

# LEARNING VISUAL SERVOING WITH DEEP FEATURES AND TRUST REGION FITTED Q-ITERATION

Alex X. Lee<sup>†</sup>, Sergey Levine<sup>†</sup>, Pieter Abbeel<sup>†‡</sup>

<sup>†</sup> UC Berkeley, Department of Electrical Engineering and Computer Science

<sup>‡</sup> OpenAI

{alexlee\_gk, svlevine, pabbeel}@cs.berkeley.edu

## ABSTRACT

Visual servoing involves choosing actions that move a robot in response to observations from a camera, in order to reach a goal configuration in the world. The visual servoing problem comes up in robotic tasks that range from target following for an aerial vehicle to robotic manipulation to spacecraft docking. Standard visual servoing approaches typically rely on manually designed features and analytical dynamics models, which limits their generalization capability and often requires extensive application-specific feature and model engineering. In this work, we study how learned visual features, learned predictive dynamics models, and reinforcement learning can be combined to learn visual servoing mechanisms. We focus on target following, with the goal of designing algorithms that can learn a visual servo using low amounts of data of the target in question, so as to be easy and quick to adapt to new targets. Our approach is based on servoing the camera in the space of learned visual features, rather than image pixels or manually-designed keypoints. We demonstrate that standard deep features, in our case taken from a model trained for object classification, can be used together with a bilinear predictive model to learn an effective visual servo that is robust to visual variation, changes in viewing angle and appearance, and occlusions. A key component of our approach is to use reinforcement learning to learn which features are best suited for the task at hand. We show that our method can learn an effective visual servo using just tens of trajectories. We evaluate our method on a complex synthetic car following benchmark, and demonstrate substantial improvement over a conventional pixel-based approach.

## 1 INTRODUCTION

Visual servoing is a classic problem in robotics that requires moving a camera or robot to match a target configuration of visual features or image intensities. Many robot control tasks that combine perception and action can be posed as visual servoing, including navigation (DeSouza & Kak, 2002; Chen et al., 2006), where a robot must follow a desired path; manipulation, where the robot must servo an end-effector or a camera to a target object to grasp or manipulate it (Malis et al., 1999; Corke, 1993; Hashimoto, 1993; Hosoda & Asada, 1994; Kragic et al., 2002); and various other problems, as surveyed in (Hutchinson et al., 1996). Most (but not all) visual servoing methods assume access to good geometric image features (Chaumette & Hutchinson, 2006; Collewet et al., 2008; Caron et al., 2013) and require knowledge of their dynamics, which are typically obtained from domain knowledge about the system. Using such hand-designed features and models prevents the method from exploiting statistical regularities in the world, and requires manual engineering for each new system.

In this work, we study how learned visual features, learned predictive dynamics models, and reinforcement learning can be combined to learn visual servoing mechanisms. We focus on target following, with the goal of designing algorithms that can learn a visual servo using low amounts of data of the target in question, so as to be easy and quick to adapt to new targets. Successful target following requires the visual servo to tolerate moderate variation in the appearance of the target, including changes in viewpoint and lighting, as well as occlusions. Learning invariances to all such

distractors typically requires a considerable amount of data. However, since a visual servo is typically specific to a particular task, it is very desirable to be able to learn the servoing mechanism very quickly, using a minimum amount of data. Prior work has shown that the features learned by large convolutional neural networks on large image datasets, such as ImageNet classification (Deng et al., 2009), tend to be useful for a wide range of other visual tasks (Donahue et al., 2014). Does the usefulness of such features extend to visual servoing?

To answer this question, we propose a visual servoing method that uses pre-trained features, in our case obtained from the VGG network (Simonyan & Zisserman, 2014) trained for ImageNet classification. Besides the visual features, our method uses an estimate of the feature dynamics in visual space by means of a bilinear model (Censi & Murray, 2015). This allows the visual servo to predict how motion of the robot’s camera will affect the perceived feature values. Unfortunately, servoing directly on the high-dimensional features of a pre-trained network is insufficient by itself to impart robustness on the servo: the visual servo must not only be robust to moderate visual variation, but it must also be able to pick out the target of interest (such as a car that the robot is tasked with following) from irrelevant distractor objects. To that end, we propose a sample-efficient trust region fitted Q-iteration procedure that automatically chooses weights for the most relevant visual features. Crucially, the actual servoing mechanism in our approach is extremely simple, and simply seeks to minimize the Euclidean distance between the weighted feature values at the next time step and the target. We show that our method can learn an effective visual servo using just tens of trajectories, and evaluate our method on a complex synthetic car following benchmark, and demonstrate substantial improvement over a conventional pixel-based approach.

## 2 RELATED WORK

Visual servoing is typically (but not always) performed with calibrated cameras and carefully designed visual features. Ideal features for servoing should be stable and discriminative, and much of the work on visual servoing focuses on designing stable and convergent controllers under the assumption that such features are available (Espiau et al., 1992; Mohta et al., 2014; Wilson et al., 1996). Some visual servoing methods do not require camera calibration (Jägersand et al., 1997; Yoshimi & Allen, 1994), and some recent methods operate directly on image intensities (Caron et al., 2013), but generally do not use learning to exploit statistical regularities in the world and improve robustness to distractors.

Learning is a relative recent addition to the repertoire of visual servoing tools. Several methods have been proposed that apply ideas from reinforcement learning to directly acquire visual servoing controllers (Lampe & Riedmiller, 2013; Sadeghzadeh et al., 2015). However, such methods have not been demonstrated under extensive visual variation, and do not make use of state-of-the-art convolutional neural network visual features. Though more standard deep reinforcement learning methods (Lange et al., 2012; Mnih et al., 2013; Levine et al., 2015; Lillicrap et al., 2015) could in principle be applied directly learn visual servoing policies, such methods tend to require large numbers of samples to learn task-specific behaviors, making them poorly suited for a flexible visual servoing algorithm that can be quickly repurposed to new tasks (e.g. to following a different object).

Instead, we propose an approach that combines learning of predictive models with pre-trained visual features. We use visual features trained for ImageNet (Deng et al., 2009) classification, though any pre-trained features could in principle be applicable for our method, so long as they provide a suitable degree of invariance to visual distractors such as lighting, occlusion, and changes in view-point. Using pre-trained features allows us to avoid the need for large amounts of experience, but we must still learn the policy itself. To further accelerate this process, we first acquire a predictive model that allows the visual servo to determine how the visual features will change in response to an action. General video prediction is an active research area, with a number of complex but data-hungry models proposed in recent years (Oh et al., 2015; Watter et al., 2015; Mathieu et al., 2015; Xue et al., 2016; Lotter et al., 2016; Brabandere et al., 2016; Walker et al., 2016; Vondrick et al., 2016). However, we observe that convolutional response maps can be interpreted as images and, under mild assumptions, the dynamics of image pixels during camera motion can be well approximated by means of a bilinear model (Censi & Murray, 2015). We therefore train a relatively simple bilinear model for short-term prediction of visual feature dynamics, which we can use inside a very simple visual servo that seeks to minimize the error between the next predicted feature values and a target image.

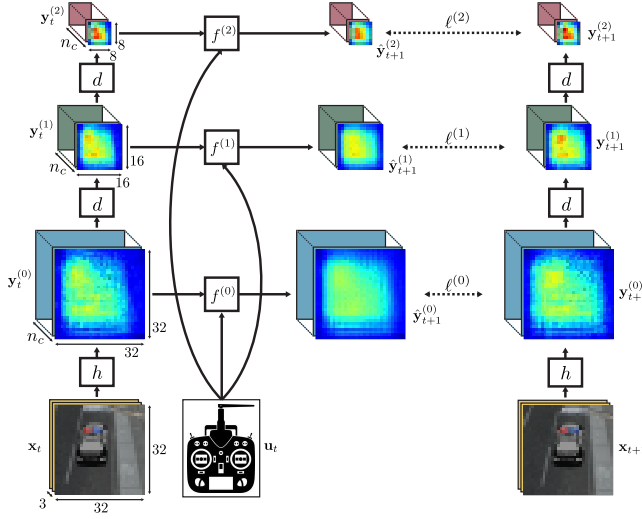


Figure 1: Multiscale bilinear model. The function  $h$  maps images  $\mathbf{x}$  to feature maps  $\mathbf{y}^{(0)}$ , the operator  $d$  downsamples the feature maps  $\mathbf{y}^{(l-1)}$  to  $\mathbf{y}^{(l)}$ , and the bilinear function  $f^{(l)}$  predicts the next feature  $\hat{\mathbf{y}}^{(l)}$ . The number of feature map channels is  $n_c$ , which is the same for all scales  $l$ .

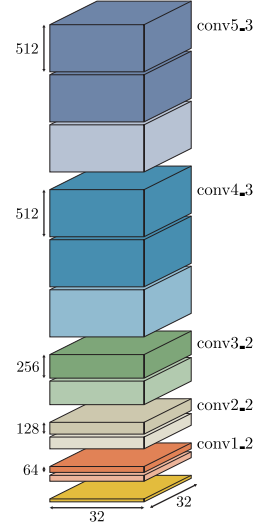


Figure 2: Dilated VGG-16 network. The outputs of dilated convolutions are shown in a lighter shade.

Unfortunately, simply training predictive models on top of pre-trained features is insufficient to produce an effective visual servo, since it weights the errors of distractor objects the same amount as the object of interest. We address this challenge by using an efficient trust-region Q-iteration algorithm to train the weights on the features to maximize the servo’s long-horizon reward. This method draws on ideas from fitted Q-iteration (Ernst et al., 2005), neural fitted Q-iteration (Riedmiller, 2005), and trust region policy optimization (Schulman et al., 2015) to develop a sample-efficient algorithm that can directly estimate the expected return of the visual servo without the use of any additional function approximator.

### 3 PROBLEM STATEMENT

Let  $\mathbf{y}_t$  be a featurization of the camera’s observations  $\mathbf{x}_t$  and let  $\mathbf{y}_*$  be some given goal feature map. For the purposes of this work, we define *visual servoing* as the problem of choosing controls  $\mathbf{u}_t$  for a fixed number of discrete time steps  $t$  as to minimize the error  $\|\mathbf{y}_* - \mathbf{y}_t\|$ .

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

$$\pi(\mathbf{x}_t, \mathbf{x}_*) = \arg \min_{\mathbf{u}} \|\mathbf{y}_* - f(\mathbf{y}_t, \mathbf{u})\|^2. \quad (1)$$

Learning this policy amounts to learning the robot dynamics and the distance metric  $\|\cdot\|$ . To learn the robot dynamics, we assume that we have access to a dataset of paired observations and controls  $\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1}$ . This data is relatively easy to obtain as it involves collecting a stream of the robot’s observations and controls. We use this dataset to learn a general visual dynamics model that can be used for any task. To learn the distance metric, we assume that the robot interacts with the world and collects tuples of the form  $\mathbf{x}_t, \mathbf{u}_t, c_t, \mathbf{x}_{t+1}, \mathbf{x}_*$ . At every time step during learning, the robot observes  $\mathbf{x}_t$  and takes action  $\mathbf{u}_t$ . After the transition, the robot observes  $\mathbf{x}_{t+1}$  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. 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,  $s_t = (\mathbf{x}_t, \mathbf{x}_*)$ , the action  $\mathbf{u}_t$  is the discrete-time control of the robot, and the cost function maps the states and action  $(s_t, \mathbf{u}_t, s_{t+1})$  to a scalar cost.

## 4 VISUAL FEATURES DYNAMICS

We learn a multiscale bilinear model to predict the visual features of the next frame given the current image from the robot’s camera and the action of the robot. An overview of the model is shown in Figure 1. The learned dynamics can then be used for visual servoing as described in Section 5.

### 4.1 VISUAL FEATURES

We consider both pixels and semantic features for the visual representation. We define the function  $h$  to relate the image  $\mathbf{x}$  and its feature  $\mathbf{y} = h(\mathbf{x})$ . Our choice of semantic features are derived from the VGG-16 network (Simonyan & Zisserman, 2014), which is a convolutional neural network trained for large-scale image recognition on the ImageNet dataset (Deng et al., 2009). Since spatial invariance is undesirable for servoing, we remove the max-pooling layers and replace the convolutions that preceded them with dilated convolutions, as done by Yu & Koltun (2015). The modified VGG network is shown in Figure 2. We use the model weights of the original VGG-16 network, which are publicly available as a Caffe model (Jia et al., 2014). We standardize the outputs of the intermediate convolutional layers and use them as our features.

We use multiple resolutions of the features for servoing. The idea is that the high-resolution representations have detailed local information about the scene, while the low-resolution representations have more global information available through the image-space gradients. The features at level  $l$  of the multiscale pyramid are denoted as  $\mathbf{y}^{(l)}$ . The features at each level are obtained from the features below through a downsampling operator  $d(\mathbf{y}^{(l-1)}) = \mathbf{y}^{(l)}$  that cuts the resolution in half.

### 4.2 BILINEAR DYNAMICS

The features  $\mathbf{y}_t^{(l)}$  are used to predict the corresponding level’s features  $\mathbf{y}_{t+1}^{(l)}$  at the next time step, conditioned on the action  $\mathbf{u}_t$ , according to a prediction function  $f^{(l)}(\mathbf{y}_t^{(l)}, \mathbf{u}_t) = \hat{\mathbf{y}}_{t+1}^{(l)}$ . We use a bilinear model to represent these dynamics, motivated by prior work (Censi & Murray, 2015). In order to servo at different scales, we learn a bilinear dynamics model at *each* scale. We consider two variants of the bilinear model in previous work in order to reduce the number of model parameters.

#### 4.2.1 CHANNELWISE FULLY CONNECTED BILINEAR DYNAMICS

To limit the number of parameters, we model the dynamics of each channel independently. When semantic features are used, this model interprets the feature maps as being abstract images with spatial information within a channel and different entities or factors of variation across different channels. This could potentially allow the model to handle moving objects, occlusions, and other complex phenomena. The dynamics of each channel are given by a *fully connected* bilinear model:

$$\hat{\mathbf{y}}_{t+1,c}^{(l)} = \mathbf{y}_{t,c}^{(l)} + \left( \mathbf{M}_c^{(l)} \cdot \mathbf{u}_t \right) \mathbf{y}_{t,c}^{(l)} + \mathbf{N}_c^{(l)} \mathbf{u}_t + \mathbf{M}_{c,0}^{(l)} \mathbf{y}_{t,c}^{(l)} + \mathbf{N}_{c,0}^{(l)}, \quad (2)$$

The  $\cdot$  denotes the tensor dot product. The tensor  $\mathbf{M}_c^{(l)}$ , matrices  $\mathbf{N}_c^{(l)}$  and  $\mathbf{M}_{c,0}^{(l)}$ , and vector  $\mathbf{N}_{c,0}^{(l)}$  are the parameters of the dynamics of the  $c$ -th channel of the features at the  $l$ -th level. The last two terms are biases that allow to model action-independent visual changes, such as moving objects.

#### 4.2.2 CHANNELWISE LOCALLY CONNECTED BILINEAR DYNAMICS

The fully connected bilinear model is quite large, so we propose a bilinear dynamics that enforces sparsity in the parameters. In particular, we constrain the prediction to depend only on the features that are in its local spatial neighborhood, leading to the following *locally connected* bilinear model:

$$\hat{\mathbf{y}}_{t+1,c}^{(l)} = \mathbf{y}_{t,c}^{(l)} + \sum_j \left( \mathbf{W}_{c,j}^{(l)} * \mathbf{y}_{t,c}^{(l)} + \mathbf{B}_{c,j}^{(l)} \right) \mathbf{u}_{t,j} + \left( \mathbf{W}_{c,0}^{(l)} * \mathbf{y}_{t,c}^{(l)} + \mathbf{B}_{c,0}^{(l)} \right). \quad (3)$$

The parameters are the 4D tensor  $\mathbf{W}_{c,j}^{(l)}$  and the matrix  $\mathbf{B}_{c,j}^{(l)}$  for each  $c, l$  and control coordinate  $j$ . The  $*$  is the locally connected operator, which is like a convolution but with untied filter weights:

$$(\mathbf{W} * \mathbf{y})_{k_h, k_w} = \sum_{i_h=k_h-\lfloor n_f/2 \rfloor}^{k_h+\lfloor n_f/2 \rfloor} \sum_{i_w=k_w-\lfloor n_f/2 \rfloor}^{k_w+\lfloor n_f/2 \rfloor} \mathbf{W}_{k_h, k_w, i_h-k_h, i_w-k_w} \mathbf{y}_{i_h, i_w}.$$

The size of the local neighborhood is  $n_f \times n_f$ , which is analogous to the filter size in convolutions.

We could have reduced the number of parameters even more by using convolutions instead of locally connected matrix multiplies. However, we found that convolutions weren't expressive enough since the translational invariance assumption in image space doesn't hold for arbitrary camera movements.

#### 4.3 TRAINING VISUAL FEATURE DYNAMICS MODELS

The loss that we use for training the bilinear dynamics is the sum of the losses of the predicted features at each level,  $\sum_{l=0}^L \ell^{(l)}$ , where the loss for each level  $l$  is the squared  $\ell$ -2 norm between the predicted features and the actual features of that level,  $\ell^{(l)} = \|\mathbf{y}_{t+1}^{(l)} - \hat{\mathbf{y}}_{t+1}^{(l)}\|^2$ .

Since the feature representation are not learned, the weights can be analytically solved using ridge regression, by solving a linear system of equations independently for each channel and scale. The fully connected dynamics of Equation (2) was learned in this way. The locally connected dynamics of Equation (3), however, was trained using ADAM (Kingma & Ba, 2014).

### 5 LEARNING VISUAL SERVOING WITH REINFORCEMENT LEARNING

We propose to use a multiscale representation of semantic features for servoing. The challenge when introducing multiple scales and multi-channel feature maps for servoing is that the features don't necessarily agree on the optimal action when the goal is unattainable or the robot is far away from the goal. To do well, it's important to use a good weighing of each of the terms in the objective. Since there are many weights, it would be impractically time-consuming to set them by hand, so we resort to learning. We want the weighted one-step lookahead objective to encourage good long-term behavior, so we want this objective to correspond to the state-action value function  $Q$ . So we propose a method for learning the weights based on fitted Q-iteration.

#### 5.1 SERVOING WITH WEIGHTED MULTISCALE FEATURES

Instead of attempting to build an accurate predictive model for multi-step planning, we use the simple greedy servoing method in Equation (1), where we minimize the error between the target and predicted features but for *all* the scales. Typically, only a few objects in the scene are relevant, so the errors of some channels should be penalized more than others. Similarly, features at different scales might need to be weighted differently. So, we use a weighting  $\mathbf{w}_{c,l} \geq 0$  per channel  $c$  and scale  $l$ :

$$\pi(\mathbf{x}_t, \mathbf{x}_*) = \arg \min_{\mathbf{u}} \sum_c \sum_{l=0}^L \frac{\mathbf{w}_c^{(l)}}{|\mathbf{y}_{*,c}^{(l)}|} \left\| \mathbf{y}_{*,c}^{(l)} - f_c^{(l)}(\mathbf{y}_{t,c}^{(l)}, \mathbf{u}) \right\|_2^2 + \sum_j \lambda_j \mathbf{u}_j^2, \quad (4)$$

where  $|\cdot|$  denotes the cardinality operator and the constant  $1/|\mathbf{y}_{*,c}^{(l)}|$  normalizes the feature errors by the corresponding resolution. We also use a separate weight  $\lambda_j$  for each control coordinate  $j$ . This optimization can be solved efficiently since the dynamics is linear in the controls (see Appendix A).

#### 5.2 Q-FUNCTION APPROXIMATION FOR THE WEIGHTED SERVOING POLICY

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 Equation (4). In particular, we use a function approximator that is linear in the weight parameters  $\boldsymbol{\theta}^\top = [\mathbf{w}^\top \quad \boldsymbol{\lambda}^\top]$ :

$$Q_{\boldsymbol{\theta},b}(s_t, \mathbf{u}) = \phi(s_t, \mathbf{u})^\top \boldsymbol{\theta} + b, \quad \phi(s_t, \mathbf{u})^\top = \left[ \left[ \frac{1}{|\mathbf{y}_{*,c}^{(l)}|} \left\| \mathbf{y}_{*,c}^{(l)} - f_c^{(l)}(\mathbf{y}_{t,c}^{(l)}, \mathbf{u}) \right\|_2^2 \right]_{c,l}^\top \quad [\mathbf{u}_j^2]_j^\top \right].$$

We denote the state of the MDP as  $s_t = (\mathbf{x}_t, \mathbf{x}_*)$  and add a bias  $b$  to the Q-function. The servoing policy is then simply  $\pi_{\boldsymbol{\theta}}(s_t) = \arg \min_{\mathbf{u}} Q_{\boldsymbol{\theta},b}(s_t, \mathbf{u})$ .

For reinforcement learning, we optimized for the weights  $\boldsymbol{\theta}$  but kept the representation and dynamics fixed. It wasn't necessary to train the Q-network end-to-end, although doing so could lead to better servoing performance.

### 5.3 LEARNING THE Q-FUNCTION WITH TRUST-REGION FITTED Q-ITERATION

Reinforcement learning methods that learn a Q-function do so by minimizing the Bellman error:

$$\left\| Q(s_t, \mathbf{u}_t) - \left( c_t + \gamma \min_{\mathbf{u}} Q(s_{t+1}, \mathbf{u}) \right) \right\|_2^2. \quad (5)$$

The agent learns the Q-function by interacting in the world and getting samples of the current state  $s_t$  and action  $\mathbf{u}_t$ , the state  $s_{t+1}$  after taking that action, and the cost  $c_t$  incurred for transitioning from  $s_t$  to  $s_{t+1}$  when taking action  $\mathbf{u}_t$ .

It is typically hard or unstable to optimize for both Q-functions that appear in the objective of Equation (5), so it's usually optimized by iteratively optimizing the current Q-function while keeping the target Q-function constant. However, we notice that the minimum over Q-values is the same for any non-negative scaling of  $\theta$  and for any bias  $b$ , regardless of the optimal action. So, we could easily optimize for the non-negative scaling  $\alpha$  and the bias  $b$  if the direction of  $\theta$  is held fixed. Therefore, to speed up the optimization of the Q-function, we first jointly solve for  $\alpha$  and  $b$ :

$$\alpha^{(k-\frac{1}{2})}, b^{(k-\frac{1}{2})} = \arg \min_{\alpha \geq 0, b} \sum_{i=1}^N \left\| Q_{\alpha \theta^{(k-1)}, b}(s_t^{(i)}, \mathbf{u}_t^{(i)}) - \left( c_t^{(i)} + \gamma \min_{\mathbf{u}} Q_{\alpha \theta^{(k-1)}, b}(s_{t+1}^{(i)}, \mathbf{u}) \right) \right\|_2^2. \quad (6)$$

This is similar to how, in policy evaluation, state values can be computed by solving a linear system. The parameters  $\theta$  can then be updated with  $\theta^{(k-\frac{1}{2})} = \alpha^{(k-\frac{1}{2})} \theta^{(k-1)}$ . We use the notation  $(k-\frac{1}{2})$  to denote an intermediate step between iterations  $(k-1)$  and  $(k)$ . Then, we optimize for  $\theta$  and  $b$  using a trust region on  $\theta$  and the intermediate values  $\theta^{(k-\frac{1}{2})}$  and  $b^{(k-\frac{1}{2})}$  for the target Q-function:

$$\begin{aligned} \theta^{(k)}, b^{(k)} = \arg \min_{\theta \geq 0, b} \sum_{i=1}^N \left\| Q_{\theta, b}(s_t^{(i)}, \mathbf{u}_t^{(i)}) - \left( c_t^{(i)} + \gamma \min_{\mathbf{u}} Q_{\theta^{(k-\frac{1}{2})}, b^{(k-\frac{1}{2})}}(s_{t+1}^{(i)}, \mathbf{u}) \right) \right\|_2^2 \\ \text{subject to } \left\| \theta - \theta^{(k-\frac{1}{2})} \right\|_2^2 \leq \epsilon. \end{aligned} \quad (7)$$

A summary of the algorithm used to learn the feature weights is shown in Algorithm 1.

---

**Algorithm 1** Trust region FQI with initialization of policy-independent parameters

---

```

1: procedure FQI( $\theta^{(0)}, \sigma_{\text{exploration}}^2$ )
2:   for  $j = 1, \dots, J$  do
3:     Gather dataset  $(s_t, \mathbf{u}_t, c_t, s_{t+1})$  of size N using exploration policy  $\mathcal{N}(\pi_{\theta^{(0)}}, \sigma_{\text{exploration}}^2)$ 
4:     for  $k = 1, \dots, K$  do
5:       Fit  $\alpha^{(k-\frac{1}{2})}$  and  $b^{(k-\frac{1}{2})}$  using (6)
6:        $\theta^{(k-\frac{1}{2})} \leftarrow \alpha^{(k-\frac{1}{2})} \theta^{(k-1)}$ 
7:       Fit  $\theta^{(k)}$  and  $b^{(k)}$  using (7)
8:     end for
9:      $\theta^{(0)} \leftarrow \theta^{(K)}$ 
10:  end for
11: end procedure

```

---

## 6 EXPERIMENTS

We evaluate the performance of the model for visual servoing in a simulated environment. The simulated quadcopter is governed by rigid body dynamics. The robot has 4 degrees of freedom, corresponding to translation and yaw angle. This simulation is inspired by tasks in which an autonomous quadcopter flies above a city, with the goal of following some target object (e.g. a car).

The dynamics of all the features were trained using a dataset of size 10000. The training set was generated from 1000 trajectories of a quadcopter following a car around the city with some randomness. Only 6 different cars were shown during training. The fully connected dynamics was fitted analytically and the locally connected dynamics was trained with ADAM (Kingma & Ba, 2014).



Figure 3: Cars used for learning the dynamics and feature weights. They were also used in some of the test experiments.



Figure 4: Cars used only in the test experiments. They were never seen during training nor validation.

We used the cross entropy method (CEM) and our FQI algorithm to learn the feature weighting. The exploration policy was the current servoing policy with additive Gaussian noise. The immediate cost received by the agent encodes the error of the target in image coordinates (details in Appendix B). We only used CEM for the pixel features since it doesn't work for high-dimensional problems, which is the case for all the VGG features. Details of the experimental setup are given in Appendix C.

We compare the servoing performance for various dynamics models and feature weightings. We hypothesize that learning a feature weighting leads to better performance since then the policy can servo more strongly with respect to the features that are relevant for the particular task, which, in our experiments, is to follow a car. We also hypothesize that using semantic features for servoing leads to better generalization of servoing at test time.

For each of the dynamics models, we compare our proposed FQI algorithm to learn the feature weights against two baselines. The first baseline doesn't use any feature weighting but just a single  $w$  for all the features. The second baseline uses CEM to search over the full space of policy parameters  $\lambda$  and the feature weights  $w$ ; the latter was initialized with the best  $w$  of the first baseline. For all of these methods, we chose the policy parameters that resulted in the best performance in 10 validation trajectories using the (noiseless) servoing policy.

We test the servoing policy on 100 test trajectories for each condition. We use the the sum of discounted costs as the performance metric and report them in Table 1. The reported costs come from test trajectories, meaning that the initial states were different as the ones used for validation.

The feature weights learned with CEM are comparable to the ones of FQI when used with pixel intensity features. This indicates that our FQI algorithm is able to learn useful feature weights.

Table 1a shows the results in which the cars used for testing were seen during training. In these experiments, all the models that used VGG features outperformed the models that used pixel intensities for servoing, if feature weighting is used. Out of all the weighted features, the ones from VGG conv2\_2 did the best. We hypothesize that the deeper features didn't lead to better performance because they weren't trained end-to-end to be spatially precise. Even though we use dilated convolutions to keep the spatial resolution throughout the network, the pretrained weights were trained for object classification where spatial invariance is desired.

In the results of Table 1b, the cars used for testing were never seen before during training nor validation. In these experiments, the deepest feature conv5\_3 with learned feature weighting achieved the best servoing performance. This shows that deep features are able to generalize for servoing. Furthermore, the total costs for both the conv4\_3 and conv5\_3 features when servoing with the unseen cars are very similar to the costs when servoing with the training cars. This suggests that the deep features are indeed robust to variations, such as different instances of cars.

Pixel dynamics performed poorly in these experiments since the model was trained and tested with different cars. However, pixel dynamics could achieve better performance in easier tasks where the target object can easily be singled out in one of the RGB channels. In fact, we found that the pixel dynamics performs better when trained and tested to follow only the bright red car of Figure 3. However, pixel dynamics didn't perform well for the blue car. Quantitative results of this single-car setting can be found in the Appendix (Table 4).

Example executions for the locally-connected dynamics of pixels and conv5\_3 features are shown in Table 2. Additional example executions can be found in the Appendix (Table 5).

feature dynamics	policy optimization algorithm			
		unweighted (1000)	CEM (12800)	FQI, ours (200)
pixel, fully connected	(17)	85.3 ± 8.8	77.8 ± 7.0	77.8 ± 9.2
pixel, locally connected	(6)	110.1 ± 11.8	92.5 ± 12.0	78.6 ± 9.5
VGG conv1_2	(2)	102.5 ± 7.0		50.6 ± 6.5
VGG conv2_2	(7)	104.5 ± 6.0		<b>34.7 ± 3.4</b>
VGG conv3_2	(46)	102.4 ± 10.8		47.4 ± 5.1
VGG conv4_3	(3)	110.4 ± 16.2		56.1 ± 7.6
VGG conv5_3	(4)	87.4 ± 5.3		47.3 ± 4.1

(a) Costs when using the set of cars seen during learning. These are the cars in Figure 3.

feature dynamics	policy optimization algorithm			
		unweighted (1000)	CEM (12800)	FQI, ours (200)
pixel, fully connected	(17)	78.8 $\pm$ 7.8	85.0 $\pm$ 9.6	92.1 $\pm$ 10.6
pixel, locally connected	(6)	120.0 $\pm$ 10.1	107.1 $\pm$ 11.0	78.3 $\pm$ 8.5
VGG conv1_2	(2)	102.4 $\pm$ 6.7		113.2 $\pm$ 7.9
VGG conv2_2	(7)	114.4 $\pm$ 12.7		60.2 $\pm$ 7.7
VGG conv3_2	(46)	107.1 $\pm$ 11.0		75.3 $\pm$ 6.1
VGG conv4_3	(3)	94.9 $\pm$ 7.9		56.1 $\pm$ 5.0
VGG conv5_3	(4)	90.9 $\pm$ 8.3		<b>49.5 <math>\pm</math> 5.6</b>

(b) Costs when using a new set of cars, none of which were seen during learning. These are the cars in Figure 4.

Table 1: Discounted sum of costs on test executions using the servoing policy for different feature dynamics and weighting of the features. The reported numbers are the mean and standard error across 100 test trajectories, of 100 time steps each. The test trajectories used different initial states as the validation trajectories. We compare our FQI algorithm used to learn a feature weighting against two baselines: one where the features are unweighted and another one where we use the cross entropy method (CEM) to learn the weighting. In the case of unweighted features, we used  $\lambda = 1$  and the best  $\mathbf{w}$  (shown in parenthesis) using validation trajectories. The CEM searches over the full space of policy parameters  $\mathbf{w}$  and  $\lambda$ , but it was only ran for pixel features since it doesn't scale for high-dimensional problems. For each of the algorithms for selecting the weights, the number of trajectories that were used are shown in parenthesis.





























































feature dynamics	observations from test trajectories										cost
pixel, locally connected											83.8
											78.0
											70.7
VGG, conv5_3											50.2
											74.4
											55.1

Table 2: Example observations seen during test trajectories in our experiments with the previously unseen cars and the corresponding discounted sum of costs for each trajectory. In each row, we show the observations of every 10 steps and the last one.



## 7 DISCUSSION

Manual design of visual features and dynamics models can limit the applicability of visual servoing approaches. We described an approach that combines learned visual features with learning predictive dynamics models and reinforcement learning to learn visual servoing mechanisms. Our experiments demonstrate that standard deep features, in our case taken from a model trained for object classification, can be used together with a bilinear predictive model to learn an effective visual servo that is robust to visual variation, changes in viewing angle and appearance, and occlusions. For control we propose to learn Q-values, building on fitted Q-iteration, which at execution time allows for one-step lookahead calculations that optimize long term objectives. Our method can learn an effective visual servo using just tens of trajectories. We evaluate our method on a complex synthetic car tracking benchmark, and demonstrate substantial improvement over a conventional pixel-based approach.

## REFERENCES

- Bert De Brabandere, Xu Jia, Tinne Tuytelaars, and Luc Van Gool. Dynamic filter networks. *CoRR*, abs/1605.09673, 2016. URL <http://arxiv.org/abs/1605.09673>.
- Guillaume Caron, Eric Marchand, and El Mustapha Mouaddib. Photometric visual servoing for omnidirectional cameras. *Autonomous Robots*, 35(2):177–193, 2013. ISSN 1573-7527. doi: 10.1007/s10514-013-9342-3. URL <http://dx.doi.org/10.1007/s10514-013-9342-3>.
- Andrea Censi and Richard M. Murray. Bootstrapping bilinear models of simple vehicles. *International Journal of Robotics Research*, 2015.
- F. Chaumette and S. Hutchinson. Visual servo control. i. basic approaches. *Robotics Automation Magazine, IEEE*, 13(4):82–90, Dec 2006. ISSN 1070-9932. doi: 10.1109/MRA.2006.250573.
- Jian Chen, Warren E Dixon, M Dawson, and Michael McIntyre. Homography-based visual servo tracking control of a wheeled mobile robot. *IEEE Transactions on Robotics*, 22(2):406–415, 2006.
- C. Collewet, E. Marchand, and F. Chaumette. Visual servoing set free from image processing. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 81–86, May 2008. doi: 10.1109/ROBOT.2008.4543190.
- Christophe Collewet and Eric Marchand. Photometric visual servoing. *IEEE Transactions on Robotics*, 27(4):828–834, 2011.
- Peter I Corke. Visual control of robot manipulators-a review. *Visual servoing*, 7:1–31, 1993.
- J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition (CVPR)*, 2009.
- Guilherme N DeSouza and Avinash C Kak. Vision for mobile robot navigation: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 24(2):237–267, 2002.
- Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*, pp. 647–655, 2014.
- Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556, 2005.
- B. Espiau, F. Chaumette, and P. Rives. A new approach to visual servoing in robotics. *IEEE Transactions on Robotics and Automation*, 8(3), 1992.
- John T Feddema and Owen Robert Mitchell. Vision-guided servoing with feature-based trajectory generation [for robots]. *IEEE Transactions on Robotics and Automation*, 5(5):691–700, 1989.
- Koichi Hashimoto. *Visual servoing: real-time control of robot manipulators based on visual sensory feedback*, volume 7. World scientific, 1993.

- Koh Hosoda and Minoru Asada. Versatile visual servoing without knowledge of true jacobian. In *Intelligent Robots and Systems' 94: Advanced Robotic Systems and the Real World', IROS'94. Proceedings of the IEEE/RSJ/GI International Conference on*, volume 1, pp. 186–193. IEEE, 1994.
- Seth Hutchinson, Gregory D Hager, and Peter I Corke. A tutorial on visual servo control. *IEEE transactions on robotics and automation*, 12(5):651–670, 1996.
- M. Jägersand, O. Fuentes, and R. C. Nelson. Experimental evaluation of uncalibrated visual servoing for precision manipulation. In *International Conference on Robotics and Automation (ICRA)*, 1997.
- Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.
- Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL <http://arxiv.org/abs/1412.6980>.
- Danica Kragic, Henrik I Christensen, et al. Survey on visual servoing for manipulation. *Computational Vision and Active Perception Laboratory, Fiskartorpsv*, 15, 2002.
- T. Lampe and M. Riedmiller. Acquiring visual servoing reaching and grasping skills using neural reinforcement learning. In *International Joint Conference on Neural Networks (IJCNN)*, 2013.
- S. Lange, M. Riedmiller, and A. Voigtlaender. Autonomous reinforcement learning on raw visual input data in a real world application. In *International Joint Conference on Neural Networks*, 2012.
- Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *CoRR*, abs/1504.00702, 2015. URL <http://arxiv.org/abs/1504.00702>.
- Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015. URL <http://arxiv.org/abs/1509.02971>.
- William Lotter, Gabriel Kreiman, and David Cox. Deep predictive coding networks for video prediction and unsupervised learning. *CoRR*, abs/1605.08104, 2016. URL <http://arxiv.org/abs/1605.08104>.
- Ezio Malis, Francois Chaumette, and Sylvie Boudet. 21/2d visual servoing. *IEEE Transactions on Robotics and Automation*, 15(2):238–250, 1999.
- Michaël Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*, 2015.
- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*, 2013.
- K. Mohta, V. Kumar, and K. Daniilidis. Vision based control of a quadrotor for perching on planes and lines. In *International Conference on Robotics and Automation (ICRA)*, 2014.
- Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.
- Martin Riedmiller. Neural fitted q iteration—first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pp. 317–328. Springer, 2005.

- Mehdi Sadeghzadeh, David Calvert, and Hussein A. Abdullah. Self-learning visual servoing of robot manipulator using explanation-based fuzzy neural networks and q-learning. *J. Intell. Robotics Syst.*, 78(1):83–104, April 2015. ISSN 0921-0296. doi: 10.1007/s10846-014-0151-5. URL <http://dx.doi.org/10.1007/s10846-014-0151-5>.
- John Schulman, Sergey Levine, Philipp Moritz, Michael I Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015.
- K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.
- Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Generating videos with scene dynamics. abs/1609.02612, 2016. URL <http://arxiv.org/abs/1609.02612>.
- Jacob Walker, Carl Doersch, Abhinav Gupta, and Martial Hebert. An uncertain future: Forecasting from static images using variational autoencoders. *CoRR*, abs/1606.07873, 2016. URL <http://arxiv.org/abs/1606.07873>.
- Manuel Watter, Jost Tobias Springenberg, Joschka Boedecker, and Martin A. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. *CoRR*, abs/1506.07365, 2015.
- LEE Weiss, ARTHUR Sanderson, and CHARLES Neuman. Dynamic sensor-based control of robots with visual feedback. *IEEE Journal on Robotics and Automation*, 3(5):404–417, 1987.
- W. J. Wilson, C. W. Williams Hulls, and G. S. Bell. Relative end-effector control using cartesian position based visual servoing. *IEEE Transactions on Robotics and Automation*, 12(5), 1996.
- Tianfan Xue, Jiajun Wu, Katherine L. Bouman, and William T. Freeman. Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks. *CoRR*, abs/1607.02586, 2016. URL <http://arxiv.org/abs/1607.02586>.
- B. H. Yoshimi and P. K. Allen. Active, uncalibrated visual servoing. In *International Conference on Robotics and Automation (ICRA)*, 1994.
- Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *CoRR*, abs/1511.07122, 2015. URL <http://arxiv.org/abs/1511.07122>.

## A LINEARIZATION OF THE BILINEAR DYNAMICS

The optimization of Equation (4) can be solved efficiently by using a linearization of the dynamics,

$$f_c^{(l)}(\mathbf{y}_{t,c}^{(l)}, \mathbf{u}) = f_c^{(l)}(\mathbf{y}_{t,c}^{(l)}, \bar{\mathbf{u}}) + \mathbf{J}_{t,c}^{(l)}(\mathbf{u} - \bar{\mathbf{u}}) = f_c^{(l)}(\mathbf{y}_{t,c}^{(l)}, \mathbf{0}) + \mathbf{J}_{t,c}^{(l)}\mathbf{u}, \quad (8)$$

where  $\mathbf{J}_{t,c}^{(l)}$  is the Jacobian matrix with partial derivatives  $\frac{\partial f_c^{(l)}}{\partial \mathbf{u}}(\mathbf{y}_{t,c}^{(l)}, \bar{\mathbf{u}})$  and  $\bar{\mathbf{u}}$  is the linearization point. Since the bilinear dynamics are linear with respect to the controls, this linearization is exact and the Jacobian matrix doesn’t depend on  $\bar{\mathbf{u}}$ . Without loss of generality, we set  $\bar{\mathbf{u}} = \mathbf{0}$ .

Furthermore, the bilinear dynamics allows the Jacobian matrix to be computed efficiently by simply doing a forward pass through the model. For the locally bilinear dynamics of Equation (3), the  $j$ -th column of the Jacobian matrix is given by

$$\mathbf{J}_{t,c,j}^{(l)} = \frac{\partial f_c^{(l)}}{\partial \mathbf{u}_j}(\mathbf{y}_{t,c}^{(l)}, \mathbf{0}) = \mathbf{W}_{c,j}^{(l)} * \mathbf{y}_{t,c}^{(l)} + \mathbf{B}_{c,j}^{(l)}. \quad (9)$$

## B COST FUNCTION FOR REINFORCEMENT LEARNING

The goal of reinforcement learning is to find a policy that maximizes the expected sum of discounted rewards, or equivalently, a policy that minimizes the expected sum of discounted costs. The cost should be one that quantifies progress towards the goal. We define the cost function in terms of the position of the target object (in the camera’s local frame) after the action has been taken,

$$c(s_t, \mathbf{u}_t, s_{t+1}) = \begin{cases} f \sqrt{\left(\frac{\mathbf{p}_{t+1}^x}{\mathbf{p}_{t+1}^z}\right)^2 + \left(\frac{\mathbf{p}_{t+1}^y}{\mathbf{p}_{t+1}^z}\right)^2 + \left(\frac{1}{\mathbf{p}_{t+1}^z} - \frac{1}{\mathbf{p}_*^z}\right)^2}, & \text{if } \|\mathbf{p}_{t+1}\|_2 \geq \tau \text{ and car in FOV} \\ (T - t + 1) c(\cdot, \cdot, s_t), & \text{otherwise,} \end{cases} \quad (10)$$

where  $f$  is the focal length of the camera (a constant) and  $T$  is the length of the trajectory. The episode terminates early if the camera is too close to the car (less than a distance  $\tau$ ) or the car is outside the camera’s field of view (FOV). The car’s position at time  $t$  is  $\mathbf{p}_t = (\mathbf{p}_t^x, \mathbf{p}_t^y, \mathbf{p}_t^z)$  and the car’s target position is  $\mathbf{p}_* = (0, 0, \mathbf{p}_*^z)$ , both in the camera’s local frame (z-direction is forward).

## C EXPERIMENT DETAILS

### C.1 TASK SETUP

The camera is attached to the vehicle slightly in front of the robot’s origin and facing down at an angle of  $\pi/6$  rad, similar to a commercial quadcopter drone. The robot has 4 degrees of freedom, corresponding to translation and yaw angle. Pitch and roll are held fixed.

In our simulations, the quadcopter follows a car that drives at  $1 \text{ m s}^{-1}$  along city roads during training and testing. The quadcopter’s speed is limited to within  $20 \text{ m s}^{-1}$  for lateral motions and  $10 \text{ m s}^{-1}$  for the other directions. The quadcopter’s angular speed is limited to within  $\pi/2 \text{ rad s}^{-1}$ . For each trajectory, a car is chosen randomly from a set of cars, and placed randomly on one of the roads. The quadcopter is initialized right behind the car, in the desired relative position for following. The image observed at the beginning of the trajectory is used as the goal observation.

### C.2 LEARNING FEATURE DYNAMICS

The dynamics of all the features were trained using a dataset of 10000 triplets  $\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1}$ . The original observations are  $480 \times 640$  RGB images but we downscale them to  $1/8$  of their resolution and take the center crop of  $32 \times 32$ . The actions are 4-dimensional vectors encoding the linear and angular (yaw) velocities. The images and the actions are normalized to between  $-1$  and  $1$ .

The training set was generated from 1000 trajectories of a quadcopter following a car around the city with some randomness. Each trajectory was 100 steps long. Only 6 different cars were shown during training, all of which are shown in Figure 3. The generation process of each trajectory is as follows: First, a car is chosen at random from the set of available cars and it is randomly placed on one of the roads. Then, the quadcopter is placed at some random position *relative* to the car. This quadcopter position is uniformly sampled in cylindrical coordinates relative to the car, with heights, radii and azimuthal angles in the intervals 10.5 m to 13.5 m, 14 m to 18 m and  $-\pi/2$  rad to  $\pi/2$  rad, where the origin of the azimuthal angle is the back of the car. The quadcopter’s yaw angle is initialized so that the quadcopter is aligned with the car. At every time step, an action is taken according to a mixed policy. With 0.25 probability, the robot takes the action that moves it towards the target position, and with 0.75 probability, it takes an action uniformly sampled from the action space. The target position is sampled from the same cylindrical space as the initial position and it is sampled once at the beginning of each trajectory.

We try the fully and locally connected dynamics for pixel intensities to better understand the performance trade-offs when assuming locally connected dynamics. We don’t use the latter for the semantic features since they are too high-dimensional for the dynamics model to fit in memory. The fully connected bilinear dynamics was fitted analytically and the locally connected dynamics was trained with ADAM (Kingma & Ba, 2014) using 10000 iterations, a batch size of 32, a learning rate of 0.001, and  $\gamma = 0.9$ . We used a weight decay of 0.0005 in both cases.

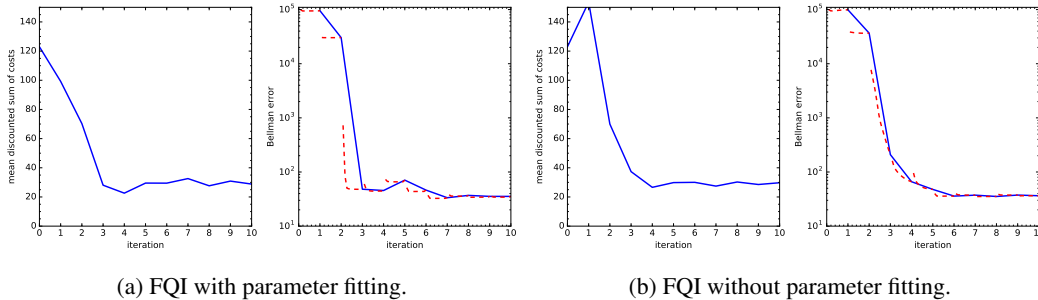


Figure 5: Learning curves of the FQI algorithm for learning the feature weighting of VGG conv1\_1 features with and without the parameter fitting of Equation (6). Each plot shows the sum of discounted costs on the left and the Bellman error on the right. The costs are computed on 10 validation trajectories and the Bellman error on the 10 training trajectories used by FQI. The solid blue lines correspond to the costs and errors after each sampling iteration  $j$  (iteration 0 is before learning). The dashed red lines correspond to the errors after each optimization iteration  $k$  (10 per sampling iteration) and they are plotted in between the corresponding sampling iterations. The Bellman error plateaus faster when the parameter fitting is used. The learning curves for the other VGG features are similar to these ones. However, the Bellman error fluctuated significantly for pixel intensities since they can’t approximate the Q-function as well.

### C.3 LEARNING FEATURE WEIGHTING VIA REINFORCEMENT LEARNING

We used the cross entropy method (CEM) and our proposed FQI algorithm to learn the feature weighting. We used the cost of Appendix B, a discount of  $\gamma = 0.9$  and trajectories of 100 steps.

We only used CEM for the pixel features since it doesn’t work for high-dimensional problems, which is the case for all the VGG features. We initialized the mean of the parameters with the best  $\mathbf{w}$  and with  $\lambda = 1$ , and the standard deviations with 10 and 1. The best  $\mathbf{w}$  was chosen by trying weights from 1 to 100 (in increments of 1) and selecting the weight that achieved the best performance in 10 trajectories. This is also how the weights were chosen in the first baseline of the experiments. Each iteration of CEM performed 117 (3 times as many parameters) noisy evaluations and selected the top 20% for the elite set. Each noisy evaluation used the mean sum of discounted costs of 10 trajectory rollouts. The rollouts used an exploration policy that sampled actions from a truncated normal distribution, with the current policy as the mean,  $\sigma_{\text{exploration}} = 0.2$  for the standard deviation, and a truncation between -1 and 1. In total, the policy learned with CEM used 12800 trajectories, out of which 1000 were used to select the best weight (out of 100) for initializing  $\mathbf{w}$ , 100 were used for validation, and the remaining ones were exploration trajectories used for learning. We used validation trajectories at each iteration to choose the parameters of the iteration that achieved the best validation performance.

The proposed FQI algorithm was initialized with  $\mathbf{w} = 10$  and  $\lambda = 1$ . The initial feature weights did not need to be chosen carefully; they just needed to be large enough so that the regularizer didn’t dominate for the first round of exploration. We used 10 sampling iterations, each of which ran FQI for 10 iterations. Each sampling iteration performed 10 trajectory rollouts also using the same exploration policy. In total, the policy learned with our method used 200 trajectories, out of which 100 were used for validation and the other 100 ones were exploration trajectories used for learning.

Typically, our proposed FQI algorithm converged in less than 10 sampling iterations, so our proposed method can potentially require even less trajectories. We also found that the fitting of Equation (6) empirically improved the converge of the FQI algorithm (see Figure 5 for an example).

### C.4 CLASSICAL IMAGE-BASED VISUAL SERVOING BASELINES

Traditional visual servoing techniques (Feddema & Mitchell, 1989; Weiss et al., 1987) use the image-plane coordinates of a set of points for control. For comparison to our method, we evaluate the servoing performance of feature points derived from bounding boxes and keypoints derived from hand-engineered features. We found that our task is challenging so we provide ground truth information for extracting and matching the features as well as for the dynamics of the points. The

servoing performance of these methods are summarized in Table 3. We provide details of the setup in the following two subsections.

#### C.4.1 BOUNDING BOX FEATURE POINTS

We use bounding boxes (the standard output in visual tracking) for visual servoing in our task. We use various variants, all of which use ground truth information:

- (a) We use ground truth 2D bounding boxes instead of using the state-of-the-art method for tracking. These boxes tightly fit around the visible portions of the car. We servo with respect to the 4 corners of the bounding box to take into account the position and scale of the car in image coordinates.
- (b) We use ground truth depth for the interaction matrices. In classical image-based visual servoing, the control law involves the so-called interaction matrix (also known as feature Jacobian), which is the Jacobian of the points in image space with respect to the camera’s control (see Chaumette & Hutchinson (2006) for details). Typically, estimated depth values are used for the Jacobian but we use the ground truth depth values.
- (c) We also try using ground truth 3D bounding boxes. These boxes are tight boxes that are axis-aligned with respect to the car’s reference frame. We servo with respect to the 8 corners of the 3D bounding box.
- (d) We also try servoing with respect to the (unprojected) 3D points, for both the 2D bounding boxes of (a) and the 3D bounding boxes of (c). This is in contrast to image-based visual servoing, where the servoing is done with respect to the projected points in the image plane.
- (e) For all of the above, the feature Jacobian assumes that the target points are static in the world frame. Since this assumption doesn’t hold for the moving car, we also try providing the ground truth dynamics of the car and incorporate it to the feature Jacobian. This amounts to adding a non-constant translation bias to the output of the dynamics function, where the translation is the displacement due to the car’s movement of the 3D point in the camera’s reference frame. Note that this is still not exactly equivalent to having the car being static since the roads have different slopes but the pitch and roll of the quadcopter is constrained to be fixed.

We ran the servoing policy for various combinations of the variants mentioned above and we report the mean discounted sum of costs in Table 3a. We used the same training cars and initial states as in Table 1a, so the results of these two tables can be directly compared to each other. We select the servoing gain hyperparameter by validating the policy on test trajectories for gains between 0.01 and 10 (in increments of 0.01 up to 0.1 and in increments of 0.1 starting at 0.1) and report the costs of the best-performing trajectories.

Despite of the ground truth information used to extract the feature points, these baselines performed worse than our method. In the case of 2D bounding boxes, the servoing is unstable when the car turns because the aspect ratio of the boxes change considerably. This problem doesn’t arise when servoing with respect to 3D bounding boxes. Still, it isn’t as good as when using the learned dynamics and feature weighting, since using the more accurate hand-specified dynamics doesn’t necessarily minimizes the cost of the task.

#### C.4.2 HAND-ENGINEERED FEATURE KEYPOINTS

We also use hand-crafted feature keypoints as a baseline. As in the previous section, we provide some ground truth information: we filter out the feature keypoints that lie outside the target object (i.e. the car), both in the goal and current images. We ran the servoing policy with SIFT, SURF and ORB features and we report the mean discounted sum of costs in Table 3b.

Servoing with respect to these features is far worse than the other baseline methods. This is, in part, because the feature extraction and matching process introduces compounding errors. Similar results were found by Collewet & Marchand (2011), who proposed photometric visual servoing (i.e. servoing with respect to pixel intensities) and showed that it outperforms, by an order of magnitude, classical visual servoing that uses SURF features.

feature points and dynamics	projected points	3D points (d)
bounding box (a) + (b)	$326.5 \pm 80.7$	$162.7 \pm 33.1$
3D bounding box (b) + (c)	$246.2 \pm 47.6$	$106.7 \pm 22.5$
bounding box with car dynamics (a) + (b) + (e)	$304.5 \pm 86.1$	$116.7 \pm 24.4$
3D bounding box with car dynamics (a) + (c) + (e)	$291.7 \pm 44.5$	$66.9 \pm 23.5$

(a) Costs when using points from bounding boxes for servoing.

feature points and dynamics	projected points	3D points (d)
SIFT	$783.1 \pm 112.7$	$730.9 \pm 133.4$
SURF	$519.9 \pm 38.9$	$475.8 \pm 73.2$
ORB	$562.1 \pm 82.3$	$589.7 \pm 74.4$

(b) Costs when using classical feature keypoints for servoing.

Table 3: Discounted sum of costs on test executions using classical image-based visual servoing with respect to feature points. The experimental setup is the same as in Table 1a, so the numbers in here can be directly compared to those numbers. We provide several variants that use ground truth information: (a) 2D bounding boxes, (b) depth for the interaction matrices, (c) 3D bounding boxes, (d) (unprojected) 3D feature points and their dynamics, and (e) car dynamics from an oracle.

feature dynamics	policy optimization algorithm		
	unweighted (1000)	CEM (12800)	FQI, ours (200)
pixel, fully connected	(9) $48.3 \pm 4.1$	$44.7 \pm 3.5$	$43.3 \pm 3.0$
pixel, locally connected	(7) $76.5 \pm 3.7$	$46.2 \pm 3.5$	$35.9 \pm 1.8$
VGG conv1_2	(2) $71.6 \pm 2.2$		$28.6 \pm 1.8$

(a) Costs when using the bright red car for learning the feature weighting and for testing.

feature dynamics	policy optimization algorithm		
	unweighted (1000)	CEM (12800)	FQI, ours (200)
pixel, fully connected	(48) $238.7 \pm 103.6$	$220.6 \pm 80.3$	$106.0 \pm 10.5$
pixel, locally connected	(40) $237.8 \pm 78.1$	$224.3 \pm 72.5$	$225.1 \pm 86.5$
VGG conv1_2	(1) $120.6 \pm 21.2$		$28.2 \pm 4.2$

(b) Costs when using the blue car for learning the feature weighting and for testing.

Table 4: Discounted sum of costs on test executions using the servoing policy for different pixel dynamics and weighting of the features. The numbers are reported in the same format as the ones in Table 1. Unlike those experiments, here we learn the feature weighting on one car instance and test the servoing with that same car. We experimented with the bright red car (a) or the blue car (b) of Figure 3.

















































































































































feature dynamics	observations from test trajectories											cost
pixel, fully connected												99.1
												56.0
pixel, locally connected												185.2
												58.4
VGG, conv1_2												105.2
												76.0
VGG, conv2_2												21.6
												32.9
VGG, conv3_2												68.8
												47.3
VGG, conv4_3												66.1
												29.2
VGG, conv5_3												69.8
												15.5

Table 5: Example observations seen during test trajectories in our experiments and the corresponding discounted sum of costs for each trajectory. In each row, we show the observations of every 10 steps and the last one. Some of the trajectories were shorter than 100 steps because of the termination condition (e.g. the car is no longer in the image). The trajectories shown in the first row of each model used one of the cars seen during training. We can see that all of the models except for VGG conv2\_2 failed to follow the red car because a fairly large distractor object (stoplight) occluded the view of the car. The trajectories shown in the second row used one of the cars that was never seen before. We can see that the VGG features were robust to the lamp post even though its color is similar to the color of the police car. Out of all the VGG features, conv5\_3 was the best at keep the camera at a good distance from the police car.