B.Tech

Project Report

Semester (VIII)

Visual Servoing for Eye-in-Hand Robotic Manipulator using Deep Deterministic Policy Gradients

Student:
Sudhir PratapYadav
(B14EE033)

Mentor:
Dr. Suril V. Shah
Assistant Professor
Department of Mechanical Engineering

॥ त्वं ज्ञानमयो विज्ञानमयोऽसि ॥
Indian Institute of Technology, Jodhpur
Department of Electrical Engineering

May 22, 2021

Abstract

Visual Servoing is a decades old problem in robotics which deals with achieving a desired target features from robot's perspective. However, in classical Visual Servoing techniques, issues related to feature choice, designing interaction matrix and singularilty of matrices pose a challenge. Recent works on Reinforcement Learning have been suc- cessful in path planning for robots where target position is not fixed. In this paper, we present our work where we try to solve Visual Servoing problem using Reinforcement Learning methods. We present a deterministic policy based actor-critic learning framework for visual servoing on UR5 robot. We are using eye in hand config- uration of the robotic manipulator. So, here the agent uses only one view of the environment which is given by the camera mounted on the end effector of UR5. Our agent derives the state from an image of the target as seen from the end effector camera. We are also making use of state of art deep neural networks for state formation. The agent receives joint angle velocities from actor-critic network in the real numbers continuous space. We show that our agent learns to achieve the target features from any random position in the image frame. Our results show that after training, the end effector reaches close to the target by slowly decreasing feature errors. We also show a comparison on how model trained for different initial position of UR5 end effector performs on test data. In the end, we present the testing results obtained on the real UR5 model.

1 Introduction and Motivation

Visual servoing is a classical problem in robotics that deals with guiding a robot to achieve a particular target image in vision. Current methods revolve around giving joint velocities based on error in image features. These features are hand crafted and are only able to achieve basic servoing. Also, there is need to design control law based on error in these features for different robots and different environments. In this paper, we present a different approach to solve this problem. We are using reinforcement learning to make a robot learn for itself to achieve this. It is a general framework and therefore it would work for different environments and different robots.Our work focuses on making a robot learn to move its end-effector near object based on camera image attached at its end effector.

1.1 Related Work

Continuous Control with DDPG: First paper to address problem of continuous action space solved using RL. It uses DDPG algorithm on various environments to show that it actually works. [1]

Towards Vision-Based Deep Reinforcement Learning for Robotic Motion Control: This paper uses DQN and a fixed camera to reach a object. But it only uses 2D plane and all training and testing on real robot is done in a 2d plane. [2]

Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large This paper uses somewhat similar approach like RL but not exactly RL. Also this uses fixed cameras, and output are not direct joint velocities. [3]

2 Methodology

2.1 Preliminaries

2.1.1 Reinforcement Learning

The essence of Reinforcement Learning(RL) is learning through interaction. An RL agent interacts with its environment by taking actions based on the present state and in return gets a reward. The end result is to maximize the numerical reward. The learner is not told which action to take, but instead discovers which action will yield the maximum reward. Reinforcement learning has found application in a wide array of robotic control tasks including target reaching and grasping. Our proposed method is the first method that outputs joint velocities from the actor network and uses it to control the robotic manipulator.

2.1.2 DDPG

The paper "Continuous control with deep reinforcement learning" [1] presented an actor-critic based on the deterministic policy gradient that can operate over continuous action spaces. We adopt this approach in our work to an actor-critic algorithm using deterministic policy gradient to learn the action value at each state of the environment. There are good reasons for choosing reinforcement learning and DDPG over conventional visual servoing techniques. Few of them are

Continuous Action Space: This algorithm is capable of handling continuous action space with ability to model policies as deterministic function instead of stochastic function.

Speed: Once the model is trained, given any random position, our end effector reaches close to the target.

Lower complexity: No need for interaction matrix, no problem of singularity in matrices

2.1.3 Deep Neural Networks

Deep-learning has been widely used to solve problems of artificial intelligence. With benefits from the improvement of computational hardware, multiple processing layers, deep convolution neural networks reached great success in solving complicated problems. In our proposed algorithm, we make use of pretrained state-of-art deep learning networks to extract relevant features from images and send it as state to our actor-critic network.

2.2 Reinforcement Learning Formulation

2.2.1 Environment

Our aim is to make the agent learn a policy which maps a sequence of images into a sequence of actions in the form of joint velocities that are applied to robot's joints. The agent used for the problem is UR5 having 6 degree of freedom. An RGB camera is mounted on the end effector to capture a 2D snapshot of 3D environment. We

simulate the 3D environment consisting of UR5 and the target object in Gazebo simulator.

2.2.2 State Space

The image captured by the camera after preprocessing forms our current Image. We stack two such feature vector, one from current image and other from previous one.These two feature vectors together form our state image of size 32X32X2.

2.2.3 Action Space

The action space in our environment are the angular velocities that are to be applied on the joints of the robot. The joint velocities are applied to the robot in Gazebo simulator. Since, UR5 consists of 6 degree of freedom, the action vector is of size 6.

2.2.4 Target Features

Since, we are doing visual servoing, we assume that we are given the target features. By target features we mean the values, which when achieved by the robotic manipulator, we can say that we have reached the target. In our problem, we have used the following as the target features

1. Blob size = 1200 ...(i)
2. Centroid of target = (41, 41) ...(ii)

So, whenever, the end effector achieves a blob size greater than or equal to (i) and distance between centroid of blob and centroid of target(ii) is less than a particular threshold, we say that we have reached in the vicinity of the target. We also, set the flag τ as 1 in this scenario. We use these target features for calculation of reward. The reward function will be discussed in a later section.

2.2.5 Reward Function

All the steps that bring the target object closer to the center of the frame in image plane are considered as favorable steps. So, for reward, we use the distance between the centroid of the target object and the center of the image frame. Smaller the distance, higher is the reward and vice-versa. Let the current distance between the centroid of the target object in image frame and the center of image frame be d_t. And, the distance between centroid of object and center of image frame is d_{t-1} for the previous state. So, we define our reward r_d as shown in eqn no. The *distconstant* is the maximum equation is used to normalize the reward.

$r_d = \{ \quad \frac{(d_{t-1}-d_t)}{distconstant}$

Also, intuitively we know that closer we move the target object, the bigger it appears. So, the area of target object should increase if we move closer to it. Hence, we also consider the area of the object captured in the image frame, higher the area higher is the reward. Ideal situation when the distance between the centroid of object and center of image plane is less than a threshold and area is also greater than a specified threshold, we give a strong positive reward(+1). Let area of blob in current image is $blobSize_t$ and area in previous image be $blobSize_{t-1}$. We define reward r_a in terms of area as shown in equation no. The *blobconstant* is a constant used to normalize

the value of reward.
$r_a = \{ \quad \frac{blobSize_t - blobSize_{t-1}}{blobconstant}$
We define another reward r_s on the basis of smoothness of actions taken by the agent. So, for each joint if it's acceleration is more than a particular threshold we assign r_s as -1 otherwise 0.
We assign a strong negative(-1) reward in cases where object is not in the frame. So using r_d, r_a and r_s, we define our final reward function as shown below:

$$R = \begin{cases} 1, & \text{if } \tau = 1, \\ -1, & \text{if } blobSize == 0, \\ w1 * r_d + w2 * r_a + w3 * r_s & \text{otherwise} \end{cases} \tag{1}$$

So for any image, we first check for target blob and if the target object is not in the image frame, we give a strong negative reward of -1. We define when distance between centroid and center of image less than a certain threshold and size of blob greater than another threshold, we say the end effector has reached the target. In this situation, we set the flag τ as 1. In all other situations, we give a weighted sum of r_d, r_a and r_s as the reward.

2.2.6 Episode

When the agent resets to intial position after reaching a terminal state, we say that the agent has completed one episode. In our environment, we end an episode in the following senarios:
i) When object goes of out of image frame.
ii) When end effector is in the vicinity of the target object and area of the blob is greater than the threshold.
In first case, the agent gets a strong negative reward(-1) while in the second case, it gets a strong positive reward(+1). In all other cases, we assign a weighted reward to the agent as discussed in the above section.

2.2.7 DDPG

We train our agent using Deep Deterministic Policy Gradient Algorithm. Each episode starts with an initial state(an image) where object is in field of view of the camera. Camera image is preprocessed and fed as input to actor-network which produces joint velocities. The robot executes these joint velocities and in return goes to next state. At this new state, the robot gets a reward. Based on reward, previous state and current state network parameters are updated so as to maximize total expected reward of an episode. An episode ends if the object gets out of field of view or end-effector reaches the vicinity of the target object.
In this RL formulation, the policy π is learned such that expected return from the start distribution is maximized.

The raw state information available to the agent is a vector of pixel values, denoted by s. At each training step, we store the experience tuple $e_t = (s_t, a_t, r_t, s_{t+1})$ in the replay memory D, where s_t is the present state, a_t is the current action, r_t is the observed reward, and 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 mini batches of size M sampled at random from the replay memory.

2.2.8 Exploration

In order to perform exploration efficiently over a continuous action space, we use the Ornstein-Uhlenbeck process whereby we add a noise sampled from a temporally correlated noise process N with inertia to the action values generated by the actor as follows:
$a_t = \mathrm{A}(\mathbf{s}_t|\boldsymbol{\omega}_A) + \mathrm{N}$

2.2.9 Policy Update

We start updating the weights only when number of iterations is greater than P. We update the weights of critic network and actor network using Adam stratergy. These updated weights are used to soft update the weights of target actor and target critic networks. The weights of the target network are then updated as follows:
$w_{\hat{A}} = \zeta w_A + (1-\zeta) w_{\hat{A}}$
$w_{\hat{C}} = \zeta w_C + (1-\zeta) w_{\hat{C}}$
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.

2.2.10 Environmental Setup

For our experiment we simulated a UR5 robot in Gazebo simulator using ROS Kinetic as the interface between our algorithm and the simulated robot. We choose Torch7 with Cuda8 as the deep learning library to construct and train our network. All the experiments are performed on an Ubuntu 16.04 system with 32 GB RAM, Intel Core-i7-4810MQ processor and NVIDIA GeForce 1080 GPU. We use four degrees of freedom of the manipulator for the reaching task.

2.2.11 RL Parameters

We design the actor and critic network as mentioned in [1]. The target actor and target critic network are initialised as clone of the actor network and critic network respectively. But as training begins, the weights of target networks are updated as explained in section. The value of learning rates and other parameters is shown in table 1.

2.2.12 CNN Parameters

The state vector obtained by preprocessing, which is of size 32X32X2, is sent as input to the actor network. We use the actor-critic network architecture employed in [1] and modify the final layer of the network 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.

Table 1: DDPG Network Parameters

Parameters	**Definition**	**Value**
P	No. of initial steps	1000
m	Minibatch size	100
ϵ	Exploitation probability	1 to 0.1
C	Target networks' update frequency	1
|M|	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
τ	Soft target update factor	0.10

3 Results and Discussion

Robot end-effector learned to reach the object. Our agent UR5 robot achieves following tasks

1. It is able to reach the target position.
2. The motion of our agent is smooth.
3. It can reach the target position from any random initial position.

Table 2 shows the results obtained after training our agent using the above discussed algorithm. The time taken to train was 48 hours. After training, the latest model was tested for 600 episodes. We found that our model successfully reached close to the target for all the 600 episodes. Also, here during training, we used the same reset position i.e. the robot always started from same initial position.

We also experimented by using the model trained for one fixed reset position and testing it for different reset positions. The success rate for different reset positions are shown in Table 3. The first row of the table shows accuracy obtained when we randomly select angle[0] of the UR5 in the range [-0.5, 0.5]. We decide this range from experiment to make sure that object remains in the camera frame. The results show that our trained model successfully reached the target with accuracy of 80% when tested on 600 test episodes. Similarly, we observed accuracy of 72 for angle[2]. The low performance of Angle[1] is dues to object being in corners of the image frame.

Parameters	Values
State Vector Size	32x32x2
Angular Velocity Range	[-1,1] (rad/sec)
Number of training steps	0.21 million
Training Time	48 hours
Number of Failures	0

Table 2: Testing Results

Angles	Range	Accuracy
Angle[0]	[-0.5,0.5]	80%
Angle[1]	[-1,0.5]	40%
Angle[2]	[1,3.5]	72%

Table 3: Model trained on one reset position and tested on random configuration

4 Conclusion and Future Work

Deep reinforcement learning shows a promising future in robotic control and manipulation. In this project we implemented DDPG algorithm to servo without obstacles. Since our control input was low level, therefore it is evident that such algorithms have capabilities of controlling robots at joint level, i.e. directly providing velocities or torque. Major limitations of our approach is not able to tackle uncertainty in visual environment, for example if we change the object we need to retrain it. One possible approach to overcome this limitation is to decouple vision and control problem in two separate policies instead of one combined policy.
As a part of our future work, we seek to investigate the performance of the agent in a more complex and uncertain environment. It is our eventual objective to establish the robustness of a trained agent in an environment with a significant amount of background clutter.

References

[1] 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] 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] L. Sergey, P. Peter, K. Alex, and Q. Deirdre. Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection. *arXiv preprint arXiv:1610.00633,* 2016.

[4] G. Shixiang, H. Ethan, L. Timothy, and L. Sergey. Deep reinforcement learning for robotic manipulation. *arXiv preprint arXiv:1603.02199,* 2016.

Student: Sudhir PratapYADAV

Mentor: Dr. Suril V. SHAH