Transferring End-to-End Visuomotor Control from Simulation to Real World for a Multi-Stage Task

Stephen James slj12@imperial.ac.uk
Andrew J. Davison a.davison@imperial.ac.uk
Edward Johns e.johns@imperial.ac.uk

Dyson Robotics Lab, Imperial College London

Abstract: End-to-end control for robot manipulation and grasping is emerging as an attractive alternative to traditional pipelined approaches. However, end-to-end methods tend to be either slow to train, exhibit little or no generalisability, or lack the ability to accomplish long-horizon or multi-stage tasks. In this paper, we show how two simple techniques can lead to end-to-end (image to velocity) execution of a multi-stage task that is analogous to a simple tidying routine, without having seen a single real image. This involves locating, reaching for, and grasping a cube, then locating a basket to drop the cube in. The first technique is to utilise the full state from a simulator to collect a series of control velocities which accomplish the task. The second technique is to utilise domain randomisation to allow the controller to generalise to the real world. Our results show that we are able to successfully accomplish the task in the real world with the ability to generalise to novel environments, including those with novel lighting conditions and distractor objects, and the ability to deal with moving objects, including the basket itself. We believe our approach to be simple, highly scalable and capable of learning long-horizon tasks that have so far not been shown with the state-of-the-art in end-to-end robot control.

Keywords: Manipulation, End-to-End Control, Domain Randomisation

1 Introduction

An emerging trend for robot manipulation and grasping is to learn controllers directly from raw sensor data in an end-to-end manner. This is an alternative to traditional pipelined approaches which often suffer from propagation of errors between each stage of the pipeline. These end-to-end approaches have had success in both real world [10, 14, 13] and in simulated worlds [3]. Learning end-to-end controllers in simulation is an attractive alternative to using physical robots due to the prospect of scalable, rapid, and low-cost data collection. However, these simulation approaches are of little benefit if we are unable to transfer the knowledge to the real world. What we strive towards are robust, end-to-end controllers that are trained in simulation and can run on the real world without having seen a single real world image.

In this paper, we accomplish the goal of transferring end-to-end controllers to the real world and demonstrate this by learning a long-horizon multi-stage task that is analogous to a simple tidying task, and involves locating a cube, reaching, grasping, and locating a basket to drop the cube in. This is accomplished by using linear paths constructed via inverse kinematics (IK) in the Cartesian space to construct a dataset that can then be used to train a reactive neural network controller that continuously accepts images along with joint angles and outputs motor velocities. We show that with the addition of auxiliary outputs (inspired by [4]) for positions of both the cube and gripper improves task performance. Transfer is made possible by simply using domain randomisation, such that through a large amount of variability in the appearance of the world, the model is likely to generalise to real world environments. Figure 1 summarises the approach, whist the video demonstrates the success of the final controller in the real world [1].

[1] Video: https://youtu.be/X3SD56hporc

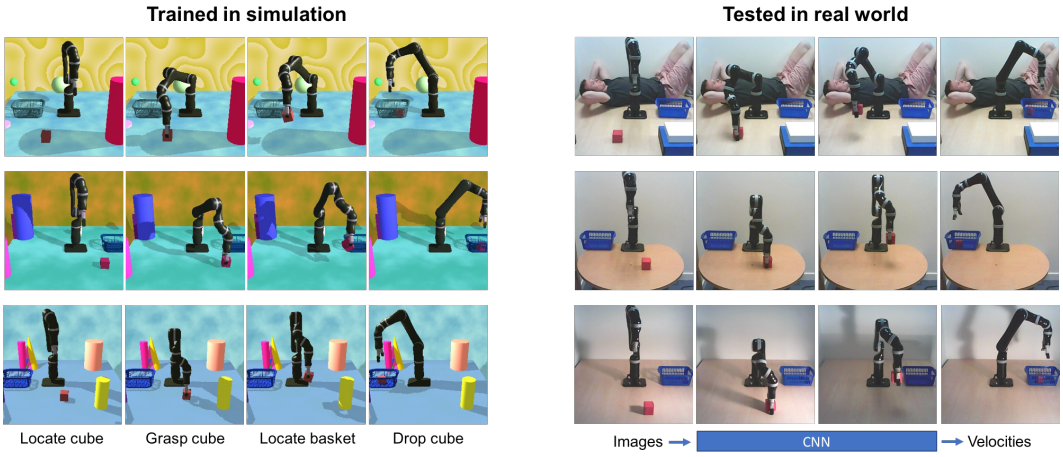


Figure 1: Our approach uses simulation to collect a series of control velocities to solve a multi-stage task. This is used to train a reactive neural network controller that continuously accepts images and outputs motor velocities. By using domain randomisation, the controller is able to run in the real world without having seen a single real image.

Our final model is not only able to run on the real world, but can accomplish the task with variations in the position of the cube, basket, camera, and initial joint angles. Moreover, the model is invariant to clutter, lighting conditions, changes in the scene, and moving objects (including people).

2 Related Work

End-to-end methods for robot control are generally trained using Reinforcement Learning (RL) or Guided Policy Search (GPS). In the past few years, classic RL algorithms have been fused with function approximators, such as neural networks, to spawn a deep reinforcement learning domain, that is capable of playing games such as Go [20] and Atari [12] to a super-human level. There have been advances in applying these techniques to both simulated robotic platforms [17, 5], and real world platforms [3], but fundamental challenges remain, such as sample inefficiency, slow convergence, and the algorithms sensitivity to hyperparameters. There have been attempts at transferring trained polices from the real world following training in simulation, but these attempts have either failed [5], or required additional training in the real world [18]. Although applying RL can work well for computer games and simulations, the same cannot be said for real world robots, where the ability to explore randomly can grow intractable as the complexity of the task increases. Our method uses the full state of the simulation to produce paths for supervision without the need for exploration, and as a result converges in far less time in comparison to RL methods which often take millions of iterations. Moreover, our method is able to generalise to new environments which is currently one of the challenges of RL.

A slightly different approach is guided policy search (GPS) [9, 8], which in addition to classic control benchmarks, has achieved success in flight [24, 7], and robot manipulation [10, 14, 13]. GPS provides a means to optimise non linear policies, such as neural networks [8], without the need to compute policy gradients. GPS uses guiding samples produced by a *teacher* algorithm, such as trajectory optimisers or trajectory-centric reinforcement learning methods, to train policies in a supervised manner. In addition to this, GPS adapts the guiding samples produced by the teacher so that they are more suited for training the final policy. Unlike most RL methods, these approaches can be trained on real world robotic platforms, but therefore have relied on human involvement which limits their scalability. To overcome human involvement, GPS has been used with domain adaptation of both simulated and real world images to map to a common feature space for putting a rope around a hook [22]; though this still requires images from the real world. Unlike GPS methods, our method has never seen a real world image before, but instead learns its task in simulation which we transfer over to the real robot. Moreover, our generated dataset can be reused to train a number of models in parallel which is not possible with GPS as the training data provided by the locally optimised trajectories are altered with respect to the global policy, and vice versa.

Figure 2: A collection of generated environments as part of our domain randomisation method.

An alternative approach for getting data from simulation is to instead use a large number of real world robots [11, 2, 16]. This was the case for [11], where 14 robots were run for a period of 2 months and collected $800,000$ grasp attempts by randomly performing grasps. A similar approach was used to learn a predictive model in order to push objects to desired locations [2]. Although the results are impressive, their is some doubt in its scalability with the high purchase cost of robots, in addition to the question of how you would get data for more complex and long-horizon tasks. Moreover, these solutions cannot generalise to new environments without also training them in that same environment.

There exists a number of works that do not specifically learn end-to-end control, but do in fact use simulation to learn behaviours with the intention to use the learned controller in the real world. Such works include [6], which uses the depth images from simulated 3D objects to train a CNN to predict a score for every possible grasp pose. This can then be used to locate a grasp point on novel objects in the real world. Another example is [1], where deep inverse models are used on the simulated control policy to decide on suitable actions to take in the real world for a back-and-forth swing of an arm using position control.

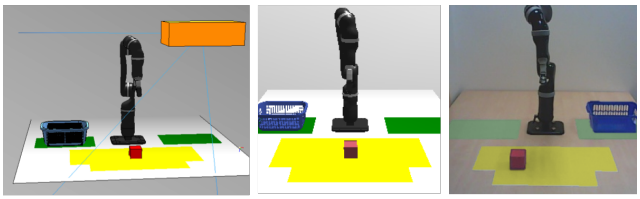
Another approach is to use transfer learning to explicitly adapt from simulated data to real world data. In [19], the focus is on learning collision-free flight in simulated indoor environments using realistic textures sampled from a dataset. They show that the trained policy can then be directly applied to the real world. Their task differs to ours in that they do not require hand-eye coordination and do not need to deal with a structured multi-stage task. Moreover, rather than sampling from a dataset of images, ours are procedurally generated, which allows for much more diversity. During development of this work, a paper emerged that also used domain randomisation [21], except that the focus was on pose estimation rather than end-to-end control. We operate at the lower level velocity control to accomplish a multi-stage task that requires not only pose estimation, but also target reaching, grasping, and control. In addition, we show that our learned controller can work in a series of stress tests, including that of human presence.

3 Approach

Our aim is to create an end-to-end reactive controller that does not require real world data to train, and is able to learn complex behaviours in a short period of time. To achieve this, we generate a large number of shortest path trajectories in simulation, which we then use to train a controller to output motor velocities, which are then mapped to torques through a PID controller to complete a multi-stage task. Through the use of domain randomisation during the data generation phase, we are able to run the controller in the real world without having seen a single real image. We now describe in detail the dataset generation and training method.

3.1 Data Collection

The success of this work comes down to the way in which we generate the training data (Figure 2). Our approach uses a series of linear paths constructed in the Cartesian space via inverse kinematics



(a) (b) (c)

Figure 3: Variations in the positioning of the cube, basket, and camera. The yellow area represents the possible locations of the cube, the green areas represents the locations of the basket, and the orange cuboid represents possible locations of the camera. The objects are positioned such that the whole object is within the highlighted area. Both (a) and (b) are from simulation, where (a) shows the variations of the controllers camera, and (b) shows the variations of the scene from the perspective of the controllers camera. (c) shows these variations visualised in the real world.

(IK) in order to construct the task sequence. At each simulation step, we record motor velocities, joint angles, gripper actions (open or close command), cube position, gripper position, and camera images. For the task of tidying, we split it into 5 stages, and henceforth, we refer to the sequence of stages as an *episode*. At the start of each episode, the cube and basket is placed randomly within an area that is shown in Figure 3.

In the first stage of the task, we place a waypoint above the cube and plan a linear path which we then translate to motor velocities to execute. Once this waypoint has been reached, we execute the second stage by performing a close action on the gripper. The third stage sets a waypoint a few inches above the cube, and we plan and execute a linear path in order to lift the cube. The fourth stage places a waypoint above the basket, where we plan and execute a final linear path to take the grasped cube above the basket. Finally, the fifth stage simply performs a command that opens the gripper to allow the cube to fall into the basket. A check is then carried out to ensure that the location of the cube is within the basket; if this is the case, then we save the episode. The data generation method can be run on multiple threads, which allows enough data to train a successful model to be collected within a matter of hours, though increasing the number of threads would further reduce this time.

Using this approach, it is already possible to train a suitable neural network to learn visuomotor control of the arm and perform the task to succeed 100% of the time when tested in simulation. However, we are concerned with applying this knowledge to the real world so that it is able to perform equally as well as it did in simulation. By using domain randomisation, we are able to overcome the reality-gap that is present when trying to transfer from the synthetic domain to the real world domain. For each episode, we list the environment characteristics that are varied, followed by an explanation of why we vary them:

- The colour of the cube, basket, and arm components are sampled from a normal distribution with the mean set as close to the estimated real world equivalent; though these could also be sampled uniformly.
- The position of the camera, light source (shadows), basket, and cube are sampled uniformly.
- The height of the arm is sampled uniformly from a small range.
- The starting joint angles are sampled from a normal distribution with the mean set such that the arm starts in a position similar to Figure 3.
- We make use of Perlin noise [15] composed with functions (such as sine waves) to generate textures that are applied to the table and background of the scene.
- We add random primitive shapes as clutter with random colours, positions, and sizes sampled from a uniform distribution.

A selection of these generated environments from the perspective of the network can be seen in Figure 2. We chose to vary the colours and positions of the objects in the scene as part of a basic domain randomisation process. Slightly less obvious is varying the height of the arm, as we cannot be sure that the position within the simulation is accurate. In order to successfully run these trained models in the real world, we also must account for non visual issues, such as the error in the starting

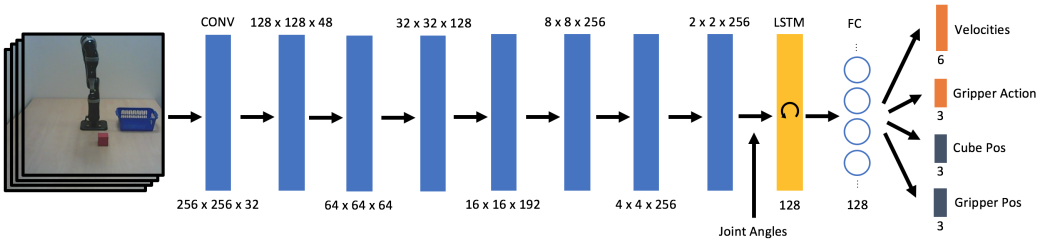


Figure 4: Our network architecture continuously maps sequences of the past 4 images to motor velocities and gripper actions, in addition to 2 axillary outputs: the 3D cube position and 3D gripper position. These axillary outputs are not used during testing, but are instead present to help the network learn informative features.

position of the joints when run on the real world. Although we could send the real world arm to the same starting position as the synthetic arm, in practice the joint angles will be slightly off, and this could be enough to put the network in states that are unfamiliar and lead to compounding errors along its trajectory. It is therefore important that the position of the robot at the start of the task is perturbed during data generation. This leads to controllers that are robust to compounding errors during execution, since a small mistake on the part of the learned controller would otherwise put it into states that are outside the distribution of the training data. This method can also be applied to the generated waypoints, but in practice this was not needed. We use procedural textures rather than plain uniform textures, as background colours in the real world are more complex than uniform textures.

3.2 Training

The network, summarised in Figure 4, consists of 8 convolutional layers each with a kernel size of 3×3, excluding the last, which has a size of 2×2. Dimensionality reduction is performed at each convolutional layer by using a stride of 2. Following the convolutional layers, the output is concatenated with the joint angles and then fed into an LSTM module (we will discuss the importance of this in the results). Finally, the data goes through a fully connected layer of 128 neurons before heading to the output layer.

The network outputs 6 motor velocities, 3 gripper actions, and 2 auxiliary outputs: cube position and gripper position. We treat the 3 gripper actions as a classification problem, where the outputs are $\{open, close, no\text{-}op\}$. During testing, the auxiliary outputs are not used at any point, but are present to aid learning and conveniently help debug the network. By rendering a small marker at the same positions as the auxiliary outputs, we are able to observe where the controller estimates the cube is, in addition to where it estimates its gripper is. This is helpful when it comes to determining whether the pose estimation is inaccurate, or whether the motor actions are inaccurate.

Our loss function $\mathcal{L}_{Total}$ is a combination of the mean squared error of the velocities (V and gripper actions (G) together with the gripper position (GP) auxiliary and cube position (CP) auxiliary, giving

$$\mathcal{L}_{Total} = \mathcal{L}_V + \mathcal{L}_G + \mathcal{L}_{GP} + \mathcal{L}_{CP}. \tag{1}$$

The loss did not use weighted terms as the scales were appropriate for learning.

4 Results

In this section we present results from a series of experiments, not only to show the success of running the trained models in different real world settings, but also to show what aspects of the domain randomisation are most important for a successful transfer. Figure 5 shows an example of the controller's ability to generalise to new environements in the real world, though many more are shown in the video[2]. We focused our experiments to answer the following questions:

[2]Video: https://youtu.be/X3SD56hporc

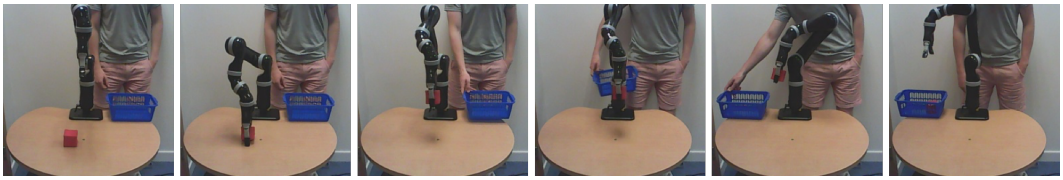


Figure 5: A sequence of images showing the networks ability to generalise. Whilst standing in the view of the camera, it is able to grasp the cube and proceed to the basket. As the arm progresses towards the basket, we move the basket to the other side of the arm. Despite this disruption, the network alters its course and proceeds to the new location. The network has never seen humans, moving baskets, or this table.

1. How does performance vary as we alter the dataset size?
2. How robust is the trained controller to new environments?
3. What is most important to randomise during domain randomisation?
4. Does the addition of auxiliary outputs improve performance?
5. Does the addition of joint angles as input to the network improve performance?

We first define how we evaluate the controller in both the simulation and real world. We place the cube using a grid-based method, where we split the area in which the cube was trained into a grid of $10cm \times 10cm$ cells. For each cell, we run the network twice, once with the basket on each side of the arm. Using the grid, the cube can be in one of 16 positions, and we run a trial twice with the basket on each side of the arm, resulting in 32 trials; therefore, all of our results are expressed as a percentage based on 32 trials. Figure 6 summarises the testing conditions, where each square represents a position, and the two triangles represent whether the basket was on the left or right side of the robot at that position.

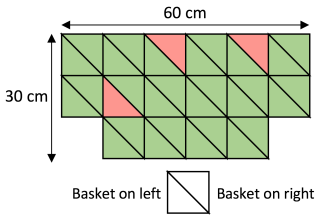


Figure 6: Controller evaluation example. Green signifies a success, whilst red signifies a failure. In this case, success would be 91%.

To answer the first question of how dataset size effects performance, we train several instances of the same network on datasets sizes ranging from 100,00 to 1 million images to accomplish the task in a non cluttered environment (such as in Figure 3), and plot in Figure 7 the success rates in both simulation and real world for each set of trained weights. The graph shows that a small dataset of 200,000 images achieves good performance in simulation, but 4 times more data is needed in order to achieve approximately the same success rate in the real world. Interestingly, both simulation and real world achieve 100% for the first time on the same dataset size (1 million).

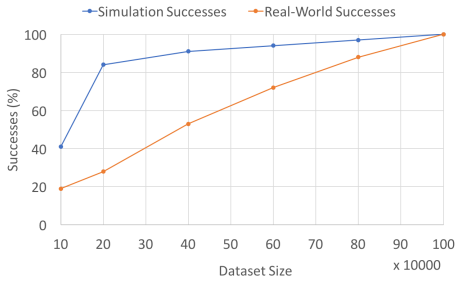


Figure 7: How dataset size effects performance in both simulation and real world.

We now turn our attention to Figure 8 and Table 1 which provide a summary of our results of the remaining research questions. First, we discuss the results in the top half of Table 1, where we evaluate how robust the network is to changes in the testing environment. Successes are categorised into cube vicinity (whether the arm got within $\approx 2cm$ of the cube), cube grasped, and full task (whether the cube was reached, grasped and dropped into the basket). The first two results show our solution achieving 100% when tested in both simulation and the real world. Although the network was trained with clutter, it was not able to achieve 100% with clutter in the real world. Note that the controller does not fail once the cube has been grasped, but rather fails during the reaching or grasping. The majority of failures in this case where when the cube was closest to the clutter in the scene. In the moving scene test, we place a 14-inch fan in the background that rotates back and

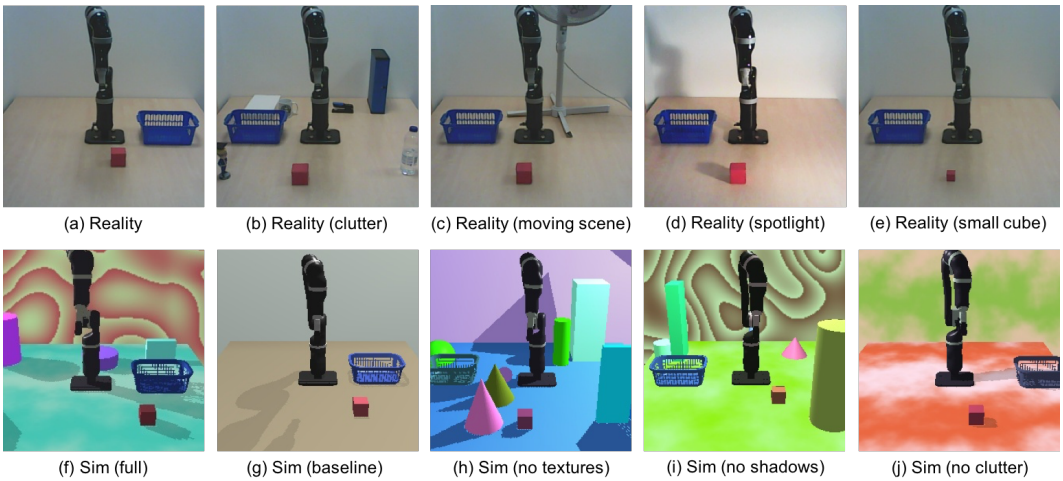


Figure 8: Images (a) to (e) are taken from the real world, whilst images (f) to (j) are taken from simulation. The real world images shows the testing scenarios from Table 1, whilst the simulated images shows samples from the training set from Table 1.

forth. Reaching was not affected in this case, but grasping performance was affected. The moving camera test was performed by continuously raising and lowering the height of the tripod where the camera was mounted within a range of 2 inches. Although never experiencing a moving scene or camera motion during training, overall task success remained high at 81% and 75% respectively. To test invariance to lighting conditions, we aimed a bright spotlight at the scene and then moved the light as the arm operates. The results show that the arm was still able to recognise the cube, getting a 84% cube vicinity success rate, but accuracy in the grasp was affected, resulting in the cube only being grasped 56% of the time. This was also the case when we replaced the cube with one that was twice as small as the one seen in training. The 89% vicinity success in comparison to the 41% grasp success shows that this new object was too different to perform accurate grasp, and would often brush the cube.

The bottom half of Table 1 focuses on the final 3 questions regarding what aspects are important to randomise, and whether auxiliary tasks and joint angles improve performance. Firstly, we wish to set a baseline that shows that naively simulating the real world is difficult for getting high success in the real world. We generate a dataset based on a scene with colors close to the real world. The baseline is unable to succeed at the overall task, but performs well at reaching. This conclusion is in line with other work [5, 23]. It is clear that having no domain randomisation does not transfer well for tasks that require high accuracy. Whilst observing the baseline in action, it would frequently select actions that drive the motors to force the gripper into the table upon reaching the cube. Training a network without clutter and testing without clutter yields 100%, whilst testing with clutter unsurprisingly performs poorly. A common characteristic of this network is to make no attempt at reaching the cube, and instead head directly to above the basket. We test our hypothesis that using complex textures yields better performance than using colours drawn from a uniform distribution. Although reaching is not effected, both grasping and full task completion degrade to 69% and 44% respectively. Moreover, swapping the table illustrated in Figure 5 leads to complete failure using plain colours.

Without moving the camera during training, the full task is not able to be completed. Despite this, target reaching seems to be unaffected. We cannot guarantee the position of the camera in the real world, but the error will be small enough such that the network is able to reach the cube, but lacks the ability to grasp. Results show that shadows play an important role following the grasping stage. Without shadows, the network can easily become confused by the shadow of both the cube and its arm. Once grasped, the arm frequently raises and lowers the cube, possibly due to the network mistaking the shadow of the cube for the cube itself.

We now observe what aspects of the network contribute to success when transferring the controllers. We alter the network in 3 distinct ways (no LSTM, no auxiliary output, and no joint angles) and evaluate how performance differs. Excluding the LSTM unit from the network causes the network

Scenario		Successes		
Train	Test	Cube vicinity	Cube grasped	Full task
Sim (full)	Sim (full)	100%	100%	100%
Sim (full)	Reality	100%	100%	100%
Sim (full)	Reality (clutter)	88%	75%	75%
Sim (full)	Reality (moving scene)	100%	81%	81%
Sim (full)	Reality (moving camera)	97%	88%	75%
Sim (full)	Reality (spotlight)	84%	56%	56%
Sim (full)	Reality (small cube)	89%	41%	41%
Sim (baseline)	Reality	72%	0%	0%
Sim (no clutter)	Reality	100%	100%	100%
Sim (no clutter)	Reality (with clutter)	53%	0%	0%
Sim (no textures)	Reality	100%	69%	44%
Sim (no moving cam)	Reality	100%	9%	3%
Sim (no shadows)	Reality	81%	46%	19%
Sim (no LSTM)	Reality	100%	56%	0%
Sim (no auxiliary)	Reality	100%	84%	84%
Sim (no joint angles)	Reality	100%	44%	25%

Table 1: Results based on 32 trials from a dataset size of 1 million images (≈ 4000 episodes) run on a square table in the real world. Successes are categorised into cube vicinity (whether the arm got within $\approx 2cm$ of the cube, based on human judgement), cube grasped, and full task (whether the cube was reached, grasped and dropped into the basket). The top half of the table focuses on testing the robustness of the full method whilst the bottom half focuses on identifying what are the key aspects that contribute to transfer. A sample of the scenarios can be seen in Figure 8.

to fail at the full task. Our reasoning for including recurrence in the architecture was to ensure that state was captured. As this is a multi-stage task, we felt it important for the network to know what stage of the task it was in, especially if it is unclear from the image whether the gripper is closed or not (which is often the case). Typical behaviour includes hovering above the cube which then causes the arm to drift into unfamiliar states, or repeatedly calling the close gripper action even after the cube has been grasped. Overall, the LSTM seems fundamental in this multi-stage task. The next component we removed was the auxiliary outputs. The task was able to achieve good performance without the auxiliaries, but not as high as with the auxiliaries. This suggests that the auxiliaries improve the network's grasping accuracy. Finally, the last modification we tested was to exclude joint angles. We found that the joint angles helped significantly in keeping the gripper in the orientation that was seen during training. Excluding the joint angles often led to the angles drifting and causing the orientation of the gripper to be very different to what was seen during training.

5 Conclusions

In this work, we have shown transfer of end-to-end controllers from simulation to the real world, where images and joint angles are continuously mapped directly to motor velocities, through a deep neural network. The capabilities of our method are demonstrated by learning a long-horizon multi-stage task that is analogous to a simple tidying task, and involves locating a cube, reaching the cube, grasping the cube, locating a basket, and finally dropping the cube into the basket. We expect the method to work well for other multi-stage tasks, such as tidying rigid objects, stacking a dishwasher, and retrieving items from shelves, where we are less concerned with dexterous manipulation. However, we expect that the method as is, will not work in instances where tasks cannot be easily split into stages, or when the objects require more complex grasps. Despite the simplicity of the method, we are able to achieve very promising results, and we are keen to explore the limits of this method in more complex tasks.

Acknowledgments

Research presented in this paper has been supported by Dyson Technology Ltd.

References

[1] P. Christiano, Z. Shah, I. Mordatch, J. Schneider, T. Blackwell, J. Tobin, P. Abbeel, and W. Zaremba. Transfer from simulation to real world through learning deep inverse dynamics model. *arXiv preprint arXiv:1610.03518*, 2016.

[2] C. Finn and S. Levine. Deep visual foresight for planning robot motion. *arXiv preprint arXiv:1610.00696*, 2016.

[3] S. Gu, E. Holly, T. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. *arXiv preprint arXiv:1610.00633*, 2016.

[4] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

[5] S. James and E. Johns. 3d simulation for robot arm control with deep q-learning. *arXiv preprint arXiv:1609.03759*, 2016.

[6] E. Johns, S. Leutenegger, and A. J. Davison. Deep learning a grasp function for grasping under gripper pose uncertainty. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 4461–4468. IEEE, 2016.

[7] G. Kahn, T. Zhang, S. Levine, and P. Abbeel. Plato: Policy learning using adaptive trajectory optimization. *arXiv preprint arXiv:1603.00622*, 2016.

[8] S. Levine and P. Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*, pages 1071–1079, 2014.

[9] S. Levine and V. Koltun. Guided policy search. In *ICML (3)*, pages 1–9, 2013.

[10] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

[11] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *arXiv preprint arXiv:1603.02199*, 2016.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[13] W. Montgomery, A. Ajay, C. Finn, P. Abbeel, and S. Levine. Reset-free guided policy search: Efficient deep reinforcement learning with stochastic initial states. *arXiv preprint arXiv:1610.01112*, 2016.

[14] W. H. Montgomery and S. Levine. Guided policy search via approximate mirror descent. In *Advances in Neural Information Processing Systems*, pages 4008–4016, 2016.

[15] K. Perlin. Improving noise. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 681–682. ACM, 2002.

[16] L. Pinto and A. Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 3406–3413. IEEE, 2016.

[17] I. Popov, N. Heess, T. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. Riedmiller. Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv preprint arXiv:1704.03073*, 2017.

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

[19] F. Sadeghi and S. Levine. (cad)2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016.

[20] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[21] 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. *arXiv preprint arXiv:1703.06907*, 2017.

[22] 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. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2016.

[23] F. Zhang, J. Leitner, M. Milford, B. Upcroft, and P. Corke. Towards vision-based deep reinforcement learning for robotic motion control. *arXiv preprint arXiv:1511.03791*, 2015.

[24] T. Zhang, G. Kahn, S. Levine, and P. Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 528–535. IEEE, 2016.