This is a repository copy of *Online state vector reduction during model predictive control with gradient-based trajectory optimisation*.

White Rose Research Online URL for this paper:
https://eprints.whiterose.ac.uk/217471/

Version: Accepted Version

Proceedings Paper:

Online state vector reduction
during model predictive control
with gradient-based trajectory optimisation

David Russell[0009−0002−5660−3890], Rafael Papallas[0000−0003−3892−1940], and
Mehmet Dogar[0000−0002−6896−5461]

School of Computer Science, University of Leeds, Leeds, LS2 9JT, UK
el16dmcr@leeds.ac.uk

Abstract. Non-prehensile manipulation in high-dimensional systems is
challenging for a variety of reasons. One of the main reasons is the com-
putationally long planning times that come with a large state space. Tra-
jectory optimisation algorithms have proved their utility in a wide variety
of tasks, but, like most methods struggle scaling to the high dimensional
systems ubiquitous to non-prehensile manipulation in clutter as well as
deformable object manipulation. We reason that, during manipulation,
different degrees of freedom will become more or less important to the
task over time as the system evolves. We leverage this idea to reduce the
number of degrees of freedom considered in a trajectory optimisation
problem, to reduce planning times. This idea is particularly relevant in
the context of model predictive control (MPC) where the cost landscape
of the optimisation problem is constantly evolving. We provide simula-
tion results under asynchronous MPC and show our methods are capable
of achieving better overall performance due to the decreased policy lag
whilst still being able to optimise trajectories effectively.

Keywords: Non-prehensile Manipulation · Model Predictive Control ·
Trajectory Optimisation · Dimensionality Reduction

1 Introduction

Trajectory optimisation algorithms are capable of synthesising complex motion
to solve a variety of challenging robotic manipulation tasks [12,13,24,26,16,18].
However, trajectory optimisation methods struggle scaling to high dimensional
systems, such as manipulation in cluttered environments or manipulation on
deformable objects. The long optimisation times caused by the curse of dimen-
sionality make it difficult to perform closed-loop model predictive control in these
high-dimensional systems.

   In this paper, we address this problem by trying to reduce the dimension-
ality of these systems dynamically so that closed-loop model predictive control
(MPC) can be achieved. Our key insight is that, in a variety of these high-
dimensional tasks, a large number of degrees of freedom (DoFs) are not relevant
to the problem at all times during task execution. This intuition makes sense on
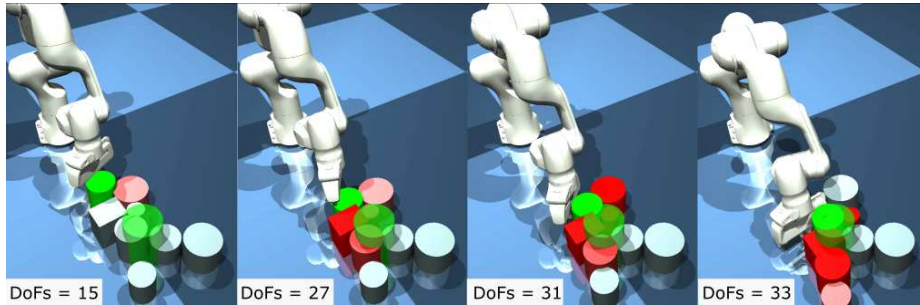
Fig. 1. A sequence of snapshots showing an example MPC trajectory generated by our method. The task is to push the green cylinder to a goal region (the green transparent cylindrical region) whilst minimally disturbing some clutter objects. The full number of DoFs in this system is 55. Our method identifies the relevant DoFs of this system at different times during execution and performs trajectory optimisation using this reduced state. Objects with stronger shades of red have more DoFs in the state vector at that point during execution: If an object is dark red, all of its six DoFs are considered; if an object is white, none of its six DoFs are considered during trajectory optimisation.

a fundamental level: Humans do not consider the full physical effects of all DoFs in a system when performing various tasks. When we fold clothes we do not consider in great detail the path each particle in the cloth will take; similarly, when we reach into a cluttered shelf, we are good at identifying which objects matter for our goals and ignoring others.

Fig. 1 shows a trajectory generated by our method for a non-prehensile manipulation task in clutter. Traditionally, during trajectory optimisation in such a scene, all six DoFs of all movable rigid objects (which contribute twelve state vector elements, the positional and velocity element of each DoF) are considered in the system state, in addition to the robot DoFs. Instead, we propose a method that can identify the relevant DoFs at different times during the task, and perform trajectory optimisation only with the reduced system state.

Reduced order models for efficient planning and control have been used in robotics before. Locomotion is one such area where reduced order models have found great performance in enabling real time closed-loop control of high-dimensional robots [3,10]. These can be pre-defined, such as inverted-pendulum models, or learned models given a task description [5]. While it is possible to take such approaches for locomotion (where the full model is almost always limited to the robot, and the reduced model is a lower-dimensional approximation of the same robot), it is more difficult to take a similar approach to object manipulation tasks as in Fig. 1, since the full model can include an arbitrary number, shape, and configuration of objects, which are unknown beforehand. That is why in this work we consider the method of reducing dimensionality *online* during task execution (MPC), given a task instance. Furthermore, the reduced state can be different for different stages of the task, as shown in the different snapshots in the figure.

It is important to note that we do *not* extract a reduced *dynamics* model of the system; rather, we only extract a reduced state vector. We show that there are significant gains (in terms of optimisation time) to be made with a reduced state vector, specifically when using gradient-based shooting trajectory optimisation methods, e.g., the iterative linear quadratic regulator (iLQR) [14]. Such methods require derivatives of the system dynamics to be calculated, often using computationally expensive *finite differencing* [22], which benefit significantly from a smaller state vector. Other operations within trajectory optimisation, such as the backwards propagation of the value function, also benefit from reduced state sizes. Therefore, in this paper, we also present our modifications to the iLQR method, which enable us to use it with a reduced state vector and make significant time gains, while the forward/dynamics model still uses the full state in a simulator [27].

We propose and compare different methods to identify the relevant DoFs of the system. We first propose a *naive* method, which considers a DoF relevant if and only if it appears in the cost function for a given task. While this in general gives a good heuristic, it can miss important DoFs that are not in the cost formulation (e.g., an object that is not considered in the cost, pushing another object that is in the cost), as well as include DoFs in the state vector that are not always relevant (e.g., an object that is in the cost formulation, but is not in a position to improve the cost given the current state and the nominal trajectory). Therefore, we also investigate methods that try to identify the relevant DoFs for the current state and around the current nominal trajectory. For this, we make use of the LQR gain matrix, K, which relates how changes in the current state should result in changes in current controls, for optimal behaviour. We propose two methods that use K: The first method directly uses the values in the columns of K to identify the state elements that can induce a large change in controls for optimal behaviour, and therefore are considered relevant. The second method performs an SVD decomposition on K to identify the principal axes and use them to identify the most important state elements. We compare the three methods above to two baselines: A vanilla iLQR that uses the full state vector at all times, and finally a method that *randomly* chooses subsets of the state vector during optimisation.

Our results show that the K-informed methods can identify the relevant DoFs of the task at different points during task execution (as can also be seen in Fig. 1), resulting in significantly lower optimisation times. Within an MPC loop, these lower optimisation times imply a lower policy lag, resulting in better MPC performance when compared with the baseline methods.

We make the following contributions in this paper[1]:

- We propose a general MPC approach that changes the elements inside the state vector *online* to minimise optimisation times, and therefore policy lag.
- We propose and compare different methods to identify the relevant DoFs inside the state vector given a task.

---

[1] Code and data: here

– We present our modifications to iLQR, to perform trajectory optimisation with a reduced state vector.
– We evaluate our methods on three high-dimensional non-prehensile manipulation tasks.

2   Related work

Trajectory optimisation in robotics has traditionally been used for finding smooth, collision-free motion plans [11,21]. Recently, its applications have expanded to more complex manipulation and locomotion tasks, such as in-hand manipulation and grasping [4,6], full-body manipulation [13], locomotion [20], and object manipulation in clutter [12,1,17].

One challenge of manipulation in clutter is physics uncertainty, where trajectories that are optimised in simulation often fail in the real-world due to unpredictable physical interactions. To address this challenge, Model Predictive Control (MPC) [8,31,19] is commonly used. MPC uses real-time model predictions to optimise control actions, allowing the system to adapt to changes and uncertainties as they occur. However, trajectory optimisation does not scale to high-dimensional spaces well. This is primarily due to the curse of dimensionality, which leads to long optimisation times. As the complexity of the task increases, so does the number of dimensions that need to be considered, making efficient optimisation and MPC increasingly difficult.

Various works explored different methods to speed up motion planning [2,23,17]. For example, Saleem and Likhachev [23] developed a planning method that uses a physics simulator only when necessary, relying on a cheaper geometric model for actions far from obstacles to reduce computation time. While these methods typically optimise a solution within several seconds, this duration poses challenges for real-time MPC.

Different methods for *dimensionality reduction* in robotics have been explored. For instance, Rapidly-exploring Random Trees (RRT) struggles to find valid motion plans as the dimensionality of the search space increases. Zheng et al. [32] consider reducing the dimensionality of the sample state space to reduce the time taken to find an optimal solution. Similarly, Vernaza et al. [28,29] consider locomotion problems for abstract robots that have a large number of DoFs that need to navigate around a cluttered space to some desired configurations whilst avoiding collisions. A different form of dimensionality reduction is considered by Hogan and Rodriguez [7], they formulate a direct transcription trajectory optimisation problem for a 2D push-slider system and consider the exponential nature of switching contact modes over a long trajectory horizon. To limit the exponential explosion, they limit the number of times the contact mode can switch over a trajectory to make the problem computationally tractable.

Manipulation of deformable objects is inherently a high-dimensional problem, often resulting in lengthy motion planning times. Wang et al. [30] consider the problem of manipulating 1D and 2D soft bodies (ropes and cloths) into goal configurations. To reduce computational costs, they computed a minimal set of

key-points that their algorithm can manipulate, enabling efficient planning of folding tasks. Mahoney et al. [15] considered reducing the dimensionality of a soft robot for computing collision free motion plans in tight environments, using principal component analysis.

Notably, Sharma and Chakravorty [25] propose a reduced order iLQR implementation similar to ours, where the roll-outs are performed using the full system dynamics. They evaluate their methods for optimisation until convergence for given high-dimensional PDEs. Our work focuses on robotic manipulation problems where analytical formulations of the PDEs are not readily available. We also focus on dynamically changing the size of the state vector used during *online* execution.

Ciorcarlie et al. [6] consider reducing the dimensionality of grasping problems by restricting the number of grasps to a set of key "eigengrasps". They show that this reduced action space is still capable grasping a wide variety of complex objects. Jin et al. [9] consider *learning* a reduced-order hybrid model for complex three finger manipulation, enabling real-time closed loop MPC. Chen et al. [5] consider the problem of learning a reduced order model for bipedal locomotion. In their work, they formulate a set of tasks for a bipedal robot to perform (walking on level ground, walking uphill etc) for which they want to compute a reduced order model offline that can be used for all their specified tasks. They manually specify a feature vector (a lower dimensional combination of features from the full-order model) and learn a set of linear weights for this feature vector offline by performing trajectory optimisation using the current reduced order model. The linear weights are adjusted via stochastic gradient descent.

These learning methods leverage prior distributions of the tasks to learn a reduced model. However, in robotic object manipulation, often the number and types of objects are unknown in advance, and each problem may require a different reduced model. We propose a method that can *dynamically* change the system state over time during MPC execution. At any given point, we use information from the trajectory optimiser to identify the most relevant DoFs for the current task configuration and optimise in this reduced state space. Our approach does not assume any prior task distribution, and we evaluate its effectiveness for both *rigid* and *deformable* object manipulation in cluttered environments.

3   Problem definition

We consider a discrete time dynamics formulation,

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) \tag{1}$$

where $\mathbf{x}_t$ and $\mathbf{u}_t$ are the state and control vectors respectively, at some time-step t along a trajectory.

We have some running cost function;

$$l(\mathbf{x}_t, \mathbf{u}_t) = (\mathbf{x}_t - \tilde{\mathbf{x}}_t)^\mathsf{T} W(\mathbf{x}_t - \tilde{\mathbf{x}}_t) + \mathbf{u}_t^\mathsf{T} R\mathbf{u}_t \tag{2}$$

where W and R are semi-positive definite cost weighting matrices and $\tilde{\mathbf{x}}_t$ is the desired state at time-step t.

The total running cost of a trajectory, J, is the summation of all the running costs as well as the terminal cost function, when a control sequence $\mathbf{U} \equiv (\mathbf{u}_0, \mathbf{u}_1, \cdots, \mathbf{u}_{T-1})$ is applied from some initial starting state $\mathbf{x}_0$;

$$J(\mathbf{x}_0, \mathbf{U}) = l_f(\mathbf{x}_T) + \sum_{t=0}^{T-1} l(\mathbf{x}_t, \mathbf{u}_t) \tag{3}$$

where l_f is the terminal cost function of the same form as Eq. 2 but with different matrices W_f and R_f. We also write the full trajectory state sequence that comes from rolling out the control sequence $\mathbf{U}$ as $\mathbf{X} \equiv (\mathbf{x}_0, \mathbf{x}_1, \cdots \mathbf{x}_T)$.

The general trajectory optimisation problem is to compute an optimal sequence of controls that minimise the total running cost of the trajectory, from some initial state:

$$\mathbf{U}^*(\mathbf{x}_0) = \arg \min_{\mathbf{U}} J(\mathbf{x}_0, \mathbf{U}) \tag{4}$$

We are interested in using trajectory optimisation within a model predictive control (MPC) framework. During each MPC iteration, the optimisation problem from Eq. 4 is solved, and the optimised controls are executed on the $real^2$ system, while another round of optimisation is initiated with the current state of the system. Therefore, the *control frequency* is determined by how quickly Eq. 4 is solved, and it has a significant effect on the performance of the MPC scheme.

Let $\underline{\mathbf{x}}_t$ represent the real system state during MPC execution at time t, and suppose the MPC executes Y controls before it stops[3]. We use `MPC_Cost` to quantify the MPC performance:

$$\texttt{MPC\_Cost} = l_f(\underline{\mathbf{x}}_Y) + \sum_{t=0}^{Y-1} l(\underline{\mathbf{x}}_t, \mathbf{u}_t) \tag{5}$$

The expression above is similar to Eq. 3 and uses the same cost functions l and l_f. The difference is that, while Eq. 3 is evaluated over the states $\mathbf{x}_t$ as predicted by the dynamics model, Eq. 5 is evaluated over the real states achieved by the system. Furthermore, while Eq. 3 sums over the optimisation horizon T, Eq. 5 sums over the complete duration of the execution, Y.

In this work, our aim is to improve the `MPC_Cost` by solving Eq. 4 faster, and therefore achieving a higher control frequency during MPC.

3.1   Definitions

We consider manipulation tasks consisting of a robot with q joints and multiple objects. We have N_O rigid objects and N_S deformable/soft objects. Rigid objects

---

[2] In this work, we use a simulator to also represent the *real* system.
[3] We stop the controller if a *Task timeout* is reached, or if a success condition is achieved.

simply have 6 degrees of freedom representing their pose. Deformable object i consists of N_{S_i} particles, and has three DoFs $\{x, y, z\}$ per particle. We use $\mathcal{F}$ to refer to the *set* of all DoFs in the system. Therefore,

$$|\mathcal{F}| = q + 6N_O + 3 \sum_{i=1}^{N_S} N_{S_i} \tag{6}$$

In this work we are interested in discovering and using a reduced set of DoFs, $\mathcal{C} \subseteq \mathcal{F}$. We will use superscript notation when referring to the reduced versions of vectors or matrices of our system. For example, $\mathbf{x}_t^{\mathcal{C}}$ refers to the reduced state vector which only includes elements corresponding to DoFs in $\mathcal{C}$. We do *not* use a superscript when the full set of DoFs, $\mathcal{F}$, is used.

We consider both positional and velocity elements for each DoF inside our state vector. Therefore, $\mathbf{x}_t \in \mathbb{R}^{2|\mathcal{F}|}$ whereas $\mathbf{x}_t^{\mathcal{C}} \in \mathbb{R}^{2|\mathcal{C}|}$. The control vector has size m, $\mathbf{u}_t \in \mathbb{R}^m$, and does not change size.

We denote the set of DoFs *not* currently in our reduced set of DoFs as $\mathcal{L}$, i.e., $\mathcal{L} = \mathcal{F} \setminus \mathcal{C}$.

4   Method

A general asynchronous MPC scheme (similar to Howell et al. [8]) can be seen in Alg. 1. An *agent* continuously executes an optimised control sequence $\mathbf{U}$. The optimiser continuously queries the current state from the agent (line 5) and optimises the control sequence $\mathbf{U}$ (line 9). Depending on how long the optimiser

---

Algorithm 1 MPC with state vector reduction (asynchronous)

1: $\mathcal{C} \leftarrow \textsc{InitialiseSubset}(\mathcal{F})$
2: $\mathcal{L} \leftarrow \mathcal{F} \setminus \mathcal{C}$
3: $\mathbf{U} \leftarrow \textsc{InitialiseControls}()$
    Optimiser Asynchronous (Planning)
4:     **while** task **not** complete **do**
5:        $\mathbf{x}_0 \leftarrow \textsc{GetCurrentState}()$
6:        $\texttt{dofs\_to\_add} \leftarrow \textsc{IdentifyDoFsToAdd}(\mathcal{L})$
7:        $\mathcal{C} \leftarrow \mathcal{C} \cup \texttt{dofs\_to\_add}$
8:        $\mathcal{L} \leftarrow \mathcal{L} \setminus \texttt{dofs\_to\_add}$
9:        $\mathbf{U}, \mathbf{K}^{\mathcal{C}} \leftarrow \textsc{Optimise}(\mathbf{x}_0, \mathbf{U}, \mathcal{C})$
10:       $\texttt{dofs\_to\_remove} \leftarrow \textsc{IdentifyDoFsToRemove}(\mathbf{K}^{\mathcal{C}})$
11:       $\mathcal{C} \leftarrow \mathcal{C} \setminus \texttt{dofs\_to\_remove}$
12:       $\mathcal{L} \leftarrow \mathcal{L} \cup \texttt{dofs\_to\_remove}$
    Agent Asynchronous (Execution)
13:     **while** task **not** complete **do**
14:       Execute first control from $\mathbf{U}$
15:       Remove first control from $\mathbf{U}$
16:       Pad $\mathbf{U}$ with the last control

---

takes to optimise the control sequence $\mathbf{U}$[4], the agent moves beyond the state that the optimiser is initialized with. The consequence of this is that the control sequence becomes less useful the longer optimisation takes; this is often referred to as *policy lag*.

In this work, we augment this general MPC formulation to dynamically change the size of the state vector currently being used in line 9, to reduce optimisation times, thus decreasing policy lag. We initialise our reduced set of DoFs in line 1: in this work we simply set $\mathcal{C} = \mathcal{F}$, but if a better heuristic is available that can be used instead. Before every optimisation call, we identify a number of unused DoFs in $\mathcal{L}$ to reintroduce to our reduced set of DoFs $\mathcal{C}$ on line 6. We then optimise a control sequence using this reduced set of DoFs and after computing a control sequence, we identify a set of DoFs that were unimportant to the previous trajectory optimisation problem on line 10. These DoFs are then removed from our reduced set of DoFs.

There are three important components of this general approach. Firstly, we need a method of optimising a trajectory using only a reduced set of DoFs (Alg. 1, line 9). This is explained in section 4.1 where we formulate performing iLQR [14,26] on our subset of reduced DoFs. Secondly, a method of determining which DoFs can be removed is required (Alg. 1, line 10). In Sec. 4.2 we outline different methods we propose to identify the DoFs to be removed. Finally, a method for reintroducing DoFs into our reduced set of DoFs is required (Alg. 1, line 6). In this work, we propose a simple random sampling strategy from the unused DoFs $\mathcal{L}$. It would be trivial to improve on the method for reintroducing DoFs into the system if specific information about the task and current trajectory was exploited. However, the purpose of this work was to suggest an abstract method for online state vector reduction that is task agnostic, as such we did not purse this line of work.

4.1 Optimise

In this section we outline how we augment the iLQR [14,26] algorithm to operate on our reduced set of DoFs $\mathcal{C}$. The high level overview is that we perform derivative computation and backwards pass calculations only for our reduced set of DoFs. We use the MuJoCo simulator to model the full system. When we perform forwards-rollouts, the full set of DoFs is updated using MuJoCo, as well as computing the total running cost of the trajectory. We name our iLQR adaption *iLQR with state vector reduction* (iLQR-SVR).

iLQR-SVR uses a first order approximation of the system dynamics where we write Eq. 1 as Eq. 7.

$$\mathbf{x}_{t+1}^{\mathcal{C}} = A_t^{\mathcal{C}} \mathbf{x}_t^{\mathcal{C}} + B_t^{\mathcal{C}} \mathbf{u}_t \tag{7}$$

---

[4] In MPC, we only perform one iteration of optimisation, not optimisation until convergence.

where $A_t^{\mathcal{C}} = \delta f(\mathbf{x}_t^{\mathcal{C}}, \mathbf{u}_t)/\delta \mathbf{x}_t^{\mathcal{C}}$ and $B_t^{\mathcal{C}} = \delta f(\mathbf{x}_t^{\mathcal{C}}, \mathbf{u}_t)/\delta \mathbf{u}_t$. iLQR also requires a first and second order approximation of the cost derivatives with respect to the state and control vector $(l_{\mathbf{x}}^{\mathcal{C}}, l_{\mathbf{xx}}^{\mathcal{C}}, l_{\mathbf{u}}, l_{\mathbf{uu}})$.

The computation of the dynamics derivatives $\mathbf{A} \equiv \{A_0, A_1, \cdots A_{T-1}\}$ and $\mathbf{B} \equiv \{B_0, B_1, \cdots B_{T-1}\}$ is often the bottleneck in gradient-based trajectory optimisation. These derivatives usually need to be computed via computationally costly *finite-differencing*. Finite-differencing requires evaluating the system dynamics for every DoF in the system as well as any control inputs. By only computing dynamics derivatives for our reduced state ($\mathbf{A}^{\mathcal{C}}$ and $\mathbf{B}^{\mathcal{C}}$) we reduce the number of dynamics evaluations to $2|\mathcal{C}| + m$ instead of $2|\mathcal{F}| + m$.

Using these approximations, optimal control modifications can be computed recursively using the dynamic programming principle [14,26]. This step is colloquially referred to as the *backwards pass.* and works by propagating the value function V from the end of the trajectory to the beginning by computing equations 8 - 10 from $t = T$ to $t = 0$. We augment the following equations to operate on our reduced set of DoFs:

$$Q_{\mathbf{x}}^{\mathcal{C}} = l_{\mathbf{x}}^{\mathcal{C}} + (A^{\mathcal{C}})^{\mathsf{T}} V_{\mathbf{x}}^{'\mathcal{C}} \tag{8a}$$

$$Q_{\mathbf{u}} = l_{\mathbf{u}} + (B^{\mathcal{C}})^{\mathsf{T}} V_{\mathbf{x}}^{'\mathcal{C}} \tag{8b}$$

$$Q_{\mathbf{xx}}^{\mathcal{C}} = l_{\mathbf{xx}}^{\mathcal{C}} + (A^{\mathcal{C}})^{\mathsf{T}} V_{\mathbf{xx}}^{'\mathcal{C}} A^{\mathcal{C}} \tag{8c}$$

$$Q_{\mathbf{uu}} = l_{\mathbf{uu}} + (B^{\mathcal{C}})^{\mathsf{T}} V_{\mathbf{xx}}^{'\mathcal{C}} B^{\mathcal{C}} \tag{8d}$$

$$Q_{\mathbf{ux}}^{\mathcal{C}} = l_{\mathbf{ux}}^{\mathcal{C}} + (B^{\mathcal{C}})^{\mathsf{T}} V_{\mathbf{xx}}^{'\mathcal{C}} A^{\mathcal{C}} \tag{8e}$$

At every time-step, these Q matrices can be used to compute an open-loop feedback term k as well as a closed-loop state feedback gain K.

$$k_t = -Q_{\mathbf{uu}}^{-1} Q_{\mathbf{u}} \tag{9a}$$

$$K_t^{\mathcal{C}} = -Q_{\mathbf{uu}}^{-1} Q_{\mathbf{ux}}^{\mathcal{C}} \tag{9b}$$

Finally the value function needs to be updated.

$$V_{\mathbf{x}}^{\mathcal{C}} = Q_{\mathbf{x}}^{\mathcal{C}} - Q_{\mathbf{u}} Q_{\mathbf{uu}}^{-1} Q_{\mathbf{ux}}^{\mathcal{C}} \tag{10a}$$

$$V_{\mathbf{xx}}^{\mathcal{C}} = Q_{\mathbf{xx}}^{\mathcal{C}} - Q_{\mathbf{u}} Q_{\mathbf{uu}}^{-1} Q_{\mathbf{ux}}^{\mathcal{C}} \tag{10b}$$

iLQR-SVR performs a forwards roll-out using the MuJoCo model, i.e., using the full non-linear system dynamics. We update the state vector for the full set of DoFs whilst only computing control modifications based on our reduced set of DoFs, as shown by Eq. 12.

$$\hat{\mathbf{x}}_0 = \mathbf{x}_0 \tag{11}$$

$$\hat{\mathbf{u}}_t = \mathbf{u}_t + \alpha k_t + K_t^{\mathcal{C}}(\hat{\mathbf{x}}_t^{\mathcal{C}} - \mathbf{x}_t^{\mathcal{C}}) \tag{12}$$

$$\hat{\mathbf{x}}_{t+1} = f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \tag{13}$$

Above, $\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t$ denotes the new computed trajectory states and controls and α is a line-search parameter between 0 and 1. If a lower cost trajectory is found, the nominal state and control trajectory is updated from the roll-out.

Remark 1. If a DoF is removed from $\mathcal{C}$, that DoF will still incur a cost when computing the running cost of a trajectory, i.e the cost function **always** operates over the full set of DoFs $\mathcal{F}$.

4.2   Reducing dimensionality

In this section, we outline how we identify DoFs to be removed from the system state. The core idea of our methods is leveraging information from the LQR gain matrices, K, which in our case, are already computed and returned by the iLQR algorithm, as the state-feedback gain matrices. In case another optimisation algorithm is used, then a linearised LQR approximation could be applied about the nominal trajectory, to compute a closed-loop state-feedback gain around it.

The $\mathbf{K} \equiv \{K_1, K_2 \cdots, K_{T-1}\}$ matrices compute a closed-loop control modification to add to the nominal control vector when performing a roll-out using full non-linear dynamics. This is achieved by calculating the difference in the current state along the new trajectory against the nominal trajectory for which derivatives were computed. This difference is then multiplied by a linear gain (i.e the K) matrix. This is shown explicitly for a single time-step in Eq. 14.

$$\begin{bmatrix} u(0) \\ u(1) \\ \vdots \\ u(m) \end{bmatrix} = \begin{bmatrix} K^{\mathcal{C}}(0,0) & K^{\mathcal{C}}(0,1) & \cdots & K^{\mathcal{C}}(0,2|\mathcal{C}|) \\ K^{\mathcal{C}}(1,0) & K^{\mathcal{C}}(1,1) & \cdots & K^{\mathcal{C}}(1,2|\mathcal{C}|) \\ \vdots & \vdots & \ddots & \vdots \\ K^{\mathcal{C}}(m,0) & K^{\mathcal{C}}(m,1) & \cdots & K^{\mathcal{C}}(m,2|\mathcal{C}|) \end{bmatrix} \begin{bmatrix} \hat{x}^{\mathcal{C}}(0) - x^{\mathcal{C}}(0) \\ \hat{x}^{\mathcal{C}}(1) - x^{\mathcal{C}}(1) \\ \vdots \\ \hat{x}^{\mathcal{C}}(2|\mathcal{C}|) - x^{\mathcal{C}}(2|\mathcal{C}|) \end{bmatrix} \tag{14}$$

We use notation $K(.,.)$ to denote indexing inside the matrix. The **columns** inside the K matrix correspond to computing entire control vector modifications based on the deviation of a specific DoF from the nominal. We reason that, DoFs that have small gain values associated with them over the entire trajectory are not important to the overall trajectory optimisation problem and can be ignored. This is because even if that DoF had a large deviation from its nominal position, it would have a minimal impact on the newly computed control.

We consider two approaches of leveraging this information, a) simply summing up the relevant values inside these matrices, b) performing singular value decomposition (SVD) to try find the most relevant DoFs. Both methods compute `dof_importance` values for every DoF in $\mathcal{C}$, we then simply use a threshold parameter ρ to determine which DoFs should remain in the $\mathcal{C}$ and which ones should be removed.

Summing The summing method simply leverages the values inside the K matrix for each DoF and how much of an impact they have on the control vector modification. If the values inside the K matrix are large for a particular DoF over the entire optimisation horizon, then that DoF is likely important to the trajectory optimisation problem currently. We compute a `dof_importance` value for each DoF using the following formula:

$$\texttt{dof\_importance}[j] = \sum_{t=0}^{T} \sum_{p=0}^{m} \Big(K_t^{\mathcal{C}}(p,j) + K_t^{\mathcal{C}}(p, j + |\mathcal{C}|) \Big) \Big/ T \qquad (15)$$

We sum over the number of controls in the control vector m as well as summing over all matrices over the optimisation horizon. Finally we sum both the columns corresponding to the positional and velocity element for the DoF j.

SVD The SVD method performs singular value decomposition (SVD) on the state feedback gain matrices:

$$K_t = U_t \Sigma_t V_t^{\mathsf{T}} \qquad (16)$$

where K is our state feedback gain matrix of size $m \times 2|\mathcal{C}|$, U is an orthogonal $m \times m$ matrix, Σ is a positive diagonal matrix of size $m \times 2|\mathcal{C}|$ and finally V is also an orthogonal matrix of size $2|\mathcal{C}| \times 2|\mathcal{C}|$.

The diagonal values of Σ are the singular values $\sigma_0 > \sigma_1 > \ldots \sigma_m$. We use the first g singular values (we used $g = 3$ in this work) and singular vectors in V to extract the dominant DoFs in the state vector. Specifically, we calculate `dof_importance` values for all DoFs in our current reduced set of DoFs, as:

$$\texttt{dof\_importance}[j] = \sum_{t=0}^{T} \sum_{n=0}^{g} \Big(\big(V_t(j,n) + V_t(j + |\mathcal{C}|, n) \big) \sigma_n \Big) \Big/ T \qquad (17)$$

We use $V(.,.)$ to denote indexing inside the matrix. Notably, for both methods of computing `dof_importance` values, we divide by the optimisation horizon so that ρ does not need to change dependant on the optimisation horizon used.

In both of these methods, we use the `dof_importance` values to determine which DoFs to remove for Alg. 1 line 10. Any DoFs that have an importance value that is below ρ are removed from the current reduced set of DoFs $\mathcal{C}$. One exception to this is that the robot DoFs are not allowed by any of the methods to be removed; i.e., the robot DoFs always appear in the state vector.

5   Results

We perform simulation experiments under an asynchronous MPC framework as described in Alg. 1. We provide results for a variety of methods. We have two baseline methods:

– *iLQR-Baseline*: A baseline method, where we perform iLQR as normal on the full set of DoFs in the system.
– *iLQR-Rand*: A baseline method where we perform iLQR on a random reduced set of DoFs. Instead of using any intelligent method to determine which DoFs, we instead randomly sample θ DoFs to use (in addition to the robot DoFs) in every optimisation iteration.

We have three methods that we evaluate against the baselines:

– *iLQR-Naive*: This method reduces the set of DoFs in our reduced set to only the DoFs that are directly considered in the cost function for the task.
– *iLQR-SVR-SVD*: Our method of dynamically changing the number of DoFs as described in Sec. 4.2-SVD.
– *iLQR-SVR-Sum*: Our method of dynamically changing the number of DoFs as described in Sec. 4.2-Summing.

All experiments were performed on a 16-core 11th Gen Intel(R) Core(TM) i7-11850H @ 2.50G with 32GB of RAM.

5.1   Task definition

We selected three high-dimensional non-prehensile manipulation tasks to provide experiments on, the main idea behind these tasks was to create tasks that were not too mechanically challenging but were high dimensional to see if our methods are capable of determining what DoFs are needed to solve the tasks efficiently. In some of the tasks that have very large state spaces, we were required to slow down the simulated agent thread by some factor to maintain some level of performance. All tasks used a Franka Panda robotic arm with 7 actuated robotic joints (not considering the grippers in the control vector).

We outline the general description of the three tasks as well as their task hyper-parameters below. T is the optimisation horizon, Δt was the model time-step. Y is the task timeout which is the number of time-steps in the agent thread until the task was finished. Finally, slowdown factor was how many times slower the agent thread was compared to the model time-step.
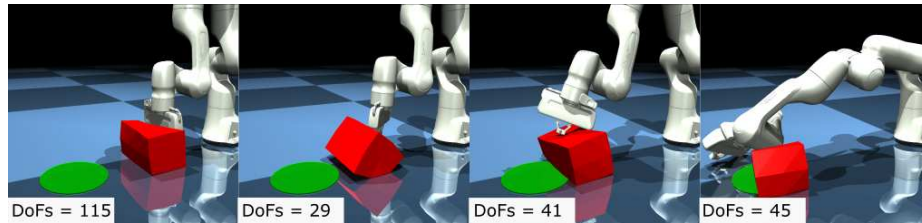

Fig. 2. A sequence of snapshots showing an example trajectory for the **soft** task. The objective is to push the red deformable object to the green flat circle on the floor. The full number of DoFs in this system is 115.
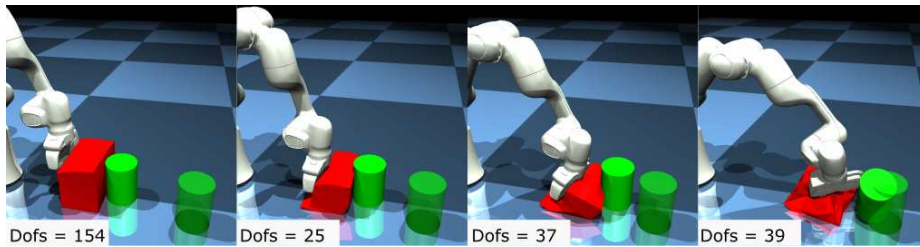
Fig. 3. A sequence of snapshots showing an example trajectory for the **soft rigid** task. The objective is to move the green cylinder to the goal region (the transparent cylindrical region) with a high-dimensional soft body being placed between the robot end-effector and the green cylinder. The full number of DoFs in this system is 154.

Clutter task: The aim of this task is to push a green cylinder to a target goal region (shown by a green silhouette) whilst minimally disturbing a set of distractor objects (example in Fig. 1). The task hyper-parameters were as follows; $T = 80$, $\Delta t = 0.004$, $Y = 2000$, Slowdown factor $= 1$.

Soft task: The aim of this task is to push a soft body. This task aimed to push a high dimensional red soft body to a goal location (example in Fig. 2). The task hyper-parameters were as follows; $T = 50$, $\Delta t = 0.004$, $Y = 1000$, Slowdown factor $= 3$.

Soft rigid task: This task aimed to push a green cylinder to a target location (green silhouette) but there is a high dimensional soft body in between the robot end-effector and the goal object. This means that the optimiser needs to reason about the dynamics between the soft body and the rigid body to achieve the task (example in Fig. 3). The task hyper-parameters were as follows; $T = 100$, $\Delta t = 0.004$, $Y = 1000$, Slowdown factor $= 5$.

5.2   Asynchronous MPC results

In this section we evaluate the performance of our proposed methods on the three outlined tasks. Earlier, in Eq. 5 we defined the `MPC_cost` as the cost of the *actual* trajectory executed in the real system. We use this as our metric of success. We show that our methods are capable of reducing the `MPC_cost` compared to the baselines, even though they are optimising only on a reduced state vector. This is because that whilst individual optimisation performance will be somewhat lower when only considering a subset of DoFs, the significant optimisation time savings reduce the detrimental effects of *policy lag*.

Table 1 shows the results from our asynchronous MPC experiments. We performed 100 runs for the clutter task and 20 runs for the soft and soft rigid tasks. The first value in the table are the mean values and the second values are 90% confidence intervals. It should be noted that we removed outliers for the **clutter** task to prevent individual data points from disproportionately affecting the results. In this task, due to cylinders being used as the goal and distractor objects, in certain occasions if a cylinder was toppled with some large velocity,

Table 1. Results of asynchronous MPC for three manipulation tasks. The values in the table are averaged over 100 trials for the clutter task and 20 trials for the soft and soft rigid task. The first value is the mean and the second is the 90% confidence interval. For the three first methods, the number of DoFs is a preset parameter, while the rest of the methods adjust the number of DoFs dynamically.

Method / Task		Clutter	Soft	Soft rigid
iLQR-Baseline	MPC cost	1.00 ± 0.034	1.00 ± 0.080	1.00 ± 0.096
	Opt time (ms)	485.71 ± 5.16	620.97 ± 7.35	4322.39 ± 366.65
	Num DoFs (preset)	55	115	154
iLQR-Rand	MPC cost	0.94 ± 0.033	1.14 ± 0.104	0.74 ± 0.071
	Opt time (ms)	148.91 ± 1.69	79.12 ± 4.13	881.54 ± 7.67
$\theta = 5$	Num DoFs (preset)	12	12	12
iLQR-Naive	MPC cost	0.89 ± 0.035	0.92 ± 0.067	$\mathbf{0.38 \pm 0.066}$
	Opt time (ms)	219.07 ± 2.11	395.52 ± 7.71	583.06 ± 85.72
	Num DoFs (preset)	23	79	9
iLQR-SVR-SVD	MPC cost	$\mathbf{0.86 \pm 0.033}$	$\mathbf{0.90 \pm 0.065}$	0.88 ± 0.071
	Opt time (ms)	335.16 ± 13.61	367.18 ± 49.60	2320.80 ± 392.91
$\theta = 10$, $\rho = 1$	Num DoFs	31.85 ± 1.12	72.55 ± 9.37	85.19 ± 18.32
iLQR-SVR-SVD	MPC cost	0.91 ± 0.036	0.91 ± 0.069	0.66 ± 0.063
	Opt time (ms)	237.79 ± 5.76	116.20 ± 4.73	1051.20 ± 32.55
$\theta = 10$, $\rho = 500$	Num DoFs	21.06 ± 0.33	23.32 ± 0.77	24.72 ± 0.17
iLQR-SVR-SVD	MPC cost	0.91 ± 0.036	0.90 ± 0.061	0.89 ± 0.068
	Opt time (ms)	271.21 ± 11.41	251.90 ± 38.30	1296.29 ± 184.13
$\theta = 5$, $\rho = 1$	Num DoFs	24.38 ± 1.00	49.59 ± 7.34	38.69 ± 8.66
iLQR-SVR-Sum	MPC cost	0.86 ± 0.032	0.94 ± 0.073	0.77 ± 0.074
	Opt time (ms)	287.05 ± 10.91	136.18 ± 10.92	1130.83 ± 54.68
$\theta = 10$, $\rho = 1$	Num DoFs	29.82 ± 1.01	26.68 ± 2.08	29.10 ± 2.92
iLQR-SVR-Sum	MPC cost	0.87 ± 0.029	1.00 ± 0.081	0.70 ± 0.071
	Opt time (ms)	192.59 ± 2.08	101.15 ± 4.73	997.71 ± 32.55
$\theta = 10$, $\rho = 500$	Num DoFs	18.69 ± 0.10	18.20 ± 0.11	24.09 ± 0.46
iLQR-SVR-Sum	MPC cost	0.94 ± 0.042	0.97 ± 0.079	0.75 ± 0.066
	Opt time (ms)	218.45 ± 8.38	114.74 ± 13.76	935.45 ± 37.03
$\theta = 5$, $\rho = 1$	Num DoFs	21.91 ± 0.86	19.94 ± 1.87	20.21 ± 0.91

it would roll into the distance away from its goal position until the task timeout. The result of this would be a disproportionately high cost, in situations where this occurred we removed these outliers. Finally, we normalise all the cost values with respect to the *iLQR Baseline* method.

The results show that, for all three tasks, our methods (*iLQR-Naive*, *iLQR-SVR-SVD* and *iLQR-SVR-Sum*) are capable of achieving a lower `MPC_Cost` than the *iLQR-Baseline*. *iLQR-Naive* on average reduces the `MPC cost` by 27% over all three tasks. Choosing the best parameterisation of our dynamic methods, they manage to lower the MPC cost by 14%. Importantly, we show that simply limiting the size of the state vector does not achieve the same level of performance as our intelligent methods as *iLQR-Rand* only reduces the `MPC cost` by 6%.
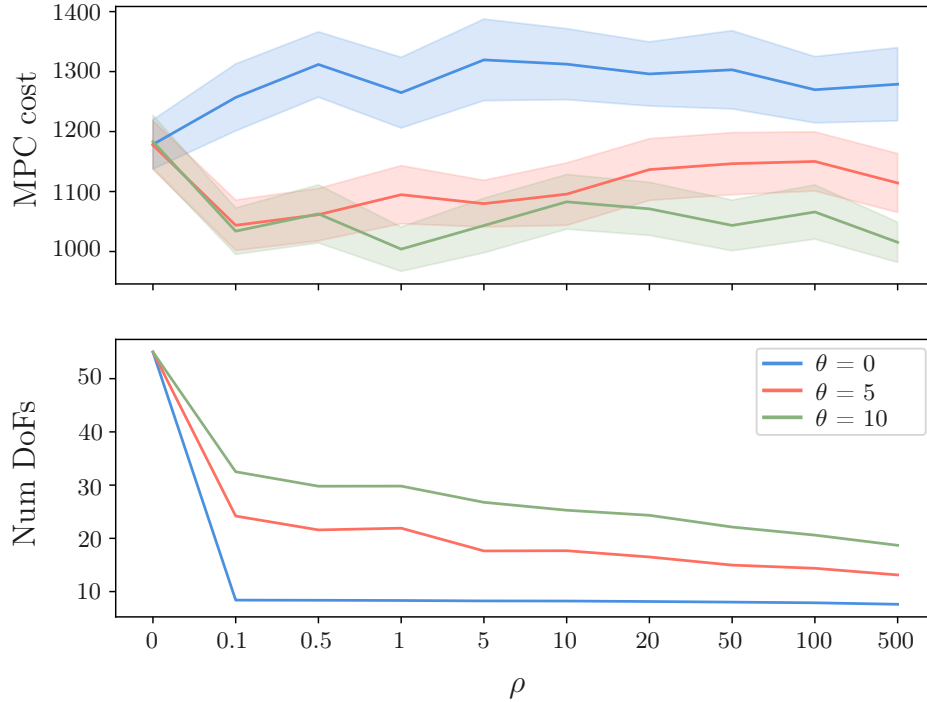
Fig. 4. Top plot shows the `MPC cost` averaged over 100 runs for the clutter task. The lightly shaded area shows the 90% confidence interval range. Bottom plot shows the average number of DoFs in our state vector. Three different parameterisations of θ were used with scaling values for ρ.

These results are skewed slightly by the **soft rigid** task, where the *iLQR-Naive* method significantly outperformed all other methods. If we only consider the **clutter** and **soft** tasks then *iLQR-Naive* reduces the `MPC_cost` by 8.5%, *iLQR-SVR* by 12%, whereas *iLQR-Rand* actually increase the `MPC_cost` by 4%.

This implies that our K-informed methods are more capable in these two tasks. This makes sense as the Naive methods always include DoFs in the state vector that are not necessarily important. For example, in the **clutter** task, when some objects are not involved in contact with the goal object / robot directly or indirectly, considering their dynamical properties adds nothing of value to the trajectory optimisation solution. Our K-informed methods are capable of deducing this automatically.

It is important to recognise that we only show three parameterisations of θ and ρ in these results and we keep them the same for all three tasks to test the generalisability of our methods. Better results could have been achieved if these values were tuned to each task specifically.

We also experimented with the effects of modifying the values for θ and ρ for the clutter task, the results of this can be seen in Fig. 4. We experiment

with three different values for θ and scale the value of ρ between 0 and 500 (please note the values are not spaced equally). Firstly, when ρ is set to zero, *iLQR-SVR* operates identically to *iLQR-Baseline*. This can be seen clearly in Fig. 4 as all three data points have nearly identical `MPC Costs`. As the value for ρ is increased, it makes our methods "pickier" at what DoFs remain in the state vector. This can clearly be seen in the bottom plots where the average number of DoFs in the state vector decreases as ρ increases.

When θ is set to zero, no DoFs are ever re-introduced to our reduced set of DoFs. This means that once a DoF has been removed it never has the opportunity to be reconsidered in optimisation. This is clearly a bad strategy as it negates the core idea of our methodology which is that DoFs importance can change over time during a task, noticeably in Fig. 4, this value of θ performs noticeably worse compared to the other two parameterisations of θ.

The optimal parameterisation for ρ would be some moderate value, as when ρ approaches zero *iLQR-SVR* performance should tend towards *iLQR-Baseline* and when ρ tends towards large values, *iLQR-SVR* performance tend towards *iLQR-Rand*.

6   Discussion

In this work, we have made several contributions, firstly we have outlined a general method for changing the size of a state vector to be used in MPC *online*. Secondly we have outlined an intelligent method of reducing the number of DoFs considered in trajectory optimisation and shown that our methods can increase performance when used in an asynchronous MPC format.

An unfortunate consequence of using a thresholding method as we have outlined in this work to decide whether DoFs should or should not be included in the state vector is that the threshold can sometimes be hard to tune and can be task dependant. More work is required to determine a better method to tune this parameter automatically. One such method that could perhaps be more robust would be to decide on a *desired* state vector size prior to performing MPC. Instead of removing DoFs if they fall below some importance threshold, we could instead retain the most important DoFs up to our desired state vector size.

In this work we only consider discrete methods of reducing the size of the state vector, i.e a DoF is either considered completely or it is not. A promising line of future work would be to consider more conventional methods of dimensionality reduction, by combing DoFs into some lower dimensional space that could be discovered using principal component analysis.

Acknowledgments

References

1. Agboh, W.C., Dogar, M.R.: Real-time online re-planning for grasping under clutter and uncertainty. In: 2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids). pp. 1–8. IEEE (2018)
2. 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: Proceedings of the 13th Workshop on the Algorithmic Foundations of Robotics 13. pp. 160–176. Springer (2020)
3. Blickhan, R.: The spring-mass model for running and hopping. Journal of biomechanics 22(11-12), 1217–1227 (1989)
4. Charlesworth, H.J., Montana, G.: Solving challenging dexterous manipulation tasks with trajectory optimisation and reinforcement learning. In: International Conference on Machine Learning. pp. 1496–1506. PMLR (2021)
5. Chen, Y.M., Posa, M.: Optimal reduced-order modeling of bipedal locomotion. In: 2020 IEEE International Conference on Robotics and Automation (ICRA). pp. 8753–8760. IEEE (2020)
6. Ciocarlie, M., Goldfeder, C., Allen, P.: Dimensionality reduction for hand-independent dexterous robotic grasping. In: 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 3270–3275. IEEE (2007)
7. Hogan, F.R., Rodriguez, A.: Feedback control of the pusher-slider system: A story of hybrid and underactuated contact dynamics. arXiv preprint arXiv:1611.08268 (2016)
8. Howell, T., Gileadi, N., Tunyasuvunakool, S., Zakka, K., Erez, T., Tassa, Y.: Predictive Sampling: Real-time Behaviour Synthesis with MuJoCo (dec 2022). `https://doi.org/10.48550/arXiv.2212.00541`, `https://arxiv.org/abs/2212.00541`
9. Jin, W., Posa, M.: Task-driven hybrid model reduction for dexterous manipulation. IEEE Transactions on Robotics (2024)
10. Kajita, S., Tani, K.: Study of dynamic biped locomotion on rugged terrain-derivation and application of the linear inverted pendulum mode. In: Proceedings. 1991 IEEE International Conference on Robotics and Automation. pp. 1405–1406. IEEE Computer Society (1991)
11. Kalakrishnan, M., Chitta, S., Theodorou, E., Pastor, P., Schaal, S.: Stomp: Stochastic trajectory optimization for motion planning. In: 2011 IEEE international conference on robotics and automation. pp. 4569–4574. IEEE (2021)
12. Kitaev, N., Mordatch, I., Patil, S., Abbeel, P.: Physics-based trajectory optimization for grasping in cluttered environments. In: 2015 IEEE International Conference on Robotics and Automation (ICRA). pp. 3102–3109. IEEE (2015)
13. Kurtz, V., Lin, H.: Contact-implicit trajectory optimization with hydroelastic contact and ilqr. arXiv preprint arXiv:2202.13986 (2022)
14. Li, W., Todorov, E.: Iterative linear quadratic regulator design for nonlinear biological movement systems. In: ICINCO (1). pp. 222–229. Citeseer (2004)
15. Mahoney, A., Bross, J., Johnson, D.: Deformable robot motion planning in a reduced-dimension configuration space. In: 2010 IEEE International Conference on Robotics and Automation. pp. 5133–5138. IEEE (2010)
16. Önol, A.Ö., Corcodel, R., Long, P., Padır, T.: Tuning-free contact-implicit trajectory optimization. In: 2020 IEEE International Conference on Robotics and Automation (ICRA). pp. 1183–1189. IEEE (2020)
17. Papallas, R., Cohn, A.G., Dogar, M.R.: Online replanning with human-in-the-loop for non-prehensile manipulation in clutter—a trajectory optimization based approach. IEEE Robotics and Automation Letters 5(4), 5377–5384 (2020)

18. Papallas, R., Dogar, M.R.: To ask for help or not to ask: A predictive approach to human-in-the-loop motion planning for robot manipulation tasks. In: 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 649–656. IEEE (2022)
19. Pezzato, C., Salmi, C., Trevisan, E., Mora, J.A., Corbato, C.H.: Sampling-based mpc using a gpu-parallelizable physics simulator as dynamic model: an open source implementation with isaacgym. In: Embracing Contacts-Workshop at ICRA 2023 (2023)
20. Posa, M., Cantu, C., Tedrake, R.: A direct method for trajectory optimization of rigid bodies through contact. The International Journal of Robotics Research **33**(1), 69–81 (2014)
21. Ratliff, N., Zucker, M., Bagnell, J.A., Srinivasa, S.: Chomp: Gradient optimization techniques for efficient motion planning. In: 2009 IEEE International Conference on Robotics and Automation. pp. 489–494. IEEE (2009)
22. Russell, D., Papallas, R., Dogar, M.: Adaptive approximation of dynamics gradients via interpolation to speed up trajectory optimisation. In: 2023 IEEE International Conference on Robotics and Automation (ICRA). pp. 10160–10166. IEEE (2023)
23. Saleem, M.S., Likhachev, M.: Planning with selective physics-based simulation for manipulation among movable objects. In: 2020 IEEE International Conference on Robotics and Automation (ICRA). pp. 6752–6758. IEEE (2020)
24. Selvaggio, M., Garg, A., Ruggiero, F., Oriolo, G., Siciliano, B.: Non-prehensile object transportation via model predictive non-sliding manipulation control. IEEE Transactions on Control Systems Technology (2023)
25. Sharma, A., Chakravorty, S.: A reduced order iterative linear quadratic regulator (ilqr) technique for the optimal control of nonlinear partial differential equations. In: 2023 American Control Conference (ACC). pp. 3389–3394. IEEE (2023)
26. Tassa, Y., Erez, T., Todorov, E.: Synthesis and stabilization of complex behaviors through online trajectory optimization. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 4906–4913. IEEE (2012)
27. Todorov, E., Erez, T., Tassa, Y.: Mujoco: A physics engine for model-based control. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 5026–5033. IEEE (2012)
28. Vernaza, P., Lee, D.: Learning dimensional descent for optimal motion planning in high-dimensional spaces. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 25, pp. 1126–1132 (2011)
29. Vernaza, P., Lee, D.D.: Learning and exploiting low-dimensional structure for efficient holonomic motion planning in high-dimensional spaces. The International Journal of Robotics Research **31**(14), 1739–1760 (2012)
30. Wang, S., Papallas, R., Leonetti, M., Dogar, M.: Goal-conditioned action space reduction for deformable object manipulation. In: 2023 IEEE International Conference on Robotics and Automation (ICRA). pp. 3623–3630. IEEE (2023)
31. Williams, G., Aldrich, A., Theodorou, E.A.: Model predictive path integral control: From theory to parallel computation. Journal of Guidance, Control, and Dynamics **40**(2), 344–357 (2017)
32. Zheng, D., Tsiotras, P.: Accelerating kinodynamic rrt* through dimensionality reduction. In: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 3674–3680. IEEE (2021)