Supplementary Material: Learning Eye-in-Hand Camera Calibration from a Single Image

Eugene Valassakis*
Department of Computing
Imperial College London
United Kingdom
pev115@ic.ac.uk

Kamil Drezckowski*
Department of Computing
Imperial College London
United Kingdom
krd115@ic.ac.uk

Edward Johns
Department of Computing
Imperial College London
United Kingdom
e.johns@imperial.ac.uk

1 Training Details

For all our models, we attempt to keep the architectures used as close as reasonably possible to each other to allow for a fair comparison. We use three core components to create all our architectures: (1) an encoder network using RGB images to output some lower-dimensional abstract feature map, (2) a decoder network that upsamples these abstract feature maps back to the original resolution, and (3) a Multi-Layer Perceptron (MLP) with dense layers processing these abstract feature maps. As such, the main differences in our architectures occur in (1) using dense layers or convolutional decoders to process the abstract feature maps obtained from the encoders, and (2) the number of output channels in the convolutional decoders which is model dependent.

Our encoder network depicted in fig. 1 and described in table 1. It takes as input a 144×256 RGB image and has a series of convolutional layers with ReLU [1] activations, batchnorm [2], and dropout [3] layers. The dropout probability is set to 0.25, and the convolutional layers use a kernel size of (3×3). We did not include a bias component in our layers, as we found this to be better early in our experimentation. The stride of the convolution alternates between 2 and 1. Padding is set so that a stride of 2 will half the input feature map's resolution, while 1 will keep the resolution the same.

Our decoder network is depicted in fig. 2 and described in table 2. It uses a series of non-parametric, bilinear upsampling layers followed by convolutional layers to increase the resolution of the abstract feature map that is the output of our encoder to the original resolution. Each bilinear upsampling layer doubles the spatial resolution of the input feature map, and its output is concatenated with the corresponding features from the encoder in a U-Net-like fashion [4]. All convolution layers apart from the last one have ReLU [1] activations, batchnorm [2], dropout of 0.25 [3], (3×3) kernels , stride 1, and padding set to keep the spatial resolution unchanged. Similarly to the encoder network, we did not include a bias component in our layers. The final convolution layer has a kernel size of (1×1), no batchnorm or dropout, and its activation and spatial resolution are model-dependent (see below).

Our fully connected MLP is depicted in fig. 3 and described in table 3 . It has three hidden fully connected layers of 16 neurons each, with batchnorm [2], 0.25 dropout [3], and ReLU [1] activations. The output layer of the network is a simple linear layer, with no batchnorm or dropout, and directly predicts the translation and orientation encoding of the extrinsic matrix.

For training all our networks, we use the Adam [5] optimiser, a batch size of 64, and a learning rate of 10^{-4}. We also use a learning rate scheduler that reduces the learning rate by a factor of 0.75 if performance stagnates. Finally, all RGB images are mapped to the range $[0, 1]$ and then normalised before being propagated through the networks.

*Joint First Author Contribution

5th Conference on Robot Learning (CoRL 2021), London, UK.

Table 1: Table detailing our encoder architecture.

Block	Layers	Parameters	Output Size	Activation
1	Conv2D, Batchnorm, Dropout	Kernel 3x3, Stride 2, Dropout 0.25	72x128x4	ReLU
2	Conv2D, Batchnorm, Dropout	Kernel 3x3, Stride 1, Dropout 0.25	72x128x4	ReLU
3	Conv2D, Batchnorm, Dropout	Kernel 3x3, Stride 2, Dropout 0.25	36x64x8	ReLU
4	Conv2D, Batchnorm, Dropout	Kernel 3x3, Stride 1, Dropout 0.25	36x64x8	ReLU
5	Conv2D, Batchnorm, Dropout	Kernel 3x3, Stride 2, Dropout 0.25	18x32x16	ReLU
6	Conv2D, Batchnorm, Dropout	Kernel 3x3, Stride 1, Dropout 0.25	18x32x16	ReLU
7	Conv2D, Batchnorm, Dropout	Kernel 3x3, Stride 2, Dropout 0.25	9x16x32	ReLU
8	Conv2D, Batchnorm, Dropout	Kernel 3x3, Stride 1, Dropout 0.25	9x16x32	ReLU

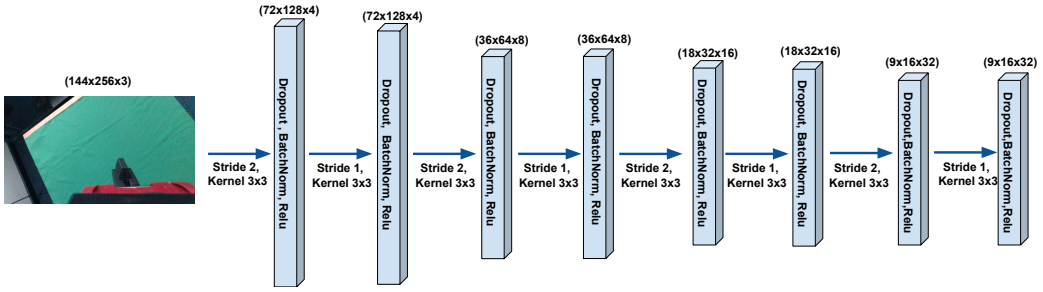

Figure 1: Illustration of the encoder architecture.

Table 2: Table detailing our decoder architecture.

Block	Layers	Parameters / Other	Output Size	Activation
1	Bilinear Upsampling		18x32x32	
2	Conv2D, Batchnorm, Dropout	Kernel 3x3, Stride 1, Dropout 0.25	18x32x32	ReLU
3	Conv2D, Batchnorm, Dropout	Kernel 3x3, Stride 1, Dropout 0.25	18x32x32	ReLU
4	Bilinear Upsampling	Skip Connection Concatenation with Encoder Feature Maps	36x64x32	
5	Conv2D, Batchnorm, Dropout	Kernel 3x3, Stride 1, Dropout 0.25	36x64x32	ReLU
6	Conv2D, Batchnorm, Dropout	Kernel 3x3, Stride 1, Dropout 0.25	36x64x32	ReLU
7	Bilinear Upsampling	Skip Connection Concatenation with Encoder Feature Maps	72x128x32	
8	Conv2D, Batchnorm, Dropout	Kernel 3x3, Stride 1, Dropout 0.25	72x128x24	ReLU
9	Conv2D, Batchnorm, Dropout	Kernel 3x3, Stride 1, Dropout 0.25	72x128x24	ReLU
10	Bilinear Upsampling	Skip Connection Concatenation with Encoder Feature Maps	144x256x24	
11	Conv2D, Batchnorm, Dropout	Kernel 3x3, Stride 1, Dropout 0.25	144x256x42	ReLU
12	Conv2D, Batchnorm, Dropout	Kernel 3x3, Stride 1, Dropout 0.25	144x256x42	ReLU
13	Conv2D	Kernel 1x1, Stride 1		

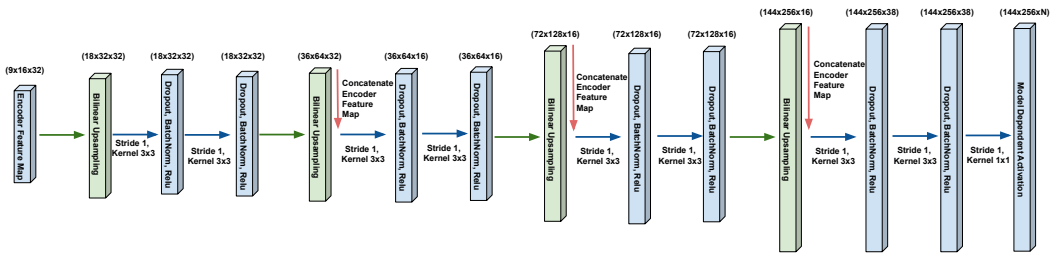

Figure 2: Illustration of the decoder architecture.

Table 3: Table detailing the MLP architecture.

Block	Layers	Parameters / Other	Output Size	Activation
1	Linear, Batchnorm, Dropout	Dropout 0.25	16	ReLU
2	Linear, Batchnorm, Dropout	Dropout 0.25	16	ReLU
3	Linear, Batchnorm, Dropout	Dropout 0.25	16	ReLU
4	Linear		9	

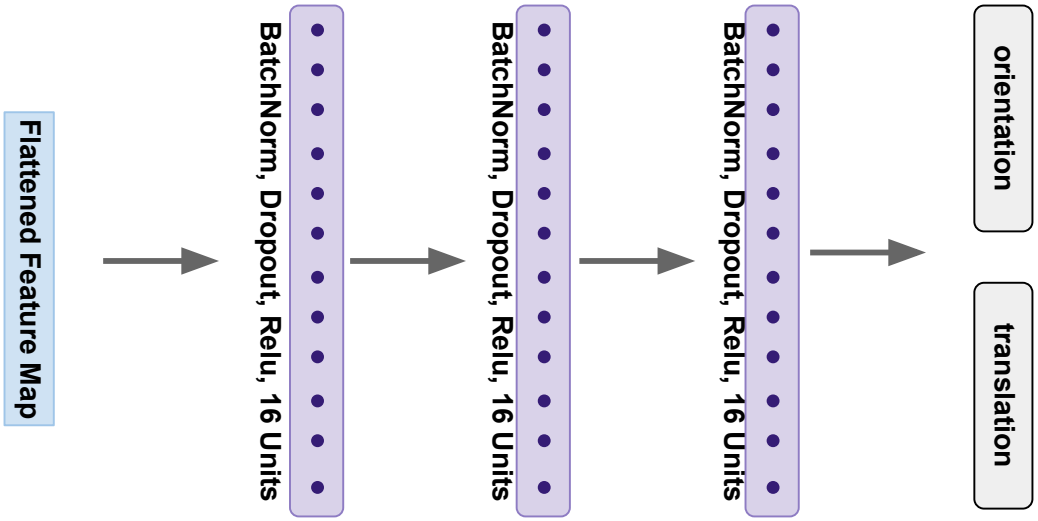

Figure 3: Illustration of the MLP architecture.

1.1 Direct Regression Details

For our direct regression model, we use our encoder network followed by our MLP network. The loss used was the mean squared error (MSE) between the network outputs and the ground truth extrinsic matrix encodings obtained from the simulated data. Finally, each output dimension was independently normalised to the range $[-1, 1]$, using min-max normalisation and the training dataset statistics.

1.2 Sparse Correspondence Model Details

Our sparse correspondence model consists of our encoder, followed by our decoder with its number of output channels being set to the number of keypoints used, and its output activation set to the spatial-soft-argmax activation [6]. The loss used was an equally-weighted linear combination of the L1 loss and the MSE loss between the network outputs and the ground truth keypoint locations obtained from the simulated data. Finally, all the keypoint locations are normalised in the range $[-1, 1]$ using the image dimensions, with any keypoint coordinate that appears out of frame (hence outside of the $[-1, 1]$ range) projected back to $[-1, 1]$. Before using the predicted 2D keypoint location to solve for the camera pose with PnP, we reject all keypoints less than 1% away from the image rim.

1.3 Dense Correspondence Model Details

For our dense correspondence model, the ICP initialisation network is the same as the direct regression network. The segmentation network consists of our encoder followed by our decoder, with a single output channel with the sigmoid activation [7]. The loss we use for this segmentation network is the $\alpha-$balanced variant of the Focal Loss [8], with the focusing parameter $\gamma = 2$, and with the weighting factor α proportional to the inverse class frequency. Finally, the depth regression network consists of our encoder network followed by our decoder network, with a single output channel fitted with a ReLU [1] activation function. The loss used for the depth regression is the L1 loss between the predicted depth map and the ground truth depth obtained from the simulated data, that is calculated over the union between the predicted and ground truth segmentation mask. Finally, the depth maps are normalised in the $[0, 1]$ range, using min-max normalisation and the training dataset statistics.

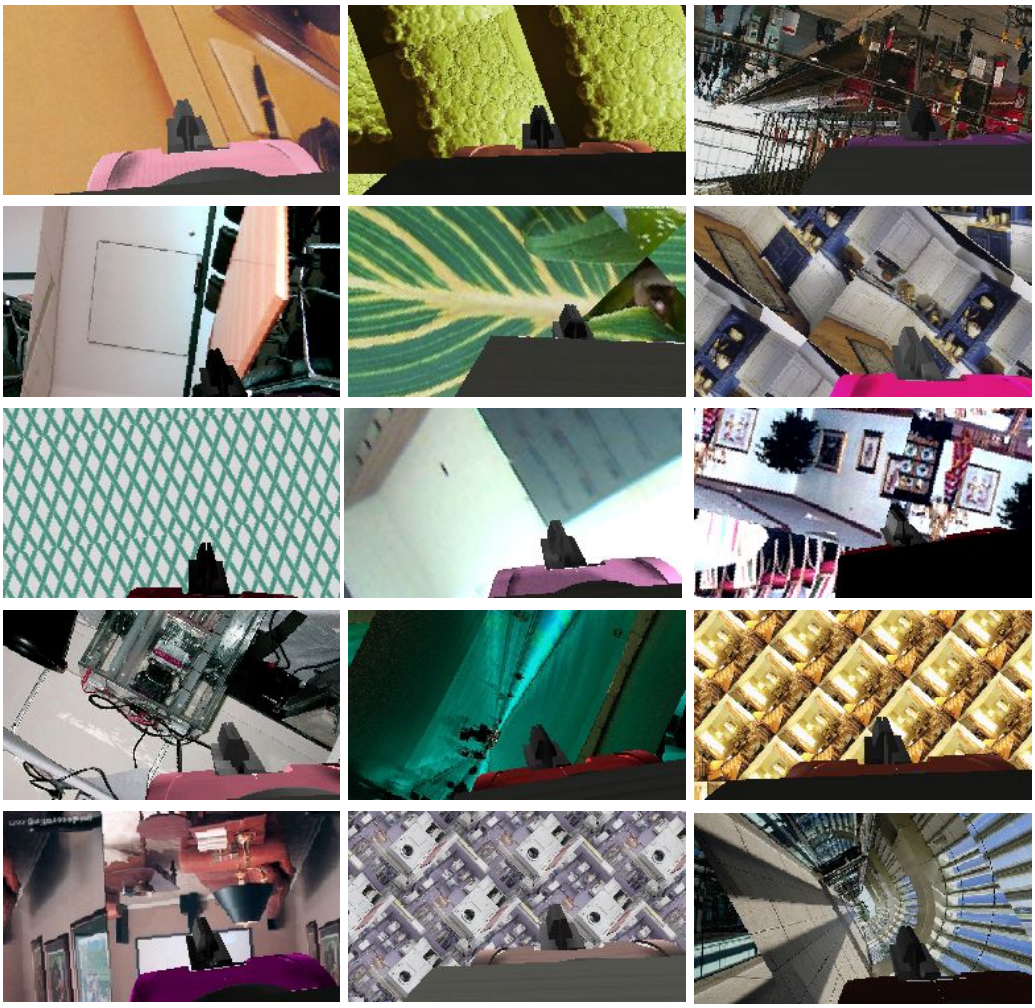

Figure 4: Examples of Domain Randomised Images

2 Dataset Generation Details

All our datasets consist of 10 000 images generated in simulation, examples of which can be seen in fig. 4. For each image in the dataset, we randomise the position of the camera in a $3cm \times 3cm \times 3cm$ volume and its orientation by $\pm 5^\circ$ around each axis. The mean of the randomisation range is estimated using the in-built Sawyer camera (which is already defined in the URDF) by taking images of the same AprilTag [10, 11] from both the in-built camera and our wrist-mounted camera and calculating the transformation between the two. We then apply random grayscale textures[†] to each of the components of the end-effector in simulation, and randomise their colours by (1) changing the hue as $h = h_{original} * (1.0 + e_h)$, (2) changing the brightness as $b = b_{original} * (1.0 + e_b)$, and (3) changing the saturation as $s = s_{original} * (1.0 + e_s)$, where e_h, e_b, and e_s are sampled uniformly from the ranges $[-0.2, 0.2]$, $[-0.45, 0.45]$ and $[-0.6, 0.5]$, respectively. We also randomise the position of two point light sources, sampled uniformly in a spherical shell around the gripper, with a radius varying in $[1.5m, 4.0m]$. Our simulator generates images at a 480×848 resolution, similarly to our real-world sensor. Before applying the background randomisation, we downsample these images to 144×256, which is our neural network input resolution. When applying the background images for background randomisation, we randomly rotate them in the range $[0^\circ, 360^\circ]$, and tile them when needed to avoid portions of the image with no background applied. Finally, we apply a colour jitter [12] operation to the full image with brighness variation of 0.2, contrast variation of 0.2, saturation variation of 0.2 and hue variation of 0.05.

[†]Textures modified from `https://github.com/tianheyu927/mil/blob/master/scripts/get_data.sh`, made available by [9].

3 Fusing Multiple Estimates

Algorithm 1: Fusing multiple estimates

Input: Individual estimates $\{\tilde{T}^i_{EC} = [\tilde{R}^i_{EC}|\tilde{t}^i_{EC}]\}^N_{i=1}$
Output: A fused estimate $\bar{T}_{EC}$
1 Initialise $\xi = []$
for $k = 1, ..., N$ **do**
2 $\quad \xi \leftarrow \xi \cup \left(\tilde{t}^k_{EC}, \phi^k\right)$ where ϕ_k are the intrinsic Euler XYZ angles that represent the rotation matrix $\tilde{R}^k_{EC}$;
3 Calculate the mean $\mu = \text{mean}(\xi) \in \mathbb{R}^6$ and estimate the covariance $\Sigma = \text{Cov}(\xi) \in \mathbb{R}^{6\times6}$
4 Discard 20% of estimates in ξ with the lowest probability density under the PDF $\mathcal{N}(\mu, \Sigma)$
5 Calculate the mean of the remaining estimates $\bar{\xi} = \text{mean}(\xi) = (\bar{t}, \bar{Euler})$
6 $\bar{T}_{EC} = [\text{Euler2Rot}(\bar{Euler})|\bar{t}]$

Algorithm 1 describes the procedure used to fuse multiple camera pose estimates into a single estimate. This algorithm is initialised with a list of camera pose estimates, $\{\tilde{T}^i_{EC} = [\tilde{R}^i_{EC}|\tilde{t}^i_{EC}]\}^N_{i=1}$, and creates an empty list $\xi = []$. It then iterates through all individual estimates $\tilde{T}^k_{EC}$ for $k = 1, ..., N$, and represents them as 6D vectors $\left(\tilde{t}^k_{EC}, \phi_k\right)$, where ϕ_k are the intrinsic Euler XYZ angles that represent the rotation matrix $\tilde{R}^k_{EC}$, and stores these representations in the list ξ. After mapping all of the input estimates to a lower dimensional representation, line 3 in the pseudocode computes the mean and an unbiased estimate of the covariance matrix of all of the estimates. In line 4, we then assume that the data follows a Gaussian distribution and discard 20% of the estimates with the lowest probability density under this Gaussian model in order to remove potential outliers. In line 5, we then calculate the mean camera pose in the 6D space, and in line 6 we transform this lower dimensional representation of the mean estimate back to a 4×4 homogeneous transformation matrix.

4 Experiments

We set up a single simulation environment for all our learning-based methods, and generate a single dataset to train them. Dataset generation took approximately 30 minutes given the ability to parallelise data collection. Training the direct regression model took approximately 30 minutes, the sparse correspondence model approximately 3.5 hours, and the dense correspondence model approximately 3 hours. Finally, we performed early stopping with a 5% validation subset.

In the following sections we use the following method abbreviations: **D**irect **R**egression (DR), **S**parse **C**orrespondense method (SC), and **D**ense **C**orrespondence method (DC). When $(fusion)$ is specified, we indicate that our fusion method for aggregating multiple estimates was used.

4.1 Simulated Experiment

Algorithm 2 outlines the procedure used to evaluate all methods in simulation. In line 1, the algorithm initialises an empty list to store the position and orientation error for each of the methods independently, so that these errors can later be averaged across N different extrinsic matrices. The position error is defined as

$$e_t(\tilde{t}, t) = ||\tilde{t} - t||_2$$

where $|| \cdot ||_2$ is the L2 norm. The orientation error is defined as

$$e_R(\tilde{R}, R) = \theta$$

where θ is the angle of rotation from the axis-angle representation of the rotation matrix R_Δ that satisfies the relationship $R = R_\Delta \tilde{R}$.

Once the lists are initialised, the outer for loop that extends from line 2 to line 18 of the algorithm is responsible for collecting position and orientation errors for various samples of the camera to end-effector pose. Within this for loop, in line 3, a camera to end-effector pose is sampled and set.

Algorithm 2: Simulated Experiment

1 $\epsilon_{pos}^{\textbf{method}} = []$ and $\epsilon_{ori}^{\textbf{method}} = []$ for each **method** $\in$ [Tsai [13], Park [14], Horaud [15], Daniilidis [16], DR, DR (fusion), SC, SC (fusion), DC, DC (fusion)]
2 **for** $i = 1, ..., N$ **do**
3 Sample and set a camera to end-effector pose $T_{EC} = [R_{EC}|t_{EC}]$
4 Collect a dataset $\{I_j, T_{BE}^j, \tilde{T}_{CO}^j\}_{j=1}^{15}$ of 15 RGB images I, end-effector to robot base poses T_{BE}, and estimated calibration object to camera poses $\tilde{T}_{CO}$
5 **for** ***method*** $\in$ *[Tsai [13], Park [14], Horaud [15], Daniilidis [16]]* **do**
6 $\tilde{T}_{EC} = [\tilde{R}_{EC}|\tilde{t}_{EC}] \leftarrow$ use_**method**_for_calibration $\left(\{T_{BE}^j, \tilde{T}_{CO}^j\}_{j=1}^{15}\right)$
7 $\epsilon_{pos}^{\textbf{method}} \leftarrow \epsilon_{pos}^{\textbf{method}} \cup e_t(\tilde{t}_{EC}, t_{EC})$
8 $\epsilon_{ori}^{\textbf{method}} \leftarrow \epsilon_{ori}^{\textbf{method}} \cup e_R(\tilde{R}_{EC}, R_{EC})$
9 **for** ***method*** $\in$ *[DR, SC, DC]* **do**
10 $T = []$
11 **for** $k = 1, ..., 15$ **do**
12 $\tilde{T}_{EC}^k = [\tilde{R}_{EC}|\tilde{t}_{EC}] \leftarrow$ use_**method**_for_calibration(I_k)
13 $T \leftarrow T \cup \tilde{T}_{EC}^k$
14 $\epsilon_{pos}^{\textbf{method}} \leftarrow \epsilon_{pos}^{\textbf{method}} \cup e_t(\tilde{t}_{EC}, t_{EC})$
15 $\epsilon_{ori}^{\textbf{method}} \leftarrow \epsilon_{ori}^{\textbf{method}} \cup e_R(\tilde{R}_{EC}, R_{EC})$
16 $\bar{T}_{EC} = [\bar{R}_{EC}|\bar{t}_{EC}] \leftarrow$ fusion(T)
17 $\epsilon_{pos}^{\textbf{method (fusion)}} \leftarrow \epsilon_{pos}^{\textbf{method (fusion)}} \cup e_t(\bar{t}_{EC}, t_{EC})$
18 $\epsilon_{ori}^{\textbf{method (fusion)}} \leftarrow \epsilon_{ori}^{\textbf{method (fusion)}} \cup e_R(\bar{R}_{EC}, R_{EC})$
19 $\varepsilon_t^{\textbf{method}} \leftarrow$ mean$(\epsilon_{pos}^{\textbf{method}})$ for all methods
20 $\varepsilon_R^{\textbf{method}} \leftarrow$ mean$(\epsilon_{ori}^{\textbf{method}})$ for all methods

Then, in line 4, a dataset of 15 corresponding RGB images and end-effector to robot base poses is collected by moving the end-effector around the AprilTag. The corresponding AprilTag to camera poses are estimated from the RGB images with the AprilTags3 library [17, 10, 11].

Between lines 5 and 8, the algorithm then estimates the camera to end-effector pose using each of the classical methods and evaluates each of the estimates. Then, between lines 9 and 18, the algorithm evaluates all of the deep learning-based methods. For each method, the algorithm begins by creating an empty list (line 10) that is used to store all individual estimates. It then iterates through all of the individual images and evaluates all of the single image estimates (lines 11-15). It then fuses all of the individual estimates using algorithm 1 and evaluates the fused estimate (lines 16-18). The algorithm ends by computing the mean of the position and orientation errors for each of the methods (lines 19 and 20).

We stress that even though the AprilTag is visible in all images used to estimate the camera to end-effector pose, it is not used by the learned methods. The same images are used to evaluate the learned methods as the ones to estimate the tag to camera pose that is required by the classical methods only to ensure a fair evaluation.

4.2 Real World Experiment

The real-world experimental procedure is shown in algorithm 3. This procedure consists of 4 independent stages. In the first stage (lines 1-3), a data bank and an evaluation dataset of RGB images, end-effector to robot base poses, and estimated AprilTag to camera poses are collected by automatically moving the robot's end-effector around an AprilTag. We estimate the AprilTag poses using the AprilTags3 library [10, 11, 17]. The lengths of the training data bank and evaluation dataset are 40 and 60 respectively.

In the second stage (lines 4-15), we obtain estimates of the camera to end-effector pose from all methods that require more than a single data point. During this stage, we sample 40 datasets of

Algorithm 3: Real World Experiment

1 $T^{\textbf{method}} = []$ for **method** $\in$ [Tsai [13], Park [14], Horaud [15], Daniilidis [16], DR, DR (fusion), SC, SC (fusion), DC, DC (fusion)]
2 Collect a training data bank $\mathcal{D}_{train} = \{I_i, T^i_{BE}, \tilde{T}^i_{CO}\}_{i=1}^{40}$
3 Collect an evaluation dataset $\mathcal{D}_{eval} = \{T^i_{BE}, \tilde{T}^i_{CO}\}_{i=1}^{60}$
4 **for** $i = 1, ..., 40$ **do**
5 Sample a dataset $\mathcal{D} = \{I_j, T^j_{BE}, \tilde{T}^j_{CO}\}_{j=1}^{15} \in \mathcal{D}_{train}$
6 **for** ***method*** $\in$ *[Tsai [13], Park [14], Horaud [15], Daniilidis [16]]* **do**
7 $\tilde{T}^i_{EC} \leftarrow$ use_**method**_for_calibration $\left(\{T^j_{BE}, \tilde{T}^j_{CO}\}_{j=1}^{15}\right)$
8 $T^{\textbf{method}} \leftarrow T^{\textbf{method}} \cup \tilde{T}^i_{EC}$
9 **for** ***method*** $\in$ *[DR, SC, DC]* **do**
10 $\xi = []$
11 **for** $k = 1, ..., 15$ **do**
12 $\tilde{T}^k_{EC} =\leftarrow$ use_**method**_for_calibration(I_k)
13 $\xi \leftarrow \xi \cup \tilde{T}^k_{EC}$
14 $\bar{T}_{EC} =\leftarrow$ fusion(ξ)
15 $T^{\textbf{method (fusion)}} \leftarrow T^{\textbf{method (fusion)}} \cup \bar{T}_{EC}$
16 **for** $I_i \in \mathcal{D}_{train}$ **do**
17 **for** ***method*** $\in$ *[DR, SC, DC]* **do**
18 $\tilde{T}^i_{EC} =\leftarrow$ use_**method**_for_calibration(I_i)
19 $T^{\textbf{method}} \leftarrow T^{\textbf{method}} \cup \tilde{T}^i_{EC}$
20 **for** ***method*** $\in$ *[Tsai [13], Park [14], Horaud [15], Daniilidis [16], DR, DR (fusion), SC, SC (fusion), DC, DC (fusion)]* **do**
 $t = []$
21 **for** $\tilde{T}^k_{EC} \in T^{\textbf{method}}$ **do**
22 **for** $\{T^i_{BE}, \tilde{T}^i_{CO}\} \in \mathcal{D}_{eval}$ **do**
 $\tilde{T}^{ik}_{BO} = [\tilde{R}^{ik}_{BO} | \tilde{t}^{ik}_{BO}] = T^i_{BE} \tilde{T}^k_{EC} \tilde{T}^i_{CO}$
23 $t \leftarrow t \cup \tilde{t}^{ik}_{BO}$
24 $\epsilon^{\textbf{method}}_{std} =$ standard devation(t)

15 data points each from the data bank. For each dataset, we estimate the camera to end-effector pose using the classical baselines. We also estimate the camera to end-effector pose from each of the RGB images using each of the learned methods and fuse all 15 estimates into a single estimate using algorithm 1. In the third stage (lines 16-19), we obtain the estimates from the learning-based methods for all of the images in the training data bank.

The final stage (lines 20-24) evaluates each of the methods. As in the real world we do not have access to the ground truth camera to end-effector pose, we use an indirect error metric [13, 16]. Let T_{BE} be the end-effector to robot base pose, $\tilde{T}_{EC}$ be an estimate of the camera to end-effector pose and $\tilde{T}_{CO}$ be an estimate of the AprilTag to camera pose. An estimate of the AprilTag to robot base pose can be computed as

$$\tilde{T}_{BO} = T_{BE} \tilde{T}_{EC} \tilde{T}_{CO}$$

Now, for a single estimate of the camera to end-effector pose, $\tilde{T}^k_{EC}$ and for each pair of end-effector to robot base and AprilTag to camera poses, $\{T^i_{BE}, \tilde{T}^i_{CO}\} \in \mathcal{D}_{eval}$, a different estimate of the AprilTag to robot base pose can be computed, $\tilde{T}^{ik}_{BO} = [\tilde{R}^{ik}_{BO} | \tilde{t}^{ik}_{BO}]$. We use the standard deviation of all estimated AprilTag to robot base positions for a single method, $\tilde{t}^{ik}_{BO}$ for all i, k, as an indirect measure of the accuracy of that method. The higher the precision of the estimated camera to end-effector matrix, the lower the spread of the estimated AprilTag positions in the robot base frame, and hence, the lower the standard deviation.

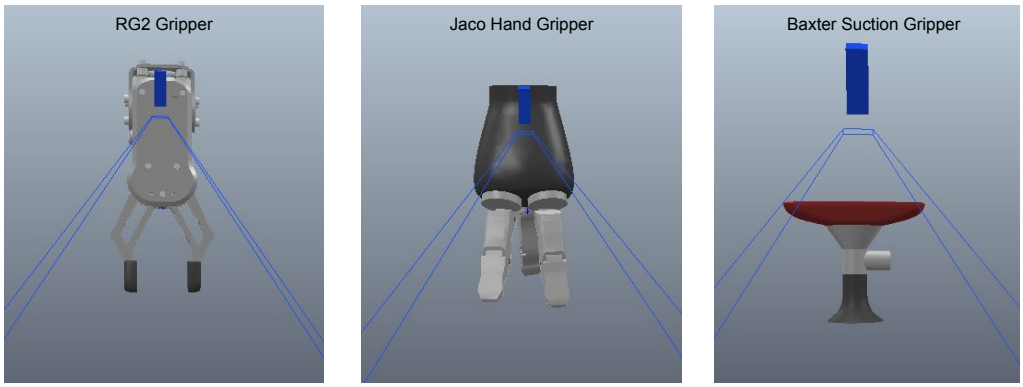

Figure 5: Illustration of additional grippers tested.

Gripper	ϵ_t [mm]	ϵ_R [degrees]
Sawyer	(13.4 ± 4.1)	(4.4 ± 1.4)
Sawyer (fusion)	(13.4 ± 4.1)	(4.4 ± 1.4)
Jaco Hand	(10.1 ± 4.0)	(3.6 ± 1.4)
Jaco Hand (fusion)	(9.9 ± 3.8)	(3.5 ± 1.3)
RG2	(10.3 ± 3.8)	(4.5 ± 1.3)
RG2 (fusion)	(10.3 ± 3.8)	(4.5 ± 1.3)
Baxter Suction	(11.2 ± 4.3)	(3.8 ± 1.4)
Baxter Suction (fusion)	(11.1 ± 4.2)	(3.8 ± 1.4)

Table 4: Direct Regression performance of four different grippers in simulation.

4.3 Additional Gripper Experiments

In order to ensure there is nothing particular about the Sawyer two-finger gripper we used in our experiments, we further test our best performing method with three additional simulated grippers, which are illustrated in fig. 5. As such, we train the direct regression model on a second two-finger gripper, a three-finger gripper, and a suction gripper. We evaluate the performance on each using our simulation testing procedure and report the results in table 4. We also append the performance on the Sawyer two-finger gripper we reported in the main paper for reference. From the table, see that the test performance on the tree grippers is very similar, which indicates that our method could be applied to any gripper requirement. We note that setting up each new gripper training is very simple and only requires the gripper model to be replaced in simulation, which can be accomplished in a time of the order of 5 minutes. As such, considering the dataset generation and training with the direct regression method, the additional time required to get the direct regression method working on a new gripper is approximately 2 hours in total.

References

[1] V. Nair and G. E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *International Conference on Machine Learning (ICML)*, 2010.

[2] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, 2015.

[3] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of Machine Learning Research (JMLR)*, 2014.

[4] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2015.

[5] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv e-prints*, 2014.

[6] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel. Deep Spatial Autoencoders for Visuomotor Learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[7] A. A. Minai and R. D. Williams. On the derivatives of the sigmoid. *Neural Networks*, 1993.

[8] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal Loss for Dense Object Detection. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[9] C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine. One-Shot Visual Imitation Learning via Meta-Learning. In *Conference on Robot Learning (CoRL)*, 2017.

[10] J. Wang and E. Olson. AprilTag 2: Efficient and robust fiducial detection. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[11] D. Malyuta. Guidance, Navigation, Control and Mission Logic for Quadrotor Full-cycle Autonomy. Master thesis, Jet Propulsion Laboratory, 2017.

[12] A. Paszke et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Conference on Neural Information Processing Systems (NeurIPS)*. 2019.

[13] R. Y. Tsai, R. K. Lenz, et al. A new technique for fully autonomous and efficient 3D robotics hand/eye calibration. *IEEE Transactions on Robotics and Automation*, 1989.

[14] F. C. Park and B. J. Martin. Robot sensor calibration: solving AX= XB on the Euclidean group. *IEEE Transactions on Robotics and Automation*, 1994.

[15] R. Horaud and F. Dornaika. Hand-eye Calibration. *The International Journal of Robotics Research (IJRR)*, 1995.

[16] K. Daniilidis. Hand-Eye Calibration Using Dual Quaternions. *The International Journal of Robotics Research (IJRR)*, 1999.

[17] Pupil-AprilTags. URL `https://pypi.org/project/pupil-apriltags/`.