Deceive the Agent: Transferring Deep Reinforcement Learning based Visual Servoing from Simulation to Reality

Virender Singh[1], Prerna Singh[1], Ayush Kumar[1], Samrat Dutta[1] and Swagat Kumar[1]

***Abstract*— In this paper, we propose a novel technique called 'Deceive the agent' for transferring policy learnt in simulation to the real world. For the purpose of our experiments, we first implement feature-agnostic image based visual servoing task on UR5 robot using state-of-art deep deterministic policy gradient(DDPG) algorithm in Gazebo simulator. Since, the real world is drastically different from the simulation environment. An agent faces different challenges like the difference in input images in real and simulation world. We attempt to reduce gap between simulated environment and real world environment by making our training robust to real world changes. During training, we add Gaussian noise to state vector used by UR5 robot for learning the policy. We present an analysis of impact of different noise levels on training performance of the agent. While transferring on real system, we propose a novel technique of converting real images to simulation images, we call this technique as 'deceive the agent'. Using this method, our agent perceives real environment images as images encountered during simulation. We analyze the similarity achieved between real images and converted simulation images. Our results show high similarity between converted simulation images and corresponding real images in terms of SSIM and PSNR. We test our algorithm on real UR5 robot. Our results show that the agent successfully reaches the target object in real environment.**

I. INTRODUCTION

Visual Servoing (VS) [12][3] or hand-eye coordination [9] uses visual feedback for controlling the motion of a robot. Visual servoing has played a significant role in industrial setting. Visual Servoing has been used in multi-arm robot in [7] and has been used along with collision avoidance in [1]. Visual servoing can be broadly divided into two categories: Image based visual servoing(IBVS) and Position based visual servoing(PBVS). In IBVS, the velocities of the robotic manipulator are directly dependent on the image feature error. This method can be further divided into classes based on robot camera configuration: **a.** eye-to-hand where the camera is stationary and external to the robotic manipulator, **b.** eye-in-hand where camera is a part of the robotic system and undergoes the same motion as that of robot. In this paper, we will be focusing on IBVS in eye-in-hand configuration. The PBVS method, unlike IBVS, uses image feature to estimate the current and desired poses in the Cartesian task space. Since, IBVS in eye-in-hand configuration involves camera motion and is a more difficult problem and therefore, we use it as a test bed for our problem. Our first objective is to solve IBVS in simulation using deep reinforcement learning.

With the advancement of artificial intelligence, the technique of deep reinforcement learning(RL) has shown promising results to perform different robotic tasks. However, it's success has been limited to the simulation environments. [9] has shown human level performance in atari games using deep RL, [4] and [12] have used deep RL to perform robotic controls. But, RL has had limited success as we move from simulation to real world. Unfortunately, the drastic difference between physics simulators and the real world make transferring behaviors from simulation to real a challenging task. In [13] authors use domain randomization technique to bridge the gap between real and simulation. The work presents introduction of variability in the simulation environment in the form of position and texture of objects, number of distracting objects, light conditions etc. This randomization helps the model to adjust to real world environment as it perceives real world as a variation of the simulation environment. In the paper [11], authors propose the use of progressive nets to bridge the gap between simulation and reality. However, these works did not include a RL agent.

Another approach used for transfer learning from simulation to real is to perform complete training on the actual hardware itself. In paper [5], authors collect 0.8 million grasp attempts from 14 real robotic manipulators and use this for training a neural network to predict whether the motion of gripper will result in a successful grasp or not. However, performing complete training on real robots is costly, potentially unsafe, and requires very lengthy training times. RL requires large amount of data to train and also applies exploration during training, this can be hazardous for a real robot in laboratory or industrial setting. In addition, running real robots for such long hours can lead to wear and tear of the motors of the robotic manipulator. Training in simulation is easier and cheaper, but transfer learning is needed to generalize from simulation to the real world. In the paper [2], authors propose the use of simulated data as a prior for training on real environment. In the paper [3], authors propose a method that uses a simulator to learn low dimensional subspace of the policy and then fine tune on the real system using much less data than would be required if training was done from scratch on the real system itself. In this paper, we attempt to transfer learning learnt in simulation to real world without training on the real system from scratch. We also perform no fine tuning in the real environment. We take inspiration from domain randomization approaches and train our agent in simulator with gaussian noise added to the state vector. We also present a unique algorithm to deceive the agent such that

[1]TCS Research and Innovation Labs, Noida albert.author@papercept.net

it perceives real environment as the simulated one. We propose a method of converting real images to simulation ones so that the agent perceives real environment as the simulation environment. Our results show that agent is able to perform the visual servoing task in real environment straight after training in simulation.

A. Contribution

In this work, we bridge the gap between simulation and reality to ensure an adaptable model in the real world which is first trained in simulation. We design an ideal robotic task environment in simulation and train our agent to output action steps for UR5 robot. We then develop a new technique to transfer the learning to real world with comparable performance. During training, we add noise to the state vector before sending it to the actor and critic network. We use gaussian noise with different levels of sigma to ensure a robust training adaptable to the input data from real world. To further ensure that the agent is able to see similar images in the real world that it saw in simulation, we have added an additional step in our pipeline. We call this step as 'deceive'. Here we create a simulation image out of a real image. The region of interest is carved out from the real image and attached to the background of a simulated frame. This step takes care of the variation in input data due to real world disturbances like different light conditions. The process of segmentation results in making the object boundary irregular. To make our agent robust to this situation, we train our agent in simulation with gaussian noise added to the image captured by robot's end-effector. The summary of the contributions made by this work are:
(1) A robust DDPG-based deep RL framework for eye-in-hand visual servoing of a 6DOF robotic manipulator is proposed. The manipulator is trained in simulation with different levels of noise in state vector. An analysis of the performance of the agent with different noise intensities is presented.
(2) A new technique is proposed to deceive the agent such that it perceives real environment similar to the simulated environment. The RGB images captured in real environment are converted into simulation images. The region of interest is carved out and is made similar to state vector encountered during simulation.
(3) The proposed algorithms have been implemented and validated in real environment with UR5 robotic manipulator.

II. Proposed Method

A. Environment

Environment is paramount for any RL setup. It is the physical world with which the agent interacts to achieve several states and get corresponding rewards. Any algorithm that agent learns is with respect to this environment. The environment for our problem statement consists of a target object in an image frame and a UR5 robot as the agent. The environment is simulated in the gazebo simulator. The Fig. 2 shows our simulation environment. The fig 3 shows our environment in real world.

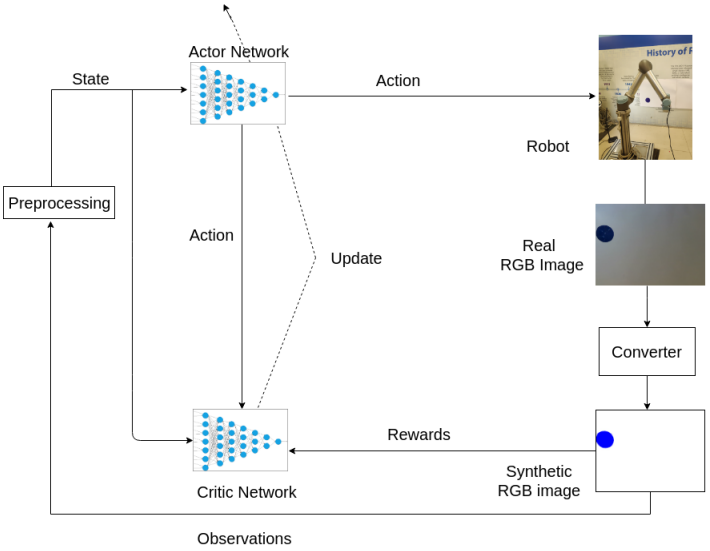


Fig. 1: The complete architecture of our approach is presented here. The image is represented as the state of the system. The action network generates the deterministic control signal. The converter block takes input images from UR5 robot in real world and converts these into simulated RGB images which are similar to the images seen by the agent during training. This makes sure that agent works similar to the simulation. The reward is collected from the environment and the critic and action networks are trained accordingly.

B. State

Condition of the agent in the environment at any instant of time is called state. Images captured by the camera mounted on the end-effector are used for obtaining the state vector. The captured RGB images are converted into YUV space and the luminance component is extracted. We prepare the current state $\mathbf{s}_t$ by resizing the luminance component of the image. The original dimension of the image captured by the end effector is $320 \times 240 \times 3$. We resize the luminance component of this image to 84×84 and use it as the state vector.

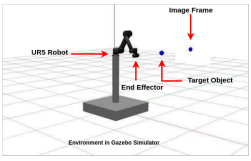


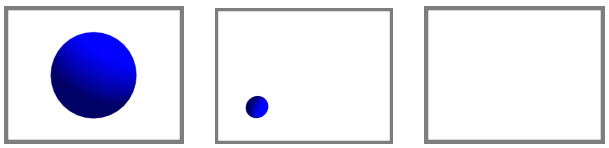
Fig. 2: Simulation environment setup

C. Deceive - Real To Simulation

The RGB images captured by UR5 robot in real environment are very different from simulation environment in terms of color of target object, background and light intensity. Therefore, the policy learnt in simulation will not perform well in a real world scenario. So, to make our agent agnostic to variation in light, color and texture, we introduce a pre-processing step in real world. The captured images are

Fig. 3: Real environment setup

first converted into simulation images. This is done using following steps:
1) Target objected is segmented out from captured RGB image by finding contours.
2) The segmented region is extracted and colored similar to the target object in simulation and the background is extracted from a sample frame in the simulation environment.
Figure 5 shows the simulation images generated from corresponding real images. We compared the real and generated simulation images using PSNR and SSIM values in table I. The high values of PSNR and SSIM indicate that we successfully converted real to simulation without significant loss in information.
In the process of conversion from real image to simulation, the usage of segmentation results in an irregular boundary around the target object. To counter this problem, we added Gaussian noise to image while training in simulation. After the generation of simulation image, we convert the image to YUV space and extract luminance component. The luminance component is resized to 84×84 for creating the real environment's state vector.



(a) (b) (c)

Fig. 4: (a) Target Image (b) Start Image and (c) Out of frame image

TABLE I: PSNR and SSIM comparison between real and generated simulation images

Image	PSNR	SSIM
a	31.065	0.714
b	31.126	0.696
c	31.203	0.747
d	29.110	0.749
e	29.191	0.759
f	28.844	0.758

D. Action Space

The action space, in our approach consists of vectors of the joint velocities of the robot. Velocity components are in radians per second. Action at any time t can be defined as $a_t = < \dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3, \dot{\theta}_4, \dot{\theta}_5, \dot{\theta}_6; t >$.
The six components of velocity or action space correspond to six joints of the UR5 robot from Base to end effector.

E. Reward Function

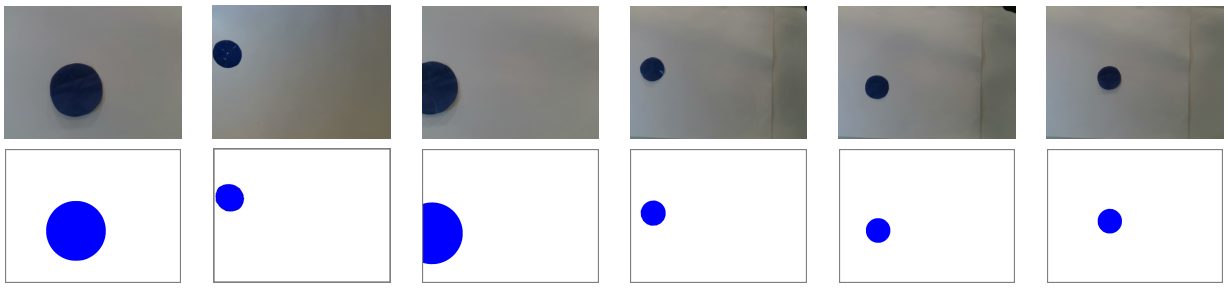
Reward function plays a crucial role in training an RL agent to perform the desired task. By means of reward, the agent learns to behave in an appropriate manner i.e. it learns to distinguish between a good move and a bad move. In classical VS, the agent tries to obtain target features similar to features in [8]. These features are used for error calculation and consequently control velocities. In our approach, we provide the agent with the target image. We do not specify any hand crafted features unlike [8]. The target image used for our problem is shown in Fig. 4.(a). We use this target image for reward calculation.

Only the 2D distance from the target has been for reward computation in [14] and [10]. This fulfills the purpose for their experiment since camera is static in eye-to-hand configuration and thus target object is always in the frame and is of fixed size. In the present work however, the camera is mounted on end-effector itself. The size of target object changes in the image frame with the motion of robot's end-effector. Hence, the distance alone is not the sufficient criteria.

In visual servoing, failure is a state when the target object is not in the camera view due to the robot movement. The failure state is represented by the "out of frame image" which must not occur during the visual servoing (Fig. 4c. shows an example of out of the frame image). We use the euclidean difference between out of frame image and the current image as the reinforcement signal in the servoing task. The idea is to reduce the error by increasing the distance between out of frame image and the current image. We define error as the euclidean distance between the current image and the target image. Inorder to reduce this error, we increase the euclidean distance between the current image and the out of view image. Since, the target image is a large spherical blue ball (see Fig. 4.a) at the center of the frame, the distance from the out of frame image is highest in this scenario. The target image and the out of view image can be roughly treated as the opposite to each other. Hence, we define our reward with help of following components:

$$R = \begin{cases} 1 & \text{if } success, \\ 0 & \text{if } failure, \\ \|current\ image - out\ of\ frame\ image\|_2^2 & \text{otherwise} \end{cases} \quad (1)$$

An episode ends in two scenarios: (1) We assign a reward 0 in cases where the target object is not in the frame and label the episode as failure. and (2) when the norm difference reaches more than a certain threshold, we say the end-effector has reached the target with success and assign it a reward of 1. In all other situations, the reward is the difference of current image with out of frame image. We make use of this reward function to train our agent.



(a) (b) (c) (d) (e) (f)

Fig. 5: Conversion from real to synthetic images which are similar to simulation images: Here the first row shows images taken from UR5 robot in a real world setup. These images are then processed to look like their counterparts in simulation environment and are shown in second row. This setup ensures that agent is not able to distinguish between real environment images and simulation environment images.

III. Algorithm

Algorithm 1 Deep Deterministic Policy Gradient Algorithm

Input: Current image frame F_t
Add gaussian noise to F_t
Create state vector s_t
Initialisation: Replay memory $\mathbf{R} = \{\mathbf{E}_1, \mathbf{E}_2, \cdots \mathbf{E}_P\}$
Initialize weights of actor, critic, target actor and target critic networks.
Algorithm:

for $t = 1$ to $numEpisodes$ **do**
  while $(\tau \neq 1)$ **do**
    Preprocess F_t to represent state $\mathbf{s}_t$
    Select random action a_t = A($\mathbf{s}_t|\omega_A$) + $\mathcal{N}$
    Execute $\mathbf{a}_t$ in simulator and obtain $< r_t, F_{t+1}, \tau >$
    Preprocess F_{t+1} to represent next state $\mathbf{s}_{t+1}$
    Add the experience $< \mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1} >$ in replay R
    if $t \geq P$ **then**
      Sample a random minibatch of size **m** consisting of transitions $< \mathbf{s}_t, \mathbf{a}_t, \mathbf{r_t}, \mathbf{s}_{t+1} >$ from R
      Update actor and critic networks.
      Soft update the target actor and critic networks.
    end if
  end while
end for

A. State-of-art DDPG

Our algorithm makes use of DDPG [6] based actor-critic network in a continuous action space. Each episode starts with an initial state where object is in field of view of the camera. A state vector s_t is created and sent as input to actor-network which produces joint velocities a_t. The robot executes these joint velocities and in return goes to next state s_{t+1} and obtains a reward r_t. At each such step, the current state s_t, the current action a_t, current reward r_t and the next state s_{t+1} are stored in the replay memory M of size S. These four components together form one experience e_t for our agent. During training, the initial P steps where $P < |M|$, random actions are chosen at each step, and corresponding experiences are stored in the replay memory, without updating the network parameters. After the first P steps, we start updating the network parameters using mini batches of size **m** from the replay memory. The complete algorithm is shown in 1. In our algorithm, we use 4 networks: actor network, target-actor network, critic network and target-critic network. When training starts, we update the weights of critic network and actor network using Adam strategy. These updated weights are used to soft update the weights of target actor and target critic networks. This step allows the weights of the target network to change slowly according to the learned weights, thereby ensuring the stability of the learning process.

IV. Experimental Results

For training in simulation, we use Gazebo7 for simulating the 6 degrees of freedom UR5 robot and ROS kinetic for communicating with the simulated robot. We choose Torch7 with Cuda 8 as the deep learning library to create and train our network. We performed all experiments on an Ubuntu 16.04 system with 32 GB RAM, Intel Core-i7-4810MQ processor and NVIDIA GeForce 1080 GPU. We also make use of different hyper parameters for training our network, the value of the parameters are shown in Table II.
For testing our algorithm in real environment, we use UR5 robotic manipulator. We attached a Real Sense SR300 camera to the end-effector of the robotic manipulator to make the UR5 in eye-in-hand configuration. We attempted to create an environment similar to simulated environment by using white sheets with a blue ball as shown in figure 3.

A. Visual Servoing in Simulation with noise variations

We introduce the concept of noise variations in the simulation during training phase. Since our main objective is to make the agent robust enough to handle different situations in real world environment, we add Gaussian noise to the images captured by end-effector of the robotic manipulator in gazebo. We use noise with different values of standard deviation for our experiments. Table III shows the number of steps taken by the network to converge for different noise levels. The agent learned to successfully reach the target object in all the situations. We observed that environment with the least noise resulted in fastest convergence i.e. least number of steps taken to converge. In order to test the

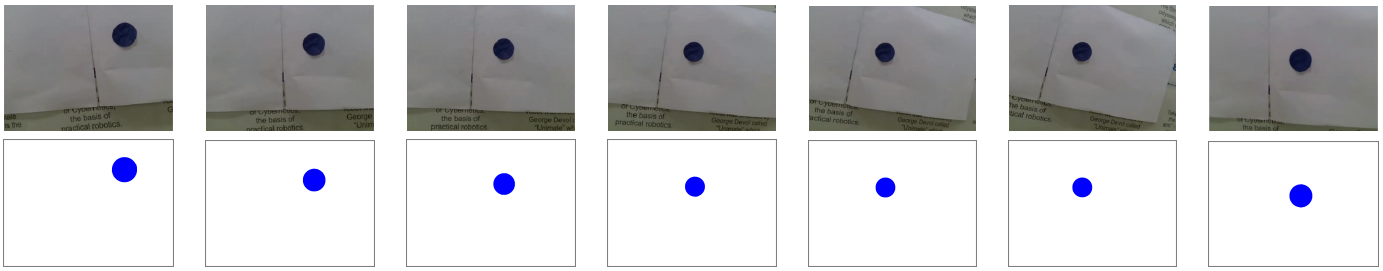


Fig. 6: First row show the end-effector view of the UR5 robot. The second row shows the corresponding synthetic images which were used to decieve the agent. The results show that the learned agent could align the target object in the center of the image frame directly using the training from simulation.

network in real environment, we select the model trained on highest level of noise i.e.σ=7.

TABLE II: Hyperparameters Used in Experiments

Parameter	Definition	Value
P	No. of initial steps after which network update starts	1000
m	Size of minibatch	100
C	Target network update frequency	100
$\|D\|$	Size of replay memory	10,0000
α	Network learning rate	0.025
γ	Discount factor	0.2

TABLE III: Test results for Experiment 1

Attributes	**DDPG**	**DDPG σ = 2**	**DDPG σ = 5**	**DDPG σ = 7**
Episodes for convergence	7000	8000	11000	14000
Number of steps	160000	190000	210000	237837

B. Visual Servoing in Real Environment

In the real UR5 robot, a real sense camera was attached to the end-effector of the robot to make it in eye-in-hand configuration. We used 'Deceive - Real to Simulation' method discussed in section C and generated simulation images for every real image captured. Figure 6 shows real images and their corresponding simulation images generated by our algorithm. Next, we tested the model trained in simulation in the real environment, with just one additional step of converting real images to synthetic images, and then using these synthetic images for creating the state vector. We found that our agent learned to bring the target object in the center of the image frame in real environment without any fine-tuning on the real system. Figure 6 shows one successful episode in real environemnt using the proposed algorithm. This shows that our proposed method was successful in deceiving the agent.

V. CONCLUSIONS

The agent successfully reached the target object in both simulation and real world environment. In future, we plan to extend our work to more complex environments like cluttered environment.

REFERENCES

[1] R. Chancharoen, V. Sangveraphunsiri, K. Sanguanpiyapan, P. Chatchaisucha, P. Dharachantra, S. Nattarorn, and S. Pongparit. Collision avoidance technique for uncalibrated visual servoing for industrial robots. In *2002 IEEE International Conference on Industrial Technology, 2002. IEEE ICIT'02.*, volume 1, pages 594–599. IEEE, 2002.

[2] M. Cutler and J. P. How. Efficient reinforcement learning for robots using informative simulated priors. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2605–2612. IEEE, 2015.

[3] J. Z. Kolter and A. Y. Ng. Learning omnidirectional path following using dimensionality reduction. In *Robotics: Science and Systems*, 2007.

[4] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[5] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen. Learning hand-eye coordination for robotic grasping with large-scale data collection. In *International Symposium on Experimental Robotics*, pages 173–184. Springer, 2016.

[6] 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.

[7] V. Lippiello, B. Siciliano, and L. Villani. Position-based visual servoing in industrial multirobot cells using a hybrid camera configuration. *IEEE Transactions on Robotics*, 23(1):73–86, 2007.

[8] Y.-H. Liu, H. Wang, and Z. Wang. Adaptive visual servoing using common image features with unknown geometry. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 87–92. IEEE, 2008.

[9] 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, 2015.

[10] S. Paul and L. Vig. Deterministic policy gradient based robotic path planning with continuous action spaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 725–733, 2017.

[11] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[12] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[13] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 23–30. IEEE, 2017.

[14] 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.