Deterministic Policy Gradient Based Robotic Path Planning with Continuous Action Spaces

Somdyuti Paul and Lovekesh Vig
TCS Research, New Delhi, India
Email : { somdyuti.p2 | lovekesh.vig}@tcs.com

Abstract

One of the most important tasks for Autonomous Robotics is the ability to manipulate objects in real world unstructured environments. Traditional path planning for robotic manipulators requires precise location of the target object in the environment based on which inverse kinematics return the required joint-angles for approaching the object. This limits their use in real domains with dynamic relative positions of objects not being readily available. Recent work on deep reinforcement learning for manipulation appear to be more succesful in adapting to different target object positions. In this paper, we present a deterministic policy based actor-critic learning framework to encode the path planning strategy irrespective of the robot pose and target object position. This reinforcement learning (RL) agent solely uses two different views of the environment to learn about path planning in order to reach a given target from a random pose, instead of relying on a depth sensor. The state-space for the RL agent is thus defined as the stereo-view of the environment whereas the action values are torques applied to the robot's joints. The reward function is defined on the relative distance between the end-effector and target object in pixels. In the episodic learning framework, the actor-critic network learns the optimal actions in the continuous space of real numbers for a given state configuration by trying to increase the expected reward. We demonstrate the validation of this approach in a simulated environment yielding 100% success rate from 100 different robot poses, with relatively few steps required on an average to reach the target. We further show that our learning strategy bests deep Q-learning based methods which have been used for similar path planning purpose. This path planning approach does not require conventional feature matching and triangulation for object localization which is error prone and inaccurate, and solves inverse kinematics and depth estimation using only the scene information.

1. Introduction

Manipulating objects in real world dynamic scenarios is one of the most challenging problems in Robotics. The difficulty of the task can be gauged from the fact that even in highly structured tasks such as picking objects from warehouse shelves, we have not achieved anywhere near human level performance as demonstrated by the results of the 2016 Amazon Picking Challenge [1]. However, recent developments in deep reinforcement learning offer directions for developing adaptive vision based controllers that can operate in dynamic environments. Reinforcement learning (RL) has enjoyed a prominent place in the field of machine learning and artificial intelligence as a powerful framework for teaching intelligent actions to an agent in an unknown or unstructured environment, in order to maximize a notion of cumulative reward. However, traditional reinforcement learning necessitates manual identification of suitable features to represent the state of the system [19]. With the progress of deep learning techniques in the recent past, this problem has become considerably simpler, with deep neural networks making it possible for the agent to automatically identify high level features encoding the state of a system from low level data such as raw pixel values. This has led to a resurgence of RL techniques to solve complicated tasks, which were intractable before its integration with deep learning. It has widely been surmised that the deep reinforcement learning (DRL) framework is capable of learning many human level tasks, having even surpassed human level performance in tasks like learning to play games as complicated as Go [17].

In this paper, we propose a vision based deep RL technique for path planning of a robotic manipulator, with the end goal of reaching a target object from any arbitrary initial pose in an unstructured 3D environment. The typical solution for this problem in real-world applications requires object localization in a 3D environment, which is used by the inverse kinematic solver for path planning. The agent trained to play Atari games using a Deep Q-Network (DQN) [15] demonstrated that we can apply RL for similar prob-

lems where action strategy can be learned purely on raw scene images. This obviates the need for solving the perception and planning problem individually. Instead we can directly learn the control strategy for the agent based on raw images, where the learning objective is to minimize the temporal difference error. Based on this approach, DQN has also been applied successfully to robotic control tasks such as path planning. However, despite the success of DQN in learning actions from high dimensional observation spaces, its performance while dealing with high dimensional action spaces remained sub-optimal. Also many physical control tasks have a continuous action space; coarse discretization such action spaces for the purpose of applying DQN severely limits the dexterity in performance of tasks which require finer control. On the contrary, a finer discretization of the action space leads to an exponential increase in the dimensionality of the action space, which is often unamenable to deep Q learning. Consequently, training a DQN to perform a task with an inherent continuous action space required enormous amount of data in order to ensure sufficient coverage of the continuouss action space after discretization failing which the agent could not learn viable actions. Thus, the need for a RL agent which can work with a continuous action space instead of discrete one was imminent in order to mitigate the tradeoff of dexterity and action space dimensionality. This has been elegantly achieved by [14] by leveraging the success of deep Q-learning for continuous control tasks using a deterministic policy gradient based actor-critic algorithm.

We adopt the approach of [14] in our work to a actor-critic algorithm using deterministic policy gradient to learn the action value at each state of the environment. The only information about the state of a system that is provided to the network is in the form of a stereo image pair taken from static RGB cameras. The action space comprises continuous and real valued torques, applied to each joint of the robot. The rewards which guide the training are computed from a relative change in distance between the end effector of the manipulator and the target object computed on the image plane. Further, to establish the advantage of using a continuous action space, we also apply a similar training strategy with a discrete action space to train a deep Q network and compare its performance to the deterministic policy gradient (DPG) based method.

The primary contributions made in this paper are: 1) A stereo vision based technique for robotic path planning in a 3D environment using a continuous action space and 2) An implicit depth estimation from a pair of grayscale stereo images using a deep actor-critic network. The trained agent is able to develop an implicit perception of depth of the target from the stereo images that it is trained upon, and is thus able to consistently guide the manipulator to approach the target object. Our experiments reveal that the DDPG based actor-critic agent trained on a continuous action space agent outperforms a learned DQN agent trained on a quantized discrete action space in terms of the number of steps required to reach the target as well as training episodes required.

The rest of the paper is organized as follows: Section 2 provides a brief overview of related works in this area; Section 3 elaborates the problem formulation; Section 4 describes the proposed technique; experimental results are presented in Section 5. Finally conclusions and avenues for future work are discussed in Section 6.

2. Related Works

RL based techniques have proven useful for tasks such as Simultaneous Localization and Mapping (SLAM), which involve efficient path planning and trajectory optimization to maximize the accuracy of the map constructed by a robot from sensor data in an unknown environment [9]. Subsequently, reinforcement learning found application in a wide array of robotic control tasks including target reaching [20] and grasping [2]. Reinforcement learning was shown to be effective in generalizing a set of parameterized motor primitives to perform different tasks such as throwing darts, and striking a ball in table tennis [8]. It was also employed to perform the visual servoing task using Neural-Fitted Q iteration (NFQ) [11]. The promise shown by deep learning in extracting visual features suitable for classification and segmentation tasks made it only natural to train a CNN for grasping, by assigning a score class to each gripper pose, with a supervised learning framework using ground truth scores [7]. However, despite the phenomenal success of deep reinforcement learning for developing autonomous agents in artificial intelligence, such as the autonomous atari game player developed in [15] recently, researchers have just began to focus on exploiting the potential of deep learning when trained in conjunction with RL strategies for robotics. We believe that DRL is a technique versatile enough to be applicable to a wide variety of robot control tasks such as path planning and manipulation.

Vision based perception is routinely used in closed loop robotic control systems to perform specific manipulation tasks in a known environment. However, such systems require significant prior knowledge about the robot poses and the environment, and is not suitable for trajectory planning or learning new manipulation skills in an unstructured environment. Convolutional Neural Networks (CNNs) and their recent variants have acheived state of the art performance in learning feature representations of visual data, with a tremendous success in image based classification tasks [10]. Subsequently, RL strategies have been successfully exploited for training a CNN regressor for object detection [4]. The versatility of deep reinforcement learning was demonstrated in [14], which used an actor critic

algorithm with deep function approximators to learn policies over a continuous action space to perform a variety of tasks such as legged locomotion, manipulation and car driving. The learning is based on the deterministic policy gradient algorithm [18]. 3D manipulation skills were learned on simulated as well as physical robots by training a deep Q-function using a variant of the Normalized Advantage Function (NAF) algorithm in [16]. A CNN based deep learning scheme was employed in [12] to learn visuomotor policies for a set of manipulation tasks by training on RGB camera images, as well as joint encoder readings of a robotic manipulator; this approach introduced a trajectory centric RL algorithm to generate guiding distributions meant to supervise the training. However, this does not rely solely on visual perception and requires joint state information of the robot. In [13] a CNN is used to learn a continuous visual servoing task for robotic grasping using only monocular images. This approach requires large scale data collection to train the CNN to predict the success of different grasp attempts and to correct the robot's motor actions through continuous servoing. The first attempt towards learning a trajectory for reaching a specified target relying entirely on visual data in the form of RGB camera images was made in [21]. Recently, a similar DRL based scheme was proposed in [6], for learning the combined task of reaching a target, and then grasping and lifting it.

To the best of our knowledge, [14], [21] and [6] are the only works to have focussed on path planning of a robotic manipulator from an arbitrary initial pose in an unstructured environment using DRL. However, our approach differs significantly from these works; In [14], a 'fixed reacher' is attempted as one of the several tasks and is learned using low dimensional feature inputs in the form of robot positions and joint angles as well as high dimensional information in the form of pixel values. Our work on the other hand utilizes only information in the form of pixel values and dispenses with the need of having robot positions and joint angles to describe the state. Both [14] and [21] use a 2D simulator for training, which makes the learned agent completely agnostic to the perception of depth and thus they fail to reach a target object when implemented on a real robot in a 3D environment as reported in [21]. Further it uses a simple reward function which assigns rewards of +1, 0 and -1, in an approach akin to that of [15]. Although a 3D simulator is used for training in [6], the depth information is not incorporated in the input to the network, which consists of a 2D image of the 3D simulator environment. Essentially, this reduces to training on a 2D image of a 3D scene, which obscures the depth information available in the 3D simulator. Although this approach has been shown to be successful while testing both on the simulator, and on a real robot, it should be pointed out that testing has been done under simplistic scenarios, and such a learning with a single view of a 3D world is unlikely to be successful in a cluttered environment where occlusions would inevitably be a common occurrence. While we also train and test in a simplistic 3D environment at present, our conjecture is that the perception of depth is indispensable to an agent learning to acquire target reaching and grasping skills.

Thereby, in order to make the agent learn an implicit perception of depth exploiting the visual cues available in a 3D environment, we propose a stereo vision based learning scheme, in which the agent is trained on pair of RGB camera images, using simulated camera models to capture snapshots of the 3D environment. We expect this approach to be able to better generalize when applied to real world scenarios. The fact that the 3D simulator (Gazebo) we use is well equipped to simulate the physics of the real world such as gravity and collision properties is also expected to improve generalization. Another notable difference of our approach with the technique of [6] lies in the reward function used; the distances used in the computation of their inverse exponential reward function are obtained directly using the x, y and z coordinates obtained from the 3D simulator. We refrain from using the 3D coordinates obtained from the simulator, and instead use 2D camera images to estimate the actual three dimensional distance between the end effector and the target. Pretrained networks trained further on real robots has the potential to vastly improve the performance of target reaching [3], but 3D coordinates would not be available while training in the real world. This makes our scheme of reward assignment much more amenable to be transferred to exploitive learning in real world scenarios than the technique of [6]. The use of a continuous action space unlike the discrete ones employed in [21] and [6] substantially aids the learning process as discussed further in Section 5.

3. RL Formulation

We formulate the problem of target reaching from an arbitrary initial pose of the robot as a Markov Decision Process (MDP), whereby the agent learns a policy π which maps a sequence of images into a sequence of actions in the form of efforts applied to the robot's joints. We train a deep Q-network to learn this task using a simulated version of a 7 degree of freedom manipulator. We simulate the 3D environment comprising the robot and the target objects using the Gazebo simulator. A pair of RGB cameras mounted at static positions are also simulated in order to capture two 2D snapshots of the 3D scene. Fig. 1 shows the simulated environment used in our experiments.

3.1. Action Space

The action space for the path planning task comprises continuous efforts applied at each joint defined as follows:

$$\mathcal{A} \in \mathbb{R}^N \tag{1}$$

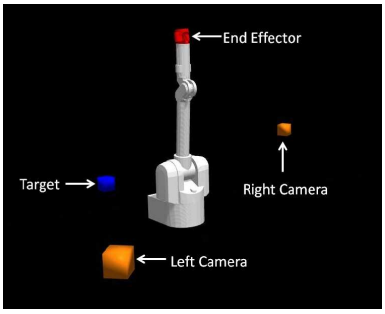


Figure 1. Simulation environment

Although the manipulator used in our experiments has 7 degrees of freedom, we make use of only 4 degrees of freedom for our initial experiments since we observe that the last 3 joints are chiefly responsible for manoeuvring the wrist and thus mostly play a redundant role in the task of reaching a target object. Since we are actuating 4 joints, N is 4 in our experiments. An action $\mathbf{a} \in \mathcal{A}$ is thus a 4 dimensional vector will real values.

3.2. State Space Representation

The state space $\mathcal{S}$ is defined in terms of 320×240 RGB images captured by the cameras are resized to 84×42, and converted to the YUV color space. The luminance (Y) channel of each image is extracted. These steps are necessary for reducing the input dimensionality. The Y channels from the images of two cameras are concatenated along their vertical dimension to create a composite image of size 84×84. Let us call this composite image $I(x, y)$, which constitute the raw state input to the network. $I(x, y)$ resized to a 7056 dimensional vector is the state $\mathbf{s}$ of the system.

3.3. Reward Assignment

We have designed a reward function for our task which facilitates incremental learning by designating all intermediate steps which bring the manipulator closer to the target as favourable steps towards the ultimate goal of reaching the target. Intuitively, the distance between the end effector and the target is a good measure in this regard. In our simulation environments we can readily compute this distance from the 3D coordinates of the end effector and the target. However, for a real robot interacting with the physical world, we would not have access to these 3D coordinates. Thereby, in order to make our framework compliant with real world constraints, we choose to compute a measure of this distance from the 2D camera images available to us.

To compute the distance between the end effector and the target, we mark them with color markers having red color for the end effector,and blue color for the target. We detect these color blobs from the images taken by the two cameras, and compute the distance between the centroids of these two blobs in the image plane. To assign the rewards based on these two distances, we must also take into account the fact that in some states, the end effector might be occluded in any one of these two views. Let $d_1(t)$ and $d_2(t)$ be the distances of the centroid of the end effector to that of the target object computed from the two camera images at the t^{th} step. Then the change in distance of the target from the end effector with respect to the previous state for the two cameras is given by:

$$\begin{aligned} \Delta d_1 &= d_1(t-1) - d_1(t) \\ \Delta d_2 &= d_2(t-1) - d_2(t) \end{aligned} \tag{2}$$

The measures of the relative change in distance between two consecutive states, Δd_1 or Δd_2 will be undefined if the end effector is in an occluded state at either the current step t or at the previous step $t-1$ in the corresponding camera view. Δd_1 and Δd_2 are normalized between -1 and +1. A state is defined as a goal state or terminal state when the end effector reaches the vicinity of the target defined by a threshold ρ applied on the average distance $(d_1(t) + d_2(t))/2$ with respect to the two cameras. The positions of the target relative to the cameras is chosen such that the end effector is unoccluded in both the camera views in the proximity of the target, so that the average distance computed from the images unambiguosly reflects the actual distance of the end effector from the target in the 3D environment. When the goal state is attained, we set the terminal flag τ indicating the end of a learning episode, i.e. $\tau = 1$ when $(d_1(t) + d_2(t))/2 \leq \rho$. We then define the reward function as follows:

$$r_t = \begin{cases} 1, & \text{if } \tau = 1, \\ -1, & \text{if } \Delta d_1 = 0 \text{ and } \Delta d_2 = 0, \\ \Delta d_1, & \text{if } \Delta d_2 \text{ is undefined}, \\ \Delta d_2, & \text{if } \Delta d_1 \text{ is undefined}, \\ (\Delta d_1 + \Delta d_2)/2 & \text{otherwise} \end{cases} \tag{3}$$

The agent is given a high positive reward of +1 whenever it takes an action that brings it to the goal state. Sometimes, a joint may reach the limit of its joint angle when consecutively applied efforts force it to move in any one direction. In such a state, the robot joint cannot move any further in that direction even if the effort applied at the next step actuates it to move in that direction. If all the joints are at their joint angle limits, and they do not move any further on application of the next action, then the system would not change its state. This indicates that the applied action at this state is a redundant one, and is therefore heavily penalized with a reward of -1. In all other cases, we assign a reward proportional to the change in distance from the target on application of the action. If the applied action reduces the end effector's distance from the target, it gets a positive reward and vice versa.

4. Proposed Method

The raw state information available to the agent is a vector of pixel values, denoted by $\mathbf{s}$. At each training step, we store the experience tuple $\mathbf{e}_t = (\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$ in the replay memory D, where $\mathbf{s}_t$ is the present state $\mathbf{a}_t$ is the current action, r_t is the observed reward, and $\mathbf{s}_{t+1}$ is the next state. For the initial P steps of the training, where $P < |D|$, random actions are chosen at each step, and the replay memory is populated with the resulting experience, without updating the network parameters. After the first P steps, learning commences, and the network is updated on minibatches of size M sampled at random from the replay memory.

We use an actor-critic network trained using the DPG approach to learn the action policy from the pixel observations representing the state of the system. As in[14], we employ a parameterized actor function $\mu(s|\theta^\mu)$ which deterministically maps states to actions using the current policy μ. The current actor policy is updated by the critic $Q(\mathbf{s}, \mathbf{a})$ which itself is learned using the Bellman equation as in Q learning.

4.1. Algorithm

RL based methods seek to maximize the expected future rewards. The discounted future reward from a state is defined as:

$$R_t = \sum_{i=t}^{T} \gamma^{t-i} r_t \tag{4}$$

Here, $\gamma \in [0, 1]$ is the discount factor for future rewards. In this RL forumulation, the policy π is learned such that expected return from the start distribution is maximized, where the start distribution $J = \mathbb{E}_{\mathbf{s}_i, \mathbf{a}_i, r_i}[R_1]$

Assuming a deterministic target policy $\mu : \mathcal{S} \leftarrow \mathcal{A}$, we use the Bellman equation for our RL paradigm to get the optimal action values of the actor as follows:

$$Q^\mu(s, a) = \mathbb{E}_{\mathbf{s}'}\big[r + \gamma \max_{\mathbf{a}'} Q^\mu(\mathbf{s}', \mathbf{a}')|\mathbf{s}, \mathbf{a}\big] \tag{5}$$

where γ is the discount factor for future rewards. Thus, the optimal action value function is arrived at by choosing that action which maximizes the expected future rewards. In this context, the CNN with weights θ^Q acts as a non-linear approximator for this action value function. The target value at the t^{th} step is approximated as follows:

$$z = r_t + \gamma \max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}'; \theta^Q) \tag{6}$$

Using these approximate target values, the loss function at the t^{th} training step is given by:

$$L_t(\theta^Q) = \mathbb{E}_{\mathbf{s}, \mathbf{a}, r}\big[(\mathbb{E}_{\mathbf{s}'}[z|\mathbf{s}, \mathbf{a}] - Q(\mathbf{s}, \mathbf{a}; \boldsymbol{\omega}_t)^2\big] \tag{7}$$

Q-learning is one such RL technique which minimizes the above loss function to learn the optimal action value function and thereafter a greedy approach is employed to select the optimal action. However for a continuous action space finding such a greedy policy is impractical. The actor-critic reinforcment learning model based on deterministic policy gradient proved to be useful in this scenario. In this learning framework, an actor function $\mu(\mathbf{s}|\theta^\mu)$ deterministically maps states to continuous action values. The critic $Q(\mathbf{s}, \mathbf{a}|\theta^Q)$ is updated using the Bellman equation similar to Q-learning. The actor is updated by applying the chain rule on the expected return from the start distribution with respect to the network parameters as follows:

$$\begin{aligned} \nabla_{\theta^\mu} J &\approx \mathbb{E}_{\mathbf{s}_t}\big[\nabla_{\theta^\mu} Q(\mathbf{s}, \mathbf{a}|\theta^Q)|_{\mathbf{s}=\mathbf{s}_t, a=\mu(\mathbf{s}_t|\theta^\mu)}\big] \\ &= \mathbb{E}_{\mathbf{s}_t}\big[\nabla_{\theta^\mu} Q(\mathbf{s}, \mathbf{a}|\theta^Q)|_{\mathbf{s}=\mathbf{s}_t, a=\mu(\mathbf{s}_t)} \nabla_{\theta^\mu} \mu(\mathbf{s}|\theta^\mu)|_{\mathbf{s}=\mathbf{s}_t}\big] \end{aligned} \tag{8}$$

Unlike supervised learning, the targets are not fixed, and depend upon the network parameters. We employ the soft target update strategy of [14] where a copy of the actor and critic networks, $\mu'(\mathbf{s}|\theta^{\mu'})$ and $Q'(\mathbf{s}, \mathbf{a}|\theta^{Q'})$ are utilized for computing the target values. The weights of the target network are then updated as follows:

$$\theta' \leftarrow \zeta\theta + (1 - \zeta)\theta' \text{ where } \zeta \ll 1 \tag{9}$$

The above update strategy lets the weights of the target network change slowly according to the learned weights, thereby ensuring the stability of the learning process.

In order to perform exploration efficiently over a continuous action space, we resort to the Ornstein-Uhlenbeck process whereby we add a noise sampled from a temporally correlated noise process $\mathcal{N}$ with inertia to the action values generated by the actor as follows:

$$\mu'(\mathbf{s}_t) = \mu(\mathbf{s}_t|\theta_t^\mu) + \mathcal{N} \tag{10}$$

The network parameters are updated at each training step after the first P steps utilized for building the replay memory. We use Adam to update the network parameters and the soft target update of Eq. 9 in order to make the target networks track the parameters of the updated network [14]

4.2. Network Architecture

The input to the network is the composite image of size 84×84 obtained after preprocessing. We use the actor-critic network architecture employed in [14], where the size of the final layer is equal to the number of degrees of freedom of the robotic manipulator used for the task. Our networks comprise 3 convolutional layers having 32 filters each followed by 2 fully connected layers, each of 200 units. All hidden layers used a rectifier non-linearity activation. Batch

Algorithm 1 Deep Q-network Training

Input: composite image I_t and goal state information τ
Initialisation :Replay memory $D = \{\mathbf{e}_1, \mathbf{e}_2, \cdots \mathbf{e}_P\}$; actor network $\mu(\mathbf{s}|\theta^\mu)$ and critic network $Q(\mathbf{s}, \mathbf{a}|\theta^Q)$ with random weights θ^μ and θ^Q respectively; target networks μ' and Q' with weights $\theta^{\mu'}$ and $\theta^{Q'}$ respectively.

for $t = 1$ to 10,000 **do**
  Initialize random process $\mathcal{N}$ for exploration.
  while $(\tau \neq 1)$ **do**
    Preprocess I_t and vectorize to represent state $\mathbf{s}_t$.
    Select random action $\mathbf{a}_t = \mu(\mathbf{s}_t|\theta^\mu) + \mathcal{N}$
    Execute $\mathbf{a}_t$ in simulator, and obtain r_t, I_{t+1} and τ
    Preprocess I_{t+1} and vectorize to represent state $\mathbf{s}_{t+1}$.
    Add the transition $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$ in D.
    if $t \geq P$ **then**
      Sample a random minibatch of size M consisting of transitions $(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}_{i+1})$ from D.
      Set $z_i = r_i + \gamma Q'(\mathbf{s}_{i+1}, \mu'(\mathbf{s}_{i+1}|\theta^{\mu'})|\theta^{Q'})$
      Update critic by minimizing the loss $L = \frac{1}{M}\sum_i (z_i - Q(\mathbf{s}_i, \mathbf{a}|\theta^Q))^2$ with respect to the network parameters θ^Q.
      Update the actor network using sampled policy gradient as follows:
      $\nabla_{\theta^\mu} J \approx \frac{1}{M}\sum_i \nabla_a Q(\mathbf{s}, \mathbf{a}|_{\mathbf{s}=\mathbf{s}_i, \mathbf{a}=\mu(\mathbf{s}_i)} \nabla_{\theta^\mu} \mu(\mathbf{s}|\theta^\mu)|_{\mathbf{s}_i}$
      Update the target actor and critic networks as follows:
      $$\theta^{Q'} \leftarrow \zeta\theta^Q + (1-\zeta)\theta^{Q'}$$
      $$\theta^{\mu'} \leftarrow \zeta\theta^\mu + (1-\zeta)\theta^{\mu'}$$
    end if
  end while
end for

normalization is employed at the input and after each hidden layer.

For the sake of comparison, we implement a deep Q network for the same path planning task using the network architecture employed in [15], but modify the fully connected output layer to account for the much larger discrete action space used in our experiments in comparison to that required. It must be pointed out that size of the fully connected layer at the output is typically much smaller for the actor-critic network than the Q network due to the discretization of the action space.

5. Experimental Results

We use a 7 degree of freedom Barrett Wide Arm Manipulator simulated in Gazebo using ROS Indigo as the interface between our algorithm and the simulated robot. We choose Torch7 with Cuda7.5 as the deep learning library to construct and train the deep Q-network. All the experiments are performed on an ubuntu 14.04 system with 16 GB RAM, Intel Core-i7-4810MQ processor and NVIDIA Quadro K2100M GPU.

We use four degrees of freedom of the manipulator for the reaching task. So, the size of the action space $N_\mathcal{A}$ is 4 in our experiments. The value of the threshold ρ is set to 10. The performance of our path planning algorithm is critically dependent on the choice of several hyperparameters. We summarize the values used for these hyperparameters along with their definitions for DDPG and DQN trainings in tables 1 and 2

For the actor network, we use a L_2 weight decay of 10^{-2}. The weights of the final layers of both the actor and critic networks are initialized from a uniform distribution $[-3 \times 10^{-4}, 3 \times 10^{-4}]$ whereas the weights of all other layers are initialized from a uniform distribution of $[-\frac{1}{\sqrt{f}}, \frac{1}{\sqrt{f}}]$ where f is the fan-in of the layer. For the deep Q-network, all layers were initialized using the Xavier initialization.

In the following sections, we analyze various aspects of the performance of our deep reinforcement learning based path planning framework.

5.1. Training Statistics

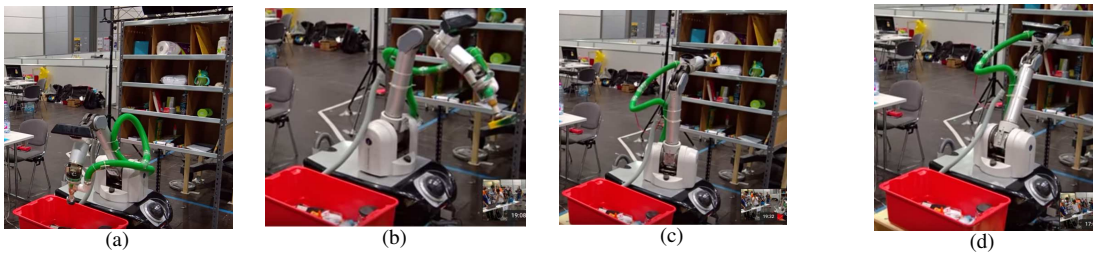
Although the combined task of reaching the target, grasping and finally lifting it as attempted in [6] is much more complicated than the task of reaching a target which we are attempting presently, reaching may be often be sufficient if a suction based actuator is used for lifting an object, which was often the case with several team in the Amazon Picking Challenge (see Figure 2). Interestingly, the agent in our approach learns to reach much faster; whereas the authors in [6] report that the agent learned to complete the entire task successfully in 1800 episodes (in their case each episode ends after 1000 iterations or completion of a task), when starting from the same initial pose, our agent learns the reaching task within just 1500 iterations. This is just 500 training steps after building the replay memory. This tremendous improvement in learning speed can be attributed to the much smaller size of the output layer in case of the policy gradient based actor-critic learning framework, which results in fewer network parameters to be learnt. Even for the method of [21], the agent used for path planning from random initial poses was trained for 5.225 million training steps, which is far greater than the number of training steps required for the agent to learn in our approach. We surmise that the larger action space used in our experiments combined with the reward assignment strategy used for incremental learning is responsible for this improvement.

Table 1. Hyperparameters used for DDPG training

Parameter	Definition	Value
P	No. of initial steps	1000
M	Minibatch size	100
ϵ	Exploitation probability	1 to 0.1 annealed linearly over 10000 steps
C	Target networks' update frequencey	1
$\|D\|$	Size of replay memory	10,000
α	Learning rate	actor - 10^{-4}, critic - 10^{-3}
γ	Discount factor	0.99
ν	Ornstein-Uhlenbeck process mean	0
σ	Ornstein-Uhlenbeck process variance	0.5
η	Ornstein-Uhlenbeck process inertia	0.15
ζ	Soft target update factor	10^{-2}

Table 2. Hyperparameters used for DQN training

Parameter	Definition	Value
P	No. of initial steps	1000
M	Minibatch size	100
ϵ	Exploitation probability	1 to 0.1 annealed linearly over 10000 steps
C	Target network update frequencey	100
$\|D\|$	Size of replay memory	10,000
α	Learning rate	0.0025
γ	Discount factor	0.2



(a) (b) (c) (d)

Figure 2. Our suction based manipulator that participated in the Amazon Picking Challenge, the arm is required to (a) reach the target object and pick it up using suction (b) transport the object to the correct location (c) choose the correct box and (d) stow the object. While this is for a retail application, it is easy to see how this may be useful for autonomous robots that are required to place small items like medicine bottles etc in their proper location

5.2. Testing Performance

A performance metric based on the average reward achieved per epidode was adopted in [5], in order to provide a benchmark for the performance of DRL based algorithms. We adopt a similar approach, but instead choose to quantify the performance of our algorithm in terms of the average number of steps required to reach the target, since unlike the rewards for the tasks in [5], our reward function is characterized by sharp discontinuities on reaching the goal, as well as on executing redundant actions. In order to evaluate the performance of our path planning algorithm, we create a test set consisting of 100 random initial poses of the manipulator. These 100 poses are obtained by setting the joint angles of the first 4 joints to random values which lie within the respective joint angle ranges, and keeping the angles of the last 3 joints (those that are responsible for the movement of the wrist) fixed at 0 radians. We evaluate the performance of the trained agent for different values of γ, and for different levels of training progress in terms of the percentage of successful attempts to reach the goal state, and the average number of steps required to reach the goal starting from each arbitrary initial pose in the test set. With these experimental setting, the performance of the DDPG algorithm summarized in 3.

For the DQN algorithm, however, training with $\gamma = 0.99$ yields worse performance than for lower values of γ. This point towards lower contribution of the previously achieved rewards on the curent action policy which is probably a consequence of the larger output layer size of the DQN. We re-

Table 3. DDPG testing Performance

γ Value	% of Successful Attempts					Average No. of Steps				
	1500	2000	5000	7000	8500	1500	2000	5000	7000	8500
0.99	100%	100%	100%	100%	100%	1.95	2.92	2.42	2.21	1.98

Table 4. DQN testing Performance

γ Value	% of Successful Attempts					Average No. of Steps				
	1500	3500	5500	7500	10000	1500	3500	5500	7500	10000
0.2	31%	100%	81%	100%	100%	21.31	2.52	8.86	3.15	2.96
0.5	62%	64%	89%	97%	96%	16.19	12.35	6.11	4.7	6.12

port the performance of the algorithm for two different values of the discount factor γ for Q learning in Table 3. Overall, $\gamma = 0.2$ gives us the best performance both in terms of success rate, and number of steps required to reach the target on an average. Increasing gamma beyond 0.5 worsens the performance, with the agent failing to reach the target in most cases for $\gamma = 0.9$. It can be seen that the technique of DDPG applied for target reaching outperforms DQN by a considerable margin both in terms of training steps required and success rate.

6. Conclusions and Future Work

A vital capability for autonomous robots is the ability to manipulate objects in unstructured environments without prior knowledge about joint angles and object positions. In this paper, we present a novel approach for robot arm control in the continuous action space using an actor-critic based RL framework trained using deterministic policy gradient. Our approach relies on stereo vision and does not require an intermediate perception module or additional information about the robot position and joint angles. We demonstrate this through the path planning of a robotic manipulator. We also employ a deep Q network to perform the path planning using the same experimental setup. Our experimental results demonstrate that the DDPG based learning technique outperforms the DQN based learning by a substantial margin both in terms of the number of learning steps required as well as success rate.

For future work, we seek to investigate the performance of the agent in a more complex environment; it is our eventual objective to establish the robustness of a trained agent in an environment with a significant amount of background clutter. With the consequent increase in occlusion, this might also necessitate adding more cameras to the simulated environment to capture views of dynamic targets which might move into and out of occlusion in the cluttered environment. Keeping this requirement in mind, we also plan to modify the network architecture to accept a volume comprising a set of camera images as input, instead of employing concatenation as in the current approach. Further, we seek to devise a way of locating and reaching different types of target objects, and eventually grasping and manipulating them. Toward this end, we propose to decouple the reaching and grasping parts by training a separate network to perform the grasping task once the end effector has reached the vicinity of the target. Finally, it would be a worthwhile endeavour to employ a transfer learning based approach [3] to exploit the knowledge acquired by a pre-trained network trained in the simulation environment to learn the same tasks with a real robot.

References

[1] Amazon picking challenge, 2016. 1

[2] T. Baier-Lowenstein and J. Zhang. Learning to grasp everyday objects using reinforcement-learning with automatic value cut-off. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1551–1556. IEEE, 2007. 2

[3] S. Barrett, M. E. Taylor, and P. Stone. Transfer learning for reinforcement learning on a physical robot. In *Ninth International Conference on Autonomous Agents and Multiagent Systems-Adaptive Learning Agents Workshop (AAMAS-ALA)*, 2010. 3, 8

[4] J. C. Caicedo and S. Lazebnik. Active object localization with deep reinforcement learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2488–2496, 2015. 2

[5] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. *arXiv preprint arXiv:1604.06778*, 2016. 7

[6] S. James and E. Johns. 3d simulation for robot arm control with deep q-learning. *arXiv preprint arXiv:1609.03759*, 2016. 3, 6

[7] E. Johns, S. Leutenegger, and A. J. Davison. Deep learning a grasp function for grasping under gripper pose uncertainty. *arXiv preprint arXiv:1608.02239*, 2016. 2

[8] J. Kober, A. Wilhelm, E. Oztop, and J. Peters. Reinforcement learning to adjust parametrized motor primitives to new situations. *Autonomous Robots*, 33(4):361–379, 2012. 2

[9] T. Kollar and N. Roy. Trajectory optimization using reinforcement learning for map exploration. *International Journal of Robotics Research*, 27(2):175–196, 2008. 2

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 2

[11] T. Lampe and M. Riedmiller. Acquiring visual servoing reaching and grasping skills using neural reinforcement learning. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pages 1–8. IEEE, 2013. 2

[12] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016. 3

[13] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, page 0278364917710318, 2016. 3

[14] 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. 2, 3, 5

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. 1, 2, 3, 6

[16] G. Shixiang, H. Ethan, L. Timothy, and L. Sergey. Deep reinforcement learning for robotic manipulation. *arXiv preprint arXiv:1610.00633*, 2016. 3

[17] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. 1

[18] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 387–395, 2014. 3

[19] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998. 1

[20] M. Tamosiunaite, T. Asfour, and F. Wörgötter. Learning to reach by reinforcement learning using a receptive field based function approximation approach with continuous actions. *Biological cybernetics*, 100(3):249–260, 2009. 2

[21] F. Zhang, J. Leitner, M. Milford, B. Upcroft, and P. Corke. Towards vision-based deep reinforcement learning for robotic motion control. *arXiv preprint arXiv:1511.03791*, 2015. 3, 6