Computing and Visualization in Science (2020) 23:8
https://doi.org/10.1007/s00791-020-00327-0

PINT 2019

Parareal with a learned coarse model for robotic manipulation

Wisdom Agboh[1] · Oliver Grainger[2] · Daniel Ruprecht[3] · Mehmet Dogar[1]

Received: 6 December 2019 / Accepted: 27 August 2020

Abstract
A key component of many robotics model-based planning and control algorithms is physics predictions, that is, forecasting a sequence of states given an initial state and a sequence of controls. This process is slow and a major computational bottleneck for robotics planning algorithms. Parallel-in-time integration methods can help to leverage parallel computing to accelerate physics predictions and thus planning. The Parareal algorithm iterates between a coarse serial integrator and a fine parallel integrator. A key challenge is to devise a coarse model that is computationally cheap but accurate enough for Parareal to converge quickly. Here, we investigate the use of a deep neural network physics model as a coarse model for Parareal in the context of robotic manipulation. In simulated experiments using the physics engine Mujoco as fine propagator we show that the learned coarse model leads to faster Parareal convergence than a coarse physics-based model. We further show that the learned coarse model allows to apply Parareal to scenarios with multiple objects, where the physics-based coarse model is not applicable. Finally, we conduct experiments on a real robot and show that Parareal predictions are close to real-world physics predictions for robotic pushing of multiple objects. Code (https://doi.org/10.5281/zenodo.3779085) and videos (https://youtu.be/wCh2o1rf-gA) are publicly available.

Keywords Parallel-in-time · Parareal · Manipulation · Robotics · Planning · Neural network · Model-predictive control · Learning

1 Introduction

We present a method for fast and accurate physics predictions during non-prehensile manipulation planning and control. An example scenario is shown in Fig. 1, where a robot arm pushes the marked cylindrical object into a target zone without pushing the other three objects off the table. We are interested in predicting the motion of the objects in a fast and accurate way.

Physics engines like Mujoco [37] and Drake [36] solve Newton's equation to predict motion. They are accurate but slow. Coarse models can be built by introducing simplifying assumptions, trading accuracy for solution speed but their lack of precision will eventually compromise the robot's chance of completing a given task successfully.

Given an initial state and a sequence of controls, the problem of predicting the resulting sequence of states is a key component of a variety of model-based planning and control algorithms [18,21,22,39]. Mathematically, such a prediction requires solving an initial value problem. Typically, those are solved through numerical integration over time-steps using e.g. semi-implicit Euler's method or Runge–Kutta methods and an underlying physics model to provide the forces. However, the speed with which these accurate physics-based predictions can be performed is still slow [9]. Faster physics-based predictions can contribute significantly

Communicated by Robert Speck.

This project was supported by an EPSRC studentship (1879668) and EPSRC grants EP/R031193/1 and EP/P019560/1.

✉ Wisdom Agboh
w.c.agboh@leeds.ac.uk

Oliver Grainger
mn17omg@leeds.ac.uk

Daniel Ruprecht
ruprecht@tuhh.de

Mehmet Dogar
m.r.dogar@leeds.ac.uk

[1] School of Computing, University of Leeds, Leeds, UK

[2] School of Mechanical Engineering, University of Leeds, Leeds, UK

[3] Lehrstuhl Computational Mathematics, Institut für Mathematik, Technische Universität Hamburg, Hamburg, Germany

Published online: 23 September 2020

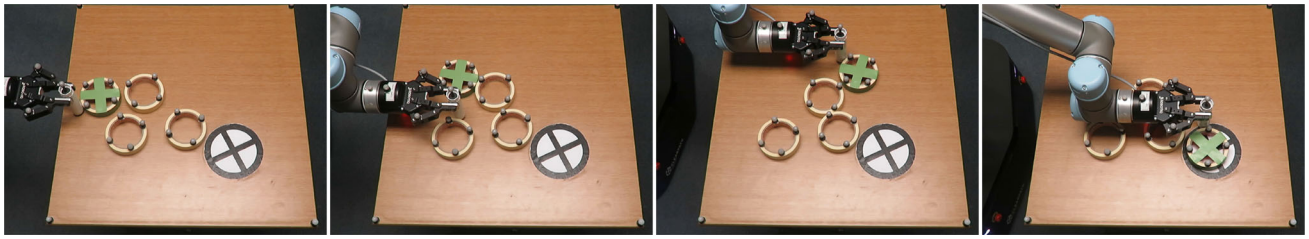

Fig. 1 Example of a robotic manipulation planning and control task using physics predictions. The robot controls the motion of the green object solely through contact. The goal is to push the green object into the target region marked X. The robot must complete the task without pushing other objects off the table or into the goal region

to contact-based/non-prehensile manipulation planning and control—especially during re-planning or model-predictive control (MPC) where a robot executes an action in the real-world, gets the resulting state and then has to generate a new physics-based plan. Such MPC methods have been used in prior work to achieve manipulation robustness to parameter uncertainty [1], stabilize complex humanoid behaviours [35], and visually manipulate fabric [19].

In a previous paper [4], we demonstrated that predictions for a robot pushing a single object can be made faster by combining a fine physics-based model with a simple, coarse physics-based model using the parallel-in-time method Parareal. Using 4 cores, Parareal was about a factor two faster than the fine physics engine alone while providing comparable accuracy and the same success rate for a push planning problem with obstacle avoidance. Here, we extend these results by investigating a deep neural network as coarse model and show that it leads to faster Parareal convergence. We also demonstrate that Parareal can be used to speed up physics prediction in scenarios where the robot pushes multiple objects.

2 Related work

Parareal has been used in many different areas. Trindade and Pereira [38], for example, use it to simulate incompressible laminar flows. Maday and Turinici [27] have tested it for to simulate dynamics in quantum chemistry. The method was introduced by Lions et al. [25]. Combinations of parallel-in-time integration and neural networks have not yet been studied widely. Very recently, Yalla and Enquist [40] showed the promise of using a machine learned model as coarse propagator for test problems. Going the other way, Schroder [32] and Günther et al. [14] recently showed that parallel-in-time integration can be used to speed up the process of training neural networks.

Results on how Parareal performs for differential algebraic equations (DAEs) are scarce. Guibert and Tromeur-Dervout [13] demonstrate that Parareal can solve DAEs, but can experiences issues with stability for very stiff problems. Cadeau and Magoules [8] propose a combination of Parareal with waveform relaxation to introduce additional parallelism. For a DAE system of size 100,000, they demonstrate that adding Parareal does provide speedup beyond the saturation point of waveform relaxation alone.

Physics predictions play a major role in robotic manipulation planning and control—to generate uncertainty averse robotic pushing plans [6], to manipulate objects in clutter through online re-planning [2], to rearrange objects in clutter through dynamic actions [17], and also to use human guidance to generate pushing motions [30]. However, planning is slow since physics predictions are computationally expensive. Parareal's potential to speed up simulations for robotic manipulation in single-object scenarios using a physics-based coarse model was recently demonstrated by Agboh et al. [4].

Furthermore, physics predictions are essential in learning physics-based manipulation policies. For example, learning gentle object manipulation through curiosity [20], learning long-horizon robotic agent behaviours through latent imagination [15], learning visuo-motor policies by formulating exploration as a latent trajectory optimization problem [26], learning policies for manipulation in clutter [7], smoothing fabric with a da Vinci surgical robot through deep imitation learning [33], and learning human-like manipulation policies through virtual reality demonstrations [16]. The training time for these policies can potentially be reduced with a parallel-in-time approach to physics predictions.

Combining different physics models for robotic manipulation has been the topic of recent research, although not with a focus on improving prediction speed. Kloss et al. [23] address the question of accuracy and generalization in combined neural-analytical models. Ajay et al. [5] focus on modeling the inherent stochastic nature of the real world physics, by combining an analytical, deterministic rigid-body simulator with a stochastic neural network.

We can make physics engines faster by using larger simulation time steps. However, this decreases the accuracy and can result in unstable behavior where objects have unreal-

istically large accelerations. To generate stable behaviour at large time-step sizes, Pan and Manocha [29] propose an integrator for articulated body dynamics by using only position variables to formulate the dynamic equation. Moreover, Fan et al. [10] propose linear-time variational integrators of arbitrarily high order for robotic simulation and use them in trajectory optimization to complete robotics tasks. Recent work has focused on making the underlying planning and control algorithms faster. For example, Giftthaler et al. [12] introduced a multiple-shooting variant of the trajectory optimizer—iterative linear quadratic regulator [24] which has shown impressive results for real-time nonlinear optimal control of complex robotic systems [28,31].

3 Robotic manipulation with parareal

3.1 Robotic manipulation

Consider the scene shown in Fig. 1. The robot's manipulation task is to control the motion of the green goal object through pushing contact from the cylindrical pusher in the robot's gripper. The robot needs to push the goal object into a goal region marked with an X. It is allowed to make contact with other sliders but not to push them off the table or into the goal region.

The system's state at time point n consists of the pose $\mathbf{q}$ and velocities, $\dot{\mathbf{q}}$ of the pusher P and N_s sliders, $S^i \dots S^{N_s}$:

$$\mathbf{x}_n = [\mathbf{q}_n^P, \mathbf{q}_n^{S^i}, \dots, \mathbf{q}_n^{S^{N_s}}, \dot{\mathbf{q}}_n^P, \dot{\mathbf{q}}_n^{S^i}, \dots, \dot{\mathbf{q}}_n^{S^{N_s}}].$$

The pose of slider i consists of its position and orientation on the plane: $\mathbf{q}^{S^i} = [q^{S^i_x}, q^{S^i_y}, q^{S^i_\theta}]^T$. The pusher's pose is $\mathbf{q}^P = [q^{P_x}, q^{P_y}]^T$ and control inputs are velocities $\mathbf{u}_n = [u_n^x, u_n^y]^T$ applied on the pusher at time n for a control duration of Δt.

A robotics planning and control algorithm takes in an initial state of the system $\mathbf{x}_0$, and outputs an optimal sequence of controls $\{\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_{N-1}\}$. However, to generate this optimal sequence, the planner needs to simulate many different control sequences and predict many resulting sequences of states $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$.

The planner makes these simulations through a physics model F of the real-world that predicts the next state $\mathbf{x}_{n+1}$ given the current state $\mathbf{x}_n$ and a control input $\mathbf{u}_n$

$$\mathbf{x}_{n+1} = F(\mathbf{x}_n, \mathbf{u}_n, \Delta t). \tag{1}$$

We use the general physics engine Mujoco [37] to model F. It solves differential algebraic equations of motion for the complex multi-contact dynamics problem

$$M(\mathbf{q})d\mathbf{v} = (\mathbf{b}(\mathbf{q}, \mathbf{v}) + \tau)\, dt + J_E(\mathbf{q})^T \mathbf{f}_E(\mathbf{q}, \mathbf{v}, \tau) + J_C(\mathbf{q})^T \mathbf{f}_C(\mathbf{q}, \mathbf{v}, \tau) \tag{2}$$

where $\mathbf{q}$, $\mathbf{v}$, and M are position vector, velocity vector, and inertia matrix respectively in generalized coordinates. $\mathbf{b}$ contains bias forces (Coriolis, gravity, centrifugal, springs), $\mathbf{f}_E$ and $\mathbf{f}_C$ are impulses caused by equality constraints and contacts respectively and J_E and J_C are the corresponding Jacobians and τ are external/applied forces. The equations are then solved numerically. Mujoco obtains a discrete-time system with two options for integrators—semi-implicit Euler or 4th order explicit Runge–Kutta.

3.2 Parareal

Normally, computing all states $\mathbf{x}_n$ happens in a serial fashion, by evaluating (1) first for $n = 0$, then for $n = 1$, etc. Parareal replaces this inherently serial procedure by a parallel-in-time integration process where some of the work can be done in parallel. For Parareal, we need a coarse physics model

$$\mathbf{x}_{n+1} = C(\mathbf{x}_n, \mathbf{u}_n, \Delta t). \tag{3}$$

It needs to be computationally cheap relative to the fine model but does not have to be very accurate. Parareal begins by computing an initial guess $\mathbf{x}_n^{k=0}$ of the state at each time point n of the trajectory using the coarse model.

This guess is then corrected via the Parreal iteration

$$\mathbf{x}_{n+1}^{k+1} = C(\mathbf{x}_n^{k+1}, \mathbf{u}_n, \Delta t) + F(\mathbf{x}_n^k, \mathbf{u}_n, \Delta t) - C(\mathbf{x}_n^k, \mathbf{u}_n, \Delta t), \tag{4}$$

for all timesteps $n = 0, \dots, N-1$. The newly introduced superscript k counts the number of Parareal iterations. The key point in iteration (4) is that evaluating the fine physics model can be done in parallel for all $n = 0, \dots, N-1$, while only the fast coarse model has to be computed serially.

After one Parareal iteration, $\mathbf{x}_1^1$ is exactly the fine solution. After two iterations, $\mathbf{x}_1^1$ and $\mathbf{x}_2^2$ are exactly the fine solutions. When $k = N$, Parareal produces the exact fine solution [11,25]. However, to produce speed up, we need to stop Parareal at much earlier iterations. This way, Parareal can run in less wall-clock time than running the fine model serially step-by-step. Below, we demonstrate that even after a small number of iterations, the solution produced by Parareal is of sufficient quality to allow our robot to succeed with different tasks. Note that, for the sake of simplicity, we assume here that the number of controls N and the number of processors used to parallelize in time are identical, but this can easily be generalised.

4 Coarse models

In this section, we introduce two coarse physics models for Parareal - a learned coarse model and the analytical coarse model from Agboh et al. [4].

4.1 Learned coarse model

As an alternative to the coarse physics model, we train a deep neural network as a coarse model for Parareal for robotic pushing.

4.1.1 Network architecture

The input to our neural network model is a state $\mathbf{x}_n$ and a single action $\mathbf{u}_n$. The output is the change in state $\Delta\mathbf{x}$ which is added to the input state to obtain the next state $\mathbf{x}_{n+1}$. We use a feed-forward deep neural network (DNN) with 5 fully connected layers. The first 4 contain 512, 256, 128 and 64 neurons, respectively, with ReLU activation function. The output layer contains 24 neurons with linear activation functions.

4.1.2 Dataset

We collect training data using the physics engine Mujoco [37]. Each training sample is a tuple $(\mathbf{x}_n, \mathbf{u}_n, \mathbf{x}_{n+1})$. It contains a randomly[1] sampled initial state, action, and next state. We collect over 2 million such samples from the physics simulator.

During robotic pushing, a physics model may need to predict the resulting state even for cases when there is no contact between pusher and slider. We include both contact and no-contact cases in the training data.

We train a single neural network to handle one pusher with at least one and at most N_s objects being pushed (also called sliders). While collecting data for a particular number of sliders, we placed the unused sliders in distinct fixed positions outside the pushing workspace. These exact positions must be passed to the neural network at test time if fewer than N_s sliders are active. For example, if $N_s = 4$, to make a prediction for a 3 slider scene, we place the last slider at the same fixed position used during training.

4.1.3 Loss function

The standard loss function for training is the mean squared error between the network's prediction and the training data. On its own, this leads to infeasible state predictions where there is pusher-slider or slider-slider penetration. We resolve this by adding a no penetration loss term such that the final loss function reads:

$$
\begin{aligned}
f_l = W_F \cdot \sum_{i=1}^{N_s} \sum_{j=i+1}^{N_s} \min(||\mathbf{p}_i^{NN} - \mathbf{p}_j^{NN}|| - (r_i + r_j), 0)^2 \\
+ W_F \cdot \sum_{i=1}^{N_s} \min(||\mathbf{p}_P - \mathbf{p}_i^{NN}|| - (r_p + r_i), 0)^2 \\
+ ||\mathbf{x}^f - \mathbf{x}^{NN}||^2. \qquad (5)
\end{aligned}
$$

Here, W_F is a constant weight, $\mathbf{x}^f$ is the next state predicted by the fine model, $\mathbf{x}^{NN}$ is the next state predicted by the DNN model. $\mathbf{p}_i^{NN}$ and $\mathbf{p}_j^{NN}$ are the new positions of sliders i and j predicted by the DNN model, respectively, and $\mathbf{p}_P$ is the position of the pusher. r_p is the radius of the pusher, and r_i, r_j represent the radius of sliders i and j, respectively. The first line of Eq. 5 penalizes slider-slider penetration, the second line penalizes pusher-slider penetration, and the third line is the standard mean squared error.

Finally, the network makes a single step prediction. However, robotic manipulation typically needs a multi-step prediction as a result of a control sequence. To do this, we start from the initial state and apply the first action in the sequence to get a resulting next state. Then, we use this next state as a new input to the network together with the second action in the sequence and so on. This way, we repeatedly query the network with its previous predictions as the current state input.

4.2 Analytical coarse model

Agboh et al. [4] have proposed a simple, kinematic coarse physics model for pushing a single object. The model moves the slider with the same linear velocity as the pusher as long as there is contact between the two. We give details below for completeness:

$$
\mathbf{q}_{n+1}^S = \mathbf{q}_n^S + [u_n^x, u_n^y, \omega]^T \cdot p_c \cdot \Delta t \qquad (6)
$$

$$
p_c = \frac{d_{contact}}{d_{contact} + d_{free}}, \quad \omega = K_\omega \cdot \frac{||\mathbf{u}_n|| \cdot \sin\theta}{||\mathbf{r}_c||} \qquad (7)
$$

$$
\dot{\mathbf{q}}_{n+1}^S = \{[u_n^x, u_n^y, \omega]^T \ if\ p_c > 0,\ \dot{\mathbf{q}}_n^S\ otherwise\} \qquad (8)
$$

$$
\mathbf{q}_{n+1}^P = \mathbf{q}_n^P + \mathbf{u}_n \cdot \Delta t, \quad \dot{\mathbf{q}}_{n+1}^P = \mathbf{u}_n. \qquad (9)
$$

Here, p_c is the ratio of contact distance $d_{contact}$ travelled by the pusher when in contact with the slider and the total pushing distance, $\mathbf{r}_c$ is a vector from the contact point to the

[1] We use rejection sampling to ensure that sampled states do not have objects in penetration, i.e. fulfill the algebraic constraints of Eq. 2.

object's center at the current state $\mathbf{q}_n^S$, θ is the angle between the pushing direction and the vector $\mathbf{r}_c$, ω is the coarse angular velocity induced by the pusher on the slider. K_ω is a positive constant.

5 Planning and control

We use the predictive model based on Parareal described above in a planning and control framework for pushing an object on a table to a target location. We take an optimization approach to solve this problem. Given the table geometry, goal position, the current state of the pusher and all sliders $\mathbf{x}_0$, and an initial candidate sequence of controls $\{\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{N-1}\}$, the optimization procedure outputs an optimal sequence $\{\mathbf{u}_0^*, \mathbf{u}_1^*, \ldots, \mathbf{u}_{N-1}^*\}$ according to some defined cost.

The predictive model is used within this optimizer to *roll-out* a sequence of controls to predict the states $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$. These are then used to compute the cost associated with those controls. The details of the exact trajectory optimizer can be found in Agboh and Dogar [3]. The cost function we use penalizes moving obstacle sliders and dropping objects from the table but encourages getting the goal object into the goal location.

We use the trajectory optimizer in a model-predictive control (MPC) framework. Once we get an output control sequence from the optimizer, we *do not* execute the whole sequence on the real-robot serially one after the other. Instead, we execute only the first action, update $\mathbf{x}_0$ with the observed state of the system, and repeat the optimization to generate a new control sequence. We repeat this process until the task is complete.

Such an optimization-based MPC approach to pushing manipulation is frequently used to handle uncertainty and improve success in the real-world [2,6,18,23]. Here, our focus is to evaluate the performance of Parareal with learned coarse model for planning and control.

6 Experiments and results

In our experiments, we investigate three key issues. First, we investigate how fast Parareal converges to the fine solution for robotic pushing tasks with different coarse models. Second, we investigate the physics prediction accuracy of Parareal with respect to real-world pushing data. Finally, we demonstrate that the Parareal physics model can be used to complete real-robot manipulation tasks.

In Sect. 6.1 we provide preliminary information used throughout the experiments. Section 6.2 investigates convergence of Parareal for two different coarse models—the analytical coarse model for single object pushing and a learned coarse model for both single and multiple object pushing. In Sect. 6.3 we present results from real-robot experiments. First, we compare the accuracy of Parareal predictions against real-world pushing physics. Then, we show several real-robot plan executions using Parareal with a learned coarse physics model as predictive model.

6.1 Preliminaries

To generate physics-based robotic manipulation plans as fast as possible, we run Mujoco at the largest possible time-step (1ms) in all our experiments. Beyond this time-step the simulator becomes unstable, leading to unrealistically large object accelerations and breakdown of the simulator. We use the 4th order Runge–Kutta integrator for Mujoco. All computations run on a standard Laptop PC with an Intel(R) Core (TM) i7-4712HQ CPU @2.3 GHz with $N = 4$ cores. Our control sequences consist of four or eight actions, each applied for a control duration $\Delta_t = 1s$.

The software version used to create training data and run experiments was Mujoco 2.00 with DeepMind DM Control bindings to Python 3.5 [34]. To develop, train and test the coarse model the Keras API was used, which is built in to TensorFlow 2.0. We used a learning rate of 5e-4 with 100 epochs and a batch size of 1024 to train the neural network model.

Our real robot setup is shown in Fig. 1. We have a Robotiq two-finger gripper holding the cylindrical pusher of radius 1.45 cm. We place markers on the pusher and sliders to sense their full pose in the environment with an OptiTrack motion capture system. Section 3.1 states were defined to include orientation of objects but, to keep experiments simple, we use cylindrical objects such that only positions play a major role. The slider radius used in all experiments is 5.12 cm.

6.2 Parareal convergence

Parareal produces the exact fine physics solution when the number of iterations is equal to the number of timeslices regardless of the coarse physics model [11,25]. The convergence rate for scalar ordinary differential equations was theoretically shown to be superlinear on bounded intervals [11]. However, for the differential algebraic equations in Eq. 2 that describe the multi-contact dynamics problem, no such theoretical result exists and we study the convergence rate numerically.

We investigate through experiments how fast Parareal converges using two coarse models—the analytic model for single object pushing and the learned model for both single object and multi-object pushing. At each iteration, we compute a root mean square (RMS) error between Parareal's predictions and the fine model's predictions of the corresponding sequence of states. We compute the RMS error

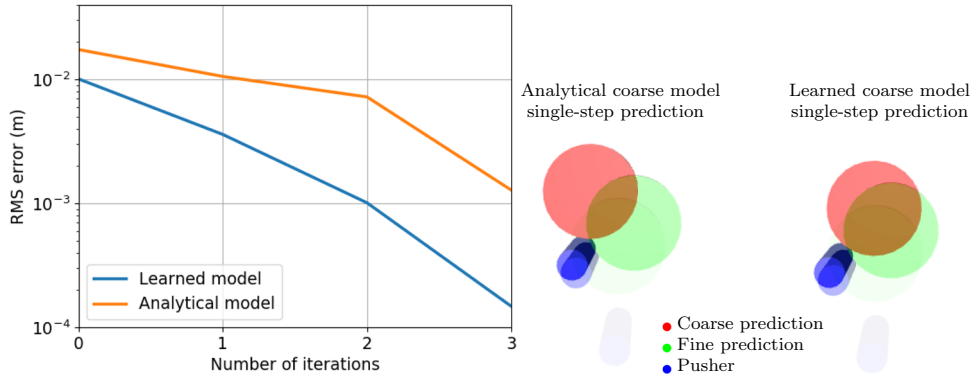

Fig. 2 Root mean square error (in log scale) of Parareal along the full trajectory for single object pushing using both a learned and an analytical coarse model (left). These results are for a control sequence with 4 actions where the average object displacement is 0.043 ± 0.033 m. The error at iteration four is 0. The learned coarse model gives a better Parareal convergence rate. Sample motions for the learned coarse model (center) and the analytical coarse model (right). The learned coarse model's prediction is closer to the fine model prediction shown in green

over only positions since we used cylindrical objects in all experiments.

6.2.1 Single object pushing

We randomly sample an initial state for the pusher and slider. We also randomly sample a control sequence where the pusher contacts the slider at least once during execution. Thereafter, we execute the control sequence starting from the initial state using Parareal. For the sample state and control sequence, we perform two runs, one using the learned model and the other using the analytical model as coarse propagator in Parareal.

We collect 100 state and control sequence samples. The analytical model makes a single step prediction 227.1 times faster than the fine model on average, while the learned model is 228.4 times faster on average. For example, to predict a 4 s long trajectory, the fine model requires 1.22 s while one iteration of Parareal requires only 0.31 s (for both models) on average. We see that both coarse models are so fast that our actual speedup in using Parareal is almost completely governed by the number of iterations.

Furthermore, for these samples, we also compute the RMS error between Parareal and the fine model run in serial. The results are shown in Fig. 2 (left) for a control sequence with 4 actions where the average object displacement is 0.043 ± 0.033 m.

We see that the learned model leads to faster convergence of Parareal than the analytical model for single object pushing. One reason for this could be that, in general, more accurate coarse models lead to better convergence. The single-step prediction of the learned model, shown in read in Fig. 2 (right), is much closer to the fine prediction shown in green than the analytical model shown in Fig. 2 (center).

6.2.2 Multi-object pushing

We randomly sample a valid initial state for the pusher and multiple sliders. Then, similar to the single object pushing case, we also sample a random control sequence that makes contact with at least one slider. We then predict the corresponding sequence of states using Parareal. However, for multi-object pushing we use only the learned model as the coarse physics model within Parareal. The analytical model for single-object pushing would need significant modifications to work for the multi-object case. Again, we collect 100 state and control sequence samples and run Parareal for each of them. Our results are shown in Fig. 3.

Figure 3 (left) shows the RMS error per slider for each Parareal iteration. While there are differences in the accuracy of the predictions for different slides, all errors decrease and Parareal converges at a reasonable pace.

These results are for a control sequence with 4 actions and where average object displacement is 0.015 ± 0.029 m. Some sample predictions are shown for a 4 slider environment in Fig. 3 (center), and for a 2-slider environment in Fig. 3 (right). In both scenes, the pusher moves forward making contact with multiple sliders and Parareal is able to predict how the state evolves.

We also investigate Parareal convergence for a longer control sequence of 8 actions. We do this for single object and multi-object pushing where all other conditions are the same as for the 4-action control sequence. Results can be found in Fig. 4 (left) for multi-object pushing and Fig. 4 (right) for single object pushing. The average object displacement for multi-object pushing is 0.034 ± 0.082 m and for single object pushing it is 0.046 ± 0.040 m. In general we find a similar convergence trend for both learned and analytical models for single and multi-object pushing.

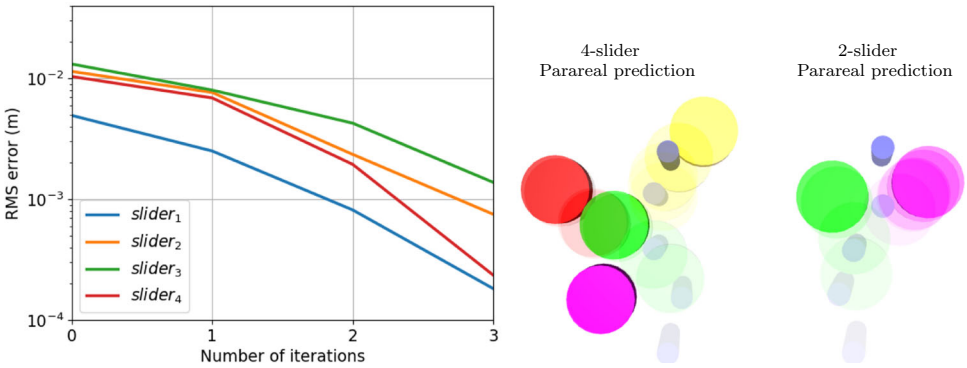

Fig. 3 Root mean square error (in log scale) along the full trajectory per slider in a 4-slider pushing experiment (left) using *only* the learned model. Two sample motions are illustrated (center and right) for multi-object physics prediction. These results are for a control sequence with 4 actions where the average object displacement is 0.015 ± 0.029 m. The error at iteration four is 0 except for accumulation of round-off errors.We find that the learned model enables Parareal convergence for the multi-object case

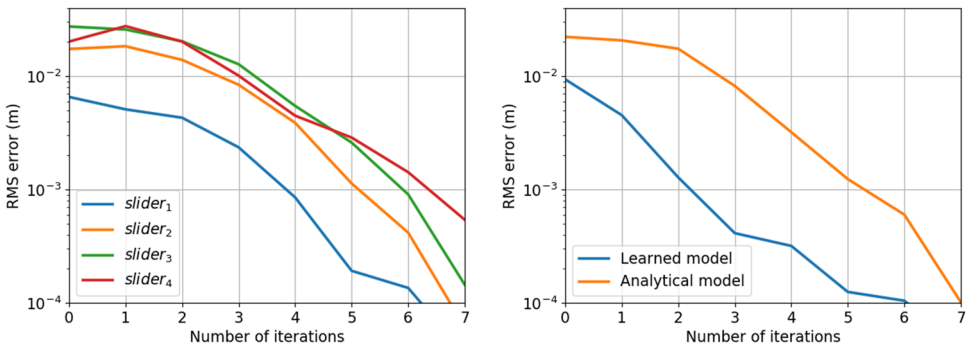

Fig. 4 Root mean square error (in log scale) along the full trajectory per object for single object pushing (right) and multiple object pushing (left) using *only* the learned model. Here we consider a control sequence of 8 actions. The average object displacement for multi-object pushing is 0.034 ± 0.082 m and for single object pushing it is 0.046 ± 0.040 m. The error at iteration eight is 0. We find that the convergence of Parareal appears similar even with a longer control sequence

Note that the shapes and sizes of the objects used are known and in fixed order. Therefore the learned model naturally does not generalize to new objects. However, it can still be used to make rather coarse predictions for similar objects.

6.3 Real robot experiments

In this section we investigate the physics prediction accuracy of Parareal with respect to real-world pushing physics. We do this for the multi-object case. In addition, we show real-world demonstrations for robotic manipulation where we use Parareal for physics prediction.

6.3.1 Parareal prediction versus real-world physics

Our coarse model neural network was trained using simulated data. Here, we demonstrate that Parareal using the trained coarse model is also able to predict real-world states. We randomly set an initial state in a real-world example by selecting positions for the pusher and sliders. This state is recorded using our motion capture system. Next, we sample a control sequence and let the real robot execute it. Again, we record the corresponding sequence of states using motion capture. Then, for the recorded initial state and control sequence pair, we use Parareal to produce the corresponding sequence of states and compare the result against the states measured for the real robot with optical tracking.

Figure 5 shows the RMS error between Parareal's prediction at different iteration numbers and the real-world pushing data. Vertical red bars indicate 95% confidence intervals.

Parareal's real-world error decreases with increasing iteration numbers and it is eventually twice as accurate as the coarse model. These results indicate that Parareal's predictions with a learned coarse model are indeed close to the real-world physics predictions. Figure 6 shows snapshots of the experiments.

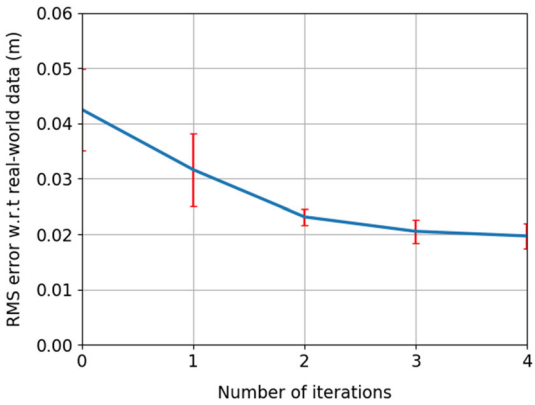

Fig. 5 Root mean square error along the full trajectory for all 4 sliders *measured with respect to the real-world pushing data*. The vertical bars indicate a 95% confidence interval of the mean. The learned coarse physics model at iteration 0 has the largest error and the fine model provides the best prediction w.r.t the real-world pushing physics

6.3.2 Planning and control

We use the Parareal predictive model for robotic manipulation to generate plans faster than using the fine model directly. In this section, we complete 3 real robot executions with Parareal at 1 iteration. We use the learned model as the coarse model in all cases.

As can be seen in Fig. 7, the robot's task is to push the green slider into the target region marked with X. The robot is allowed to make contact with other sliders. An execution fails when a non-goal object is pushed into the goal region or over the edge of the table.

The robot was successful for all 3 sample scenes. Some sample plans for two scenes are shown in Fig. 7. The third scene is shown in Fig. 1. We find that using Parareal with a learned coarse model for physics predictions, a robot can successfully complete complex real-world pushing manipulation tasks involving multiple objects. At 1 Parareal iteration, we complete the tasks about 4 times faster than directly using the fine model.

In general, we trade-off physics prediction accuracy with respect to time. An important question then is how many iterations of Parareal to use for physics-based robotic manipulation i.e. how accurate should the physics predictions be? This depends on the manipulation task. For example, physics prediction accuracy should be higher when a robot is tasked with pushing an object on a narrow strip versus a large table where the chances of failure are lower.

Figure 5 shows coarse physics errors (iteration 0) w.r.t. the real-world data of up to 5 cm which is about the radius of a slider. Therefore, we conclude that the coarse model alone is not sufficient to complete the robotic manipulation task

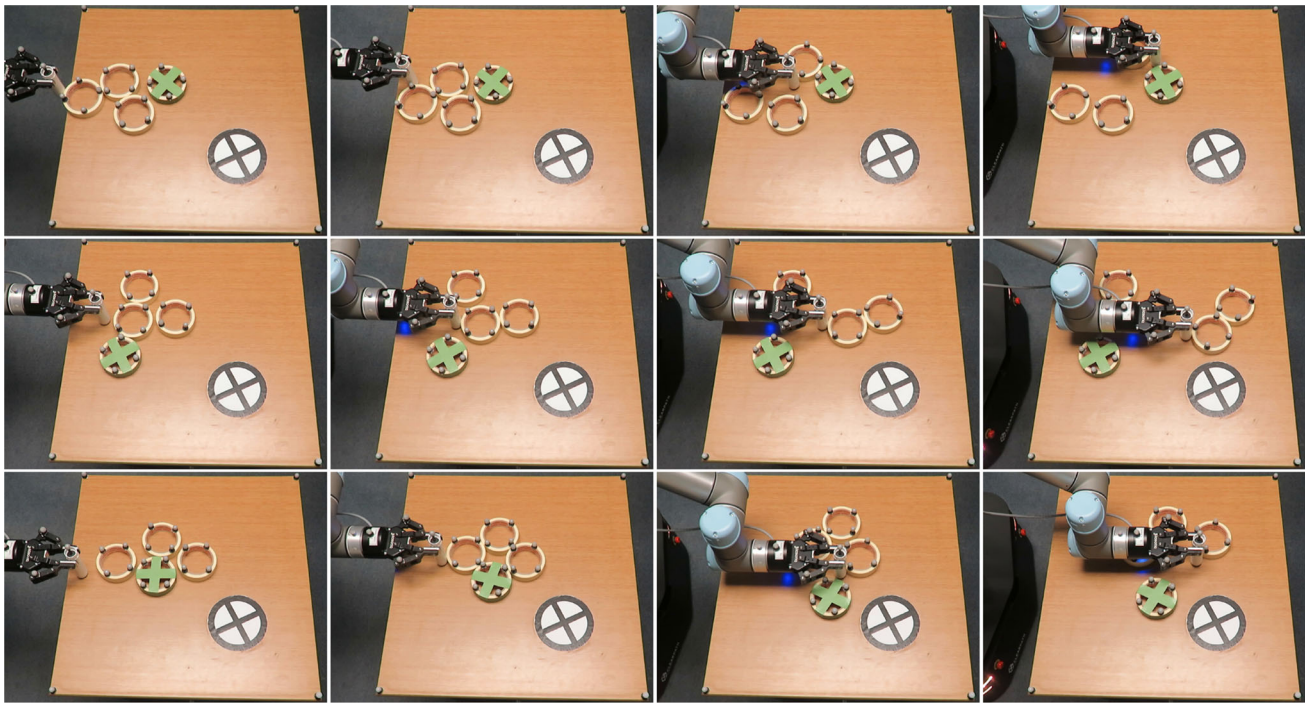

Fig. 6 The resulting sequence of states for applying a random control sequence starting from some random initial state in the real-world. Our goal is to assess the accuracy of the Parareal physics models with respect to real-world physics. We collect 50 such samples. These are some snapshots for 3 of such scenes—one per row with initial state on the left and final state on the right

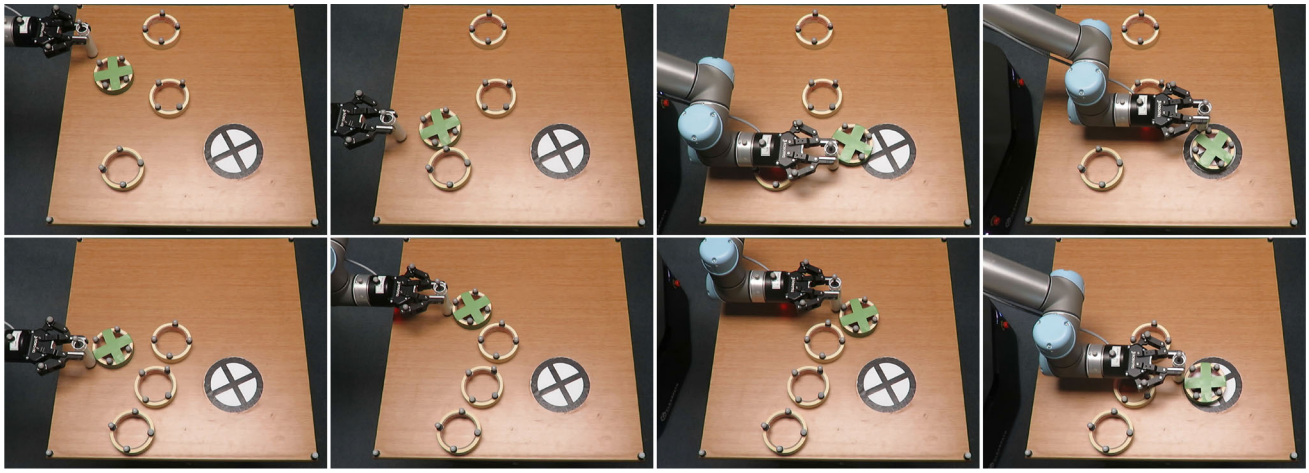

Fig. 7 Robotic manipulation planning and control for 2 different scenes. The robot succeeds in all scenes using Parareal with a learned coarse model for physics predictions. The third planning and control scene is in Fig. 1

considered here—an object can easily fall-off the table due to an inaccurately planned action.

Furthermore, there is uncertainty during robotic pushing in the real-world [41]. Agboh et al. [4] showed that physics predictions with errors below real-world stochasticity (e.g. position standard deviation at the end of a real-world push) have similar planning success rates. Hence it is usually pointless to have physics predictions as accurate as the fine model.

7 Summary

We demonstrate the promise of using Parareal to parallelize the predictive model in a robot manipulation task involving multiple objects. As coarse model, we propose a neural network, trained with a physics simulator. We show that for single object pushing, Parareal converges faster with the learned model than with a coarse physics-based model we introduced in earlier work. Furthermore, we show that Parareal with the learned model as coarse propagator can successfully complete tasks that involve pushing multiple objects. We also show that although a simulator is used to provide training data, Parareal with a learned coarse model can accurately predict experiments that involve pushing with a real robot.

References

1. Abraham, I., Handa, A., Ratliff, N., Lowrey, K., Murphey, T.D., Fox, D.: Model-based generalization under parameter uncertainty using path integral control. IEEE Robot. Autom. Lett. **5**(2), 2864–2871 (2020)
2. Agboh, W.C., Dogar, M.R.: Real-time online re-planning for grasping under clutter and uncertainty. In: IEEE-RAS International Conference on Humanoid Robots (2018)
3. Agboh, W.C., Dogar, M.R.: Pushing fast and slow: task-adaptive planning for non-prehensile manipulation under uncertainty. In: Algorithmic Foundations of Robotics XIII, pp. 160–176 (2020)
4. Agboh, W.C., Ruprecht, D., Dogar, M.R.: Combining coarse and fine physics for manipulation using parallel-in-time integration. In: International Symposium on Robotics Research (2019)
5. Ajay, A., Wu, J., Fazeli, N., Bauza, M., Kaelbling, L.P., Tenenbaum, J.B., Rodriguez, A.: Augmenting physical simulators with stochastic neural networks: case study of planar pushing and bouncing. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (2018)
6. Arruda, E., Mathew, M.J., Kopicki, M., Mistry, M., Azad, M., Wyatt, J.L.: Uncertainty averse pushing with model predictive path integral control. In: IEEE-RAS International Conference on Humanoid Robots (2017)
7. Bejjani, W., Dogar, M.R., Leonetti, M.: Learning physics-based manipulation in clutter: combining image-based generalization and look-ahead planning. In: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (2019)
8. Cadeau, T., Magoules, F.: Coupling the Parareal algorithm with the waveform relaxation method for the solution of differential algebraic equations. In: International Symposium on Distributed Computing and Applications to Business, Engineering and Science, pp. 15–19 (2011)
9. Erez, T., Tassa, Y., Todorov, E.: Simulation tools for model-based robotics: comparison of bullet, havok, mujoco, ode and physx. In: IEEE International Conference on Robotics and Automation (2015)

10. Fan, T., Schultz, J., Murphey, T.: Efficient computation of higher-order variational integrators in robotic simulation and trajectory optimization. In: Algorithmic Foundations of Robotics XIII, pp. 689–706 (2020)
11. Gander, M., Vandewalle, S.: Analysis of the parareal time-parallel time-integration method. SIAM J. Sci. Comput. **29**(2), 556–578 (2007)
12. Giftthaler, M., Neunert, M., Stäuble, M., Buchli, J., Diehl, M.: A family of iterative gauss-newton shooting methods for nonlinear optimal control. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (2018)
13. Guibert, D., Tromeur-Dervout, D.: Adaptive parareal for systems of ODEs. In: Widlund, O., Keyes, D. (eds.) Domain Decomposition Methods in Science and Engineering XVI. Lecture Notes in Computational Science and Engineering, vol. 55, pp. 587–594. Springer, Berlin (2007)
14. Günther, S., Ruthotto, L., Schroder, J.B., Cyr, E.C., Gauger, N.R.: Layer-parallel training of deep residual neural networks. SIAM J. Math. Data Sci. **2**(1), 1–23 (2019)
15. Hafner, D., Lillicrap, T., Ba, J., Norouzi, M.: Dream to control: Learning behaviors by latent imagination. In: International Conference on Learning Representations (ICLR) (2020)
16. Hasan, M., Warburton, M., Agboh, W.C., Dogar, M.R., Leonetti, M., Wang, H., Mushtaq, F., Mon-Williams, M., Cohn, A.G.: Human-like planning for reaching in cluttered environments. In: IEEE International Conference on Robotics and Automation (2020)
17. Haustein, J.A., King, J., Srinivasa, S.S., Asfour, T.: Kinodynamic randomized rearrangement planning via dynamic transitions between statically stable states. In: IEEE International Conference on Robotics and Automation (2015)
18. Hogan, F.R., Rodriguez, A.: Feedback control of the pusher-slider system: a story of hybrid and underactuated contact dynamics. In: Algorithmic Foundations of Robotics XII, pp. 800–815 (2020)
19. Hoque, R., Seita, D., Balakrishna, A., Ganapathi, A., Tanwani, A.K., Jamali, N., Yamane, K., Iba, S., Goldberg, K.: Visuospatial foresight for multi-step, multi-task fabric manipulation. In: Robotics: Science and Systems (RSS) (2020)
20. Huang, S.H., Zambelli, M., Kay, J., Martins, M.F., Tassa, Y., Pilarski, P.M., Hadsell, R.: Learning gentle object manipulation with curiosity-driven deep reinforcement learning (2019)
21. Johnson, A.M., King, J., Srinivasa, S.: Convergent planning. IEEE Robot. Autom. Lett. **1**(2), 1044–1051 (2016)
22. Kalakrishnan, M., Chitta, S., Theodorou, E., Pastor, P., Schaal, S.: Stomp: Stochastic trajectory optimization for motion planning. In: International Conference on Robotics and Automation (2011)
23. Kloss, A., Schaal, S., Bohg, J.: Combining learned and analytical models for predicting action effects. CoRR (2017). arXiv:1710.04102
24. Li, W., Todorov, E.: Iterative linear quadratic regulator design for nonlinear biological movement systems. In: International Conference on Informatics in Control, Automation and Robotics (2004)
25. Lions, J.L., Maday, Y., Turinici, G.: A "parareal" in time discretization of PDE's. Comptes Rendus de l'Académie des Sciences - Series I - Mathematics **332**, 661–668 (2001)
26. Luck, K.S., Vecerik, M., Stepputtis, S., Amor, H.B., Scholz, J.: Improved exploration through latent trajectory optimization in deep deterministic policy gradient. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2019)
27. Maday, Y., Turinici, G.: Parallel in time algorithms for quantum control: parareal time discretization scheme. Int. J. Quantum Chem. **93**(3), 223–228 (2003)
28. Neunert, M., Stäuble, M., Giftthaler, M., Bellicoso, C.D., Carius, J., Gehring, C., Hutter, M., Buchli, J.: Whole-body nonlinear model predictive control through contacts for quadrupeds. IEEE Robot. Autom. Lett. **3**(3), 1458–1465 (2018)
29. Pan, Z., Manocha, D.: Time integrating articulated body dynamics using position-based collocation method. In: Algorithmic Foundations of Robotics XIII, pp. 673–688 (2020)
30. Papallas, R., Dogar, M.R.: Non-prehensile manipulation in clutter with human-in-the-loop. In: IEEE International Conference on Robotics and Automation (ICRA) (2020)
31. Plancher, B., Kuindersma, S.: A performance analysis of parallel differential dynamic programming on a gpu. In: Algorithmic Foundations of Robotics XIII, pp. 656–672 (2020)
32. Schroder, J.: Parallelizing over artificial neural network training runs with multigrid (2017). arXiv:1708.02276
33. Seita, D., Ganapathi, A., Hoque, R., Hwang, M., Cen, E., Tanwani, A.K., Balakrishna, A., Thananjeyan, B., Ichnowski, J., Jamali, N., Yamane, K., Iba, S., Canny, J., Goldberg, K.: deep imitation learning of sequential fabric smoothing policies. In: International Symposium on Robotics Research (ISRR) (2019)
34. Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., de Las Casas, D., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., Lillicrap, T., Riedmiller, M.: DeepMind control suite. Tech. rep,, DeepMind (2018). arXiv:1801.00690
35. Tassa, Y., Erez, T., Todorov, E.: Synthesis and stabilization of complex behaviors through online trajectory optimization. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (2012)
36. Tedrake, R., the Drake Development Team: Drake: Model-based design and verification for robotics (2019). https://drake.mit.edu
37. Todorov, E., Erez, T., Tassa, Y.: Mujoco: A physics engine for model-based control. In: IEEE International Conference on Intelligent Robots and Systems (2012)
38. Trindade, J.M.F., Pereira, J.C.F.: Parallel-in-time simulation of two-dimensional, unsteady, incompressible laminar flows. Numer. Heat Transf. Part B Fundam. **50**(1), 25–40 (2006)
39. Williams, G., Aldrich, A., Theodorou, E.: Model predictive path integral control using covariance variable importance sampling. CoRR (2015)
40. Yalla, G.R., Engquist, B.: Parallel in time algorithms for multiscale dynamical systems using interpolation and neural networks. In: Proceedings of the High Performance Computing Symposium, pp. 9:1–9:12 (2018)
41. Yu, K.T., Bauza, M., Fazeli, N., Rodriguez, A.: More than a million ways to be pushed. A high-fidelity experimental dataset of planar pushing. In: IEEE International Conference on Intelligent Robots and Systems (IROS) (2016)