Coarse-to-Fine for Sim-to-Real: Sub-Millimetre Precision Across Wide Task Spaces
Supplementary Material

Eugene Valassakis, Norman Di Palo, and Edward Johns

1 Overview

In this document we detail supplementary information on our work. In section 2, we present details about our policy implementations, including neural network architectures and training hyperparameters. In section 3, we present in detail the expert policies used to generate the states and labels used in training. In section 4, we provide further details about our simulation environments, enumerating all the relevant simulator aspects that are being randomised in order to overcome the reality gap.

2 Network Details and Hyperparameters

Our main network architecture consists of an encoder-decoder that first encodes input images into a keypoint representation [2] before decoding them into a depth reconstruction and a control action outputs. In some experiments, we also use a standard convolutional neural network (CNN) architecture that consists of a simple convolutional encoder without any depth reconstruction. In all cases, the image inputs are taken from the wrist-mounted camera, and the output control actions consist of velocity commands in end-effector space. In this section, we describe implementation details for our network architectures, followed by network training implementation details.

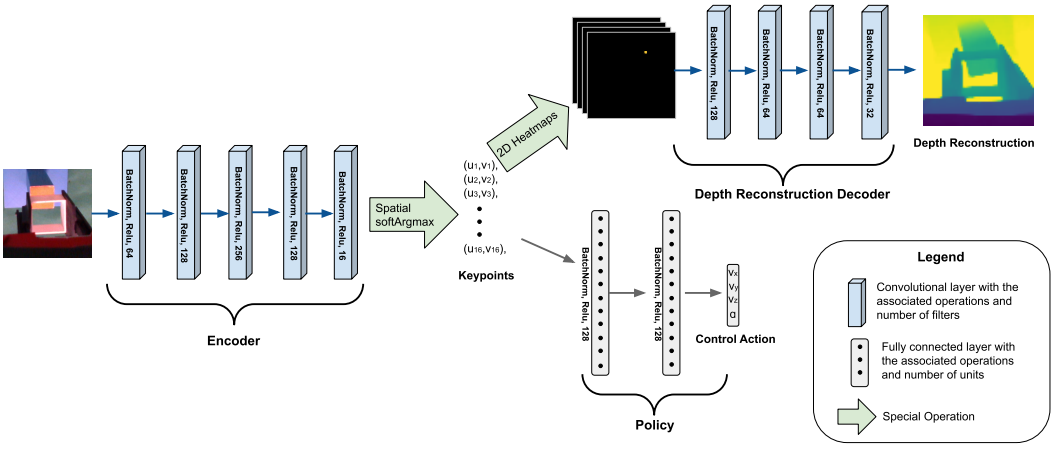

Figure 1: Our keypoint-based, encoder-decoder architecture

2.1 Network Architecture Implementation Details

2.1.1 Encoder Decoder Architectures

Our encoder-decoder networks are inspired from the deep spatial autoencoder architecture [2], and encode an image into a set of keypoints that are input into a policy and a depth reconstruction branch. This is illustrated in Fig 1.

Our encoder consists of 5 convolutional layers, each followed by 2-dimensional Batch Normalisation (BatchNorm) [4] and ReLU activations [7]. The layers have [64, 128, 256, 128, 16] filters of kernel size [5, 3, 3, 3, 3], and stride of [2, 1, 2, 1, 1], respectively. From the last convolutional layer, 16 keypoints are extracted using the spatialSoftArgmax operation [2].

Our depth reconstruction decoder takes in those keypoints and first constructs 16, 64×64 heatmaps, using Laplace distributions with fixed scale set to 0.05, concatenated into a $64 \times 64 \times 16$ tensor. This tensor is then processed by 5 convolutional layers, with BatchNorm and ReLU activations in all but the last layer which has a sigmoid activation. The intermediate layers have [128, 64, 64, 32] filters, and the final one has 1 channel for the depth reconstruction map (or 2 when using concatenated stereo IR inputs to reflect the depth reconstructions from both point of views). The kernel sizes for each layer are [5, 3, 3, 3, 3] and the stride is 1 throughout.

Our policy takes in the flattened keypoints and is composed of 3 fully-connected layers with [128, 128, 4] units. Each hidden layer is followed by BatchNorm and ReLU activations, and the final layer has a simple linear

activation.

We note that (1) for experiments where we used an encoder-decoder architecture without keypoints we kept the same structure but removed the spatialSoftArgmax [2] and Laplacian map operations, using convolutional features directly instead, and (2) for our fully end-to-end experiments, the dimensionality of the input images, Laplacian maps and depth reconstruction outputs is 128×128.

2.1.2 CNN Architecture

In some of our experiments we also compare our encoder-decoder architectures with a simple convolutional encoder without any depth reconstruction, consisting on a standard CNN architecture. In making this architecture we aimed to keep the number of parameters used as close as possible to our encoder-decoder networks, while increasing its performance to the best of our ability. Inspired by widely used and successful architectures [6], our CNN is composed of 5 convolutional layers, each with BatchNorm, ReLU activation, and Dropout [8]. The layers have [32, 64, 128, 256] and 32 channels respectively, with kernel sizes [5, 3, 3, 3, 1] and stride [1, 1, 2, 2, 1]. The Dropout probability is set to 0.1. The feature map output from the last convolutional layer is then flattened into a 1-dimensional vector further processed by two fully connected layers of [64, 4] units, respectively. The hidden layer is followed by a ReLU activation and Dropout, while the output layer has a simple linear activation.

2.2 Training Details

We use the same training hyperparameters for all our networks: the input images are normalised such that the input features are in the $[0, 1]$ range. We further apply min-max normalisation to the output actions, so that they are also in the $[0, 1]$ range. We use a batch size of 32, and the Adam optimiser [5] with a learning rate of $1e-3$ that is halved every 10 epochs. Finally, we apply an L2 weight decay regularisation with a weight of $1e-6$. In an attempt to minimise the effect of training hyperparameters on the final performance, for the networks we trained we test 30 and 40 epochs of training, keep the best performing version for the final experiments.

3 Expert Policy Details

In this section we describe in detail the expert policies we used in order to generate the demonstrations when gathering our datasets.

For the coarse-to-fine policies, the demonstrations start at an initial pose sampled from around the *bottleneck*. Specifically, the initial position and orientation offsets are sampled from a $2.5cm \times 2.5cm \times 1.5cm$ rectangular volume and a $[-0.45, 0.45]rad$ range, respectively. The expert policy is then deployed to carry out the insertion. It relies on an *alignment waypoint* to align the ring with the peg before completing the insertion through a downward push. This alignment waypoint has the same orientation and x, y position as the bottleneck but is on a different position on the vertical axis. The expert policy consists of a proportional controller in end-effector space reaching to first the *alignment waypoint*, and second to a final goal pose that completes the insertion. As such, the linear velocity $\mathbf{v}$ is obtained from the position error $\mathbf{e}$ by applying some proportional gains $\mathbf{k}$: $\mathbf{v} = \mathbf{e} \odot \mathbf{k}$, where $\odot$ is element-wise multiplication. The angular speed ω is pegged to the linear velocity $\mathbf{v}$ such that , $\omega = \frac{\alpha}{d/||\mathbf{v}||}$, where α and d are the remaining angular and linear distances to the current target, respectively. For the screw insertion task, the expert policy also has a second stage with a twisting motion of constant angular velocity of $0.0873\ rad/s$, which is engaged after insertion.

In order to allow for a good coverage of the state space, encourage robustness, and limit dataset imbalances, we found it important to (1) randomise the position of the alignment waypoint, as well as how closely it needs to be reached before it switches to the final target goal (see table 3), (2) limit the controller's policy speed to a maximum of $5mm/s$ with a further cut-off of the resulting labels at $2mm/s$, and (3) increase the gains of the x, y components in the proportional controller relative to the z component, setting $\mathbf{k} = (2.5, 2.5, 1)$.

4 Simulation Randomisation Details

In this section we describe all the simulation aspects that we randomise during our domain randomisation (DR) procedure [10, 11]. We break these down into three categories: Visual randomisation in 4.1, Dynamics randomisation in 4.2, Geometry randomisation in 4.3.

4.1 Visual Randomisation

When transferring image-based policies between simulation and the real world perhaps the most obvious hurdle to overcome are the visual differences between the two domains.

These are most often due to (1) colours which can vary significantly in the real world, depending on external conditions such as lighting or sensor properties, (2) textures, with real world objects rarely being composed of smooth monochrome surfaces, and (3) lights including shadows and the colour variations that they induce in the surface of objects. In order to overcome all these differences we use visual domain randomisation [10], with the parameters we randomised being described in Table 1.

When using an assisted stereo depth images however, the differences between the simulation and the real world mostly manifest as noise in the depth maps and missing depth values. In order to account for these we build on the procedure described in [9] and use the depth noise profile described in [1]. In Table 2, we describe in some more detail the parameters we randomise in order to get variability in the depth maps.

Table 1: Table describing the aspects of the simulation randomised for visual DR.

Randomised Aspect	**Description**
Light source position	At each of timestep, two omnidireactional light sources are positioned relative to the object being manipulated, by choosing $r \sim [1.5, 3.5]$ meters, $\theta \sim [0.26, \pi/2]$ radians, and $\phi \sim [-\pi, \pi]$ radians in spherical coordinates.
Light source colour	At each timestep, the colour of the light emitted is set by sampling the corresponding R,G,B values.
Ambient Light intensity	At each timestep, the ambient light intensity is varied by setting CoppeliaSim's *sim_arrayparam_ambient_light* parameter to a white light with an intensity sampled uniformly from $[0.7, 1.4]$.
Colours Shift	At each timestep, the colour of each component of the simulation is randomised around its mean by shifting the R,G,B channels independently by up to 30%.
Colours Damping	The brightness in the colour of each simulation component is lowered by up to 20% at each simulation timestep to help avoid the over saturation of pixel values.
Grayscale textures	Random grayscale images are sampled uniformly to apply textures to the different simulation components. To obtain these images we download the folder found here* [3] , which contains a large number of images that can be used as textures. We then convert all the images to grayscale, increase their brightness by a random factor $f_b \sim [1.5, 2.5]$, and decrease their contrast by a random factor $f_c \sim [0.2, 0.5]$.

*https://github.com/tianheyu927/mil/blob/master/scripts/get_data.sh , made available by [3]

Table 2: Table describing the parameters randomised when generating synthetic depth images.

Randomised Aspect	**Description**
Proportional depth noise	The value of the depth d at each pixel is augmented with gaussian noise of mean 0 and standard deviation $0.001063 + 0.0007278 * d + 0.003949 * d^2$ [1]
Minimum depth range	The minimum depth range is sampled uniformly within $[16.0cm, 18.0cm]$. Any depth pixel with a value smaller than the minimum is set to 0.
Morphological opening kernel sizes	The kernel size for each morphological opening operation on the depth masks are randomly chosen from $\{2, 3, 5\}$.
Morphological dilation kernel sizes	The kernel size for each morphological dilation operation on the depth masks are randomly chosen from $\{0, 3, 5, 7, 9, 11\}$.
Medial filtering kernel sizes	The kernel size for each median filtering operation on the depth masks are randomly chosen from $\{0, 3, 5, 7\}$.
Gaussian filtering scale	The standard deviation for the gaussian filtering operation on the depth maps is sampled uniformly within $[0., 3.5]$.
Perlin noise parameters	We wrap our depth maps using perlin noise with the procedure and parameters described in [12] and [9].

4.2 Dynamics Randomisation

Differences in the dynamics between the simulation and the real world result in the actions taken in the two domains to have different effects. This can result from various sources, such as differences in the physical properties of the two domains or hard to model physical processes including gear backlash and time delays that occur due to the continuous time nature of the real domain versus the stopping time nature of simulations. We note that in our setup the biggest difference seemed to emerge from computation times, where even though both control loops were set to 20Hz, in practice the real execution frequency could become up to several times slower due to the time needed to do computations on each iteration. In order to account for any such differences, we add diversity in the distribution of states visited in simulation, which has been shown to be an effective approach. We diversify the state visitation by (1) augmenting the control actions at each simulation timestep with various sources of noise, and (2) adding diversity in the expert policy trajectories. Both aspects are described in Table 3.

Table 3: Table describing the simulation aspects randomised for overcoming the dynamics discrepancies between the simulation and the real world.

Randomised Aspect	**Description**
Additive action noise	At each time step, white gaussian noise is added to each action. Each dimension of an action is treated independently, with the linear velocity noise having a standard deviation of 2 mm/s and the angular velocity noise having a standard deviation of 0.035 rad/s.
Action rescaling factor	At each time step, each action dimension is re-scaled by up to 5%. All dimensions are rescaled by the same factor at a given timestep.
Systematic action noise	At each time step, a systematic error of up to 0.5 mm/s is added to each linear velocity component of the action, and one of up to 0.0035 rad/s is added to the angular velocity component. This error is sampled uniformly at the beginning of every new trajectory, and remains constant throughout the episode.
Alignment waypoint position	The position of the alignment waypoint along the vertical axis is randomised at the beginning of each episode, varying between 1.1cm and 2.1cm above the top of the peg.
Alignment acceptance radius	How closely the ring needs align with the alignemnt waypoint before switching to reach towards the final goal position is varying between 0.5mm and 2.5mm, and is randomised at the beginning of each episode.

4.3 Geometry Randomisation

Differences in the geometry between the simulated and real world environments can also have a large effect on the final policies. In our work, we assume that the dimensions of the models (mesh files) imported in simulation are accurate, but that their relative poses may be subject to errors. We overcome these by randomising the associated simulation parameters, which we describe in Table 4.

Table 4: Table describing the randomised aspects in the geometry of the simulated environments

Randomised Aspect	**Description**
Camera pose	At each time step, the camera pose is randomised relative to the end-effector, with all the sensors on the camera (RGB, IR, Depth) getting the same transformation. The camera position is sampled uniformly inside a $0.0085938m \times 0.01780764m \times 0.0049902m$ rectangular volume centred around a mean obtained by calibration. The camera orientation is also sampled uniformly inside a $0.06698968rad \times 0.03490336rad \times 0.00682072rad$ volume, centred around an orientation mean obtained by calibration. We note that each side of these volumes corresponds to four times the standard deviation of the corresponding calibration estimate.
Vision sensor relative pose	At each time step, we also randomise the pose of each vision sensor relative to each other. After applying the camera randomisation to all sensors, each is further independently randomly shifted within a $(0.6mm)^3$ and a $(0.008rad)^3$ volumes centered around their current positions and orientations.
Depth light emitter pose	The depth light emitter pose is first randomised with the whole camera along with the vision sensors. Then, its position is further randomly shifted within a $20mm \times 0.3mm \times 0.3mm$ volume, and its orientation within a $(0.004rad)^3$ volume.
Ring pose	At each time step, the ring pose relative to the end-effector is randomised by sampling uniformly its position and orientation around its initial pose. For the square ring, its position is sampled within a $0.4mm \times 2.0mm \times 3.0mm$ volume and its orientation within a $0.026rad \times 0.017rad \times 0.008rad$ volume. For the round and screw rings, those volumes are increased to $0.4mm \times 8.0mm \times 4.0mm$ and $0.21rad \times 0.034rad \times 0.35rad$ respectively, since they were harder to accurately place in the real world gripper.
Gripper finger position	At each time step, the gripper finger positions are randomised relative to the end effector. Both get the same transformation, sampled uniformly within a $4mm \times 2mm \times 1mm$ volume.
Gripper finger relative position	The position of each finger is further randomised independently within a $(0.5mm)^3$ volume.
Finger pad pose	The pose of each finger pad is independently randomised around their initial pose. The pad positions are sampled within a $(0.5mm)^3$ volume, and their orientations are sampled within a $0.02rad \times 0.02rad \times 0.025rad$ volume.
Image crop position	The images input to the networks are cropped around the end-effector position in the image. However, locating the exact position of the end-effector in the real images is subject to noise, so we also randomise the end-effector position in the simulated images to account for this. As such, on each simulated image we shift the crop centre by up to 9 pixels on both the x and y directions independently before applying the image crop.

References

[1] M. S. Ahn, H. Chae, D. Noh, H. Nam, and D. Hong. Analysis and noise modeling of the intel realsense d435 for mobile robots. In *2019 16th International Conference on Ubiquitous Robots (UR)*, pages 707–711. IEEE, 2019.

[2] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE, 2016.

[3] C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine. One-shot visual imitation learning via meta-learning. In *Conference on Robot Learning*, pages 357–368. PMLR, 2017.

[4] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

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

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[7] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

[8] 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*, 15(1):1929–1958, 2014.

[9] S. Thalhammer, K. Park, T. Patten, M. Vincze, and W. Kropatsch. Sydd: Synthetic depth data randomization for object detection using domain-relevant background. *TUGraz OPEN Library*, pages 14–22, 2019.

[10] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 23–30. IEEE, 2017.

[11] E. Valassakis, Z. Ding, and E. Johns. Crossing the gap: A deep dive into zero-shot sim-to-real transfer for dynamics. In *International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[12] S. Zakharov, B. Planche, Z. Wu, A. Hutter, H. Kosch, and S. Ilic. Keep it unreal: Bridging the realism gap for 2.5 d recognition with geometry priors only. In *2018 International Conference on 3D Vision (3DV)*, pages 1–11. IEEE, 2018.