Visual servoing using fitted Q-iteration

Ravi Pipaliya
ASU ID: 1217413275

Rakshith Vishwanatha
ASU ID: 1217756241

Mohit Mukul Bhagwat
ASU ID: 1217374990

***Abstract*—This work explores Visual-servoing of a two joint robotic manipulator using an approximate reinforcement learning method. We focus on target following using camera images with our target being a simple object confined to 2-dimensional space. Our goal is to explore fitted Q-iteration that can learn a visual servo of the target object in question. We use a relatively simple gradient-based servoing policy that minimizes the distance between the goal feature map and the one-step prediction. A key component of project is to setup an environment with a robotic manipulator in gazebo with a camera attached on the end-effector. We also learn a bilinear model for the arm to predict the next state dynamics of the object position in the image output.**

***Index Terms*—Visual-servoing, Reinforcement Learning, Fitted Q-Iteration, Robotic Manipulator**

I. Introduction

Visual-servoing is a technique which uses visual feedback to provide appropriate control commands to reach a goal configuration in the world.

In this project, we implement the visual-servoing task for a 2 degrees of freedom, Robotic arm [5] well-defined in the Gazebo world. The visual system can observe the defined object (goal) with the camera attached to the end-effector of the arm (eye-in-hand configuration). OpenCV [4] library was used to identify the object and locate its position in the camera frame. We then learn to servo the arm such that the difference between one step predicted position and the target position of object is minimized. To do so we fit a bilinear model similar to one used in [1] to get a one-step dynamics of the object position.

We plan to implement approximate reinforcement learning based control approach to achieve the assigned visual servoing task of tracking the object in 2-dimensional space. The objective of our control task would be to align the object in the world to coincide with the center of our camera frame, that is the Robotic arm would look at the object head-on.

Under the umbrella of Reinforcement learning we will use the fitted Q-iteration method, inspired by [1] where they have used the approach to control the quad-copter to track the vehicles within the simulated city environment. Q-iteration is quite a popular approach that learns the action-value function in an exact sense for a given problem. Thus is it suitable for problems with discrete states and actions. In our case we have represent our problem with continuous states and inputs and thus we use fitted Q-iteration. This approach is an approximate reinforcement learning approach as it will try to learn the Q-function based on the approximator used.

II. Methods

A. PROBLEM STATEMENT

Considering the Robotic arm defined in [5], we plan to control it such that the arm always points in the direction such that object placed in the center of camera frame. The below problem statement was completely motivated from [1]. Let y_t be a featurization of the camera's observation x_t and let y_* be some goal feature map. In this project we define visual servoing as the problem of choosing controls u_t for a fixed number of discrete time steps t as to minimize the error $||y_* - y_t||$.

We use a relatively simple gradient-based servoing policy that uses one-step feature dynamics, $f : \{y_t, u_t\} \rightarrow y_{t+1}$. The policy chooses the control that minimizes the distance between the goal feature map and the one-step prediction:

$$\pi(x_t, x_*) = arg \min_u ||y_* - f(y_t, u)||^2 \tag{1}$$

Learning this policy amounts to learning the robot dynamics and the distance metric $||.||$.

To learn the robot dynamics, we assume that we have access to a dataset of paired observations and controls x_t,u_t,x_{t+1} . This data is relatively easy to obtain as it involves collecting a stream of the robot's observations and controls.

To learn the distance metric, we assume that the robot interacts with the world and collects tuples of the form x_t,u_t,c_t, x_{t+1},x_*. At every time step during learning, the robot observes x_t and takes action u_t . After the transition, the robot observes x_t+1 and receives an immediate cost c_t . This cost is task-specific and it quantifies how good that transition was in order to achieve the goal. At the beginning of each trajectory, the robot is given a goal observation x_* , and it is the same throughout the trajectory. We define the goal feature map to be the featurization of the goal observation. We learn the distance metric using reinforcement learning and we model the environment as a Markov Decision Process (MDP). The state of the MDP is the tuple of the current observation and the episode's target observation, $s_t = (x_t, x_)$, the action u_t is the discrete-time continuous control of the robot, and the cost function maps the states and action $(s_t, u_t, s_t + 1)$ to a scalar cost c_t .

B. VISUAL FEATURE DYNAMICS

We use feature extraction process to detect object position y_t in the camera image x_t. Here x_t is 480 X 640 pixel image obtained from the camera mounted at the end of the manipulator. We define the feature vector in our case as the position of object in pixel co-ordinate. We define our pixel coordinates as (row,column). Our uber objective thus will be to maintain the position of the object in pixel coordinates as close as possible to (240,320) pixel.

$$y_t = \begin{bmatrix} row\ pixel \\ column\ pixel \end{bmatrix}$$

In case the feature position is not at the desired center location, we try to control the manipulator bring it to the center. The inputs for the robotic manipulator are the change in joint co-ordinates input to the arm. Let q_1 , q_2 be the joint position of the manipulator. We constrain the workspace of the manipulator by confining its joint coordinates such that camera can view the 2-D space in front of the manipulator where the object is placed. We define the actions as change in the joint position of the two joints on the manipulator.

$$u = \begin{bmatrix} \Delta q_1 \\ \Delta q_2 \end{bmatrix}$$

Based on the control objective defined in the problem statement we need to acquire a model to predict the one-step dynamics of the position of the object. Using the position co-ordinates and a given input action we need to predict the next position of the object in the frame using a simple bilinear model as described in [1].

C. LEARNING VISUAL SERVOING WITH REINFORCEMENT LEARNING

We use reinforcement learning to find the optimal control input with an objective of reducing the one-step difference between the predicted next step position of object and target position as well as penalize for high control inputs. We correspond this objective with state-action value function Q for learning the weights W and λ.

$$\pi(x_t, x_*) = arg\min_u W||y_* - f(y_t, u)||_2^2 + \sum_j \lambda_j u_j^2 \quad (2)$$

The objective above represents cost that we want to minimize. In Reinforcement learning literature usually Q represents expected reward. We can transform between reward and cost objective by changing max to min and multiplying with negative sign. We choose a Q-value function approximator that can represent the servoing objective such that the greedy policy with respect to the Q-values results in the policy of (2). In particular, we use a function approximator that is linear in the weight parameters $\theta^T = [W\ \lambda]$:

$$Q_{\theta,b}(s_t, u) = \phi(s_t, u)^T\theta + b \quad (3)$$

$$\phi(s_t, u)^T = \begin{bmatrix} ||y_* - f(y_t, u)||_2^2 & [u]^2 \end{bmatrix} \quad (4)$$

We denote the state of the MDP as $s_t = (x_t, x^*)$ and add a bias b to the Q-function. The servoing policy then simply $\pi_\theta(s_t) = arg\min_u Q_{\theta,b}(s_t, u)$. For reinforcement learning, we optimize for the weights θ but keep the feature representation and its dynamics fixed.

D. Fitted Q-iteration

Below we include a general procedure to do fitted Q-iteration as stated in [2]. Here Q represents expected reward value. In fitted Q-iteration, the agent iteratively gathers a dataset $S = (x_j, u_j, x_j^{'}, r_j)|j = 1, \ldots, S$, where $x_j^{'}$ is next state and r_j is the corresponding reward. At iteration l, when the parameters are θ_l , fitted Q-iteration computes the Bellman target $q_{l+1,j} = r_j + \gamma \max_{u' Q'(x_j', u'; \theta_l)}$ for each transition sample. Then, least-squares regression is run on the input-output samples $(x_j, u_j) \rightarrow q_{l+1,j}$ to obtain the next parameters θ_{l+1}. Algorithm below summarizes the procedure.

```
Algorithm   Fitted Q-iteration.
Input: $\gamma$, dataset $\mathcal{S}$
1: initialize parameter vector, e.g. $\theta_0 \leftarrow 0$
2: repeat at every iteration $\ell = 0, 1, 2, \ldots$
3:     $q_{\ell+1,j} = r_j + \gamma \max_{u'} \hat{Q}(x_j', u'; \theta_\ell)$, for $j = 1, \ldots, S$
4:     $\theta_{\ell+1} = \arg\min_\theta \sum_{j=1}^{S} \left[q_{\ell+1,j} - \hat{Q}(x_j, u_j; \theta)\right]^2$
5: until $\theta_{\ell+1}$ is satisfactory
Output: $\hat{Q}^*$, $\hat{\pi}^*$ greedy in $\hat{Q}^*$ (implicitly represented via $\hat{Q}^*$)
```

We assume the Q-function approximator is such that greedy actions can be efficiently found and use this fact to sidestep the requirement of representing policies. We find greedy actions on demand, at any state where they are needed. The usual way to achieve this is to discretize the actions into a few values, and then maximize by enumeration as follows,

$$\pi^*(x) \in arg\min_u Q^*(x, u) \quad (5)$$

The u in above equation are a discretized actions amongst which the best action is selected at the state.

E. Alternate approach

If necessary, an alternate approach that can be tried is to discretize the camera image into different states and apply a grid-based Markov process to achieve the necessary visual servoing task. This approach would be similar to the example discussed in class with respect to how actions and policies are obtained for a grid world problem. Here the goal would be to have the box moved to the central goal state with discrete robot arm actions. Q-iteration and policy iteration are techniques that can be explored.

III. Implementation and simulation

A robotic arm in a gazebo environment is being used to simulate, develop and test the reinforcement learning model. The URDF files for the robotic arm in gazebo were taken from [5]. The two-link robotic manipulator has two degree of freedom q_1 and q_2 in joint space. ROS is being used to actuate and sense within the gazebo environment. A camera is attached to the end-effector which publishes images of the environment. The object to be tracked is placed such that it is located within the camera's field of view. The object is then moved to different locations, and based on the learnt policy the manipulator should change its configuration to track the object and center it within the camera image. Fig. 1 shows the simulated environment in gazebo with manipulator and the object.

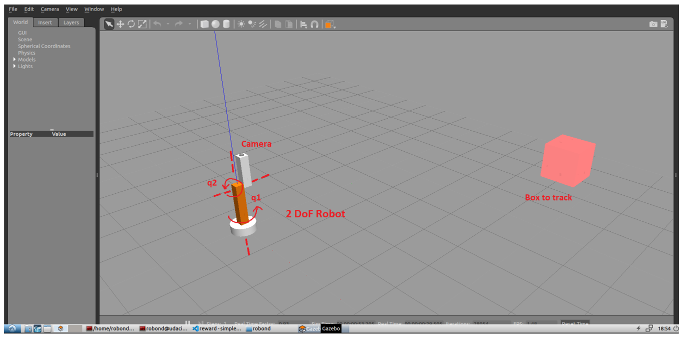


Fig. 1. Gazebo Environment with Robot and Box

IV. Results

The current progress and the pending steps are mentioned below.

Progress:

- ROS nodes to change the configuration of the joints, use the end-effector camera and process the camera images have been written. The diagram below illustrates the nodes and topics being used currently in the ROS-Gazebo environment.

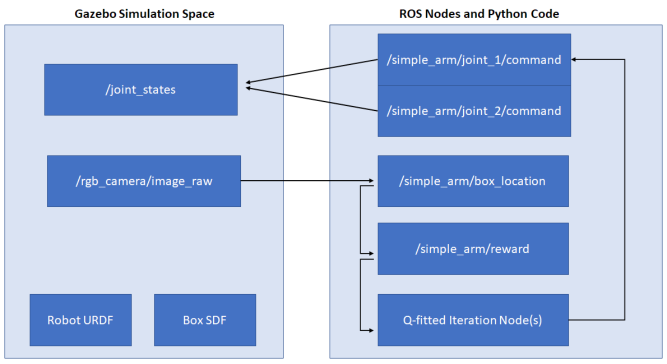


Fig. 2. ROS-Gazebo Overview

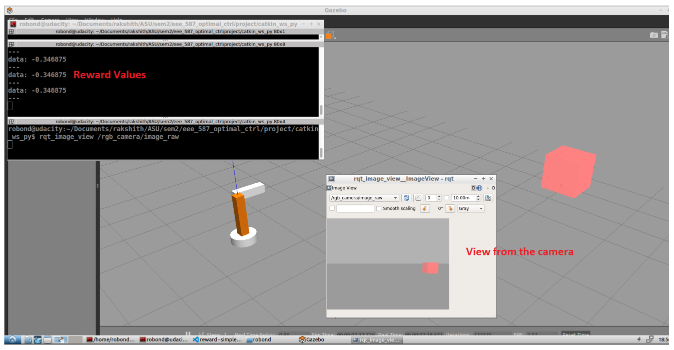
- As per the above figure, /joint_states node and /simple_arm/joint_1/command topics are used to control the joints of the robot. Also, using the ros parameters node, desired (q_{min}, q_{max}) restrictions can be placed on the joints.
- The /rgb_camera/image_raw topic and /simple_arm/box_location node are used to obtain camera images and located the object's position within the camera frame. To detect the location of the box to be servoed, the box has been coloured in bright red. This allows for a simple color channel based feature detection. Using OpenCV [4], the red channel is extracted and a value thresholding is performed. A location average is computed on the pixels that surpass the threshold to reliably obtain the location of the object.
- Now, it is required to provide a mechanism by which the environment can give the robot rewards based on the actions it performs. Since the goal of the robot is to move its end effector such that the box is placed at the center of its camera image, a reward function that utilizes the how far away the box is from the center of the image is used. The /simple_arm/reward node subscribes to the box location data being published by /simple_arm/box_location node and uses the euclidean distance metric to compute a reward/cost for the robot.

$$Reward = -\sqrt{(box_x - center_x)^2 + (box_y - center_y)^2} \quad (6)$$

- When the box is withing a 20x20 pixel region around the the center of the image, the goal state is said to be reached and a positive reward of $+1$ is given.

$$Reward_{goal} = +1 \quad (7)$$

The images below shows two cases in which the robot might see the box in the camera. The first is where the box is within the camera frame but not centered within the camera. Notice here the reward is -0.34.

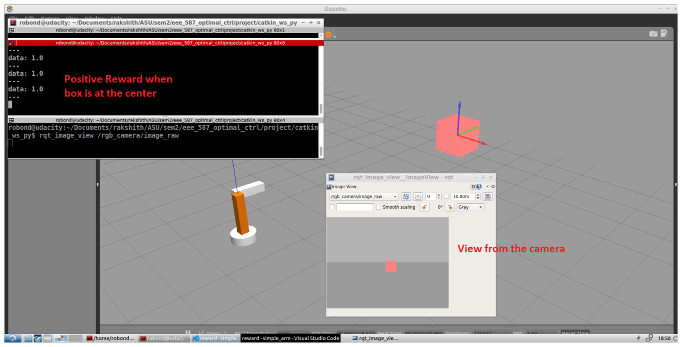


```
robond@udacity: ~/Documents/rakshith/ASU/sem2/eee_587_optimal_ctrl/project/catkin_ws_py
---
data: -0.346875
---
data: -0.346875
---
data: -0.346875
---
```

```
robond@udacity:~/Documents/rakshith/ASU/sem2/eee_587_optimal_ctrl/project/catkin_ws_py$ rqt_image_view /rgb_camera/image_raw
```

Fig. 3. Penalty or Negative Reward for Off-centered Box.

The second scenario is where the box is identified to be at the center of the image. Here the robot receives a reward of $+1$.



```
robond@udacity: ~/Documents/rakshith/ASU/sem2/eee_587_optimal_ctrl/project/catkin_ws_py 80x8
---
data: 1.0
---
data: 1.0
---
data: 1.0
---
```

```
robond@udacity: ~/Documents/rakshith/ASU/sem2/eee_587_optimal_ctrl/project/catkin_ws_py 80x4
robond@udacity:~/Documents/rakshith/ASU/sem2/eee_587_optimal_ctrl/project/catkin
_ws_py$ rqt_image_view /rgb_camera/image_raw
```

Fig. 4. Positive Reward for Centered Box.

Another condition that could occur is of the box not being present in the camera frame altogether. In this case, when no features of the box are detected, the robot receives a large penalty of -999 for having lost track of the box entirely.

Now, we go over the next steps of implementation

Next Steps:

- Generating data based on the set up environment and learning the dynamics model
- Implementing q-fitted iteration algorithm to learn a optimal policy
- If we get successful results for tracking a static object we will try to learn policy for a complex object trajectories

V. Discussions and Conclusions

Q fitted iteration was understood and a plan on how the continuous state space will be represented and learnt has been mapped as given in previous sections. There is now a clear understanding on the implementation of the Q-fitted iteration.

An environment to support the development and training of a reinforcement learning algorithm is now complete, as it allows us to interact with the gazebo world to perform actions and receive rewards based on the result of the action performed.

Data is being collected, to perform the fitted Q-iteration algorithm, which then needs to be implemented to learn a policy to track the object.

References

[1] Alex X. Lee, Sergey Levine, Pieter Abbeel (2017) "Learning Visual Servoing with Deep Features and Fitted Q-Iteration" In the International Conference on Learning Representations (ICLR).
[2] Busoniu, Lucian de Bruin, Tim Tolić, Domagoj Kober, Jens Palunko, Ivana. (2018). Reinforcement learning for control: Performance, stability, and deep approximators. Annual Reviews in Control. 10.1016/j.arcontrol.2018.09.005.
[3] R. S. Sutton and A. G. Barto. 2018. Reinforcement Learning: An Introduction. The MIT Press.
[4] OpenCV, 2015. Open Source Computer Vision Library,
[5] Udacity, Simple Robotic Arm: https://github.com/udacity/simplearm
[6] Robot Operating System [ROS] Kinetic, Gazebo