Solving Challenging Dexterous Manipulation Tasks With Trajectory Optimisation and Reinforcement Learning

Henry Charlesworth, Giovanni Montana
Warwick Manufacturing Group, University of Warwick
Coventry, United Kingdom
H.Charlesworth.1@warwick.ac.uk G.Montana@warwick.ac.uk

Abstract

Training agents to autonomously learn how to use anthropomorphic robotic hands has the potential to lead to systems capable of performing a multitude of complex manipulation tasks in unstructured and uncertain environments. In this work, we first introduce a suite of challenging simulated manipulation tasks that current reinforcement learning and trajectory optimisation techniques find difficult. These include environments where two simulated hands have to pass or throw objects between each other, as well as an environment where the agent must learn to spin a long pen between its fingers. We then introduce a simple trajectory optimisation that performs significantly better than existing methods on these environments. Finally, on the challenging PenSpin task we combine sub-optimal demonstrations generated through trajectory optimisation with off-policy reinforcement learning, obtaining performance that far exceeds either of these approaches individually, effectively solving the environment.

1 Introduction

Developing dexterous multi-fingered robotic arms capable of performing complex manipulation tasks is both highly desirable and extremely challenging. In industry today, a majority of robots make use of simple parallel jaw grippers for manipulation. Whilst this works well in structured settings, as we look to develop more autonomous robots capable of performing a wider variety of tasks in more complicated environments it will be necessary to develop considerably more sophisticated manipulators. Probably the most sophisticated and versatile manipulator that we know of is the human hand — capable of tasks ranging from complex grasping to writing and tool use. As such, it is natural to attempt to create robotic hands that mimic the human hand and that can perform similarly complicated manipulation tasks.

This is difficult for traditional robotic control approaches, both with real robotic hands and in simulation. The primary challenges stem from the need to precisely coordinate numerous joints as well as complex, discontinuous contact patterns that can arise at a large number of potential contact points between the hand and the object being manipulated. This motivates the use of techniques that can learn directly via interactions with the environment, without requiring accurate physics models of the hand-object system. In recent years, there has been significant success in applying both reinforcement learning (RL) and gradient-free trajectory optimisation to a range of dexterous manipulation tasks, both in simulation (Plappert et al. 2018; Rajeswaran et al. 2018; Lowrey et al. 2019) and with real robotic hands (Andrychowicz et al. 2018; Akkaya et al. 2019; Nagabandi et al. 2019). Nevertheless, many of these successes involve tasks that by human standards are relatively simple, and complex manipulation tasks still remain a significant challenge.

In this paper we make three primary contributions. Firstly, we introduce a suite of challenging new manipulation tasks based on OpenAI Gym's simulated shadow-hand environments. These include tasks that require coordination between two hands, such as handing over and catching objects, as well as a challenging "PenSpin" environment where a hand has to learn to spin a long pen between its fingers. We demonstrate that many of these tasks are extremely challenging for existing RL/trajectory optimisation techniques, making them potentially useful benchmarks for testing new algorithms.

Secondly, we introduce a simple trajectory optimisation algorithm that is able to significantly outperform existing methods that have been applied to dexterous manipulation, achieving high success fractions on most of the tasks considered.

Finally, we carry out a case study using the "PenSpin" environment. This is arguably the most challenging environment we consider, and the trajectory optimisation algorithm we introduce is only capable of partially solving it — producing sub-optimal demonstrations that eventually either get stuck or drop the pen. Nevertheless, we show that we can combine these sub-optimal demonstrations with an off-policy RL algorithm in order to achieve performance on this task that can achieve significantly better performance than both the trajectory optimisation algorithm and the RL algorithm individually. Given that the task of spinning a pen in this way is challenging for a majority of humans, we would argue that this represents one of the most striking examples to date of a system autonomously learning to complete a complex dexterous manipulation task.

2 Related Work

Dexterous Manipulation: There is a large body of work that has looked at both designing and developing controllers for anthropomorphic robotic hands. Some work has found success by simplifying the design of the hands (Gupta et al. 2016), however a number of trajectory optimisation/policy search methods have also been found to produce successful behaviour for more realistic robotic hands (Mordatch, Popović, and Todorov 2012; Kumar et al. 2014; Lowrey et al. 2019). Lowrey et al. (2019) is particularly relevant to us since their method only requires access to sampling interactions with the simulated environment and not any detailed model of the physics of the system under consideration. Here, they combine a trajectory optimisation technique known as "model predictive path integral control" (MPPI) (Williams, Aldrich, and Theodorou 2015) with value function learning and coordinated exploration in order to perform in-hand manipulation of a cube using a simulated ADROIT robotic hand. In fact, they also demonstrate that this in-hand manipulation can also be performed only using the MPPI trajectory optimisation algorithm itself.

In terms of applying RL to anthropomorphic robotic hands, Rajeswaran et al. (2018) introduced a suite of dexterous manipulation environments again based on a simulated ADROIT hand, including in-hand manipulation and tool use tasks. Whilst they demonstrated that on-policy RL could be used to solve the tasks when carefully shaped reward functions were designed, they noted that this still took a long time and produced somewhat unnatural movements. They found that they could greatly improve the sample efficiency as well as learn behaviours that looked "more human" by augmenting the RL training with a small number of demonstrations generated via motion capture.

Plappert et al. (2018) introduced a number of multi-goal hand manipulation tasks using a simulated Shadow hand robot and integrated these into OpenAI Gym (Brockman et al. 2016). These environments form the basis for the new environments we introduce in the next section. In this work they also carried out extensive experiments using Hindsight Experience Replay (HER) (Andrychowicz et al. 2017), a state-of-the-art off-policy RL algorithm designed for multi-goal, sparse reward environments. This worked well for some of the environments, however for the more challenging ones (particularly HandManipulateBlockFull and HandManipulatePenFull) they found only limited success. A number of other works have built upon HER (Liu et al. 2019; He, Zhuang, and Li 2020), however they only provide minor improvements on the challenging hand manipulation tasks.

Nagabandi et al. (2019) introduce a model-based approach in order to solve a number of dexterous manipulation tasks in both simulation and on a real robotic hand, including manipulating two Baoding balls and a simplified handwriting task. Their method involves learning an ensemble of dynamics models and then planning each action using MPPI within these learned models, and is sample efficient enough that it can be used to learn using a real robotic hand.

Andrychowicz et al. (2018) use large-scale distributed training with domain randomization to train an RL agent in simulation that can transfer and operate on a real anthropomorphic robotic hand. Here, they are able to teach the hand to perform in-hand manipulation to rotate a block to a given target orientation, and show that it is sometimes capable of achieving over 50 different goals in a row without needing to reset. Akkaya et al. (2019) extend this work, introducing automatic domain randomization and training the hand to learn how to solve a Rubik's cube (more precisely, it learns how to implement the instructions given to it by a separate Rubik's cube solving algorithm). This requires considerably more dexterity than the block manipulation task, and represents one of the most impressive feats of learned dexterous manipulation to date, particularly as it is capable of operating on a real robotic hand. Nevertheless, it is worth noting that the amount of compute needed to obtain these results was enormous — using 64 Nvidia V100 GPUs and $\sim 30,000$ CPU cores, training for several months to obtain the best Rubik's cube policy.

Combining RL with demonstrations: A number of methods that make use of demonstrations in order to improve/speed up RL have been proposed. Hester et al. (2017) introduce a method to combine a small number of demonstrations to speed up Deep Q-learning, sampling demonstration data more regularly and using a large margin supervised loss that pushes the values of the demonstrator's actions above the values of other actions. Vecerík et al. (2017) extend this to environments with continuous action spaces, combining demonstrations with deep deterministic policy gradient (Lillicrap et al. 2016). Nair et al. (2018) use a similar approach but for sparse-reward, multi-goal environments. Here they combine demonstrations with HER, supplementing the RL with a behavioural cloning loss and a "Q-filter" which ensures this only acts on expert actions with higher predicted value than the policy's proposed action. This is particularly relevant to this work, as they also consider resetting episodes to randomly chosen demonstration states to aid with exploration, which is the basis for our approach of combining demonstrations generated by our trajectory optimisation algorithm with RL. Ding et al. (2019) also introduce a method designed for multi-goal environments, effectively combining HER with generative adversarial imitation learning (Ho and Ermon 2016).

3 New Dexterous Manipulation Tasks

In this section we introduce "Dexterous-gym" — a suite of challenging hand-manipulation environments based on modifications to the openAI gym shadow hand environments (Plappert et al. 2018)[1]. These include a number of environments that require coordination between two hands, such as handing over an object and playing catch with either one or two objects. This means that their state/action spaces are at least twice as large as the standard OpenAI gym manipulation tasks. All of the two-handed environments are goal-based in the same way as the original environments, meaning that a new goal is generated for each episode, and each environment is available with both sparse and dense reward variants (sparse rewards given only when the desired goal is achieved, vs. a dense reward based on a measure of distance

[1] All environments are available open-sourced here.

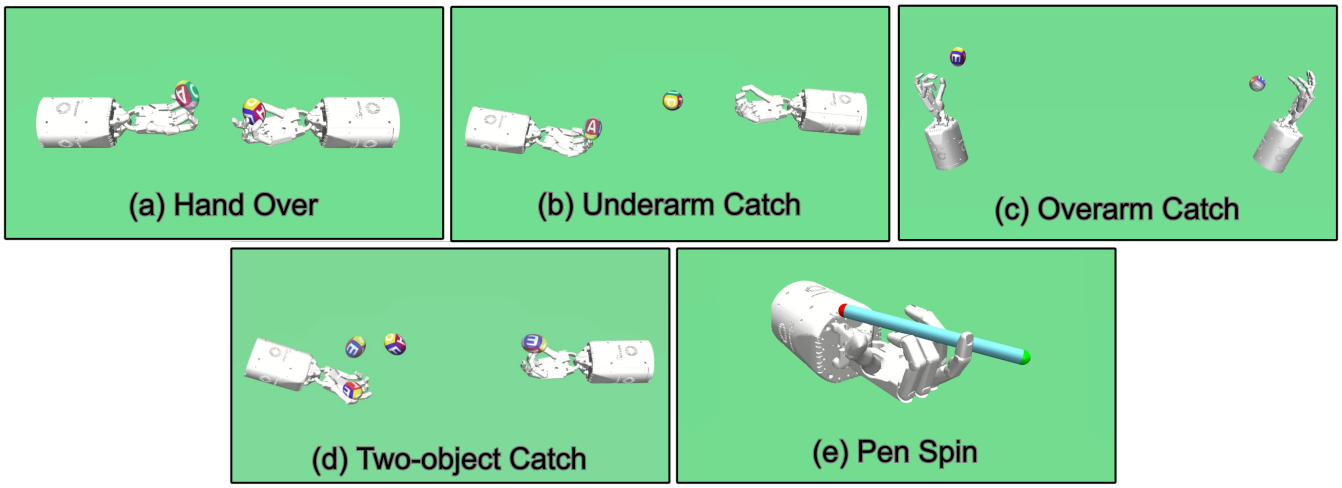

Figure 1: Overview of the types of new environments we introduce in "dexterous-gym". For (a)-(d) transparent objects visualise the desired goals, whilst opaque objects are the physical objects being manipulated.

to the desired goal). For each, the object can be either an egg, block or a pen.

We also introduce an environment we call "PenSpin", which is the only non goal-based, single-hand environment. This is a simple modification of the HandManipulatePen environment from OpenAI Gym where we re-define the reward function to encourage the pen to be spun around (whilst remaining horizontal relative to the hand). The basic types of environment introduced are shown in Figure 1.

(a) Hand Over environments: In these environments we have two hands with fixed bases slightly separated from each other. The goal position/orientation of the object will always only be reachable by the hand that does not start with the object, so to achieve the goal the object must first be passed to the other hand (e.g. by flicking it across the gap). Observation space: 109-dimensional, action space: 40-dimensional, goal space: 7 dimensional.

(b) Underarm Catch environments: Similar to the Hand Over environments, except now the bases are not fixed and have translational and rotational degrees of freedom that allow them to move within some range. However, the hands are too far apart to directly pass over the object and so it must be thrown and subsequently caught by the other hand, before being moved to the desired goal. Observation space: 133-dimensional, action space: 52-dimensional, goal space: 7-dimensional.

(c) Overarm Catch environments: Similar to Underarm Catch, except the hands are oriented vertically, requiring a different throwing/catching technique. Observation space: 133-dimensional, action space: 52-dimensional, goal space: 7-dimensional.

(d) Two-object Catch environments: Similar to Underarm Catch but now each hand starts with its own object. The goal is then the position/orientation of both objects, which are always only reachable by the other hands. This means that both objects have to be thrown in order to be swapped over. This requires significantly more coordination than the single-object variant. Observation space: 146-dimensional, action space: 52-dimensional, goal space: 14-dimensional.

(e) Pen Spin environment: The set-up is completely identical to HandManipulatePen from OpenAI Gym, with a single hand interacting with a long pen, however we remove the notion of a goal and instead define a new reward function; $r = \omega_3 - 15|b_z - t_z|$, where ω_3 is the third-component of the pen's angular velocity (in the generalised coordinates used by Mujoco) and b_z and t_z are the z-positions of each end of the pen. The first term encourages the pen to be spun, whilst the second penalises the pen for becoming vertical. Although lower-dimensional than the other tasks, this environment requires a significant amount of dexterity, and indeed many humans are not able to perform a similar task successfully. Observation space: 61-dimensional, action space: 20-dimensional.

4 Methods

Trajectory Optimisation for Precise Dexterous Manipulation (TOPDM)

In this section we introduce TOPDM, a trajectory optimisation algorithm that we apply to both the environments introduced in the previous section as well as the most challenging OpenAI gym manipulation tasks. Our method shares a number of similarities with MPPI (Williams, Aldrich, and Theodorou 2015), and so we begin by describing that in detail before going on to highlight the key differences with our approach.

MPPI is a gradient-free trajectory optimisation technique that can plan actions purely through sampling trajectories, either with a simulator (as in Lowrey et al. (2019)) or using a learned model (as in Nagabandi et al. (2019)). It is a form of model-predictive control where the next action is chosen by planning over a finite number of future time steps, τ, before taking a single action and then re-planning again from the next state. The procedure is quite simple — to plan the next action from the current state s_t, N mean action sequences

Algorithm 1: Trajectory Optimisation for Precise Dexterous Manipulation (TOPDM)

Initialise: τ (planning horizon), N_t (number of trajectories per iteration), N_i (number of iterations), T (trajectory length), β (action coupling parameter), f_n (fraction of non-zero noise-dimensions), f_b (fraction of best trajectories retained for next iteration), σ_n (noise standard deviation), σ_i (initial standard deviation), a_d (action-dimension), E (environment)

begin

- Initialise $\mu \in \mathbb{R}^{N \times \tau \times a_d} \sim \mathcal{N}(0, \sigma_i)$, $a \in \mathbb{R}^{N \times \tau + 1 \times a_d}$ as zeroes
- **for** $t = 1 : T$ **do**
  - **for** $i = 1 : N_i$ **do**
    - add noise σ_n to $\lceil f_n \times a_d \rceil$ randomly chosen dimensions of μ
    - **for** $s = 1 : \tau$ **do**
      - $a[:, s, :] = \beta\, \mu[:, s, :] + (1 - \beta)\, a[:, s-1, :]$
    - Evaluate all N_t of these action trajectories with E. Sort based on returns
    - Take fraction f_b of best performing trajectories. Duplicate these to uniformly fill μ and a with copies of these best trajectories. These act as the starting point for the next iteration
  - Take μ_b as the best performing mean action sequence from any iteration. Step real environment E with $a_t = \beta \mu_b[1] + (1 - \beta) a_{t-1}$
  - $\mu[:, 1 : \tau - 1, :] = \mu_b[2 : \tau, :]$ (carry over rest of action sequences for next search)
  - $a[:, 0, :] = a_t$ (set previous action)

are initialised with random noise: $\{\mu^{(n)}_{t:t+\tau}\}_{n=1}^{N}$. Rather than executing these mean actions directly, a coupling between subsequent actions is introduced such that the actual actions executed are $a^{(n)}_{t'} = \beta \mu^{(n)}_{t'} + (1-\beta) a^{(n)}_{t'-1}$, where $\beta \in [0, 1]$. The motivation for this is that an intermediate β value can produce smoother action sequences and effectively reduces the dimensionality of the search space (by excluding some possible action sequences). These action sequences are then all evaluated using the simulator (or model) and their returns over the τ time steps considered, R_n, are recorded. These are then used to update the mean action sequence by calculating $\mu_{t:t+\tau} = \frac{\sum_{n=1}^{N} \mu^{(n)}_{t:t+\tau} e^{\kappa R_n}}{\sum_{k=1}^{K} e^{\kappa R_k}}$, i.e. an exponentially weighted average of all of the mean action sequences weighted by their returns (with κ as a hyperparameter). If this is the final iteration, we return $a_t = \beta \mu_t + (1 - \beta) a_{t-1}$ as the action to actually execute in the environment, otherwise we duplicate $\mu_{t:t+\tau}$ N times, add noise, and repeat the procedure.

The trajectory optimisation algorithm we implement shares the same basic structure as MPPI, but we make a number of key changes that we find significantly boosts performance on the environments considered in this paper:

- Firstly, rather than taking a weighted average of action sequences we instead choose some fraction f_b of the best performing trajectories at the end of an interation and uniformly duplicate them so that there are N in total. These get carried over as the initialisation of the next iteration of the plan (on the final iteration we return the first action from the single best performing trajectory we have found throughout the full planning process). This is similar to the "cross-entropy method" used for trajectory optimisation in Hafner et al. (2019), except there they choose the top f_b fraction of trajectories in order to update the parameters (mean and standard deviation) of a normal distribution that is used to sample the action trajectories for the next iteration. The motivation for doing away with the averaging procedure is that when we are considering high-dimensional tasks that require significant precision, it might be very rare to see a trajectory that significantly improves upon the return. This means that any averaging procedure can easily perturb or effectively ignore any trajectories that have unusually high returns and significantly reduce overall performance.
- Secondly, we change the way in which we add noise to the mean action sequences. At the start of a given iteration of planning we can consider the collection of mean action sequences as a multi-dimensional array, $\mu \in \mathbb{R}^{N \times \tau \times a_d}$, where a_d is the dimension of a single action. Rather than adding noise to this whole vector, for a given iteration we instead choose to add noise only to a fraction f_n of randomly chosen dimensions. The motivation here is that if we have already found a reasonably good trajectory then adding noise to all dimensions of all time-steps can very easily perturb the trajectory away from good performance. If we only add noise to some of the dimensions at some time steps we make it more likely that we can find slight improvements to certain parts of the trajectory without disturbing the rest of the sequence.
- Finally, rather than starting each plan from scratch, we instead carry over the best trajectory from the previous planning step and use this to initialise the mean action sequence. That is, given the best mean action sequence from the previous planning step, $\mu^{b}_{t:t+\tau}$, we use μ^{b}_{t} to choose the next action actually taken in the environment, but rather than discarding $\mu^{b}_{t+1:t+\tau}$ we instead use it to initialise the first $\tau - 1$ actions for all of the mean action sequences for the next planning step. The reason for not doing this in MPPI is stated as wanting to encourage exploration and not getting stuck in local optima, however in our experiments this isn't really an issue we see. Also, since the problems considered require precise coordination we find it is much more useful to make use of

as many iterations of improvement as possible (carrying over the previous best mean sequence effectively means building upon all of the iterations carried out in the previous plan, rather than restarting the search from scratch).

Using Demonstrations Gathered via Trajectory Optimisation to Guide RL

There are two main issues with the trajectory optimisation approach described in the previous section. Firstly, despite being highly parallelisable, for the challenging manipulation environments we consider in this paper the algorithm still requires a lot of compute in order to generate high quality trajectories. On the other hand, a neural network may take a long time to be trained with RL, however once it has been trained it can easily generate new trajectories on the fly in real time. The second issue with the trajectory optimisation approach is that it is necessary to plan over a finite horizon, τ. We find that in our experiments it is not practical to extend τ much beyond ~ 25, which is fine for many of the environments considered but can lead to situations where optimising reward over a relatively short period of time misses strategies that over a longer time-frame can lead to much higher rewards. As we shall see in the next section, this turns out to be the case for the PenSpin environment. This is another drawback that RL does not suffer from, as with RL we aim to maximise the (usually discounted) sum of future rewards, which in principle allows us to learn strategies that maximise rewards over a much longer period of time.

Both of these issues motivate combining the demonstrations generated via trajectory optimisation with RL. We focus on the PenSpin environment for two reasons — firstly because it is the only "standard" RL environment (i.e. non goal-based), but mainly because it is the environment which requires the most dexterity and which is the most challenging for the trajectory optimisation algorithm.

The approach we take is extremely simple — we train an agent using the off-policy RL algorithm TD3 (Fujimoto, Van Hoof, and Meger 2018). However, rather than gathering data by resetting the environment and running an episode, we instead consider gathering segments of data. With some probability a segment will come from a normal ongoing episode, but with some probability we will instead load the simulator to a randomly chosen demonstration state and gather the segment of data from there instead. We start with a high probability of gathering data starting from a demonstration state and then gradually anneal this throughout the training.

The motivation for this approach is that it allows us to gather lots of data that start from desirable configurations where we know it is possible to obtain high-rewards, as well as states that we know can lead to these desirable configurations. If we consider that for a particular task there may be states that are potentially highly rewarding but which are very difficult to discover through random exploration then there are essentially two challenges. Firstly, the agent needs to be able to discover these states at all. But even if the agent can eventually discover the states it needs to be able to estimate their values. If an agent finally discovers a hard to reach state but has no concept of its value, then it will not know that it wants to return there again later and will most likely not discover it again for a long time. The approach we propose here therefore helps with exploration in two ways — firstly, it allows the agent to learn better value estimates of states that are potentially very difficult to reach via random exploration much earlier on, meaning that when an agent does eventually discover them it will know to return. Secondly, it also helps with learning to reach these states by sometimes starting the simulation from states that are much closer to the desirable states than the initial state at the start of a normal episode.

We also employ the same action-coupling technique used in the trajectory optimisation during the training of the RL agent. That is, the agent's policy takes in the current observation o_t as well as the previous action, a_{t-1}, and outputs $\mu_\theta(t)$ (θ represent the parameters of the policy). The actual action taken in the environment is then $\beta\mu_\theta(t)+(1-\beta)a_{t-1}$. In Appendix B we show that this is crucial to reliably train an agent which can solve this task.

Finally, we note that this kind of approach is not completely new and similar ideas have been suggested in other contexts. Nair et al. (2018) include resetting to demonstration states when using demonstrations to aid with exploration in multi-goal RL, however it is more of a detail that is sometimes found to be useful and not at the core of their method (which mixes behavioural cloning and hindsight experience replay). GoExplore (Ecoffet et al. 2019) is another method which uses a similar principle and gathers data by resetting the simulator to promising states that it has saved in an archive. However, this requires defining some kind of similarity measure between states so that they can be distinguished and stored in a finite archive, whilst with our approach we already have small finite set of demonstrations generated through trajectory optimisation which we can choose from randomly when we choose to load the simulator to a demonstration state.

5 Results

OpenAI Gym Manipulation Environments

We start by evaluating our method on the two most challenging OpenAI Gym manipulation tasks — HandManipulateBlockFull and HandManipulatePenFull. These both require manipulating an object to a desired orientation *and* desired position simultaneously, rather than only a desired orientation. HandManipulatePenFull is particularly challenging as it is easy for the pen to get stuck in between the fingers. Figure 2 shows the results of our trajectory optimisation algorithm alongside a number of baselines. For each environment we plot both the final success rate (the fraction of episodes that end with the desired goal being achieved) as well as the average sum of rewards obtained in an episode. This latter measure is useful to include as well since it allows us to evaluate how close a method might get the goal even if it doesn't successfully achieve it at the last time step of an episode. There are a few things here worth noting: firstly, we slightly modify the reward from the default environments so that they lie in the range $[0, 1]$ for each time step. Secondly, we change the maximum number of time steps down from

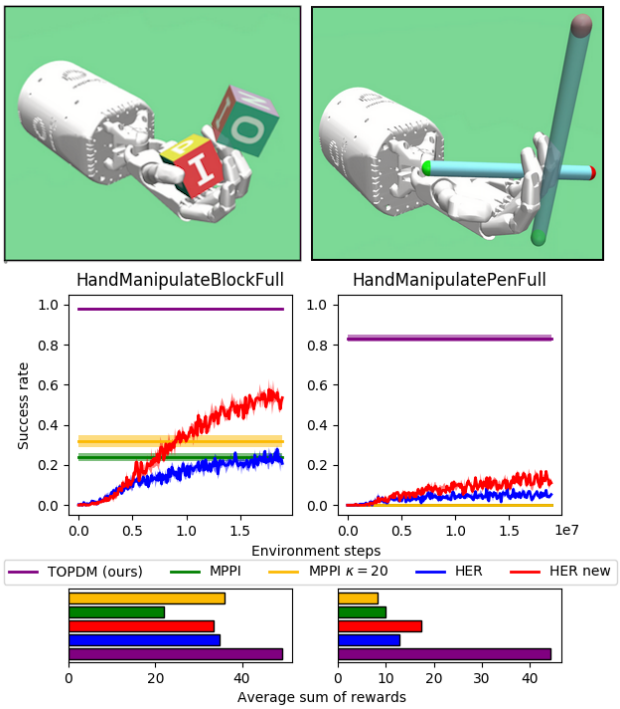

Figure 2: Performance of TOPDM compared with various baselines on the two most challenging dexterous manipulation environments from OpenAI Gym. The top two panels show screenshots of the two environments considered, with the transparent objects representing the target object goals. The plots show the performance in terms of both success rate and average rewards obtained in an episode for our method and a number of baselines. See text for more details.

the default of 100 to 75 as this does not significantly effect the success rate of any of the methods considered. Thirdly, we note that Hindsight Experience Replay (HER) actually works better with sparse rewards, and so for this comparison we train using the sparse-reward version of the environment but still evaluate the average sum of rewards using the dense version once a policy is trained. Finally, we note that by default the HandManipulatePenFull environment has a distance tolerance of 0.05 (i.e. the centre of mass of the pen must be within this distance of the desired position before the positional part of the goal is considered achieved), whilst the other environments have this set to 0.01. This means that the goal can be considered achieved even though visually it is clear that the pen does not even overlap at all with the desired goal. We change this value back to 0.01 in order to match up with the other environments (making the task significantly harder, which explains why the HER result is lower than previously reported in Plappert et al. (2018)).

For most of the trajectory optimisation experiments we use $\tau = 20$, $N_i = 40$ and $N_t = 1000$ (full hyperparameters are detailed in Appendix A). This means that this approach, whilst highly parallelisable, is computationally intensive and trajectories have to be computed offline.

We also compare our trajectory optimisation method

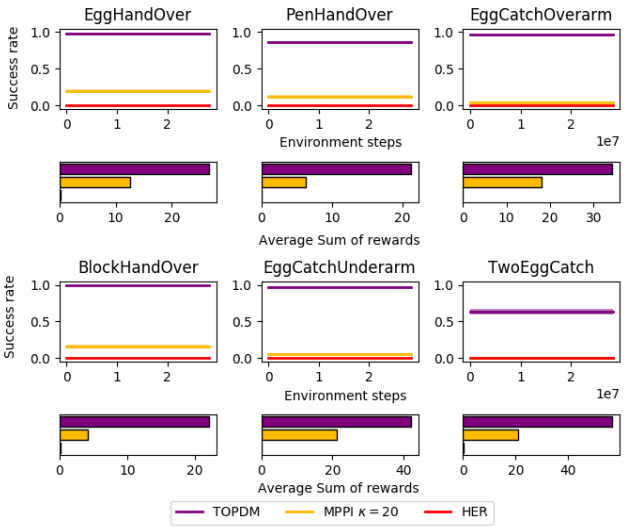

Figure 3: TOPDM and baseline performance on goal-based environments introduced in Section 3.

against MPPI, which is the most directly comparable approach and has been used before in various dexterous manipulation tasks. We consider two variants, firstly setting $\kappa = 1$ (as in Lowrey et al. (2019)) which we just call "MPPI", and secondly setting $\kappa = 20$ (which was used for some experiments in Nagabandi et al. (2019)), which we label "MPPI $\kappa = 20$". Increasing κ changes the weight we give to trajectories, with higher values providing significantly more weight to the highest performing trajectories ($\kappa = 20$ often corresponds to giving almost all of the weight to the trajectory with highest return). To give the fairest comparison possible, for both of these we scale up the number of iterations as well as the number of trajectories sampled per iteration such that they are the same as used in our trajectory optimisation method.

Interestingly, when running the HER experiments we found that by changing one of the hyperparameters away from the value used in the original paper we significantly improved its performance on HandManipulateBlockFull (we changed the probability of a random action from 0.3 to 0). We include results with both the original hyperparameters ("HER") as well as with the improved values ("HER new").

We see that our method substantially outperforms all of the baselines, both in terms of final success rate as well with the average sum of rewards, achieving close to 100% success rate of HandManipuilateBlockFull. The results on the more challenging HandManipulatePenFull are even more striking, as none of the other methods are able to achieve more than $\sim$15% success rate whilst ours achieves $\sim$83% along with a much higher average sum of rewards. Side-by-side videos of the final performance of all of these methods can be found on the project website.[2]

Results on Our Custom Goal-based Environments

Figure 3 shows the results of our trajectory optimisation algorithm on some of the goal-based variants of the environ-

[2]https://dexterous-manipulation.github.io

ments introduced in section 3. We see that all of the environments are very difficult for HER, where effectively the success rate is zero and very little reward is gathered. MPPI performs better, generally making a reasonable attempt at completing the task but nevertheless lacking the required precision to reliably achieve *and maintain* the goals until the end of the episode. On the other hand, our method achieves $\sim 100\%$ success rate on four of the environments, and then $\sim 80\%$ and $\sim 60\%$ respectively on the more challenging PenHandOver and TwoEggCatch environments. Note that we omit the results for MPPI $\kappa = 1$ and HER original, as these both achieve practically zero success rate and close to zero sum of rewards for each of the six environments considered here.

Solving the PenSpin environment

Figure 4 summarises the results on arguably the most challenging environment — PenSpin. Here we see that both MPPI and TD3 (a state of the art off-policy RL algorithm) both achieve very low scores and are never really able to get the pen to spin properly at all. This is probably because the environment requires a fairly precise set of actions at the start in order to move the pen into a good position to start spinning it, and finding this consistently through random exploration is very unlikely. On the other hand, our trajectory optimisation algorithm (TOPDM) performs significantly better, and almost always starts spinning the pen for some period of time. However, in none of the trials gathered was the agent able to continuously spin the pen for the whole 250 time steps without either dropping it or getting it stuck between its fingers at some point.

Nevertheless, the imperfect demonstrations generated by TOPDM can be combined with TD3 as described at the end of section 4 (TD3 + demo restarts in Figure 4). We see that this combination significantly outperforms both TOPDM and TD3 by themselves (note that we plot the median rewards over 10 runs, with confidence intervals based on the maximum and minimum scores. Two of the runs still did not learn to improve beyond the pure TD3 results (the other 8 all did), which is the reason for the large error bars shown. In Appendix B we plot each trajectory individually when comparing $\beta = 0.7$ vs $\beta = 1.0$).

This is a nice demonstration of how imperfect demonstrations (generated autonomously with trajectory optimisation) can be combined with RL in order to generate significantly improved performance on a challenging environment. We also note that the final trained policy can be run beyond 250 time steps (the standard episode length) and appears to be robust enough to continue to spin the pen indefinitely. This can be seen from watching the video on the project website.

In Appendix B we investigate the importance of using the action-coupling parameter β. Surprisingly, we find that not using any action coupling ($\beta = 1.0$) massively reduces performance, with only 1/10 trials achieving significantly better performance than TD3 on its own, compared with 8/10 trajectories with $\beta = 0.7$. Whilst this trick has been used previously with trajectory optimisation, we are not aware of it being used in RL before, and these results suggest that action-coupling could prove to be useful more generally when using RL for high-dimensional continuous control tasks.

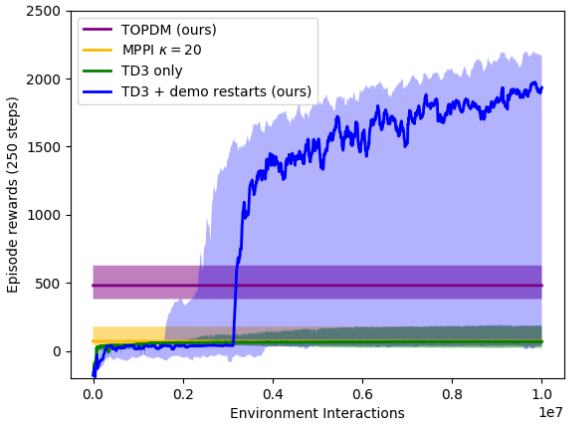

Figure 4: Results on the PenSpin environment.

6 Conclusion

We have introduced a suite of challenging dexterous manipulation tasks on which current reinforcement learning/ trajectory optimisation algorithms fail to achieve good performance. We then introduced a new trajectory optimisation technique that performs significantly better, obtaining impressive results on all of the environments considered. However, this comes at the cost of computational expense, requiring the solutions to be computed offline. We went on to show how the solutions generated by this trajectory optimisation algorithm could be used to guide the training of an RL agent which, once trained, can be deployed in real time. Applying this to the task of learning to spin a pen between the fingers of a robotic hand leads to performance that far exceeds either trajectory optimisation or RL on its own. We would argue that these results on this challenging tasks represent one of the most impressive examples to date of autonomously learning to solve a complex dexterous manipulation task.

In the future we would like to combine the trajectories generated by TOPDM with imitation learning (IL)/ RL to train an agents that can solve the goal-based tasks we introduced as well. Whilst not our primary focus, in some preliminary experiments we found that existing methods (Ding et al. 2019; Nair et al. 2018) did not work well here. We also tried implementing a similar technique as with solving the PenSpin environment, but using HER rather than TD3 as the base RL algorithm. Whilst this performed slightly better than HER alone we were unable to attain performance close to a similar level of TOPDM. As such, as part of the code release we also include the trajectories generated for each task by TOPDM, and leave combining these with RL/IL as a challenge for future research. The environments introduced here also represent a significant challenge for future approaches based purely on RL.

References

Akkaya, I.; Andrychowicz, M.; Chociej, M.; Litwin, M.; McGrew, B.; Petron, A.; Paino, A.; Plappert, M.; Powell,

G.; Ribas, R.; Schneider, J.; Tezak, N.; Tworek, J.; Welinder, P.; Weng, L.; Yuan, Q.; Zaremba, W.; and Zhang, L. 2019. Solving Rubik's Cube with a Robot Hand.

Andrychowicz, M.; Baker, B.; Chociej, M.; Józefowicz, R.; McGrew, B.; Pachocki, J. W.; Pachocki, J.; Petron, A.; Plappert, M.; Powell, G.; Ray, A.; Schneider, J.; Sidor, S.; Tobin, J.; Welinder, P.; Weng, L.; and Zaremba, W. 2018. Learning Dexterous In-Hand Manipulation. *CoRR* abs/1808.00177. URL http://arxiv.org/abs/1808.00177.

Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Pieter Abbeel, O.; and Zaremba, W. 2017. Hindsight Experience Replay. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 30*, 5048–5058. Curran Associates, Inc. URL http://papers.nips.cc/paper/7090-hindsight-experience-replay.pdf.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI Gym.

Ding, Y.; Florensa, C.; Abbeel, P.; and Phielipp, M. 2019. Goal-conditioned Imitation Learning. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d' Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 32*, 15324–15335. Curran Associates, Inc. URL http://papers.nips.cc/paper/9667-goal-conditioned-imitation-learning.pdf.

Ecoffet, A.; Huizinga, J.; Lehman, J.; Stanley, K. O.; and Clune, J. 2019. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995* .

Fujimoto, S.; Van Hoof, H.; and Meger, D. 2018. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477* .

Gupta, A.; Eppner, C.; Levine, S.; and Abbeel, P. 2016. Learning dexterous manipulation for a soft robotic hand from human demonstrations. 3786–3793. doi:10.1109/IROS.2016.7759557.

Hafner, D.; Lillicrap, T.; Fischer, I.; Villegas, R.; Ha, D.; Lee, H.; and Davidson, J. 2019. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, 2555–2565.

He, Q.; Zhuang, L.; and Li, H. 2020. Soft Hindsight Experience Replay.

Hester, T.; Vecerík, M.; Pietquin, O.; Lanctot, M.; Schaul, T.; Piot, B.; Sendonaris, A.; Dulac-Arnold, G.; Osband, I.; Agapiou, J.; Leibo, J. Z.; and Gruslys, A. 2017. Learning from Demonstrations for Real World Reinforcement Learning. *CoRR* abs/1704.03732. URL http://arxiv.org/abs/1704.03732.

Ho, J.; and Ermon, S. 2016. Generative Adversarial Imitation Learning. In Lee, D. D.; Sugiyama, M.; Luxburg, U. V.; Guyon, I.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 29*, 4565–4573. Curran Associates, Inc. URL http://papers.nips.cc/paper/6391-generative-adversarial-imitation-learning.pdf.

Kumar, V.; Tassa, Y.; Erez, T.; and Todorov, E. 2014. Real-time behaviour synthesis for dynamic hand-manipulation. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 6808–6815.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2016. Continuous control with deep reinforcement learning. In *ICLR (Poster)*. URL http://arxiv.org/abs/1509.02971.

Liu, H.; Trott, A.; Socher, R.; and Xiong, C. 2019. Competitive experience replay. In *International Conference on Learning Representations*. URL https://openreview.net/forum?id=Sklsm20ctX.

Lowrey, K.; Rajeswaran, A.; Kakade, S.; Todorov, E.; and Mordatch, I. 2019. Plan Online, Learn Offline: Efficient Learning and Exploration via Model-Based Control. In *International Conference on Learning Representations (ICLR)*.

Mordatch, I.; Popović, Z.; and Todorov, E. 2012. Contact-Invariant Optimization for Hand Manipulation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA 12, 137144. Goslar, DEU: Eurographics Association. ISBN 9783905674378.

Nagabandi, A.; Konoglie, K.; Levine, S.; and Kumar, V. 2019. Deep Dynamics Models for Learning Dexterous Manipulation. In *Conference on Robot Learning (CoRL)*.

Nair, A.; McGrew, B.; Andrychowicz, M.; Zaremba, W.; and Abbeel, P. 2018. Overcoming Exploration in Reinforcement Learning with Demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 6292–6299.

Plappert, M.; Andrychowicz, M.; Ray, A.; McGrew, B.; Baker, B.; Powell, G.; Schneider, J.; Tobin, J.; Chociej, M.; Welinder, P.; Kumar, V.; and Zaremba, W. 2018. Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research .

Rajeswaran, A.; Kumar, V.; Gupta, A.; Vezzani, G.; Schulman, J.; Todorov, E.; and Levine, S. 2018. Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. In *Proceedings of Robotics: Science and Systems (RSS)*.

Vecerík, M.; Hester, T.; Scholz, J.; Wang, F.; Pietquin, O.; Piot, B.; Heess, N.; Rothörl, T.; Lampe, T.; and Riedmiller, M. A. 2017. Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards. *CoRR* abs/1707.08817. URL http://arxiv.org/abs/1707.08817.

Williams, G.; Aldrich, A.; and Theodorou, E. 2015. Model Predictive Path Integral Control using Covariance Variable Importance Sampling .

A Experimental details

Trajectory optimisation (TOPDM)

The default hyperparameters used for the trajectory optimisation experiments are the following:

- $\tau = 20$ (planning horizon)
- $N_i = 40$ (number of iterations for each planning step)
- $N_t = 1000$ (number of trajectories of length τ generated for each iteration of the planning step)
- $f_b = 0.05$ (fraction of highest performing trajectories carried over at each iteration)
- $\beta = 0.7$ (action-coupling parameter)
- $\sigma_i = 0.9$ (initialisation noise)
- $\sigma_n = 0.3$ (standard deviation of noise added at each iteration)
- $f_n = 0.3$ (fraction of action dimensions that noise is added to)

An episode for the OpenAI Gym environments is taken to be 75 time steps. For our custom goal-based environments we take an episode to be 50 time steps, and for PenSpin we take a standard episode to be 250 time steps. For TwoEggCatchUnderarm we use $N_i = 80$, $N_t = 2000$ and for PenSpin we use $N_i = 80$, $N_t = 4000$.

TD3 plus demos

In our experiments with TD3 plus demonstrations generated by TOPDM we used largely standard parameters as reported in Fujimoto et al. (2018) for the core TD3 algorithm.

- start timesteps: 25,000 (initial steps gathered with random policy)
- total timesteps: 10,000,000
- exploration noise: 0.1
- batch size: 256
- γ : 0.98 (discount factor)
- target network update rates: 0.005
- policy noise: 0.2
- target policy noise clip threshold: 0.5
- update policy frequency: 2 (policy is updated every 2 critic updates)
- β : 0.7 (action coupling parameter)
- segment length: 15 (number of timesteps per segment)
- initial probability of starting segment from demo reset: 0.7
- decay factor: 0.999996 (each step probability of demo reset multiplied by this)

Neural networks for the policy/critic and target policy/target critic were all fully connected with two hidden layers of size 256, and use the RelU activation function.

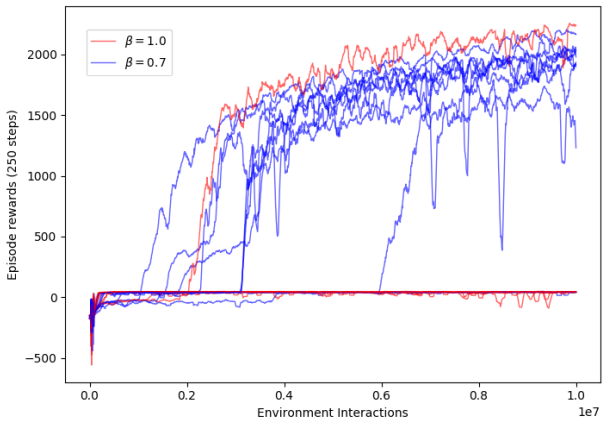

Figure 5: 10 runs for $\beta = 1.0$ (no action coupling) vs $\beta = 0.7$ (action coupling)

B Effect of action-coupling with RL plus demos for PenSpin

Here we investigate the effect of including action-coupling when training the RL agent with demonstration resets on the PenSpin environment. We run 10 experiments with different initial seeds for two values of the coupling parameter, $\beta = 1.0$ (no action-coupling) and $\beta = 0.7$ (the value used in the results shown in Figure 4). We see that including the action-coupling allows for the agent to learn a high-quality policy significantly more often, whilst only one run with no action-coupling ($\beta = 1.0$) succeeds at the task.