zhang, and Shaoning Pang. A sandpile model for reliable actor-critic reinforcement learning. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 4014–4021. IEEE, 2017.
- [138] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.
- [139] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [140] Adarsh Sehgal, Muskan Sehgal, and Hung Manh La. Aacher: Assorted actor-critic deep reinforcement learning with hindsight experience replay. *arXiv preprint arXiv:2210.12892*, 2022.