Sim2Real Viewpoint Invariant Visual Servoing by Recurrent Control

Fereshteh Sadeghi[†§] Alexander Toshev[§] Eric Jang[§] Sergey Levine[‡§]

[†]University of Washington [‡]University of California, Berkeley [§]Google Brain

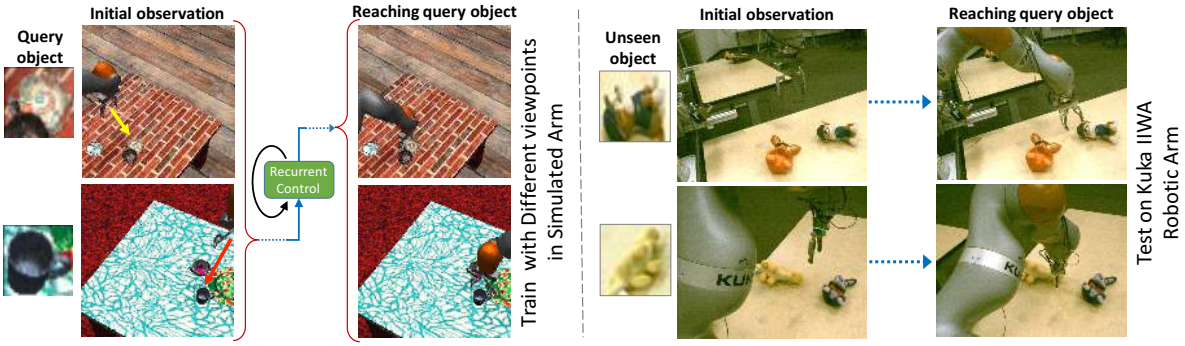

Figure 1. Illustration of our learned recurrent visual servoing controller. Training is performed in simulation (left) to reach varied objects from various viewpoints. The recurrent controller learns to implicitly calibrate the image-space motion of the arm with respect to the actions issued in the unknown coordinate frame of the robot. The model is then transferred to the real world by adapting the visual features, and can reach previously unseen objects from novel viewpoints (right).

Abstract

Humans are remarkably proficient at controlling their limbs and tools from a wide range of viewpoints. In robotics, this ability is referred to as visual servoing: moving a tool or end-point to a desired location using primarily visual feedback. In this paper, we propose learning viewpoint invariant visual servoing skills in a robot manipulation task. We train a deep recurrent controller that can automatically determine which actions move the end-effector of a robotic arm to a desired object. This problem is fundamentally ambiguous: under severe variation in viewpoint, it may be impossible to determine the actions in a single feed-forward operation. Instead, our visual servoing approach uses its memory of past movements to understand how the actions affect the robot motion from the current viewpoint, correcting mistakes and gradually moving closer to the target. This ability is in stark contrast to previous visual servoing methods, which assume known dynamics or require a calibration phase. We learn our recurrent controller using simulated data, synthetic demonstrations and reinforcement learning. We then describe how the resulting model can be transferred to a real-world robot by disentangling perception from control and only adapting the visual layers. The adapted model can servo to previously unseen objects from novel viewpoints on a real-world Kuka IIWA robotic arm. For supplementary videos, see: https://www.youtube.com/watch?v=oLgM2Bnb7fo

1. Introduction

Humans and animals can quickly recognize the effects of their actions through visual perception: when we see ourselves in a mirror, we quickly realize that the motion of our reflected image is reversed as a function of our muscle movements, and when we observe ourselves on camera (e.g., a security camera in a grocery store), we can quickly pick ourselves out from a crowd simply by looking for the motions that correlate with our actions. We can even understand the effects of our actions under complex optical transformations, such as in the case of a surgeon performing a procedure using a laparoscope. In short, we can quickly discover our own "end-effector" (either our own hand, or even a tool) and visually guide it to perform a desired task.

The ability to quickly acquire visual servoing skills of this sort under large viewpoint variation would have substantial implications for autonomous robotic systems: if a robot can learn to quickly adapt to any viewpoint, it can be dropped without any calibration into novel situations and autonomously discover how to control its joints to achieve a desired servoing goal. However, this poses a substantial technical challenge: the problem of discovering how the controllable degrees of freedom affect visual motion can be ambiguous and underspecified from a single frame. Consider the two scenes shown on the right side of Figure 1. Which way should the robot move its end-effector in order to reach the unseen query object? In the two settings, the action in the robot's (unknown) coordinate frame has almost the opposite effects on the observed image-space motion. Af-

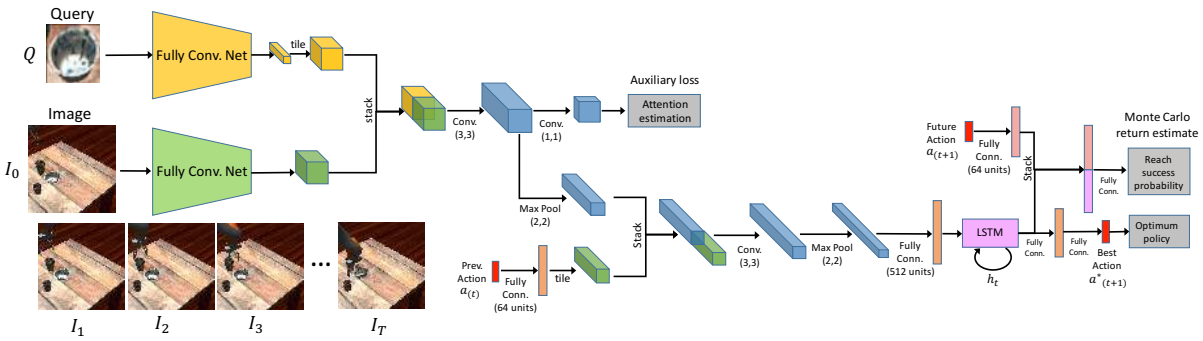

Figure 2. The input to our network is a query image (top-left) and the observed image at step t (left). The images are processed by separate convolutional stacks; their features are concatenated and are fed into an LSTM layer. The output is the policy (bottom right) which is an end-effector movement in the frame of the robot . The previously selected action is provided to LSTM, enabling it to implicitly calibrate the effects of actions on image-space motion. **Value prediction:** a separate head (top right) predicts the Q-value of the action trained with Monte Carlo return estimates. **Auxiliary loss:** An auxiliary loss function minimizes the localization error for the query object in the observed image. Also used in order to adapt the convolutional layers with a few labeled real images.

ter commanding an action and observing the movement, it is possible to deduce this relationship. However, identifying the effect of actions on image-space motion and successfully performing the servoing task requires a robust perception system augmented with the ability to maintain a memory of past actions.

In this paper, we show that view invariant visual servoing skills can be learned by deep neural networks, augmented with recurrent connections for memory. In classical robotics, visual servoing refers to controlling a robot in order to achieve a positional target in image space, typically specified by positions of hand-designed keypoint features [35, 15]. We instead take an open-world approach to visual servoing: the goal is specified simply by providing the network with a small picture of the desired object, and the network must select the actions that will cause the robot's arm to reach that object, without any manually specified features, and in the presence of severe viewpoint variation. This mechanism automatically and implicitly learns to identify how the actions affect image-space motion, and can generalize to novel viewpoints and objects not seen during training.

The main contribution of our work is a novel recurrent convolutional neural network controller that learns to servo a robot arm to previously unseen objects while it is invariant to the viewpoint. To learn perception and control for such viewpoint invariant servoing, we propose a training procedure that uses automatically generated demonstration trajectories in simulation as well as reinforcement learning (RL) policy evaluation. We train our viewpoint invariant controller primarily in a randomized simulation setup where we generate diverse scenes. Using a small amount of real-world images, we adapt the visual features to enable successful servoing on a real robotic arm while the overwhelming majority of training data is generated in a randomized simulator. Our experimental results evaluate the importance of recurrence for visual servoing on an extensive simulated benchmark and show that incorporating the value prediction function improves the results. We also evaluate the effectiveness of our method in several real-world servoing scenarios both quantitatively and qualitatively.

2. Related Work

Visual servoing has a long history in computer vision and robotics [6, 15]. Our proposed visual servoing method aims to address a similar problem, but differs in several important aspects of the visual servoing problem formulation, with the aim of producing a method that is more general and practical for open-world settings. We depart from a common assumption that the camera intrinsics and extrinsics are calibrated [23, 9], and make no assumptions about the 3D structure of the scene [9, 23, 7]. Several prior visual servoing methods also address servoing with uncalibrated cameras [36, 16, 21], but all of them address an "eye-in-hand" setting, where the goal is to servo the camera toward a target view by using previously known geometric features and estimate the image Jacobian within an image based visual servoing setup. In contrast, our visual servoing setting involves servoing a robotic arm to a visually indicated target, provided via a query image while the camera viewpoint is unknown and changes between trials. In contrast to the the eye-in-hand setup, the query image is *not* the desired image that the camera should see, but rather an object that arm should reach while the camera observes the scene from an unknown viewpoint. This requires the servoing mechanism to learn to match visual features between the query object and current observation, recognize the motion of the arm, and account for differences in viewpoint between trials.

Specifying the target by providing an image of the query object, instead of specifying low-level keypoints, is most similar to photometric visual servoing [5]. However, while photometric visual servoing aims to match a target image (e.g., by moving the camera), our method aims to direct the arm to approach the visually indicated object. The query image provides no information about *how* to approach the object, just which object is desired.

Our model must therefore both localize the object and direct the robot's motion.

Similarly to recent work on self-supervised robotic learning [18, 19, 2, 10, 26], our method uses observed images and self-supervision to train deep convolutional networks to predict action-conditioned task outcome. In contrast to these prior methods, our camera viewpoint is not fixed and can change drastically from one episode to another. Our approach incorporates fast adaptation via recurrence to adapt the visual servo to a novel viewpoint within a single episode of task execution, in addition to an outer-level self-supervised learning process performed with conventional gradient descent.

The use of recurrent networks for control has previously been used in a number of works on reinforcement learning, including methods for visual navigation [22, 24], continuous control [13, 38], and physics simulation to real world transfer [25] without visual observations. However, to our knowledge, no prior method has demonstrated that recurrence can be used to enable real-world robotic visual servoing from novel viewpoints. The closest work to this topic has taken a system identification approach for unknown physical parameters, such as masses and friction parameters [37] and does not use either of image observation or recurrence.

We use randomized simulated experience to train our visual servoing system. In order to use it for visual servoing in the real world, we also introduce an adaptation procedure based on finetuning of the visual features with an auxiliary objective. Most prior approaches to domain adaptation either train domain invariant representation [20, 4, 11, 3], learn a representation transformation from one domain to another via transfer learning [12, 34, 29], or employ domain randomization in simulation [30, 33] which produces robust models that can generalize broadly and can directly be deployed in the real world. Our approach combines domain randomization with transfer learning: we learn the controller entirely in a randomized simulation environment and then finetune only the visual features with a small amount of real world images, effectively transforming the model's representation into the real-world domain. We show that our final finetuning procedure produces an effective visual servoing mechanism in the real world, even though the recurrent motor control layers are not finetuned on the real-world data.

3. Recurrent View Invariant Visual Servo

Our aim is to learn visual servoing policies that can generalization to new viewpoints. To this end, our policy network should implicitly learn to "self-calibrate" and discover the relationship between motor commands and motion in the image. We set up visual servoing scenarios where a robot arm must reach objects using monocular camera observations captured from an arbitrary viewpoint. The reaching target is indicated by an image of the query object, and the network must figure out where this object is in the image and how to actuate the robot arm in order to reach it.

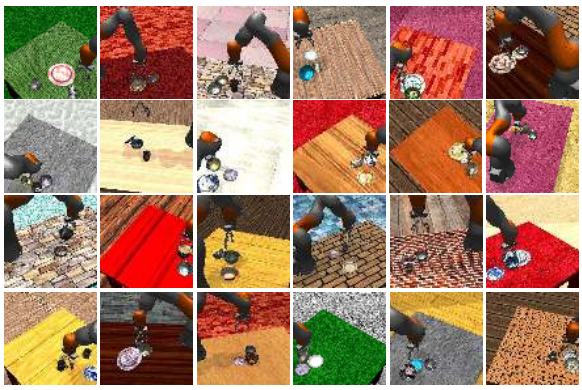

Figure 3. Our randomized simulated scenes and viewpoint randomization to learn viewpoint invariant visual servoing skills.

The principal challenge in this problem setup comes from the inherent ambiguity over the motion of the arm in the image in terms of the actions. Most standard visual servoing methods assume knowledge of the Jacobian – the relationship between actions and motion of desired visual features. Our approach not only has no initial knowledge of the Jacobian, but it does not even have any prior visual features, and must learn everything from data. Determining the right actions from a single image is generally not possible. Instead, we must incorporate temporal context, using the outcomes of past actions to inform future ones. To that end, we use a recurrent neural net (RNN), whose internal state is trained to extract and capture knowledge about hand-eye coordination during the course of a single episode. Our network must take a few initial actions, observe their outcomes, and implicitly "self-calibrate" to understand how actions influence image-space motion.

Another challenging aspect of our problem is the visual scene complexity. We generate diverse simulated scenes for each episode by randomly selecting a query object and distractor objects from a set of 3D shapes and rendering scene components(table, plane, objects) with random textures and under varied lighting conditions. The objects are placed on the table with random location and orientation. This setup enforces the model to learn to distinguish between objects and as such it needs to implicitly perform object localization in 3D.

We denote our controller model as π_θ. This model is a function, with parameters θ, that accepts as input the current image observation o_t, the query image q as well as the previous internal state h_{t-1} representing the memory. The previously chosen action a_{t-1} is also provided, so that it can infer how actions affect image-space motion. The output consists of the action a_t and the new internal state h_t, such that the policy is defined as

$$a_{t+1}, h_{t+1} = \pi_\theta(o_t, a_{t-1}, q, h_t) \quad (1)$$

We implemented recurrence in our controller using an LSTM, and the action is defined as a displacement $a = (\partial x, \partial y, \partial z)$ of the end-effector of the arm in the

Figure 4. The set of objects used in the real-world experiments. The seen plush toys are used for adapting the visual layers to natural images, while the unseen objects are used for testing.

robot's frame of reference (which is not known to the model). For the purpose of exploration, the policy that is used to collect experience during training is stochastic, and corresponds to a Gaussian with mean given by the model output. When the model is used to select actions for T steps, it produces a sequence of observation and actions $\tau = (o_1, a_2, \cdots, o_T, a_T)$. As illustrated in Fig. 2, the observation o_t and query image q are processed with separate convolutional stacks based on the VGG16 architecture [31], with o_t having an input size of 256×256 and q resized to 32×32. These networks are trained from scratch, without pretraining and produce vector representations of both images. The previous action a_{t-1} is transformed via a fully connected ReLU layer into a 64-dimensional feature vector. The observation embedding at step t, the query embedding and the action embeddings are concatenated in one vector as input to the recurrent motor control system, which uses a single-layer LSTM with 512 units [14]. The state of this LSTM corresponds to the memory h_{t-1}, which allows the model to capture information from past observations needed to perform implicit calibration.

4. Training

We train our visual servoing model with a combination of supervised learning, which is analogous to learning from demonstration, and outcome prediction, which corresponds to RL objective. In our implementation, the model is trained entirely in simulation (see Sec. 5), which provides full access to object locations and robot states. This allows us to produce supervision that corresponds to ground truth actions or synthetic "demonstrations." These demonstrations directly supervise the action output a_t. However, this supervision does not directly teach the long term effects of an action to the network. We found it beneficial to also augment the training process with a value prediction loss for RL. This loss trains the model to also predict the state-action value function associated with each action using multi-step Monte Carlo policy evaluation, which is the reward that the model expects to obtain for the entire episode by following its policy to take actions. As shown in our experiments, this RL loss leads to improved performance, since the resulting internal representations become better adapted to the long-term goal of the task.

4.1. Learning from Synthetic Demonstration

We synthesize demonstration trajectories by generating a large set of episodes with varied camera locations, objects, and textures as described in Sec. 5. Each episode contains ground truth actions that servo the arm to the query object, perturbed by Gaussian noise to provide some degree of exploration, which we found beneficial for producing robust polices. The training loss corresponds to the sum of squared Euclidean distances between the output action and the vector from the end-effector to the target object and can be written as

$$Loss = \sum_{t=1}^{T} || \frac{y - x_t}{||y - x_t||} - a_t ||^2. \quad (2)$$

To keep the action magnitudes within a bound, we learn normalized action direction vectors and use constant velocity so that a_t is independent of the number of steps. Here, we are not proposing a planning method for arbitrary tasks (e.g. presence of obstacles, etc.), but specifically aim to solve a visual servoing task, where the robot should move *directly* toward an object. Therefore, at each time t, the optimum action is the normalized direction vector towards the object and is computed by subtracting the end-effector position x_t from the object location y. The sampled trajectories provide starting points of the arm and past actions from which the model needs to recover. After unrolling the policy and formulating the above loss we use stochastic gradient descend over the parameters θ to minimize $Loss$. Following the DAgger framework [27], once our model converges, we generate additional on-policy trajectories by running the current policy and label them with the ground truth actions. This new data is then used to retrain the model and we repeat this procedure for two iterations.

4.2. Learning the Value Function

The above supervised learning procedure can quickly lead to a reasonable policy, but it is also myopic, in the sense that it does not consider the long term effects of an action. We found that the final performance of our model could be improved by also incorporating an RL objective, in the form of state-action value function prediction, also known as the Q-function [32]. This allows us to then select the action that minimizes the predicted long term reward. We formulate a reward function that indicates whether the arm has reached the target at the end of the episode, such that $r(s_t, a_t) = 1$ and $r(s_t, a_t) = 0$ otherwise. Here, we use s_t to denote the (unobserved) underlying state of the system, which includes the arm pose and query position. The target Q-values are then computed according to

$$Q(s_t, a_t) = r(s_t, a_t) + E_{\tau \sim \pi_\theta} \left[\sum_{t'=t+1}^{T} \gamma^{t'-t} r(s_{t'}, a_{t'}) \right]$$

where γ is a discount factor. These target values are used with a squared error regression loss applied to a second

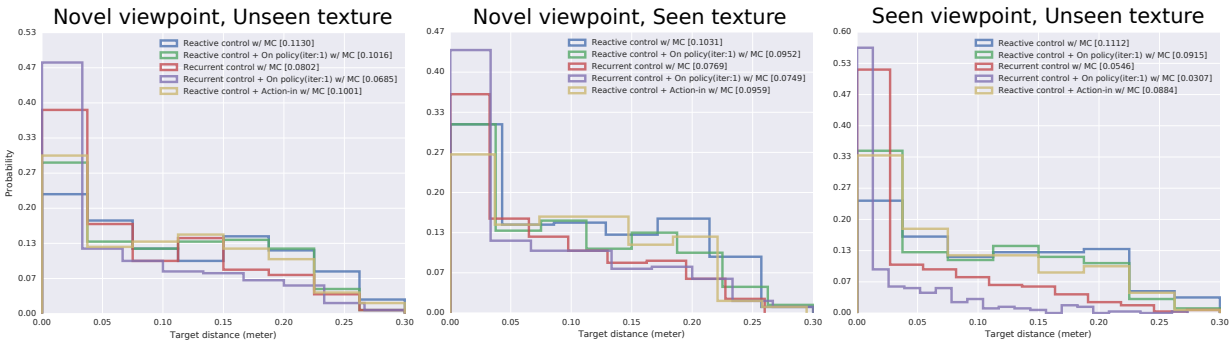

Figure 5. Comparing recurrent vs reactive control in test scenarios with different levels of difficulty and three random objects.

head of the model (see Fig. 2). The rewards and target Q-values are computed along trajectories sampled by running the policy. In practice, we found it beneficial to unroll the policy multiple times from each visited state and action and average together the corresponding returns to estimate the expectation with respect to π_θ. This corresponds to multi-step Monte Carlo policy evaluation [32], which provides a simple and stable method for learning Q-functions [30]. We optimize with respect to the input action a_t to choose actions according to this Q-function. In our implementation, we use cross-entropy method (CEM) [28] to perform this optimization, which provides a simple gradient-free optimization procedure for choosing actions that maximize predicted Q-values.

We implement our models in TensorFlow [1]. We use a buffer of one million unrolls for each policy learning round and deploy the Adam optimizer [17] with a learning rate of $1.5e-5$ and an exponential decay schedule. Each round of training converges after one million steps.

5. Simulated Training and Transfer

One of the main challenges in robot learning is data collection. While deep models are shown to work well on huge amount of data, large scale robot data collection is infeasible in many scenarios and results in challenges for training models with high generalization. To address this challenge, we train our controller in simulation where we can generate a large, diverse range of scenes captured from various viewpoints and obtain the supervision needed to train our model efficiently. We use domain randomization [30] to learn robust visual features and boost our performance in the real world by also incorporating visual adaptation with small amount of real world images.

Simulated Environment: We use the Bullet physics engine simulator [8], with a simulated 7 DoF Kuka IIWA arm and a variety of objects placed on a table in front of the arm. The objects are randomly selected from a set of 50 scanned objects of various dishware – plates, mugs, cups, bowls, teapots and etc. The objects are dropped on the table, so that their pose and location is randomized. We also randomize textures, lighting, and the appearance of the table and ground plane. This randomization procedure serves two important purposes: first, it forces the controller to learn the underlying geometric patterns that are actually important to the task, rather than picking up on extraneous aspects of object appearance that might correlate with actions, and second, it serves to enable the model to generalize more easily to real-world scenes, by essentially forcing it to solve a harder generalization task (widely different appearances), as discussed in prior work [30]. Each simulated trial consists of a random camera viewpoint, up to three randomly selected objects and randomized appearance parameters. Fig. 3 shows examples of our randomized simulation environment from various robot camera viewpoints.

Adaptation to the Real World: To perform visual servoing with a real robotic arm the model parameters should be able to generalize to real world. The randomization procedure described in the previous section already provides some degree of generalization [33, 30]. We also found that an additional adaptation step can improve generalization to real world scenarios. Obtaining ground truth actions and rewards in the real world requires costly manual labeling. Instead, we leverage the fact that the motor control portion of our model can remain largely unchanged between simulation and the real world, and only the visual features can be adapted using a weaker form of supervision to finetune only the convolutional layers of the model.

To that end, we use the auxiliary adaptation loss at the last layer of the visual stack (see Fig. 2, top-left), which predicts the presence or absence of the query object on a 8×8 grid overlaid on the image, by using computed logits at the last fully convolutional layer. These logits are fed into a cross entropy loss to finetune the vision stack. We use 22 sequences of the arm executing random actions, and we annotate the first frame in each video with bounding boxes for the objects that are present on the table which resulted in a total of 76 bounding boxes. Some of these scenes are shown in Fig. 10. Since the episodes remain stationary during each episode, we can propagate the labels automatically through each sequence. The actual loss is constructed by sampling a batch of sequences, and for each sequence sampling one object to use as the query object by cropping out one bounding box. To make our localization robust against object poses, for each sequence we randomly select the query image from a pool of query images of the same query

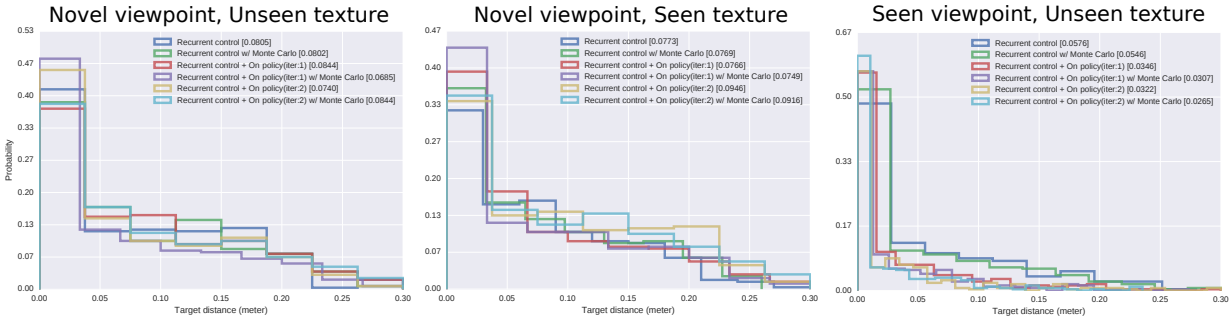

Figure 6. Comparison for different iterations of on-policy data collection, and the benefit of value prediction objective by using Monte-Carlo policy evaluation with three object test scenarios. Test scenarios with different levels of difficulty from left to right.

object category. The loss then describes the error in localizing the query object in the spatial image frame.

6. Experiments

Our experiments consists of detailed evaluations in simulation as as well as real-world evaluation with Kuka IIWA arm to study generalization to real-world. Prior visual servoing methods are not directly applicable to our problem setting, since we do not assume knowledge about the camera position or the action to image Jacobian. Also our target is specified by a picture of the object that the arm should reach for. Therefore, we compare to ablated variants of our method. To evaluate the importance of memory for learned implicit self-calibration, we compare to non-recurrent visual servo architectures trained in exactly the same way as our method. We also analyze the importance of combining supervised learning from demonstrated trajectories with RL value prediction.

6.1. Simulated Reaching

In our simulation experiments, we aim to answer the following questions: (1) How effective is our proposed recurrent controller compared to a feedforward reactive policy? (2) How robust is our model to viewpoint variation and visual diversity? (3) What is the benefit of incorporating on-policy data? (4) How beneficial is the value prediction objective?

Model setup: In addition to the recurrent controller, we also train two reactive non-recurrent policies which may or may not take the previous action as input and use a feed-forward network which has same layers as in Fig. 2, but has two fully connected layers instead of the recurrent LSTM layer. We call these baseline architectures as reactive+action-in and reactive policies, respectively.

Test setup: For testing, we generate new simulated scenarios using the randomization procedure described in Sec. 5. We generate scenes with two or three novel objects not seen during training. The test viewpoints are sampled uniformly at random in a region around the workspace, with the orientation chosen to always point toward the table, such that the query object is visible. We randomly select of the viewpoints for training episodes, while keeping a held-out set of test viewpoints to test generalization. We evaluate the performance of our method on test scenarios with different levels of difficulty: (a) Novel textures and novel viewpoints. (b) Previously seen textures and novel viewpoints. (c) Novel textures and previously seen viewpoints.

Evaluation criteria: In each simulation experiment, we run the policy for 300 trials, each with a fixed length of 10 steps. At the end of each trial, we compute the Euclidean distance of the robot's end-effector to the query object, using the closest points on the arm and the object mesh. This metric is in meters, and is zero when the arm touches the object. We report the average distance to the query object is the last time step over 300 test trials.

Reactive vs. recurrent policies: According to the final distance distributions illustrated in Figure 5, both reactive policies are substantially less proficient at reaching the query objects. When the testing scenario is the most challenging, with novel textures and novel viewpoints, and without any on-policy data collection, the average final distance obtained by the reactive policies are 0.10m and 0.11m, while the recurrent policy reaches an average final average distance of 0.08m. Incorporating on-policy data for training our proposed approach results in a final distance of 0.07m in the novel viewpoint and unseen texture condition. The results also indicate that the novel camera viewpoints are indeed more challenging when it comes to generalization. According to Table 1 there is $\sim$ 4-6 cm difference between reactive and recurrent controller performance which is 57%-86% of the open gripper (with width of 7cm), respectively. This is significant for robotic applications.

Random policy: We also compared our performance with random walk policy for commanding the end-effector. The random policy obtains average euclidean distance of 0.169m to the target object in 300 trials of 10 steps which validates the promising performance of our learned recurrent policy.

The effect of using on-policy data: The effect of using different numbers of iterations of on-policy training is shown in Figure 6 and also summarized in Table 2. Performing two iterations of retraining with on-policy data produces the best performance on the seen viewpoints and unseen texture scenarios, while resulting in poorer performance in the scenarios with novel viewpoints. On

Table 1. Average distance to target in meter for various test settings with two and three objects scenes. (VP: Viewpoint, T: Texture)

	Three objects			Two objects		
	Novel VP Unseen T	Novel VP Seen T	Seen VP Unseen T	Novel VP Unseen T	Novel VP Seen T	Seen VP Unseen T
Reactive w/ MC	0.1130	0.1031	0.1112	0.1062	0.1067	0.1051
Reactive + On policy w/ MC	0.1016	0.0952	0.0915	0.0935	0.0909	0.0950
Reactive+Action-in w/ MC	0.1001	0.0959	0.0884	0.0990	0.0938	0.0898
Recurrent w/ MC	0.0802	0.0769	0.0546	0.0730	0.0757	0.0461
Recurrent + On policy w/ MC	0.0685	0.0749	0.0307	0.0678	0.0741	0.0226

Table 2. Average distance to target in meter for evaluating the effect of the value prediction loss by using Monte-Carlo policy evaluation (MC) and on-policy data. (VP: Viewpoint, T: Texture)

	Three objects			Two objects		
	Novel VP Unseen T	Novel VP Seen T	Seen VP Unseen T	Novel VP Unseen T	Novel VP Seen T	Seen VP Unseen T
Recurrent	0.0805	0.0773	0.0576	0.0729	0.0819	0.0437
Recurrent w/ MC	0.0802	0.0769	0.0546	0.0730	0.0757	0.0461
Recurrent + On policy	0.0844	0.0766	0.0346	0.0747	0.0751	0.0235
Recurrent + On policy w/ MC	0.0685	0.0749	0.0307	0.0678	0.0741	0.0226
Recurrent + On policy(iter:2)	0.0740	0.0946	0.0322	0.0883	0.0863	0.0270
Recurrent + On policy(iter:2) w/ MC	0.0844	0.0916	0.0265	0.0865	0.0903	0.0229

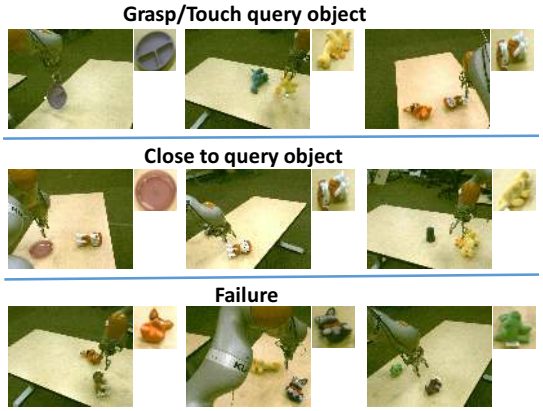

Figure 7. A successful reach occurs if the gripper touches the query object or gets very close to it. Failure examples include cases where the gripper ends up with a far distance from the query object or approaching the wrong object.

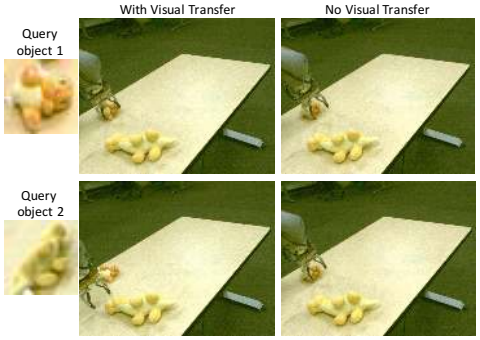

Figure 8. The network trained with only simulated data becomes confused between two objects with similar color and fails in the reaching task, while the visually adapted network can distinguish between the two object.

the other hand, using one iteration of retraining with on-policy data improves the performance in all scenarios. This result suggests that one iteration of on-policy data collection can address the distribution mismatch problem, though additional iterations can potentially result in becoming more specialized to the training viewpoints. Therefore, if the task emphasis is on mastery, using more on-policy data can improve performance.

The effect using Monte Carlo policy evaluation for value prediction : We conducted simulated experiments with and without the value prediction loss. When the value prediction loss is used, it is denoted by w/Monte Carlo in Figure 6 and Table 2. When using w/Monte Carlo, we compute the action based on the action prediction output of the model, at each time step. We then use CEM to perturb this action with Gaussian noise with standard deviation $\sigma = 0.003$ to generate 150 candidate actions and evaluate them via the value prediction head. The executed action is sampled at random from the top 5 actions with highest values. The results in Figure 6 shows that, in most conditions, incorporating the value prediction head which is trained using Monte-Carlo return estimates results in improved performance.

6.2. Real-World Robotic Reaching

We evaluated the generalization capability of our viewpoint invariant visual servoing model on a real 7DoF Kuka IIWA robotic arm. We used two sets of novel objects for the test experiments. The test objects include a variety of plush toys and dishware objects as shown in Figure 4. In the experiments, we placed the camera at various locations and arranged the table with objects at arbitrary locations and poses.

Quantitative Results: Here, we compare our recurrent controller with adapted visual layers to one that was trained entirely in simulation without any additional adaptation. The two models were compared head-to-head on each viewpoint and object arrangement, to provide a low-variance comparison. The tests were divided into scenarios with either one or two objects on the table. Table 3 summarizes the performance on the real-world reaching task. We performed a total of 42 trials, 18 with

Table 3. Real world reaching task results with novel viewpoints (Percentage of successful trials).

	Simulation only controller			Visually adapted controller		
	Success rate	Grasp/Touch query object	Close to query object	Success rate	Grasp/Touch query object	Close to query object query object
One object	88.9	55.6	33.3	94.4	61.1	33.3
Two objects	54.1	33.3	20.8	70.8	25.0	45.8

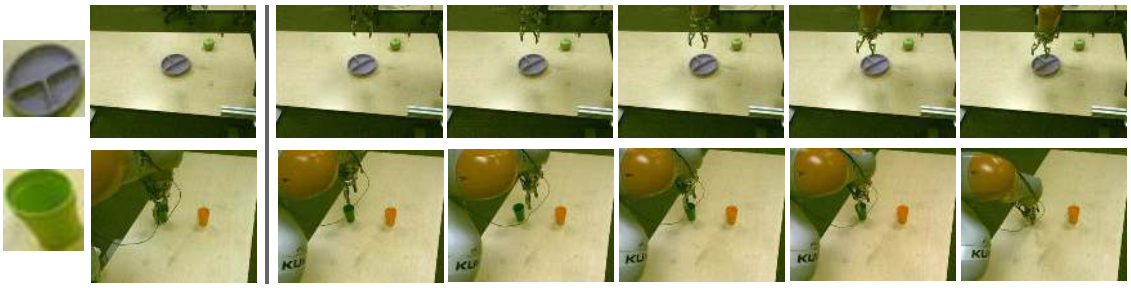

Figure 9. In both scenarios, the arm successfully reaches the object. Note that, in the second sequence, the arm first moves to the right, and then observes the effect of this action and corrects, moving toward the query object. This suggests that the controller can observe action outcomes and incorporate these observations to correct servoing mistakes.

single objects, and 24 with two objects. These trials were recorded from a total of 18 camera viewpoints. We count the number of times the arm moves towards the right query object and reaches it. A successful reach occurs if the gripper touches the object or gets very close to it. If the arm moves in the wrong direction or is confused between the two objects, we count the trial as a failure. We added a fixed procedure at the end of each robot trial to model a pointing action. In this procedure, the gripper is first closed and the arm is pulled up. Then the arm is moved downward and the gripper is opened. Note that, while our model is trained for reaching, and not particularly for the grasping task, using the aforementioned procedure can sometimes result in a successful grasp.

Figure 8 illustrates examples of successful and unsuccessful reaching attempts. The first row of this figure shows trials where the gripper touches the object or grasps it. The second row shows successful reach attempts where the gripper gets very closed to the query object, and is far from the distractor object. The last row shows several failure cases, where the controller is confused between two objects and the trial ends with the gripper at a considerable distance from the query object.

The single object scenarios provide a simpler test setting, where success is mainly dependent on the ability of the method to determine which actions move the arm toward the object. On the other hand, the two-object scenarios require the model to both generalize to a novel viewpoint and distinguish the query object from the distractor. This is significantly more challenging, especially since the test objects differ significantly from the simulated objects seen during training. As seen in Table 3, adapting the visual features with a small amount of real-world data substantially improves the performance of the network in both scenarios, with a success rate of 70.83% in the harder two-object setting. Table 3 summarizes the outcome of successful trials in detail and outlines the percentage of trials that result in the gripper touching or grasping the object.

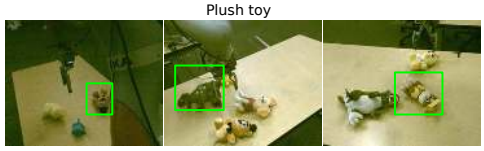

Figure 10. Examples of real-world scenes used for testing.

Qualitative Results: We visualize two interesting reaching sequences. In Figure 9 we see successful reaches with exploratory motions, where the arm first moves in the wrong direction, then observes the image-space motion and corrects. In Figure 8, we observe that the network that is entirely trained in simulation makes more mistakes when the query object and distractor object are visually similar. The network after adaptation is more robust to these kinds of visual ambiguities. For supplementary videos with more qualitative results, see: https://fsadeghi.github.io/Sim2RealViewInvariantServo.

7. Discussion and Future Work

In this paper, we described a learning-based visual servoing approach which can automatically and implicitly "self-calibrate" a robot in the process of a manipulation task from an unseen viewpoint. Our method is based on training a deep convolutional recurrent neural network that can control a robot to reach user-specified query objects, implicitly learning to identify the effects of actions in image-space from the past history of observations and actions. The network is trained primarily in simulation, where supervised demonstrated data is easy to obtain automatically, and a novel adaptation procedure is used to adapt the visual layers of this model to the real world, using only a small number of labeled images. An exciting direction to explore in future work is how more complex manipulation skills can be performed from any viewpoint using a similar approach as well as incorporating meta learning for fast adaptation.

Acknowledgement We thank Erwin Coumans and Yunfei Bai for providing pybullet and Vincent Vanhoucke for helpful discussions.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] P. Agrawal, A. V. Nair, P. Abbeel, J. Malik, and S. Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in Neural Information Processing Systems*, 2016.

[3] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. *arXiv preprint arXiv:1709.07857*, 2017.

[4] K. Bousmalis, G. Trigeorgis, N. Silberman, D. Krishnan, and D. Erhan. Domain separation networks. In *NIPS*, 2016.

[5] G. Caron, E. Marchand, and E. M. Mouaddib. Photometric visual servoing for omnidirectional cameras. *Autonomous Robots*, 2013.

[6] F. Chaumette and S. Hutchinson. Visual servo control. i. basic approaches. *IEEE Robotics & Automation Magazine*, 2006.

[7] C. Collewet, E. Marchand, and F. Chaumette. Visual servoing set free from image processing. In *ICRA*, 2008.

[8] Y. B. Erwin Coumans. pybullet, a python module for physics simulation in robotics, games and machine learning. `http://pybullet.org/`, 2016–2017.

[9] B. Espiau, F. Chaumette, and P. Rives. A new approach to visual servoing in robotics. *ieee Transactions on Robotics and Automation*, 1992.

[10] D. Gandhi, L. Pinto, and A. Gupta. Learning to fly by crashing. *arXiv preprint arXiv:1704.05588*, 2017.

[11] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, 2016.

[12] R. Gopalan, R. Li, and R. Chellappa. Domain adaptation for object recognition: An unsupervised approach. In *ICCV*, 2011.

[13] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015.

[14] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 1997.

[15] S. Hutchinson, G. D. Hager, and P. I. Corke. A tutorial on visual servo control. *IEEE transactions on robotics and automation*, 1996.

[16] M. Jagersand, O. Fuentes, and R. Nelson. Experimental evaluation of uncalibrated visual servoing for precision manipulation. In *IEEE International Conference on Robotics and Automation*, 1997.

[17] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *JMLR*, 2016.

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

[20] M. Long, Y. Cao, J. Wang, and M. Jordan. Learning transferable features with deep adaptation networks. In *ICML*, 2015.

[21] A. massoud Farahmand, A. Shademan, and M. Jagersand. Global visual-motor estimation for uncalibrated visual servoing. In *IROS*. IEEE, 2007.

[22] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

[23] K. Mohta, V. Kumar, and K. Daniilidis. Vision-based control of a quadrotor for perching on lines. In *ICRA*. IEEE, 2014.

[24] J. Oh, V. Chockalingam, S. Singh, and H. Lee. Control of memory, active perception, and action in minecraft. *arXiv preprint arXiv:1605.09128*, 2016.

[25] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. *arXiv preprint arXiv:1710.06537*, 2017.

[26] L. Pinto and A. Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *ICRA*. IEEE, 2016.

[27] S. Ross, G. Gordon, and A. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. *JMLR*, 2011.

[28] R. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and computing in applied probability*, 1999.

[29] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell. Sim-to-real robot learning from pixels with progressive nets. *arXiv preprint arXiv:1610.04286*, 2016.

[30] F. Sadeghi and S. Levine. CAD2RL: Real single-image flight without a single real image. In *Robotics: Science and Systems(RSS)*, 2017.

[31] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

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

[33] J. Tobin, W. Zaremba, and P. Abbeel. Domain randomization and generative models for robotic grasping. *arXiv preprint arXiv:1710.06425*, 2017.

[34] E. Tzeng, C. Devin, J. Hoffman, C. Finn, P. Abbeel, S. Levine, K. Saenko, and T. Darrell. Adapting deep visuomotor representations with weak pairwise constraints. *CoRR, vol. abs/1511.07111*, 2015.

[35] W. J. Wilson, C. W. Hulls, and G. S. Bell. Relative end-effector control using cartesian position based visual servoing. *IEEE Transactions on Robotics and Automation*, 1996.

[36] B. H. Yoshimi and P. K. Allen. Active, uncalibrated visual servoing. In *IEEE International Conference on Robotics and Automation*, 1994.

[37] W. Yu, C. K. Liu, and G. Turk. Preparing for the unknown: Learning a universal policy with online system identification. *arXiv preprint arXiv:1702.02453*, 2017.

[38] M. Zhang, Z. McCarthy, C. Finn, S. Levine, and P. Abbeel. Learning deep neural network policies with continuous memory states. In *ICRA*. IEEE, 2016.