University of Nevada, Reno

Genetic Algorithm as Function Optimizer in Reinforcement Learning and Sensor Odometry.

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering

by

Adarsh Sehgal

Dr. Hung M. La - Thesis Advisor
May 2019

THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

ADARSH SEHGAL

Entitled

Genetic Algorithm As Function Optimizer In Reinforcement Learning And Sensor Odometry

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

MASTER OF SCIENCE

Hung M. La, Ph.D., Advisor

Sushil Louis, Ph.D., Committee Member

Wanliang Shan, Ph.D., Graduate School Representative

David W. Zeh, Ph.D., Dean, Graduate School

May, 2019

Abstract

Reinforcement learning (RL) enables agents to make a decision based on a reward function. However, in the process of learning, the choice of values for learning algorithm parameters can significantly impact the overall learning process. In this thesis, we use a genetic algorithm (GA) to find the values of parameters used in the Deep Deterministic Policy Gradient (DDPG) combined with Hindsight Experience Replay (HER) algorithm, to help speed up the learning agent. We used this method on fetch-reach, slide, push, pick and place, and door opening in robotic manipulation tasks. Our experimental evaluation shows that our method leads to significantly better performance, faster than the original algorithm. This thesis also deals with Lidar-Monocular Visual Odometry (LIMO), an odometry estimation algorithm, which combines camera and LIght Detection And Ranging sensor (LIDAR) for visual localization by tracking camera features as well as features from LIDAR measurements, and it estimates the motion of sensors using Bundle Adjustment based on robust key frames. For rejecting outliers, LIMO uses semantic labelling and weights of vegetation landmarks. A drawback of LIMO as well as many other odometry estimation algorithms is that they have many parameters that need to be manually adjusted according to dynamic changes in the environment in order to decrease translational errors. In this thesis, we also present and argue the use of Genetic Algorithms to optimize parameters with reference to LIMO and to maximize LIMO's localization and motion estimation performance. We evaluate our approach on the well known KITTI odometry dataset and show that the genetic algorithm helps LIMO to significantly reduce translation error in different datasets.

Dedication

Dedicated to Pushpa Sehgal

Acknowledgments

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.

I also thank my committee members, Dr. Sushil Louis, and Dr. Wanliang Shan, for taking time to review this thesis and providing guidance on my research.

Ultimately, I thank my family, especially my parents for being pillars of financial and emotional support throughout my time in this school.

This material is based upon work supported by the National Aeronautics and Space Administration (NASA) Grant No. NNX15AI02H issued through the NVSGC-RI program under sub-award No. 19-21, and the RID program under sub-award No. 19-29, and the NVSGC-CD program under sub-award No. 18-54. This work is also partially supported by the Office of Naval Research under Grant N00014-17-1-2558.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Accuracy and efficiency play an influential part in developing as well as existing technologies. One of the many areas of robotics where efficiency matters, deals with intelligent robots [6]. Robots have long been able to function teleoperated. The world is already moving towards adding artificial intelligence (AI) to robots. Many technologies help robots make decisions. The measure of how well the robot learns is a matter of efficiency. Let's say a robot takes a few months just to learn how to open a door, which is definitely not efficient because of the amount of time this learning process would take. Hence, there's a need for efficiency in self-learning robots. The robots should be able to learn faster, which in turn save resources and time. We specifically deal with Reinforcement Learning (RL) [7] in this thesis to increase the efficiency of learning robots (agents).

Turning to the subject of accuracy, in the case of self-driving cars [8], it matters because there are human lives involved. Tesla's Model X, while running on adaptive cruise control, slammed into a barrier in California because of some flaw in Tesla's semi-autonomous driving system. High accuracy GPS is unaffordable and problematic in places with no GPS signal, so it is important for the self-driving car to use other sensors and still be able to track its exact location on the road. Different kinds of sensors have been used to estimate the motion and odometry of the self-driving cars. While some researches have used generalized cameras [9], others have shown how different sensors are used [10]. Furthermore, it has been tested if these kinds of cars can be trusted, because there are many possible kinds of attacks that can reduce the reliability of such sensors. Recent work on sensor odometry has shown that LIDAR in combination with a monocular camera, can be used for odometry estimation. One such algorithm is Lidar-Monocular Visual Odometry (LIMO) [4]. This algorithm has error in relation to the ground truth, hence, the algorithm has a notable scope of improvement.

Hence, in order to increase the efficiency and accuracy of above-said systems, optimization algorithms such as Genetic Algorithms (GA) can assume a greater role.

1.2 Background on Genetic Algorithm (GA)

Genetic Algorithms (GAs) [11–13] were designed to search poorly-understood spaces [14], where exhaustive search may not be feasible, and where other search approaches perform poorly. When used as function optimizers, GAs try to maximize a fitness tied to the optimization objective. Evolutionary computing algorithms in general and GAs specifically have had much empirical success on a variety of difficult design and optimization problems. They start with a randomly initialized population of candidate solutions typically encoded in a string (chromosome). A selection operator focuses search on promising areas of the search space while crossover and mutation operators generate new candidate solutions.

We used ranking selection [15] to select parents for crossover and mutation. Rank selection probabilistically selects higher ranked (higher fitness) individuals. Unlike fitness proportional selection, ranking selection pays attention to the existence of a fitness difference rather than to the magnitude of fitness difference. Children are generated using uniform crossover [16], which are then mutated using flip mutation [13]. Chromosomes are binary encoded with concatenated parameters.

1.3 Background on Deep Reinforcement Learning (DRL)

Q-learning [17] methods have been applied to a variety of tasks by autonomous robots [18], and much research has been done in this field starting many years ago [17], with some work specific to continuous action spaces [19–22] and others to discrete action spaces [23]. Reinforcement Learning (RL) [7] has been applied to locomotion [24] [25] and also to manipulation [26, 27].

A lot of work specific to robotic manipulators also exists [28, 29]. Some of this work used fuzzy wavelet networks [30], while others used neural networks to accomplish their tasks [31] [32]. Off-policy algorithms [33] such as the Deep Deterministic Policy Gradient algorithm (DDPG) [34] and Normalized Advantage Function algorithm (NAF) [35] are helpful for real robot systems. A complete review of recent deep reinforcement learning (DRL) methods for robot manipulation is given in [36]. We are specifically using DDPG combined with Hindsight Experience Replay (HER) [37] for our experiments. Recent work on using experience ranking to improve the learning speed of DDPG + HER was reported in [38].

RL has been widely used in training/teaching both a single robot [39, 40] and a multi-robot system [41–45]. Previous work has also been done on both model-based and model-free learning algorithms. Applying model-based learning algorithms to real world scenarios relies significantly on a model-based teacher to train deep network policies.

Similarly, there is also much work in GA's [11] [46] and the GA operators of crossover and mutation [47], applied to a variety of problems. GA has been specifically applied to variety of RL problems [47–50].

1.4 Background on Lidar-Monocular Visual Odometry (LIMO)

Motion estimation has long been a popular subject of research in which many techniques have been developed over the years [51]. Varied types of work have been done related to Visual Simultaneous Localization and Mapping (VSLAM), also referred to as Visual Odometry [52], which simultaneously estimates the motion of the camera and the 3D structure of the observed environment. A recent review of SLAM techniques for autonomous car driving can be found in [53]. Bundle Adjustment is the most widely used method for VSLAM. Bundle Adjustment is a procedure that minimizes the re-projection error between the observed point (landmarks in reference to LIMO) and the predicted points. Recent developments make use of offline VSLAM for mapping and localization [54–56].

Figure 1.1 illustrates the structure of the VSLAM pipeline [4]. Algorithms such as Robust Outlier Criterion for Camera-based odometry (ROCC) [57] and Stereo visual Odometry based on Feature selection and Tracking (SOFT) [58] rely on pre-processing and feature extraction, which is in contrast to most of the methods that obtain scale information from a camera placed at a different viewpoint [1, 56, 59, 60]. SOFT and

ROCC extract robust and precise features and select them using special techniques, and have managed to attain high performance on the KITTI Benchmark [61], even without Bundle Adjustment.

The major disadvantage of a stereo camera is it's reliance on extrinsic camera calibration. It was later observed that performance can be enhanced by learning a compensation for the calibration bias through a deformation field [62]. LIght Detection And Ranging sensor LIDAR-camera calibration is also an expanding topic of research [63, 64]. Previous work has been done with VSLAM and LIDAR [65–68]. LIMO [4], uses the feature tracking capability of the camera and combines it with depth measurements from a LIDAR sensor but suffers from translation and rotation errors. Later on, we describe our approach for increasing LIMO's robustness to translation errors.

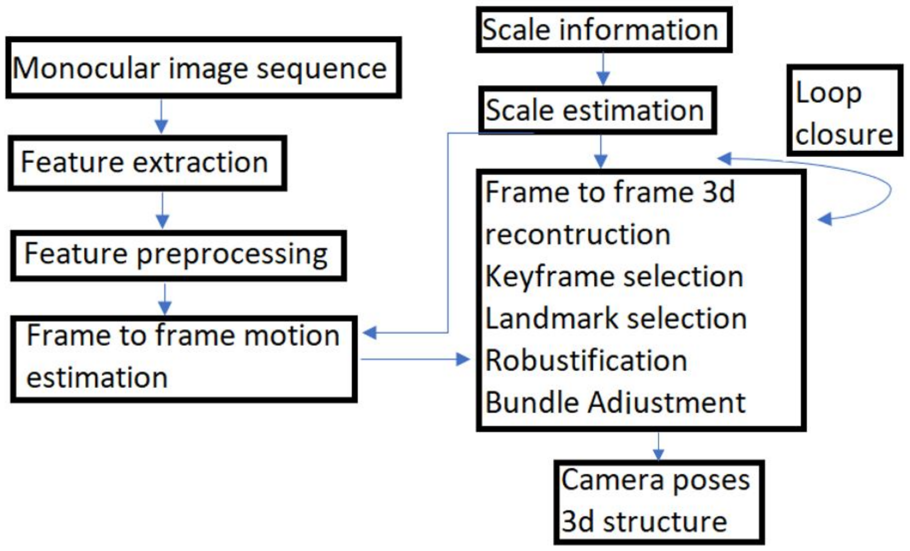


Figure 1.1: VSLAM pipeline. The input is a temporal sequence of images, and the system outputs a sparse reconstruction of the observed environment and the camera poses [1–4]. In this work, LIMO does not perform loop closure [5].

LIMO takes advantage of depth information from LIDAR, which is used for feature detection in the image. Outliers are rejected if they do not meet the local plane assumptions, and points on the ground plane are treated for robustness. As illustrated in figure 1.1, in the VSLAM pipeline, depth information is fused with monocular feature detection techniques. Another approach is taken for prior estimation, landmark selection and key frame selection to fulfill real time constraints. Unlike the approach in [66], LIMO does not use any LIDAR-SLAM algorithms such as Iterative Closest Point (ICP). The major drawback of LIMO is that it has many parameters, which needs to be manually tuned. LIMO suffers from translation and rotation errors even more than existing algorithms such as Lidar Odometry and Mapping (LOAM) [68] and Vision-Lidar Odometry and Mapping (V-LOAM) [69]. Typically, researchers tune parameters (in LIMO as well) in order to minimize these errors but there always exists the possibility of finding better parameter sets that may be optimized for specific camera and LIDAR hardware or for specific scenarios. Hence, there is a need to use optimization algorithms to increase LIMO's performance. In this thesis, we propose using a genetic algorithm (GA) to efficiently search the space of possible LIMO parameter values to find precise parameters that maximizes performance. Our experiments with this new GA-LIMO algorithm show that GA-LIMO performs statistically significantly better than stock LIMO.

Much empirical evidence shows that evolutionary computing techniques such as Genetic Algorithms (GAs) work well as function optimizers in poorly-understood, non-linear, discontinuous spaces [13, 70–73]. GAs [11, 74] and the GA operators of crossover and mutation [47] have been tested on numerous problems. Closer to our research, GAs have been applied to early SLAM optimization problems [75], mobile localization using ultrasonic sensors [76] [77], and in deep reinforcement learning [78]. This provides good evidence for GA efficacy on localization problems, and our main contribution in this thesis is a demonstration of significantly smaller translation error when using a GA to tune LIMO parameter values compared to the stock LIMO algorithm [4]. Our experiments show that translation error is non-linearly related to LIMO parameters, that is, translation error can vary non-linearly based on the values of the LIMO's parameters. The following sections describe the LIMO, the GA and GA-LIMO algorithms. We then show results from running LIMO with GA tuned parameters on the KITTI odometry data sequences [79].

1.5 GA on DRL and LIMO

GA as a function optimizer can be used with various optimization problems. This thesis focuses on the DRL and LIMO, background on which was presented earlier in this chapter. GAs can be used to optimize the parameters used in the system based on their fitness values. GA tries to maximize the fitness function. An objective function can be converted to a fitness function using various mathematical formulations.

Existing DRL algorithms use a fixed set of parameters. GA when applied to DRL, finds better set of parameters, which helps the learning agent to learn faster. The inverse of the number of epochs serves as the fitness value to this problem. GA offers a promising way to increase the efficiency of the system.

On the other hand, LIMO also uses a constant value of parameters for estimating the sensor odometry. Our experiments show that there is a scope of increasing the accuracy of the system in this scenario. GA in combination with LIMO finds an better set of parameters, which increases the accuracy of odometry estimation. The inverse of translation error is considered as the fitness value in this GA implementation. The GA-LIMO system works with higher accuracy.

1.6 Content

The following chapters of this thesis are as follows: Chapter 2 introduces DRL algorithms, discusses the open problem, proposes an algorithm to solve the problem and the experimental results. In Chapter 3, LIMO is discussed in detail along with the GA-LIMO algorithm, which helps to reduce the translation error in sensor tracking compared to ground truth. Corresponding experimental results follow in this chapter. Lastly, the conclusion and future work are provided in the last chapter of this thesis.

Chapter 2

Genetic Algorithm optimization for Deep Reinforcement Learning

2.1 Reinforcement Learning

Consider a standard RL setup consisting of a learning agent, which interacts with an environment. An environment can be described by a set of variables where S is the set of states, A is the set of actions, $p(s_0)$ is a distribution of initial states, $r : S \times A \to R$ is a reward function, $p(s_{t+1}|s_t, a_t)$ are transition probabilities and $\gamma \in [0, 1]$ is a discount factor.

A deterministic policy maps from states to actions: $\pi : S \to A$. The beginning of every episode is marked by sampling an initial state s_0. For each timestep t, the agent performs an action a_t based on the current state s_t: $a_t = \pi(s_t)$. The performed action gets a reward $r_t = r(s_t, a_t)$, and the distribution $p(.|s_t, a_t)$ helps to sample the environment's new state. The discounted sum of future rewards is: $R_t = \sum_{i=T}^{\infty} \gamma^{i-t} r_i$. The agent's goal is to try to maximize its expected return $E[R_t|s_t, a_t]$ and an optimal policy denoted by π^* can be defined as any policy π^*, such that $Q^{\pi^*}(s,a) \geq Q^{\pi}(s,a)$ for every $s \in S, a \in A$ and any policy π. The optimal policy, which has the same Q-function, is called an optimal Q-function, Q^*, which satisfies the *Bellman* equation:

$$Q^*(s,a) = E_{s' \ p(.|s,a))}[r(s,a) + \gamma \underset{a' \in A}{max} Q^*(s',a'))]. \tag{2.1}$$

2.2 Deep Q-Networks (DQN)

A *Deep Q-Network (DQN)* [80] is defined as a model free reinforcement learner [81], designed for discrete action spaces. In a DQN, a neural network Q is maintained, which approximates Q^*. $\pi_Q(s) = argmax_{a \in A} Q(s,a)$ denotes a greedy policy w.r.t. Q. A - greedy policy takes a random action with probability ϵ and action $\pi_Q(s)$ with probability $1 - \epsilon$.

Episodes are generated during training using a ϵ-greedy policy. A *Replay buffer* stores transition tuples (s_t, a_t, r_t, s_{t+1}) experienced during training. The neural network training is interlaced by a generation of new episodes. A Loss $\mathcal{L}$ defined by $\mathcal{L} = E(Q(s_t, a_t) - y_t)^2$ where $y_t = r_t + \gamma max_{a' \in A} Q(s_{t+1}, a')$ and tuples (s_t, a_t, r_t, s_{t+1}) are being sampled from the replay buffer.

The *target network* changes at a slower pace than the main network, which is used to measure targets y_t. The weights of the target networks can be set to the current weights of the main network [80]. Polyak-averaged parameters [82] can also be used.

2.3 Deep Deterministic Policy Gradients (DDPG)

In *Deep Deterministic Policy Gradients (DDPG)*, there are two neural networks: an Actor and a Critic. The actor neural network is a target policy $\pi : S \rightarrow A$, and critic neural network is an action-value function approximator $Q : S \times A \to R$. The critic network $Q(s, a|\theta^Q)$ and actor network $\mu(s|\theta^\mu)$ are randomly initialized with weights θ^Q and θ^μ.

A behavioral policy is used to generate episodes, which is a noisy variant of the target policy, $\pi_b(s) = \pi(s) + \mathcal{N}(0, 1)$. The training of a critic neural network is done like the Q-function in a DQN but where the target y_t is computed as $y_t = r_t + \gamma Q(s_{t+1}, \pi(s_{t+1}))$, where γ is the discounting factor. The loss $\mathcal{L}_a = -E_a Q(s, \pi(s))$ is used to train the actor network.

2.4 Hindsight Experience Replay (HER)

Hindsight Experience Reply (HER) tries to mimic human behavior to learn from failures. The agent learns from all episodes, even when it does not reach the original goal. Whatever state the agent reaches, HER considers that as the modified goal. Standard experience replay only stores the transition $(s_t||g, a_t, r_t, s_{t+1}||g)$ with original goal g. HER tends to store the transition $(s_t||g', a_t, r'_t, s_{t+1}||g')$ to modified goal g' as well. HER does great with extremely sparse rewards and is also significantly better for sparse rewards than shaped ones.

2.5 Open Problem Discussion

DDPG + HER suffer from an efficiency problem. The performance of most of the robotic tasks can be improved by using a better set of parameters used in the algorithm. The performance can be measured based on the number of epochs it takes for the learning agent to learn a given robotic task. Further sections of this chapter show how the change in values of various parameters significantly impacts the learning rate of the agent. The solution to this problem is also presented later in this chapter and the supporting experimental results showing that the proposed solution outperforms the existing technique for reinforcement learning.

2.6 DDPG + HER and GA

In this section, we present the primary contribution of our thesis: the genetic algorithm searches through the space of parameter values used in DDPG + HER for values that maximize task performance and minimize the number of training epochs. We target the following parameters: discounting factor γ; polyak-averaging coefficient τ [82]; learning rate for critic network α_{critic}; learning rate for actor network α_{actor}; percent of times a random action is taken ϵ; and standard deviation of Gaussian noise added to not completely random actions as a percentage of maximum absolute value of actions on different coordinates η. The range of all the parameters is 0-1, which can be justified using the equations following in this section.

Our experiments show that adjusting the values of parameters did not increase or decrease the agent's learning in a linear or easily discernible pattern. So, a simple hill climber will probably not do well in finding optimized parameters. Since GAs were designed for such poorly understood problems, we use our GA to optimize these parameter values.

Specifically, we use τ, the polyak-averaging coefficient to show the performance non-linearity for values of τ . τ is used in the algorithm as show in Equation (2.2):

$$\begin{aligned} \theta^{Q'} &\leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}, \\ \theta^{\mu'} &\leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}. \end{aligned} \tag{2.2}$$

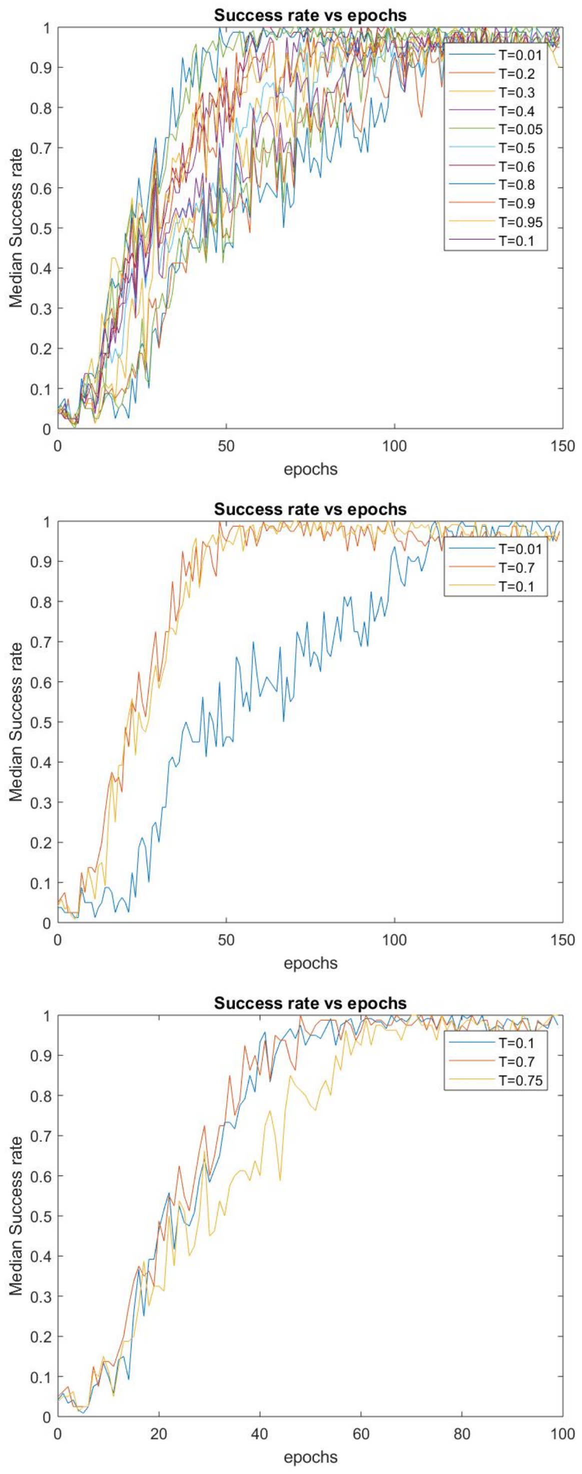
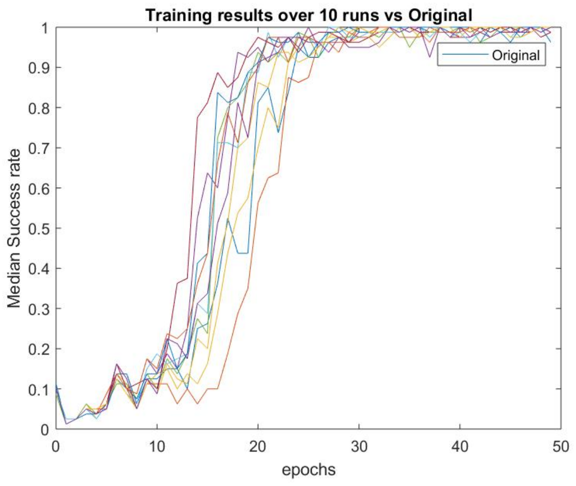
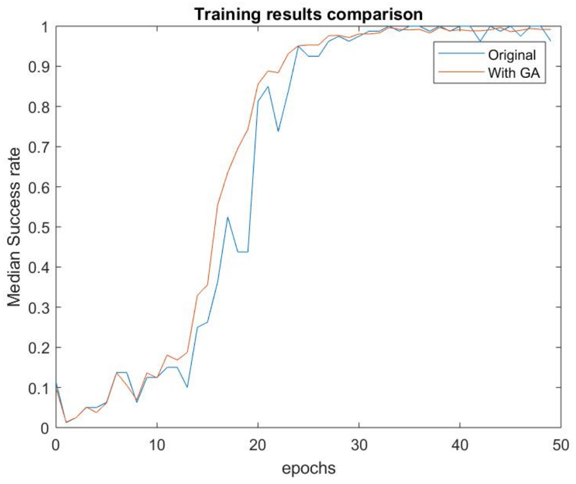


Figure 2.1: Success rate vs. epochs for various τ for *FetchPick&Place-v1* task.



(a) GA-DRL over 10 runs, vs. Original



(b) GA-DRL averaged over 10 runs, vs. Original

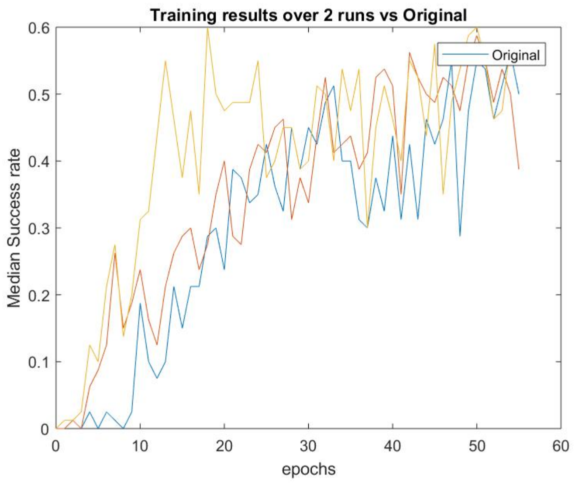
Figure 2.2: Success rate vs. epochs for *FetchPush-v1* task when τ and γ are found using the GA.

Equation (2.3) shows how γ is used in the DDPG + HER algorithm, while Equation (2.4) describes the Q-Learning update. α denotes the learning rate. Networks are trained based on this update equation.

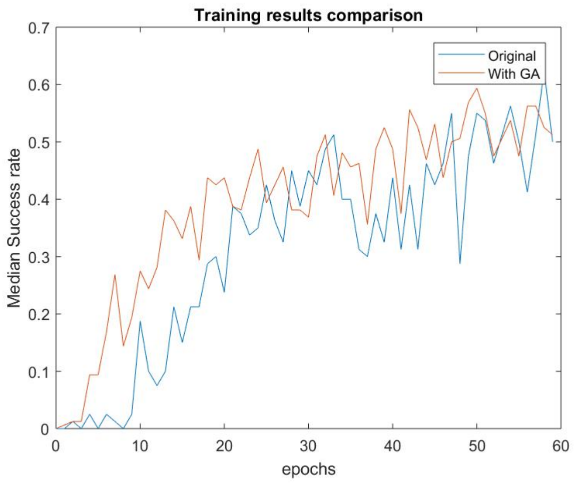
$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{t+1}|\theta^{\mu'})|\theta^{Q'}), \tag{2.3}$$

Algorithm 1: DDPG + HER and GA

```
1  Choose population of $n$ chromosomes
2  Set the values of parameters into the chromosome
3  Run the DDPG + HER to get number of epochs for which the algorithm first
     reaches success rate $\geq 0.85$
4  for all chromosome values do
5      Initialize DDPG
6      Initialize replay buffer $R \leftarrow \phi$
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
13         for t=0, T-1 do
14             $r_t := r(s_t, a_t, g)$
15             Store the transition $(s_t||g, a_t, r_t, s_{t+1}||g)$ in $R$
16             Sample a set of additional goals for replay $G := S($current
                 episode$)$
17             for $g' \in G$ do
18                 $r' := r(s_t, a_t, g')$
19                 Store the transition $(s_t||g', a_t, r', s_{t+1}||g')$ in $R$
20             end
21         end
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
26     end
27     return $1/epochs$
28 end
29 Perform Uniform Crossover
30 Perform Flip Mutation at rate 0.1
31 Repeat for required number of generations to find optimal solution
```



(a) GA-DRL over 2 runs, vs. Original



(b) GA-DRL averaged over 2 runs, vs. Original

Figure 2.3: Success rate vs. epochs for *FetchSlide-v1* task when τ and γ are found using the GA.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) \\ -Q(s_t, a_t)]. \tag{2.4}$$

Since we have two kinds of networks, we will need two learning rates, one for the actor network (α_{actor}), another for the critic network (α_{critic}). Equation (2.5) explains the use of percent of times that a random action is taken, ϵ.

$$a_t = \begin{cases} a_t^* & \textit{with probability } 1-\epsilon, \\ \textit{random action} & \textit{with probability } \epsilon. \end{cases} \tag{2.5}$$

Figure 2.1 shows that when the value of τ is modified, there is a change in the agent's learning, further emphasizing the need to use a GA. The original (untuned) value of τ in DDPG was set to 0.95, and we are using 4 CPUs. All the values of τ are considered up to two decimal places, in order to see the change in success rate with change in value of the parameter. From the plots, we can clearly tell that there is a great scope of improvement from the original success rate.

Algorithm 1 explains the integration of DDPG + HER with a GA, which uses a population size of 30 over 30 generations. We are using *ranking selection* [15] to select parents. The parents are probabilistically based on rank, which is in turn decided based on the relative fitness (performance). Children are then generated using *uniform crossover* [16]. We are also using *flip mutation* [13] with probability of mutation to be 0.1. We use a binary chromosome to encode each parameter and concatenate the bits to form a chromosome for the GA. The six parameters are arranged in the 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 completely random actions as a percentage of maximum absolute value of actions on different coordinates. Since each parameter requires 11 bits to be represented to three decimal places, we need 66 bits for 6 parameters. These string chromosomes then enable domain independent crossover and mutation string operators to generate new parameter values. We consider parameter values up to three decimal places, because small changes in values of parameters causes considerable change in success rate. For example, a step size of

0.001 is considered as the best fit for our problem.

The fitness for each chromosome (set of parameter values) is defined by the inverse of the number of epochs it takes for the learning agent to reach close to maximum success rate (≥ 0.85) for the very first time. Fitness is inverse of the number of epochs because GA always maximizes the objective function and this converts our minimization of number of epochs to a maximization problem. Since each fitness evaluation takes significant time an exhaustive search of the 2^{66} size search space is not possible and thus we use a GA search.

2.7 Experimental Results

Figure 2.4, shows the environments used to test robot learning on five different tasks: *FetchPick&Place-v1*, *FetchPush-v1*, *FetchReach-v1*, *FetchSlide-v1*, and *DoorOpening*. We ran the GA separately on these environments to check the effectiveness of our algorithm and compared performance with the original values of the parameters. Figure 2.2 (a) shows the result of our experiment with *FetchPush-v1*, while Figure 2.3 (a) shows the results with *FetchSlide-v1*. We let the system run with the GA to find best values of parameters τ and γ. Since the GA is probabilistic, we show results from 10 runs of the GA and the results show that the optimized parameters found by the GA can lead to better performance. The learning agent can run faster, and can reach the maximum success rate, faster. In Figure 2.2 (b), we show one learning run for the original parameter set and the average learning over these 10 different runs of the GA.

Figure 2.3 (b) compares one run for original with averaged 2 runs for optimizing parameters τ and γ. For this task, we have run it for only 2 runs because these tasks can take a few hours for one run. The results shown in Figures 2.2 and 2.3 show

Parameters	DRL	GA-DRL
γ	0.98	0.88
τ	0.95	0.184
α_{actor}	0.001	0.001
α_{critic}	0.001	0.001
ϵ	0.3	0.055
η	0.2	0.774

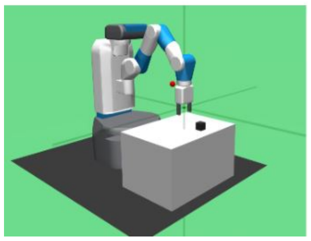
Table 2.1: DRL vs GA-DRL values of parameters

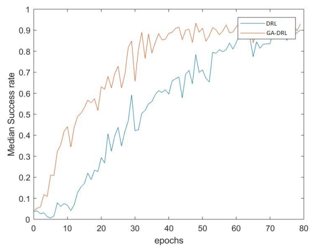
changes when only two parameters are being optimized as we tested and debugged the genetic algorithm, we can see the possibility for performance improvement. Our results from optimizing all five parameters justify this optimism and are described next.

The GA was then run to optimize all parameters and these results were plotted in Figure 2.4 for all the tasks. Table 2.1 compares the GA found parameters with the original parameters used in the RL algorithm. Though the learning rates α_{actor} and α_{critic} are same as their original values, the other four parameters have different values than original. The plots in the figure 2.4 shows that the GA found parameters outperformed the original parameters, indicating that the learning agent was able to learn faster. All the plots in the above mentioned figure are averaged over 10 runs.

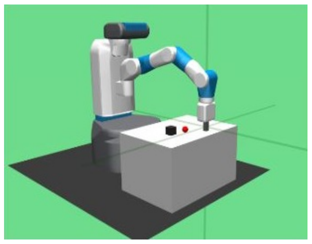
2.8 Summary

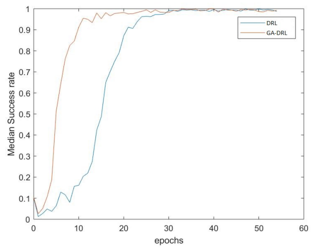
This chapter discusses about RL, DQN, DDPG and HER. The main contribution, as described in [78], is also presented in this chapter. Experimental results were compared between DRL [83] and GA-DRL [78] scenarios. It was found that GA found parameters outperformed the original parameters.


(a) FetchPick&Place environment

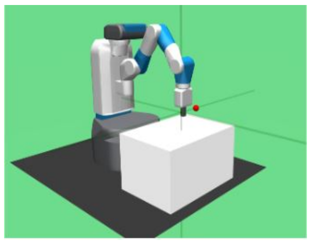


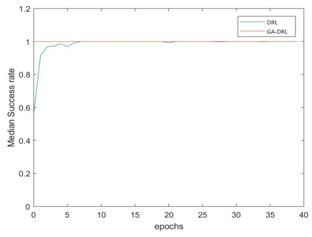
(f) FetchPick&Place plot


(b) FetchPush environment

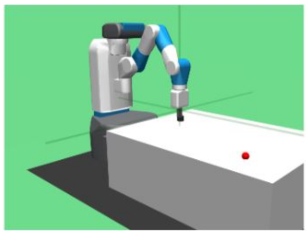


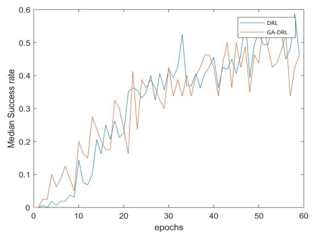
(g) FetchPush plot


(c) FetchReach environment

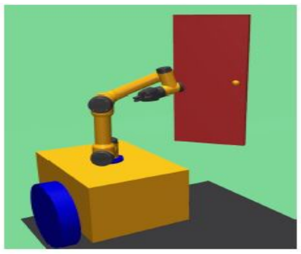


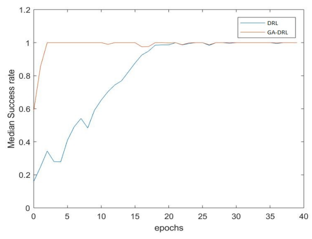
(h) FetchReach plot


(d) FetchSlide environment



(i) FetchSlide plot


(e) Door Opening environment



(j) DoorOpening plot

Figure 2.4: Environments and the corresponding DRL vs GA-DRL plots, when all the 6 parameters are found by GA. All plots are averaged over 10 runs.

Chapter 3

Genetic Algorithm optimization for Lidar-Monocular Visual Odometry

3.1 LIMO

In this section, we present prior work related to our GA-LIMO algorithm. We first describe the VSLAM pipeline and then the LIMO algorithm.

3.1.1 Feature extraction and pre-processing

Figure 1.1 shows feature extraction's procedure in the pipeline. Feature extraction consists of tracking the features and associating the features using the Viso2 library [59]. It is further used to implement feature tracking, which comprises non-maximum suppression, sub-pixel refinement and outlier rejection by flow. Deep learning is used to reject landmarks that are moving objects. The neighborhood of the feature point in a semantic image [84] is scanned, and if the majority of neighboring pixels categorize to a dynamic class, like vehicle or pedestrian, the landmark is excluded.

3.1.2 Scale Estimation

For scale estimation, the detected feature points from the camera are mapped to the depth extracted from LIDAR. LIMO uses a one shot depth estimation approach. Initially LIDAR point cloud is transformed into the camera frame and then it is projected onto the image plane. In detail, the following steps are executed for every feature point f:

1. A region of interest is selected around f, which is a set F consisting of projected LIDAR points.
2. A new set called foreground set F_{seg} is created by segmenting the elements of F.
3. The elements of F_{seg} are fitted with a plane p. A special fitting algorithm is used in case f belongs to the ground plane.
4. To estimate the depth, p is intersected with the line of sight corresponding to f .
5. For the previous estimated depth a test is performed. Depth estimates that are more than 30m are rejected since they can be uncertain. In addition, the angle between the line of sight of the feature point and the normal of the plane must be smaller than a threshold.

From the point clouds, neighborhoods for ordered point clouds can be selected directly. However, projections of the LIDAR points on the image are used, and the points within a rectangle in the image plane around f are selected in case the point clouds are unordered. Before the plane estimation is performed, the foreground F_{seg} is segmented. In the next step, a histogram of depth having a fixed bin width of $h = 0.3$m is created and interpolated with elements in F. LIDAR points of the nearest bin is used to perform segmentation using all detected feature points. For estimating the local surface around f precisely, fitting the plane to F_{seg} can help. Three points are chosen from the points in F_{seg}, which traverse the triangle F_Δ with maximum area. Depth estimation is avoided if the area of F_Δ is too small, to evade incorrectly estimated depth.

However, the above technique cannot be used to estimate the depth of points on the ground plane because LIDAR has a lower resolution in a perpendicular direction than in a level direction. A different approach is followed to enable depth estimation for a relevant ground plane. For solving this, RANSAC with refinement is used on the LIDAR point cloud to extract the ground plane [55]. In order to estimate feature points on the road, points that correspond to the ground plane are segmented. Outliers are extracted by allowing only local planes that lie close to the ground plane.

3.1.3 Frame to Frame Odometry

Perspective-n-Point-Problem [55] serves as the starting point of the frame to frame motion estimation.

$$\underset{x,y,z,\alpha,\beta,\gamma}{argmin} \sum_i ||\varphi_{i,3d\to 2d}||_2^2 \tag{3.1}$$

$$\varphi_{3d\to 2d} = \bar{p}_i - \pi(p_i, P(x, y, z, \alpha, \beta, \gamma)), \tag{3.2}$$

where $\bar{p_i}$ is the observed feature point in the current frame, p_i is the 3D-point corresponding to $\bar{p_i}$, the transform from the previous to the current frame is denoted by freedom $P(x, y, z, \alpha, \beta, \gamma)$, which has three translation and three rotation degrees of freedom. $\pi(...)$ is the projection function from the 3D to 2D domain. The extracted features with valid estimated depth may be very small in the environments that has low structure and large optical flow. LIMO introduces epipolar error as $\varphi_{2d \to 2d}$ [54].

$$\varphi_{2d \to 2d} = \bar{p}_i F(\frac{x}{z}, \frac{y}{z}, \alpha, \beta, \gamma)\bar{p}_i, \tag{3.3}$$

where fundamental matrix F can be calculated from the intrinsic calibration of the camera and from the frame to frame motion of the camera. LIMO suggests the loss function to be Cauchy function [54]: $\rho_s(x) = a(s)^2.log(1 + \frac{x}{a(s)^2})$, where $a(s)$ is the fix outlier threshold. For frame to frame motion estimation, the optimization problem can be denoted as:

$$\underset{x,y,z,\alpha,\beta,\gamma}{argmin} \sum_i \rho_{3d \to 2d}(\|\varphi_{i,3d \to 2d}\|_2^2) + \rho_{2d \to 2d}(\|\varphi_{i,2d \to 2d}\|_2^2). \tag{3.4}$$

3.1.4 Backend

LIMO proposes a Bundle Adjustment framework based on keyframes , with key components as selection of keyframes, landmark selection, cost functions and robustification measures. The advantage with this approach is that it retains the set that carries information, which is required for accurate pose estimation as well as excludes the unnecessary measurements. Keyframes are classified as required, rejected and sparsified keyframes. Required frames are crucial measurements. Frame rejection is done when the vehicle does not move. The remaining frames are collected, and the technique selects frames every 0.3s. Finally in keyframe selection, length of optimization window is chosen.

An optimal set of landmarks should be well observable, small, free of outliers and evenly distributed. Landmark selection divides all landmarks into three bins, near, middle and far, each of which has fixed number of landmarks selected for the Bundle Adjustment. Weights of the landmarks are then determined based on the semantic information. The estimated landmark depth is taken into consideration by an additional cost function,

$$\xi_{i,j}(i_i, P_j) = \begin{cases} 0, & if\ l_i\ has\ no\ depth\ estimate \\ \hat{d}_{i,j} - \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \tau(l_i, P_j), & else, \end{cases} \tag{3.5}$$

where l_i denotes the landmark, τ mapping from world frame to camera frame and $\hat{d}$ denotes the depth estimate. The combination of indices i, j denote the landmark-pose. A cost function ν punishes deviations from the length of translation vector,

$$\nu(P_1, P_0) = \hat{s}(P_1, P_0) - s, \tag{3.6}$$

where P_0, P_1 are the last two poses in the optimization window and $\hat{s}(P_0, P_1) = \|translation(P^{-1}P_1)\|_2^2$, where s is a constant with value of $\hat{s}(P_1, P_0)$ before optimization.

While outliers need to be removed because they do not let Least-Square methods to converge [85, 86], semantics and cheirality only does preliminary outlier rejection. The LIMO optimization problem can now be formulated as:

$$\underset{P_j \in P, l_i \in L, d_i \in D}{argmin} \quad w_0 \|\nu(P_1, P_0\|_2^2) +$$

$$\sum_i \sum_j w_1 \rho_\phi (\|\phi_{i,j}(l_i, P_i)\|_2^2) + w_2 \rho_\xi (\|\xi_{i,j}(l_i, P_j)\|_2^2), \tag{3.7}$$

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

3.2 Open Problem Discussion

LIMO suffers from accuracy problem. LIMO when running with various environments experiences an error with respect to the ground truth. In order to address this problem, this thesis proposes an algorithm called GA-LIMO, which aims to reduce the translation error in sensor odometry estimation. The proposed algorithm and the experimental results are shown further in this chapter.

3.3 GA-LIMO

In this section, one of the main contributions of this thesis is presented. The detailed description is also conferred in [87]. The specific GA searches through the space of parameter values used in LIMO for the values that maximizes the performance and minimizes the translation error as a result of pose estimation. We are targeting the following parameters: outlier rejection quantile δ; maximum number of landmarks for near bin ϵ_{near}; maximum number of landmarks for middle bin ϵ_{middle}; maximum number of landmarks for far bin ϵ_{far} and weight for the vegetation landmarks μ. As described in the background section, rejecting outliers, δ, plays an important role in converging to minimum, the weight of outlier rejection thus has notable impact on the translation error. The landmarks are categorized into three bins, which also have great significance in translation error calculation. Trees that have a rich structure result in feature points that are good to track, but they can move. So, finding an optimal weight for vegetation can significantly reduce translation error. δ and μ range from 0 to 1, while ϵ_{near}, ϵ_{middle} and ϵ_{far} range from 0 to 999. We have set these ranges based on early experimental results.

Algorithm 2: GA-LIMO

```
1  Choose population of $n$ chromosomes
2  Set the values of parameters into the chromosome
3  Run LIMO with the GA selected parameter values
4  for all chromosome values do
5      Run LIMO on KITTI odometry data set sequence 01
6      Compare LIMO estimated poses with ground truth
7      Translation error $\sigma_1$ is found
8      Run LIMO on KITTI odometry data set sequence 04
9      Compare LIMO estimated poses with ground truth
10     Translation error $\sigma_4$ is found
11     Average error $\sigma_{avg} = \frac{\sigma_1+\sigma_4}{2}$
12     return $1/\sigma_{avg}$
13 end
14 Perform Uniform Crossover
15 Perform Flip Mutation at rate 0.1
16 Repeat for required number of generations for optimal solution
```

Our experiments show that adjusting the values of parameters did not decrease or increase the translation error in a linear or easily appreciable pattern. So, a simple hill climber will probably not do well in finding optimized parameters. We thus use a GA to optimize these parameters.

Algorithm 2 explains the combination of LIMO with the GA, which uses a population size of 50 runs for 50 generations. We used ranking selection [15] to select the parents for crossover and mutation. Rank selection probabilistically selects higher ranked (higher fitness) individuals. Unlike fitness proportional selection, ranking selection pays attention to the existence of a fitness difference rather than to the magnitude of fitness difference. Children are generated using uniform crossover [16], which are then mutated using flip mutation [13]. Chromosomes are binary encoded with concatenated parameters. δ and μ are considered up to three decimal places, which means a step size of 0.001, because changes in values of parameters cause considerable change in translation error. All the parameters require 11 bits to represent their range of values, so we have a chromosome length of 55 bits, with parameters arranged in the order: δ, ϵ_{near}, ϵ_{middle}, ϵ_{far}, μ.

The algorithm starts with randomly generating a population of n individuals. Each chromosome is sent to LIMO to evaluate. LIMO evaluates the parameter set represented by each individual by using those parameters to run on the KITTI dataset [61]. The KITTI benchmarks are well known and provide the most popular benchmark for Visual Odometry and VSLAM. This dataset has rural, urban scenes along with highway sequences and provides gray scale images, color images, LIDAR point clouds and their calibration. Most LIMO configurations are as in [4]. In our work, we focus on two sequences in particular: sequence 01 and 04. Sequence 01 is a highway scenario, which is challenging because only a road can be used for depth estimation. Sequence 04 is an urban scenario, which has a large number of landmarks for depth estimation. We consider both sequences for each GA evaluation because we want a common set of parameters that will work well with multiple scenes.

The fitness of each chromosome is defined as the inverse of translation error. This translates the minimization of translation error into a maximization of fitness as required for GA optimization. Since each fitness evaluation takes significant amount of time, an exhaustive search of the 2^{55} size search space is not possible, hence we are using the GA. During a fitness evaluation, the GA first runs the LIMO with sequence 01. It then compares the LIMO estimated poses with ground truth (also found in [61]) and finds the translation error using the official KITTI metric [61]. The same steps are followed for sequence 04. The fitness value of each chromosome is the average of

Figure 3.1: Camera data while GA-LIMO is in action.

the inverse translation errors from the two sequences.

$$\sigma_{avg} = \frac{\sigma_1 + \sigma_4}{2}. \tag{3.8}$$

Selected chromosomes (ranked selection) are then crossed over and mutated to create new chromosomes to form the next population. This starts the next GA iteration of evaluation, selection, crossover, and mutation. The whole system takes significant amount of time since we are running $50 * 50 = 2500$ LIMO evaluations to determine the best parameters. The next section shows our experiments with individual and combined sequences, with and without the GA. Our results show that the GA-LIMO performs better than the results of LIMO [4].

3.4 Experimental Results

In this section we show our experiments with individual KITTI sequences, a combination of sequences, and overall results. First, we run the GA-LIMO with sequences 01 and 04 separately. We show the translation error and the error mapped onto trajectory, compared to the ground truth (reference) [79]. We then show our results when GA-LIMO runs with evaluations on both sequences 01 and 04. Finally, we compare the values of parameters found by GA-LIMO versus LIMO.

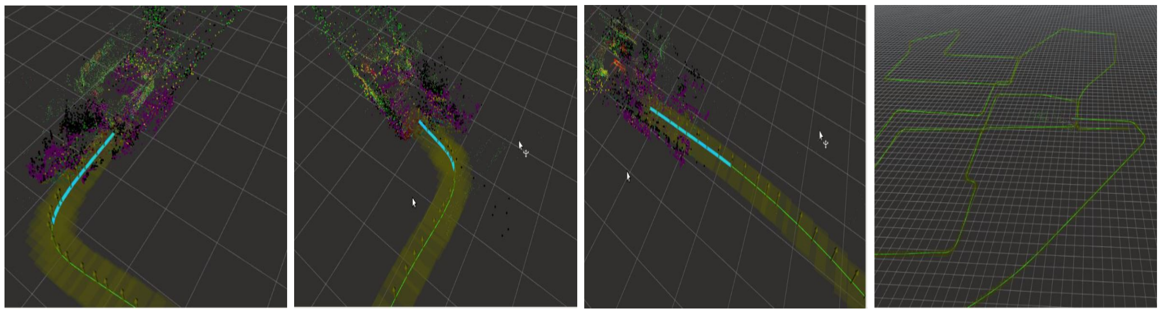
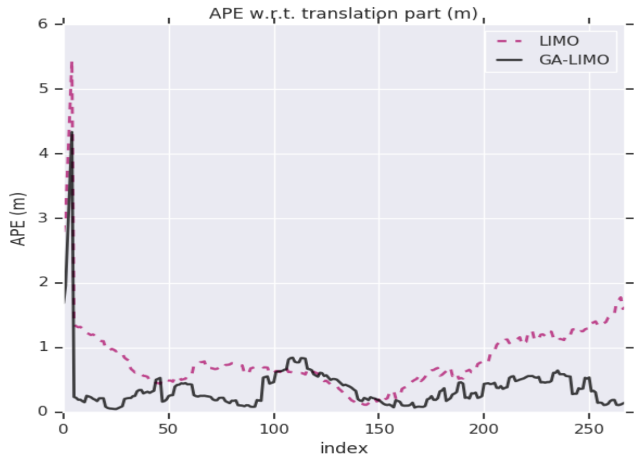


Figure 3.2: GA-LIMO estimating the pose. The video for this visualization can be found on: $https://ara.cse.unr.edu/?page\_id = 11$ and $https://www.youtube.com/watch?v = \_4peUcYy6 - g$

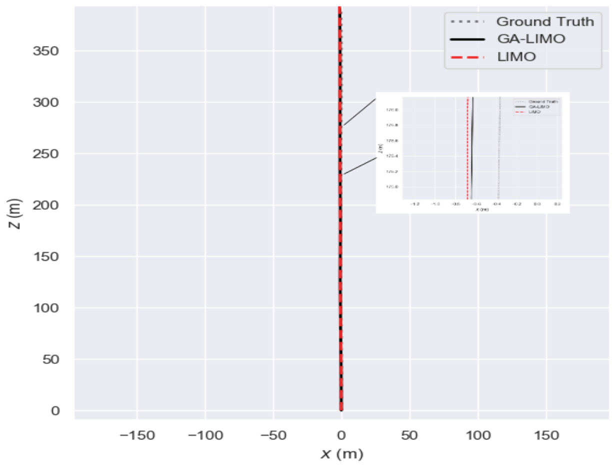
Algorithm 3: GA-LIMO individual

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

Figure 3.1 shows camera data while GA-LIMO is estimating the pose from that data in figure 3.2. Figure 3.3 compares LIMO performance with GA-LIMO on sequence 04. Here the GA was run on this sequence individually to find best values of parameters, as in algorithm 3. Absolute Pose Error (APE) and Root Mean Squared Error (RMSE) are one of the important measures [88]. The translation error for each sequence is the RMSE calculated with respect to ground truth. Figure 3.3a compares the translation error over the poses, while figure 3.3b compares the error mapped onto the trajectory with the zoomed in trajectory. Table 3.1 compares the values of parameters for LIMO and GA-LIMO. Our results show that the GA-LIMO trajectory is closer to ground truth compared to LIMO. We found that the translation error was



(a) Translation error comparison over the poses.

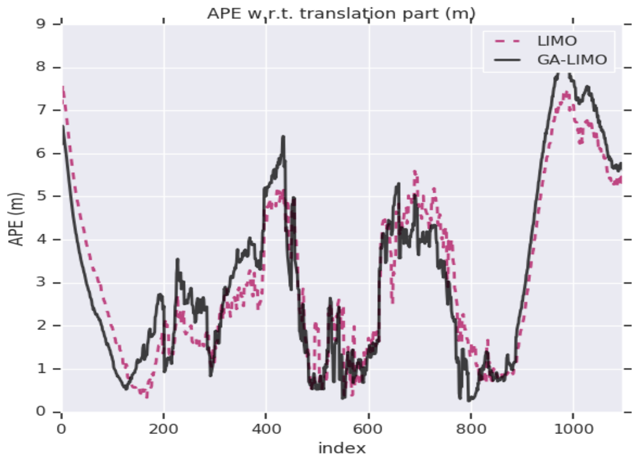


(b) Trajectory comparison for sequence 04, when GA-LIMO was run on this sequence individually (algorithm 3).

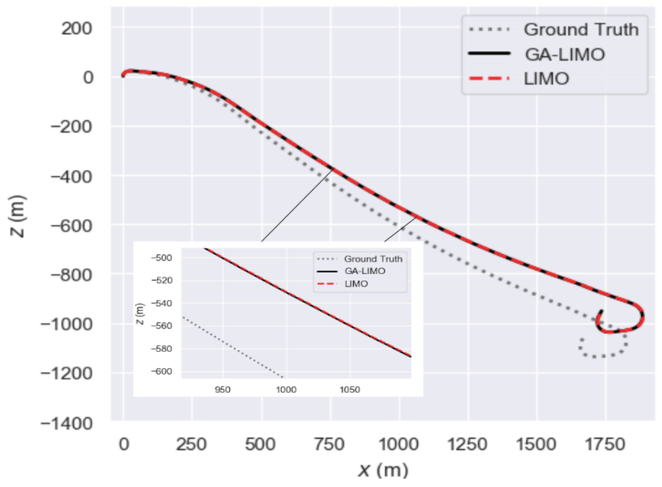
Figure 3.3: Results comparison for sequence 04 (algorithm 3). LIMO has 1.01% translation error, while GA-LIMO has about half this error with 0.56%.

0.56% with GA-LIMO, in contrast to 1.01% with LIMO.

Figure 3.4 compares the performance of LIMO with GA-LIMO when the system is run on just sequence 01. In this case, first, the GA was run on sequence 01 (Algorithm 3) and then the GA-LIMO parameters were used to test the same sequence. Table 3.2 compares the original and GA found parameter values. Figure 3.4a compares translation error, while figure 3.4b shows the error mapped onto the trajectory for



(a) Translation error comparison over the poses.



(b) Trajectory comparison.

Figure 3.4: Results comparison for sequence 01 (algorithm 3). LIMO has 3.71% translation error, while GA-LIMO has 3.8%.

Parameters	LIMO	GA-LIMO
δ	0.95	0.986
ϵ_{near}	400	999
ϵ_{middle}	400	960
ϵ_{far}	400	859
μ	0.9	0.128

Table 3.1: LIMO vs GA-LIMO values of parameters when GA was run on LIMO with sequence 04 individually.

LIMO and GA-LIMO. As shown in the zoomed in figure 3.4b, GA-LIMO is closer to the ground truth. The translation error for LIMO is found to be around 3.71% and 3.8% in case of GA-LIMO, with sequence 01. GA found parameters that did not outperform the original parameters, when GA-LIMO was run on just sequence 01.

Parameters	LIMO	GA-LIMO
δ	0.95	0.958
ϵ_{near}	400	999
ϵ_{middle}	400	593
ϵ_{far}	400	877
μ	0.9	0.813

Table 3.2: LIMO vs GA-LIMO values of parameters when GA was run on LIMO with sequence 01 individually.

Parameters	LIMO	GA-LIMO
δ	0.95	0.963
ϵ_{near}	400	999
ϵ_{middle}	400	554
ϵ_{far}	400	992
μ	0.9	0.971

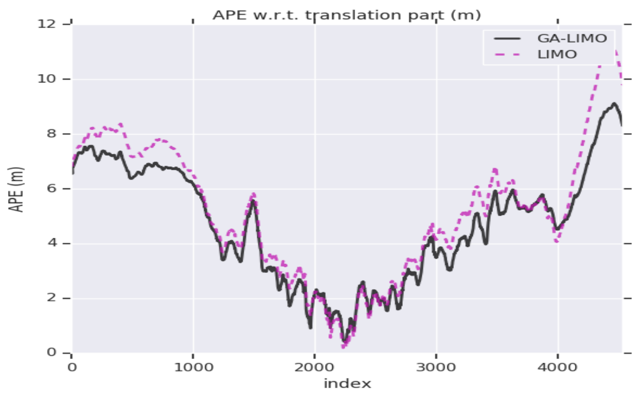
Table 3.3: LIMO vs GA-LIMO values of parameters when GA is run on LIMO with combined sequence 01 and 04.

We finally ran the system with both sequences 01 and 04 as described in Algorithm 2. The fitness of each evaluation is the average of translation errors of the sequences when run using the input parameters. The parameters found in GA-LIMO as shown in table 3.3, were then tested on sequences sequences 00, 01 and 04, as shown in figure 3.5 and 3.6. It is evident that GA-LIMO performed better than LIMO in all three sequences. The zoomed in figures show a closer view on one part of the trajectories. GA-LIMO trajectories are closer to the ground truth and have lesser translation errors. GA-LIMO has a translation error of 5.13% with sequence 00, 3.59% with sequence 01 and 0.65% with sequence 04, in contrast with 5.77% with sequence 00, 3.71% with sequence 01 and 1.01% with sequence 04 using original parameters.

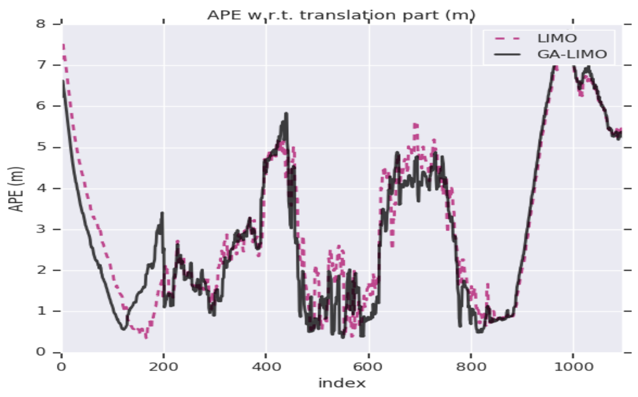
Our method helped to find a common set of GA-LIMO parameters, which works better and hence lead to better performance in different kinds of environments.

3.5 Summary

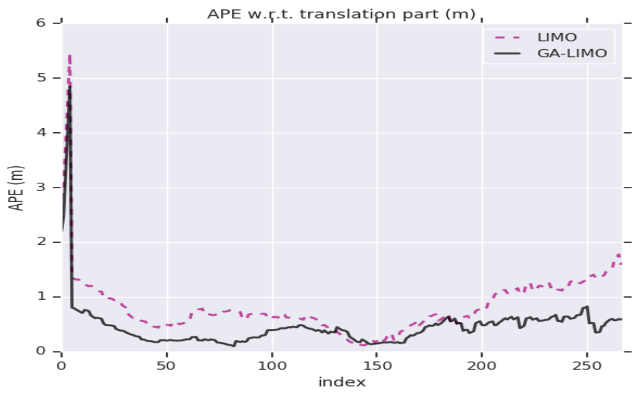
In this chapter, the LIMO algorithm is described. The chapter also proposes the GA-LIMO algorithm, compares the performance of LIMO [4] with that of GA-LIMO [87], and shows that the GA-LIMO performs better than the LIMO.



(a) Translation error comparison over the poses for sequence 00. LIMO has 5.77% translation error while GA-LIMO has 5.13%.

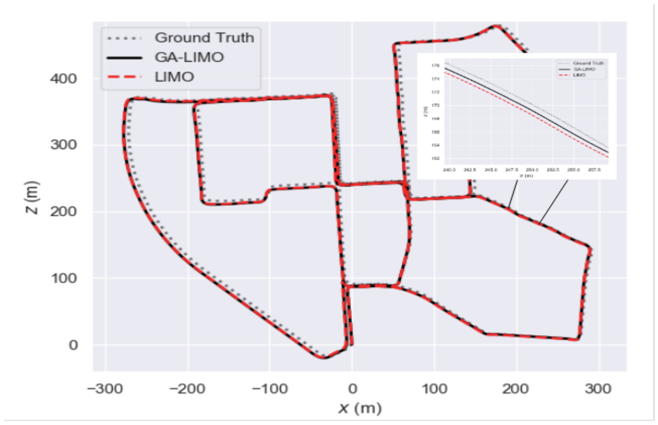


(b) Translation error comparison over the poses for sequence 01. LIMO has 3.71% translation error while GA-LIMO has 3.59%.

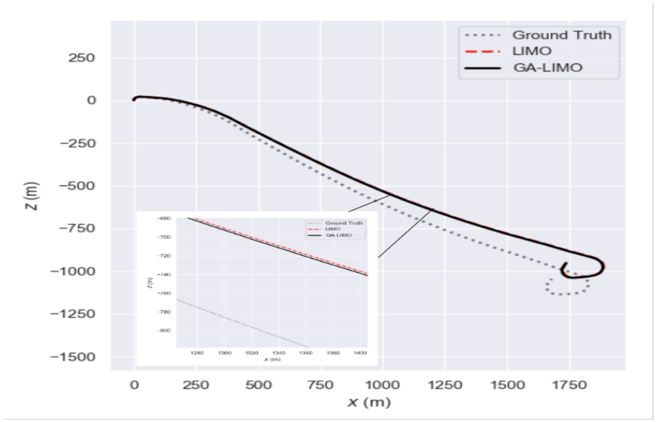


(c) Translation error comparison over the poses for sequence 04. LIMO has 1.01% translation error while GA-LIMO has 0.65%.

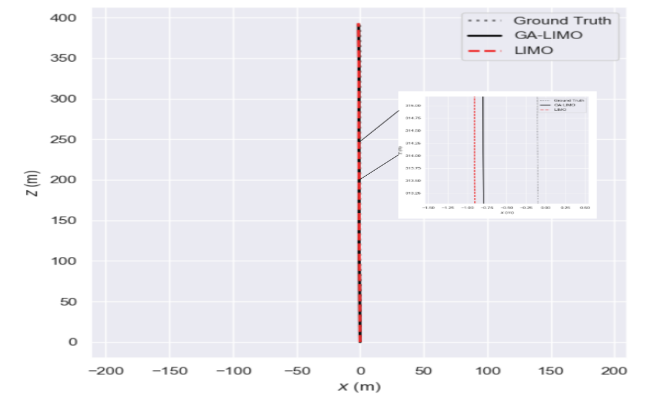
Figure 3.5: The parameters are found using GA-LIMO, using a combination of sequences 01 and 04 (Algorithm 2). These parameters are then tested on three sequences. In all three sequences, GA-LIMO performs better than LIMO.



(a) Sequence 00 trajectories showing GA-LIMO closer to ground truth.



(b) Sequence 01 trajectories showing GA-LIMO closer to ground truth.



(c) Sequence 04 trajectories showing GA-LIMO closer to ground truth.

Figure 3.6: Trajectory comparison when GA-LIMO was run in Algorithm 2. In all three sequences, GA-LIMO performs better than LIMO.

Chapter 4

Conclusion and Future Work

4.1 Conclusion

This thesis showed initial results that demonstrated that a genetic algorithm can tune reinforcement learning algorithm parameters to achieve better performance, illustrated by faster learning rates at five manipulation tasks. It was further shown that the genetic algorithm can also be used to tune sensor odometry algorithm parameters to achieve more accuracy in various environments. We discussed existing work in reinforcement learning in robotics and sensor odometry, presented the GA-LIMO algorithm along with an algorithm that integrates DDPG + HER with GA to optimize the number of epochs required to achieve maximal performance, and explained why a GA might be suitable for such optimization. Initial results bore out the assumption that GAs are a good fit for such parameter optimization and our results on the five manipulation tasks show that the GA can find parameter values that lead to faster learning and better (or equal) performance at our chosen tasks. Furthermore, the results for GA-LIMO illustrated that GA found parameters produce more accurate sensor odometry.

4.2 Future Work

We provided further evidence that heuristic search as performed by genetic and other similar evolutionary computing algorithms are a viable computational tool for optimizing reinforcement learning and sensor odometry performance. Adaptive Genetic Algorithms can also be deployed to have different sets of parameters during the process of running the system. This may point towards online parameter tuning, which will help any system have better performance, irrespective of the domain or type of testing environment.

Bibliography

[1] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[2] J. Gräter, T. Schwarze, and M. Lauer, "Robust scale estimation for monocular visual odometry using structure from motion and vanishing points," in *2015 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2015, pp. 475–480.

[3] G. Nützi, S. Weiss, D. Scaramuzza, and R. Siegwart, "Fusion of imu and vision for absolute scale estimation in monocular slam," *Journal of intelligent & robotic systems*, vol. 61, no. 1-4, pp. 287–299, 2011.

[4] J. Graeter, A. Wilczynski, and M. Lauer, "Limo: Lidar-monocular visual odometry," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 7872–7879.

[5] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1271–1278.

[6] Y. Bar-Cohen and C. Breazeal, "Biologically inspired intelligent robots," in *Smart Structures and Materials 2003: Electroactive Polymer Actuators and Devices*

(EAPAD), vol. 5051. International Society for Optics and Photonics, 2003, pp. 14–21.

[7] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning.* MIT press Cambridge, 1998, vol. 135.

[8] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[9] G. Hee Lee, F. Faundorfer, and M. Pollefeys, "Motion estimation for self-driving cars with a generalized camera," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 2746–2753.

[10] C. Yan, W. Xu, and J. Liu, "Can you trust autonomous vehicles: Contactless attacks against sensors of self-driving vehicle," *DEF CON*, vol. 24, 2016.

[11] L. Davis, "Handbook of genetic algorithms," 1991.

[12] J. H. Holland, "Genetic algorithms," *Scientific american*, vol. 267, no. 1, pp. 66–73, 1992.

[13] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine learning*, vol. 3, no. 2, pp. 95–99, 1988.

[14] K. De Jong, "Learning with genetic algorithms: An overview," *Machine learning*, vol. 3, no. 2-3, pp. 121–138, 1988.

[15] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," in *Foundations of genetic algorithms.* Elsevier, 1991, vol. 1, pp. 69–93.

[16] G. Syswerda, “Uniform crossover in genetic algorithms,” in *Proceedings of the third international conference on Genetic algorithms.* Morgan Kaufmann Publishers, 1989, pp. 2–9.

[17] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[18] H. M. La, R. Lim, and W. Sheng, “Multirobot cooperative learning for predator avoidance,” *IEEE Transactions on Control Systems Technology*, vol. 23, no. 1, pp. 52–63, Jan 2015.

[19] C. Gaskett, D. Wettergreen, and A. Zelinsky, “Q-learning in continuous state and action spaces,” in *Australasian Joint Conference on Artificial Intelligence.* Springer, 1999, pp. 417–428.

[20] K. Doya, “Reinforcement learning in continuous time and space,” *Neural computation*, vol. 12, no. 1, pp. 219–245, 2000.

[21] H. V. Hasselt and M. A. Wiering, “Reinforcement learning in continuous action spaces,” 2007.

[22] L. C. Baird, “Reinforcement learning in continuous time: Advantage updating,” in *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on*, vol. 4. IEEE, 1994, pp. 2448–2453.

[23] Q. Wei, F. L. Lewis, Q. Sun, P. Yan, and R. Song, “Discrete-time deterministic q-learning: A novel convergence analysis,” *IEEE transactions on cybernetics*, vol. 47, no. 5, pp. 1224–1237, 2017.

[24] N. Kohl and P. Stone, “Policy gradient reinforcement learning for fast quadrupedal locomotion,” in *Robotics and Automation, 2004. Proceedings.*

ICRA'04. 2004 IEEE International Conference on, vol. 3. IEEE, 2004, pp. 2619–2624.

[25] G. Endo, J. Morimoto, T. Matsubara, J. Nakanishi, and G. Cheng, "Learning cpg-based biped locomotion with a policy gradient method: Application to a humanoid robot," *The International Journal of Robotics Research*, vol. 27, no. 2, pp. 213–228, 2008.

[26] J. Peters, K. Mülling, and Y. Altun, "Relative entropy policy search." in *AAAI*. Atlanta, 2010, pp. 1607–1612.

[27] M. Kalakrishnan, L. Righetti, P. Pastor, and S. Schaal, "Learning force control policies for compliant manipulation," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 4639–4644.

[28] M. P. Deisenroth, C. E. Rasmussen, and D. Fox, "Learning to control a low-cost manipulator using data-efficient reinforcement learning," 2011.

[29] L. Jin, S. Li, H. M. La, and X. Luo, "Manipulability optimization of redundant manipulators using dynamic neural networks," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 6, pp. 4710–4720, June 2017.

[30] C.-K. Lin, "HâĹđ reinforcement learning control of robot manipulators using fuzzy wavelet networks," *Fuzzy Sets and Systems*, vol. 160, no. 12, pp. 1765–1786, 2009.

[31] Z. Miljković, M. Mitić, M. Lazarević, and B. Babić, "Neural network reinforcement learning for visual control of robot manipulators," *Expert Systems with Applications*, vol. 40, no. 5, pp. 1721–1736, 2013.

[32] M. Duguleana, F. G. Barbuceanu, A. Teirelbar, and G. Mogan, "Obstacle avoidance of redundant manipulators using neural networks based reinforcement learn-

ing," *Robotics and Computer-Integrated Manufacturing*, vol. 28, no. 2, pp. 132–146, 2012.

[33] R. Munos, T. Stepleton, A. Harutyunyan, and M. Bellemare, "Safe and efficient off-policy reinforcement learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 1054–1062.

[34] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[35] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," in *International Conference on Machine Learning*, 2016, pp. 2829–2838.

[36] H. Nguyen and H. M. La, "Review of deep reinforcement learning for robot manipulation," in *2019 Third IEEE International Conference on Robotic Computing (IRC)*. IEEE, 2019, pp. 590–595.

[37] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, 2017, pp. 5048–5058.

[38] H. Nguyen, H. M. La, and M. Deans, "Deep learning with experience ranking convolutional neural network for robot manipulator," *arXiv:1809.05819, cs.RO*, 2018.

[39] H. X. Pham, H. M. La, D. Feil-Seifer, and L. V. Nguyen, "Autonomous uav navigation using reinforcement learning," *arXiv:1801.05086, cs.RO*, 2018.

[40] ——, "Reinforcement learning for autonomous uav navigation using function approximation," in *2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Aug 2018, pp. 1–6.

[41] H. M. La, R. S. Lim, W. Sheng, and J. Chen, "Cooperative flocking and learning in multi-robot systems for predator avoidance," in *2013 IEEE International Conference on Cyber Technology in Automation, Control and Intelligent Systems*, May 2013, pp. 337–342.

[42] 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*, vol. 45, no. 1, pp. 1–12, Jan 2015.

[43] H. X. Pham, H. M. La, D. Feil-Seifer, and A. Nefian, "Cooperative and distributed reinforcement learning of drones for field coverage," *arXiv:1803.07250, cs.RO*, 2018.

[44] A. D. Dang, H. M. La, and J. Horn, "Distributed formation control for autonomous robots following desired shapes in noisy environment," in *2016 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, Sep. 2016, pp. 285–290.

[45] M. Rahimi, S. Gibb, Y. Shen, and H. M. La, "A comparison of various approaches to reinforcement learning algorithms for multi-robot box pushing," in *International Conference on Engineering Research and Applications.* Springer, 2018, pp. 16–30.

[46] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[47] P. W. Poon and J. N. Carter, "Genetic algorithm crossover operators for ordering applications," *Computers & Operations Research*, vol. 22, no. 1, pp. 135–147, 1995.

[48] F. Liu and G. Zeng, "Study of genetic algorithm with reinforcement learning to solve the tsp," *Expert Systems with Applications*, vol. 36, no. 3, pp. 6995–7001, 2009.

[49] D. E. Moriarty, A. C. Schultz, and J. J. Grefenstette, "Evolutionary algorithms for reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 11, pp. 241–276, 1999.

[50] S. Mikami and Y. 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.* IEEE, 1994, pp. 223–228.

[51] D. Cremers, "Direct methods for 3d reconstruction and visual slam," in *2017 Fifteenth IAPR International Conference on Machine Vision Applications (MVA).* IEEE, 2017, pp. 34–38.

[52] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual slam algorithms: A survey from 2010 to 2016," *IPSJ Transactions on Computer Vision and Applications*, vol. 9, no. 1, p. 16, 2017.

[53] A. Singandhupe and H. La, "A review of slam techniques and security in autonomous driving," in *2019 Third IEEE International Conference on Robotic Computing (IRC).* IEEE, 2019, pp. 602–607.

[54] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision.* Cambridge university press, 2003.

[55] R. Szeliski, *Computer vision: algorithms and applications.* Springer Science & Business Media, 2010.

[56] M. Sons, H. Lategahn, C. G. Keller, and C. Stiller, "Multi trajectory pose adjustment for life-long mapping," in *2015 IEEE Intelligent Vehicles Symposium (IV).* IEEE, 2015, pp. 901–906.

[57] M. Buczko and V. Willert, "Flow-decoupled normalized reprojection error for visual odometry," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC).* IEEE, 2016, pp. 1161–1167.

[58] I. Cvišić and I. Petrović, "Stereo odometry based on careful feature selection and tracking," in *2015 European Conference on Mobile Robots (ECMR).* IEEE, 2015, pp. 1–6.

[59] A. Geiger, J. Ziegler, and C. Stiller, "Stereoscan: Dense 3d reconstruction in real-time," in *2011 IEEE Intelligent Vehicles Symposium (IV).* Ieee, 2011, pp. 963–968.

[60] I. Cvišic, J. Cesic, I. Markovic, and I. Petrovic, "Soft-slam: Computationally efficient stereo visual slam for autonomous uavs," *Journal of field robotics*, 2017.

[61] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[62] I. Krešo and S. Šegvic, "Improving the egomotion estimation by correcting the calibration bias," in *10th International Conference on Computer Vision Theory and Applications*, 2015.

[63] A. Geiger, F. Moosmann, Ö. Car, and B. Schuster, "Automatic camera and range sensor calibration using a single shot," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 3936–3943.

[64] J. Gräter, T. Strauss, and M. Lauer, "Photometric laser scanner to camera calibration for low resolution sensors," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2016, pp. 1552–1557.

[65] Y. Xu, P. Dong, J. Dong, and L. Qi, "Combining slam with muti-spectral photometric stereo for real-time dense 3d reconstruction," *arXiv preprint arXiv:1807.02294*, 2018.

[66] J. Zhang and S. Singh, "Visual-lidar odometry and mapping: Low-drift, robust, and fast," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 2174–2181.

[67] T. Caselitz, B. Steder, M. Ruhnke, and W. Burgard, "Monocular camera localization in 3d lidar maps," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 1926–1931.

[68] J. Zhang and S. Singh, "Loam: Lidar odometry and mapping in real-time." in *Robotics: Science and Systems*, vol. 2, 2014, p. 9.

[69] Y. Balazadegan Sarvrood, S. Hosseinyalamdary, and Y. Gao, "Visual-lidar odometry aided by reduced imu," *ISPRS International Journal of Geo-Information*, vol. 5, no. 1, p. 3, 2016.

[70] J. H. Holland *et al.*, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[71] M. Mitchell, *An introduction to genetic algorithms.* MIT press, 1998.

[72] S. Gibb, H. M. La, and S. Louis, "A genetic algorithm for convolutional network structure optimization for concrete crack detection," in *2018 IEEE Congress on Evolutionary Computation (CEC).* IEEE, 2018, pp. 1–8.

[73] A. Tavakkoli, A. Ambardekar, M. Nicolescu, and S. Louis, "A genetic approach to training support vector data descriptors for background modeling in video data," in *International Symposium on Visual Computing.* Springer, 2007, pp. 318–327.

[74] D. Kalyanmoy, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[75] T. Duckett *et al.*, "A genetic algorithm for simultaneous localization and mapping," 2003.

[76] L. Moreno, J. M. Armingol, S. Garrido, A. De La Escalera, and M. A. Salichs, "A genetic algorithm for mobile robot localization using ultrasonic sensors," *Journal of Intelligent and Robotic Systems*, vol. 34, no. 2, pp. 135–154, 2002.

[77] H. M. La, T. H. Nguyen, C. H. Nguyen, and H. N. Nguyen, "Optimal flocking control for a mobile sensor network based a moving target tracking," in *2009 IEEE International Conference on Systems, Man and Cybernetics*, Oct 2009, pp. 4801–4806.

[78] A. Sehgal, H. La, S. Louis, and H. Nguyen, "Deep reinforcement learning using genetic algorithm for parameter optimization," in *2019 Third IEEE International Conference on Robotic Computing (IRC).* IEEE, 2019, pp. 596–601.

[79] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[80] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[81] T. Degris, P. M. Pilarski, and R. S. Sutton, "Model-free reinforcement learning with continuous action in practice," in *2012 American Control Conference (ACC)*. IEEE, 2012, pp. 2177–2182.

[82] B. T. Polyak and A. B. Juditsky, "Acceleration of stochastic approximation by averaging," *SIAM Journal on Control and Optimization*, vol. 30, no. 4, pp. 838–855, 1992.

[83] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines," https://github.com/openai/baselines, 2017.

[84] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3213–3223.

[85] P. H. Torr and A. Zisserman, "Mlesac: A new robust estimator with application to estimating image geometry," *Computer vision and image understanding*, vol. 78, no. 1, pp. 138–156, 2000.

[86] P. H. Torr and A. W. Fitzgibbon, “Invariant fitting of two view geometry,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 5, pp. 648–650, 2004.

[87] A. Sehgal, A. Singandhupe, H. M. La, A. Tavakkoli, and S. J. Louis, “Lidar-monocular visual odometry with genetic algorithm for parameter optimization,” *arXiv preprint arXiv:1903.02046*, 2019.

[88] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, “A benchmark for the evaluation of rgb-d slam systems,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems.* IEEE, 2012, pp. 573–580.