

# Model Predictive Control with Efficient Trajectory Optimisation for Contact-based Manipulation



David Mackenzie Charles Russell

School of Computer Science

The University of Leeds

Submitted in accordance with the requirements for the degree of

*Doctor of Philosophy*

5th June 2025

Dedicated to my parents, without whom I would have never begun my PhD, and to Sophie, without whom I would have never completed it.

## Declaration

I confirm that the work submitted is my own, except where work which has formed part of jointly authored publications has been included. My contribution and the other authors to this work has been explicitly indicated below. I confirm that appropriate credit has been given within the thesis where reference has been made to the work of others.

Some of the results and work presented in this thesis have been published in the following papers:

- Russell, David, Rafael Papallas, and Mehmet Dogar. “Adaptive approximation of dynamics gradients via interpolation to speed up trajectory optimisation.” 2023 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2023.
- Russell, D., R. Papallas, and M. Dogar. “Online state vector reduction during model predictive control with gradient-based trajectory optimisation.” Springer Proceedings in Advanced Robotics (SPAR). Springer, 2024.

The above publications are primarily the work of myself. The co-authors on these papers helped in an advisory role along with proof reading publications. The development, experimentation and analysis was performed entirely by myself.

This copy has been supplied on the understanding that it is copyright material and that no quotation from my thesis may be published without proper acknowledgment. The right of David Russell to be identified as author of this work has been asserted by David Russell in accordance with the Copyright, Designs and Patents Act 1988.

# Acknowledgements

There are many people I would like to thank for helping me complete this thesis. First and foremost, I extend my sincere thanks to my supervisor, Professor Mehmet Dogar. Before starting this PhD, I knew very little about robotics—and even less about what it takes to be a good researcher. Throughout the PhD journey, there were many times when I struggled: sometimes with understanding new concepts, other times with developing my algorithms or communicating my research to a wider audience. I would not have been able to overcome these challenges without your constant encouragement, guidance, and friendship. I truly could not have asked for a better supervisor.

I would like to express my gratitude to my examiners, Bruno Vilhena Adorno and Songyan Xin, for taking the time to evaluate and give constructive feedback on my thesis. Your invaluable input has not only strengthened this thesis but also inspired exciting avenues for future research.

I would like to thank my fellow PhD students at the University of Leeds Robotic Manipulation Lab. Shengyin Wang and Zisong Xu were here before I started and were always so friendly. I learned a great deal from both of you, and have enjoyed watching your works progress over the years. Xiao Wang, Yulei Qiu, Longrui Chen and Ga Jun Locher joined our lab during my PhD and it has been so nice getting to know you all. I wish you all the best of luck in the future.

Special thanks to Rafael Papallas. You have been a great mentor to me over the last few years and I learned a great many things from you: how to write clean and concise code, how to be a better researcher and how to fix things that were broken in a systematic way. More than that, you were a great friend; I will particularly miss our games of squash.

A PhD is not completed in isolation, often there are people behind the scenes offering their support and encouragement. This thesis is no exception, as such, I would like to thank all my family for their constant support over the years. To my parents, Fiona and Matthew, and my siblings, Edward and Jennie, along with my siblings-in-law, Jenna and Matt - thank you for always being there for me. A special thank you to my mum, who always let me ramble on about everything I was working on, despite never having a clue what I was talking about. A special mention goes to my three wonderful nieces: Ava and Mia, who have brought so much laughter and joy into my life, and little Hannia, who I am certain will be just as wonderful. Even though he can not read, I would like to thank my beautiful Golden Retriever, Buster. You were the best boy, and I will miss you dearly.

Finally, thanks to my Sophie. You were always there for me, listening to me rant and stress about everything relating to my general PhD work and also with writing this thesis. Whilst this PhD journey has been great and I am excited to have achieved my doctorate, I truly believe the best thing to come out of this PhD was meeting you. Thank you for agreeing to come with me on this journey called life.

# Abstract

This thesis is concerned with enabling robots to manipulate their environments in efficient ways, similar to how humans do. Humans employ a variety of manipulation actions to efficiently manipulate their environments using non-prehensile manipulation. However, enabling robots to make use of these efficient non-prehensile manipulation actions is challenging, particularly when tasking robots to manipulate cluttered environments. There are a variety of issues that need to be addressed to enable robots to reliably operate in cluttered environments. These include, but are not limited to, difficulty predicting contact-interactions between multiple objects, as well as the curse of dimensionality that occurs from scaling levels of clutter.

General purpose physics simulators can be used to predict, to differing levels of accuracy, how multiple objects will interact with one another, subject to the motion of a robotic manipulator. Using these physics simulators, trajectories can be computed to enable a robotic manipulator to perform complex manipulations in clutter to achieve some objective. However, these physics predictions can never perfectly emulate the real world. If trajectories that were computed inside a physics simulator are executed on real robotic hardware open-loop, they are often bound to fail.

A general method to address this issue is model predictive control (MPC). MPC can account for the stochastic difference between a physics simulator and the real world by constantly re-planning trajectories from the current state of the real world. The key idea is that this re-planning needs to occur fast to be effective. Current methods for planning trajectories using physics simulators can be computationally costly, especially when considering manipulation in clutter.

This thesis addresses the issue of computational expense for high dimensional manipulation in clutter tasks, specifically focusing on trajectory optimisation techniques. Two main methods are investigated. Firstly, this thesis explores how approximations can be used to accelerate the computation of dynamics derivatives required by certain trajectory optimisation methods. Secondly, this thesis examines how the dimensionality of a trajectory optimisation problem can be adjusted online during task execution, leveraging the fact that not all objects in a scene are relevant at all times. The thesis concludes by investigating the challenges involved in transferring solutions from simulation to real-world execution, demonstrated through a packing through clutter task.

## Abbreviations

<b>MPC</b>	model predictive control
<b>DoFs</b>	degrees of freedoms
<b>RRT</b>	rapidly-exploring random trees
<b>PRM</b>	probabilistic roadmaps
<b>RL</b>	reinforcement learning
<b>IL</b>	imitation learning
<b>NLP</b>	nonlinear program
<b>SQP</b>	sequential quadratic programming
<b>QP</b>	quadratic programming
<b>KKT</b>	Karush-Kuhn-Tucker
<b>DDP</b>	differential dynamic programming
<b>iLQR</b>	iterative linear quadratic regulator
<b>AD</b>	automatic differentiation
<b>FD</b>	finite-differencing

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Challenges of Manipulation in Clutter . . . . .	5
1.2	Main themes . . . . .	6
1.3	Contributions . . . . .	7
1.4	Structure . . . . .	8
1.5	Publication Note . . . . .	8
<b>2</b>	<b>Literature Review</b>	<b>9</b>
2.1	Classical Motion Planning . . . . .	9
2.1.1	Useful terminology . . . . .	10
2.1.2	Sampling-based motion planning . . . . .	12
2.2	Trajectory Optimisation Fundamentals . . . . .	14
2.2.1	General formulation . . . . .	14
2.2.2	Shooting methods . . . . .	15
2.2.3	Direct methods . . . . .	19
2.2.4	Collision-free trajectory optimisation . . . . .	22
2.2.5	Remarks . . . . .	23
2.3	Contact-Based Trajectory Optimisation . . . . .	24
2.3.1	Shooting methods . . . . .	24
2.3.2	Direct methods . . . . .	26
2.4	Other Methods for Contact-based Manipulation / Locomotion . . . . .	28
2.4.1	Learning-based approaches . . . . .	29
2.5	Approximations for Speeding up Robotic Motion Planning . . . . .	35
2.5.1	Dimensionality reduction . . . . .	36
2.5.2	Other approximation methods . . . . .	37
2.6	Remarks . . . . .	39
<b>3</b>	<b>Background</b>	<b>40</b>
3.1	The Optimisation State Vector . . . . .	40
3.2	Trajectory Optimisation (iLQR and SCVX) . . . . .	41
3.2.1	iLQR . . . . .	42
3.2.2	SCVX . . . . .	44

3.3	Differentiation Methods . . . . .	47
3.3.1	Finite-differencing . . . . .	47
3.3.2	Automatic-differentiation . . . . .	48
<b>4</b>	<b>Speeding up Trajectory Optimisation Via Approximated Dynamics</b>	
	<b>Derivatives</b>	<b>49</b>
4.1	Introduction . . . . .	49
4.1.1	Contributions . . . . .	52
4.1.2	Organisation . . . . .	53
4.2	Problem formulation . . . . .	53
4.3	Key-point Selection Methods . . . . .	55
4.4	Approximation Error Bound for the Pendulum System . . . . .	60
4.5	Task Specifications and Testing Setup . . . . .	63
4.5.1	Optimisers . . . . .	64
4.5.2	Tasks . . . . .	64
4.5.3	Key-point parametrisations . . . . .	67
4.6	Results . . . . .	68
4.6.1	Evaluating interpolation accuracy . . . . .	69
4.6.2	Impact of contact on optimisation and key-point selection . . . . .	71
4.6.3	Long horizon optimisation performance . . . . .	74
4.6.4	Short horizon optimisation performance . . . . .	77
4.6.5	Execution performance on hardware . . . . .	80
4.7	Discussion . . . . .	82
4.8	Conclusion . . . . .	84
<b>5</b>	<b>Online State Vector Reduction during Model Predictive Control</b>	<b>86</b>
5.1	Introduction . . . . .	86
5.1.1	Contributions . . . . .	89
5.1.2	Organisation . . . . .	89
5.2	Problem Formulation . . . . .	89
5.2.1	Definitions . . . . .	91
5.3	Method . . . . .	91
5.3.1	Optimise . . . . .	92
5.3.2	Reducing dimensionality . . . . .	94
5.4	Results . . . . .	96
5.4.1	Task definition . . . . .	97
5.4.2	Asynchronous MPC results . . . . .	98
5.5	Conclusion . . . . .	101



<b>6</b>	<b>Challenges of MPC on Real Robotic Hardware</b>	<b>103</b>
6.1	Introduction . . . . .	103
6.1.1	Contributions . . . . .	105
6.1.2	Organisation . . . . .	105
6.2	MPC Performance . . . . .	105
6.2.1	Controllers . . . . .	107
6.2.2	Time indexing methods . . . . .	108
6.2.3	Results . . . . .	109
6.3	Packing an Object as an Optimisation Problem . . . . .	116
6.4	Conclusion . . . . .	119
<b>7</b>	<b>Conclusions &amp; Future Work</b>	<b>121</b>
7.1	Conclusions . . . . .	121
7.2	Remaining Problems / Limitations . . . . .	123
7.3	Future Work . . . . .	124
7.3.1	Efficient picking/packing . . . . .	124
7.3.2	Change of basis online state vector reduction . . . . .	125
7.3.3	Combining learning with trajectory optimisation . . . . .	125
7.3.4	Using GPU-based physics simulation . . . . .	126
7.4	Final Remarks . . . . .	127

# List of Figures

1.1	An example illustration of a contact-aware picking/packing system. The dark gray shapes represent a pump end-effector attachment for a robot. In this example, the high level objective is to retrieve a purple object from a bin. Frame A shows the initial scene, where the purple object is obscured by a green and dark blue object. In frame B, the robot slides the green object into the corner of the bin. Then, in frame C, the robot pushes the dark blue object out of the way, subsequently moving the brown object also. These actions have then cleared enough space so that the vacuum pump tip can reach in and grab the purple object. Finally, frame D shows the robot removing the purple object from the bin. . . . .	1
1.2	An example illustration of model predictive control (MPC) being used for a contact-based manipulation task to correct for the deviations between a planned trajectory in simulation (blue rectangles) and the executed trajectory in the real world. The top row shows the real trajectory of the pushed object when plan is executed open-loop. The bottom row shows the path of pushed object when correct feedback is applied via MPC. The bottom trajectory is more successful at keeping the object on its planned path. . . . .	3
2.1	An example illustrated solution for the rapidly-exploring random trees (RRT) [71] and probabilistic roadmaps (PRM) [61] algorithms for a 2D path planning problem. $\mathbf{q}_{start}$ is the starting configuration and $\mathbf{q}_{goal}$ is the goal configuration. Black dots are sampled configurations in $C_{free}$ that were added to the graph/tree. . . . .	11

4.1	Two finalised trajectories for pushing a green cylinder to a target location (green silhouette) through clutter. The top row shows the baseline method taking 9.48 s of optimisation time, whereas the middle row shows a trajectory produced from one of the methods proposed in this chapter which only took 2.66 s of optimisation time, whilst achieving a similar solution. The bottom graph shows a derivative value (one of many) over the course of the trajectory along with its approximation. Through key-points, the number of derivative evaluations was reduced from 1000 to 48. . . . .	50
4.2	Illustrative one-dimensional example of the Velocity Change method. The top plot shows the underlying true derivative value (black curve) with a set of key-points (green points) placed as determined by the Velocity change method. The bottom curve shows the velocity profile over the trajectory for some DoF. When velocities are large (at the start of the trajectory), or the velocity changes direction often (near the middle of the trajectory), key-points are placed more densely, and when the velocity is small or more constant, key-points are placed more sparsely. . . . .	56
4.3	Illustrative one-dimensional example of the Adaptive Jerk method. The top plot shows the underlying true derivative value (black curve) with a set of key-points (green points) placed as determined by the Adaptive Jerk method. The bottom curve shows the jerk profile over the trajectory for some DoF. When the jerk exceeds the jerk threshold ( $\bar{J}$ ), key-points are placed more densely, and when jerk is below the threshold, key-points are placed more sparsely. . . . .	57
4.4	Illustrative one-dimensional example of the Iterative Error method. The black curve is the true underlying derivative, the blue dashed line is the current linear approximation. Three iterations of the approximation being refined are shown. The top image is the first iteration where it selects only three key-points (blue). Halfway between the key-points, the approximate derivative value (red) is compared against the accurate derivative (green). If the error between these values is above a threshold, the segment is subdivided. This process happens recursively until all segments are complete. . . . .	59
4.5	Free body diagram of the pendulum. . . . .	60

4.6	Approximation accuracy for different key-point methods and different key-point parametrisations values, for the Acrobot task. The colour of each data-point represents the specific key-point selection method used and each data-point is a different parameterisation for that method. Three parametrisations for each key-point method are explicitly shown by a <b>star</b> , <b>cross</b> and <b>diamond</b> symbol and labelled in the top right of the figure. . . . .	70
4.7	Optimisation performance for 100 instantiations of the Acrobot task versus scaling values for the velocity change threshold. Green line shows normalised final cost of the trajectory, blue line shows the average percentage of derivatives computed accurately. . . . .	71
4.8	Two snapshots from the 1D toy contact task. The red piston is actuated and limited to 1D of motion. The green object is an un-actuated cube. The goal is to push the green cube to the goal location (green silhouette). . . . .	72
4.9	A 2x2 subset of graphs that show how values inside the <b>A</b> matrix for the toy contact task change over the initial trajectory. The subset shown are how the cube's position and velocity are affected by the piston's position and velocity. The blue line shows the <i>accurate</i> values, the red line shows the approximation subject to the computed key-points (yellow dots). The green dashed vertical line denotes moment of contact $t_c$ . . . . .	73
4.10	Optimisation performance versus smoothing effect after one iteration of optimisation. . . . .	73
4.11	MPC results for the Walker model over a variety of receding horizon lengths for the different key-point methods. The top plot is the performance of the executed trajectory when evaluated with the cost function. The bottom plot shows the effective control frequency that could be attained for the various methods. . . . .	79
4.12	Real robot scene setup, scene 1 (left) and scene 2 (right). Objective is to push purple object to goal location (small green sticker) whilst minimising the disturbance to the clutter obstacles (red and green cylinders). . . . .	81

5.1	A sequence of snapshots showing an example MPC trajectory generated by the method introduced in this chapter. 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. The method identifies the relevant degrees of freedoms (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. . . . .	87
5.2	A sequence of snapshots showing an example trajectory for the <b>soft</b> 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.	97
5.3	A sequence of snapshots showing an example trajectory for the <b>soft rigid</b> 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. . . . .	98
5.4	Top plot shows the MPC <b>cost</b> 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 the state vector. Three different parameterisations of $\theta$ were used with scaling values for $\rho$ . . . . .	100
6.1	A set of plots showing the deviations between commanded velocity (blue) and actual velocity (orange) of the real Franka Panda joints. The top plots shows the best example (Inverse-Dynamics-6) of velocity tracking from Table 6.1, whereas the bottom sets of plots shows the worst (Torque-PD-4). 7th joint omitted for the sake of space. . .	113
6.2	Three example sequences (from left to right) of packing a grasped object (tomato sauce) to a goal position obstructed by two obstacles (tomato soup and hot chocolate powder) on real robotic hardware. . .	118

7.1	The contact-aware picking/packing system that was discussed in Chapter 1. The dark gray shapes represent a pump end-effector attachment for a robot. To recap: In this example, the high level objective is to retrieve a purple object from a bin. Frame A shows the initial scene, where the purple object is obscured by a green and dark blue object. In frame B, the robot slides the green object into the corner of the bin. Then, in frame C, the robot pushes the dark blue object out of the way, subsequently moving the brown object also. These actions have then cleared enough space so that the vacuum pump tip can reach in and grasp the purple object. Finally, frame D shows the robot removing the purple object from the bin. . . . .	121
-----	---	-----

# List of Tables

4.1	Tasks used for simulation experiments. The left column shows the task name and an image of the task. Right column gives a brief description of the task objective followed by the optimiser used for the task. The costs used for optimisation are shown below the task description with cost notation as defined previously. $\Delta t$ is the model time-step. . . . .	66
4.2	Summary of long-horizon optimisation performance. OT is the total optimisation time in seconds, CR is cost reduction and NI is the number of optimisation iterations. For <b>iLQR-FD</b> and <b>SCVX</b> , the values reported in the tables are mean values averaged over 100 start/goal states. For <b>iLQR-AD</b> the values were single runs. . . . .	76
4.3	MPC results for mini-cheetah locomotion using different key-point methods. . . . .	80
4.4	Results from real robot experiments. . . . .	82
5.1	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. . . . .	99
6.1	Summary of real robot control performance under different control parametrisations. LP is whether a low pass filter was used, TRL was whether a torque rate limiter was used and FF is whether the feed forward term was used. The values reported in the table are mean values plus/minus the standard deviation, averaged over 5 runs for each control parametrisation. TC is whether the task was completed and RS is whether the robot was stopped early, with the values showing how many times this event occurred. . . . .	111

6.2	Summary of real robot performance data from Table 6.1 grouped by different categories (Controller used, time indexing method, whether the feed forward term was used or not and finally whether a low pass filter or torque rate limiter were used). The values reported in this table are mean values plus/minus the standard deviation, except for TC and RS columns which report the number of times these events occurred. . . . .	115
-----	--	-----



# List of Algorithms

1	Simplified iLQR algorithm . . . . .	43
2	Simplified SCVX algorithm . . . . .	46
3	Finite-differencing . . . . .	47
4	TrajOptKP implementation of GET DERIVATIVES . . . . .	55
5	Velocity Change implementation of SELECTKEYPOINTS . . . . .	56
6	Adaptive Jerk implementation of SELECTKEYPOINTS . . . . .	58
7	Iterative Error implementation of SELECTKEYPOINTS . . . . .	60
8	<i>Execution with MPC.</i> . . . .	78
9	MPC with state vector reduction (asynchronous) . . . . .	92
10	MPC real robot . . . . .	106

# Nomenclature

$\mathbf{q}$	Robot joint configuration.
$C$	Configuration space.
$\mathbf{x}$	System state vector.
$\mathbf{u}$	System control vector.
$n_x$	Size of system state vector.
$n_u$	Size of system control vector.
$T$	Trajectory optimisation horizon.
$J$	Total running cost of a trajectory.
$\mathbf{U}$	Sequence of control vectors over a trajectory.
$\mathbf{X}$	Sequence of state vectors over a trajectory.
$\mathbf{U}^*$	Optimal sequence of control vectors over a trajectory.
$\mathbf{X}^*$	Optimal sequence of state vectors over a trajectory.
$\bar{\mathbf{U}}$	Nominal sequence of control vectors over a trajectory.
$\bar{\mathbf{X}}$	Nominal sequence of state vectors over a trajectory.
$\hat{\mathbf{U}}$	New rollout sequence of control vectors over a trajectory.
$\hat{\mathbf{X}}$	New rollout sequence of state vectors over a trajectory.
$\mathbf{A}$	First order partial dynamics derivatives with respect to state vector.
$\mathbf{B}$	First order partial dynamics derivatives with respect to control vector.
$l(\mathbf{x}_t, \mathbf{u}_t)$	Cost function at time-step $t$ .
$\epsilon$	Small perturbation used for finite-differencing.
$\hat{\mathbf{A}}$	Accurate first order partial dynamics derivatives with respect to state vector.

$\hat{\mathbf{B}}$	Accurate first order partial dynamics derivatives with respect to control vector.
$N_{min}$	Minimum interval between key-points.
$N_{max}$	Maximum interval between key-points.
$\Delta\bar{\mathbf{v}}$	Array of velocity change thresholds for each DoF in the system.
$\bar{\mathbf{J}}$	Array of jerk thresholds for each DoF in the system.
$\bar{\epsilon}$	Iterative error threshold.
$\boldsymbol{\tau}$	Torque vector.
$\mathbf{W}$	State dependent cost matrix.
$\mathbf{W}_f$	Terminal state dependent cost matrix.
$\mathbf{R}$	Control dependent cost matrix.
$\mathbf{K}$	State dependent feedback matrix.
$\tilde{\mathbf{x}}$	Desired state vector.
$N_o$	Number of rigid objects in a scene.
$N_s$	Number of soft body objects in a scene.
$\rho$	DoF importance threshold.
$\theta$	Number of DoFs to be randomly re-sampled.
$\mathcal{C}$	Set of reduced DoFs considered in state vector.
$\mathcal{F}$	Set of all DoFs in the system.
$\mathcal{L}$	Set of unused DoFs.
$w_i$	Weight scalar for $i$ -th residual.
$n_i(\cdot)$	Twice differentiable norm function for $i$ -th residual.
$r_i(\mathbf{x}, \mathbf{u})$	Residual function.

# Chapter 1

## Introduction

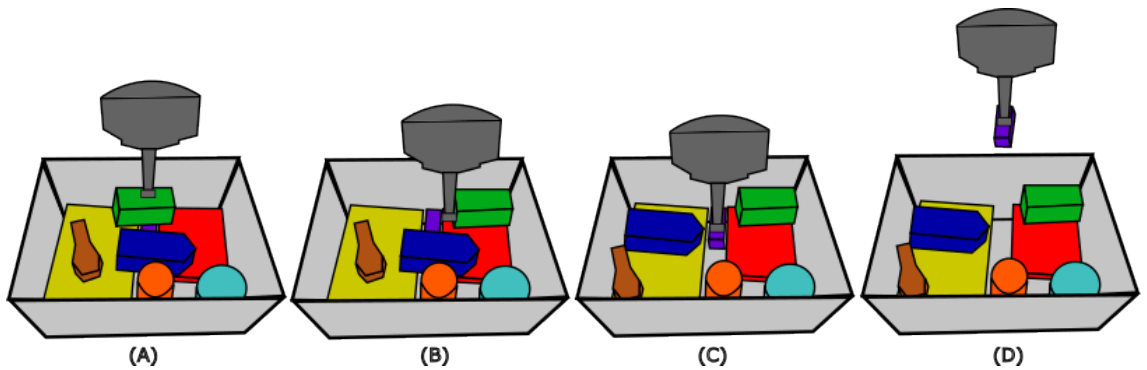


Figure 1.1: An example illustration of a contact-aware picking/packing system. The dark gray shapes represent a pump end-effector attachment for a robot. In this example, the high level objective is to retrieve a purple object from a bin. Frame A shows the initial scene, where the purple object is obscured by a green and dark blue object. In frame B, the robot slides the green object into the corner of the bin. Then, in frame C, the robot pushes the dark blue object out of the way, subsequently moving the brown object also. These actions have then cleared enough space so that the vacuum pump tip can reach in and grab the purple object. Finally, frame D shows the robot removing the purple object from the bin.

There are a variety of reasons why the automation of certain jobs is beneficial to society. Firstly, automation can improve the efficiency in creating consumer goods, making products and services cheaper. Certain jobs can be dangerous (e.g., maintenance work on high-voltage cables or decommissioning old nuclear reactors) and even with significant health and safety equipment and protocols, there will always be some inherent risk to the individuals who perform these jobs. Then there are jobs that are not necessarily dangerous but are physically demanding and can cause strain to a human body over the course of a lifetime. Warehouse jobs are one such example, often requiring workers to lift heavy objects or adopt unnatural postures that can be potentially harmful. Finally, there is the idea of fulfilment in

---

every day life. Imagine a society in which everyone is able to pursue what they truly enjoy. If there are jobs that are necessary for society, but lack human interest or willingness, then robotic automation can fill this need.

Let us consider an example where robotic automation has had a significant impact on the world. In the 1960/70s, robotic automation was introduced in the manufacturing of automotive vehicles [122]. This robotic automation considerably increased the efficiency of producing cars, making them substantially more affordable. Due to this, cars have become ubiquitous in modern day civilisation.

Much like the automotive industry has already experienced, numerous sectors stand to gain significant advantages by automating their services. For instance, automation in food sorting warehouses for custom food orders could greatly improve efficiency. In such systems, a wide variety of goods need to be selected and packed for household delivery, a process that is currently highly labour-intensive and lacking in automation. Workers in these settings often face physically demanding tasks that can lead to repetitive strain injuries, such as back problems, and must endure uncomfortable working conditions, particularly in chilled or frozen warehouses.

What are the challenges of automation in this setting? If robotic automation has been so widely adopted in car manufacturing over 60 years ago, why is this particular problem so challenging? Let us consider Fig. 1.1, which illustrates a robotic end-effector with a suction gripper attachment that is operating in a cluttered and unstructured warehouse box. The goal is to retrieve a purple object from this box, however initially this object is not directly reachable, because it is covered by the blue and green objects. In frame 2, the robot end-effector pushes the green object to one side. After this, in frame 3 it pushes a blue object to the other side, which also displaces a brown object. Finally, in frame 4, the purple object is now reachable so the end-effector picks up the purple object and removes it from the bin.

This hypothetical system has significant implementation challenges and differences when compared to the robotic systems used to automate car manufacturing. In automated car manufacturing, the environments are structured and well-defined, and the robots are typically performing pre-defined actions repetitively. In a food delivery warehouse there can be thousands of unique objects that could be in any number of warehouse bins. This means that the robotic system to automate this needs to be able to adapt to different environments programmatically.

Another significant challenge of this proposed system is that the robot needs to manipulate objects in the bin using *non-prehensile* manipulation actions. Non-prehensile manipulation is defined as manipulation actions that occur when an object is not firmly grasped by a robot. Examples of these actions include pushing, sweeping, rolling, toppling and throwing. These non-prehensile manipulation skills are used by humans efficiently in our everyday lives without much thought.

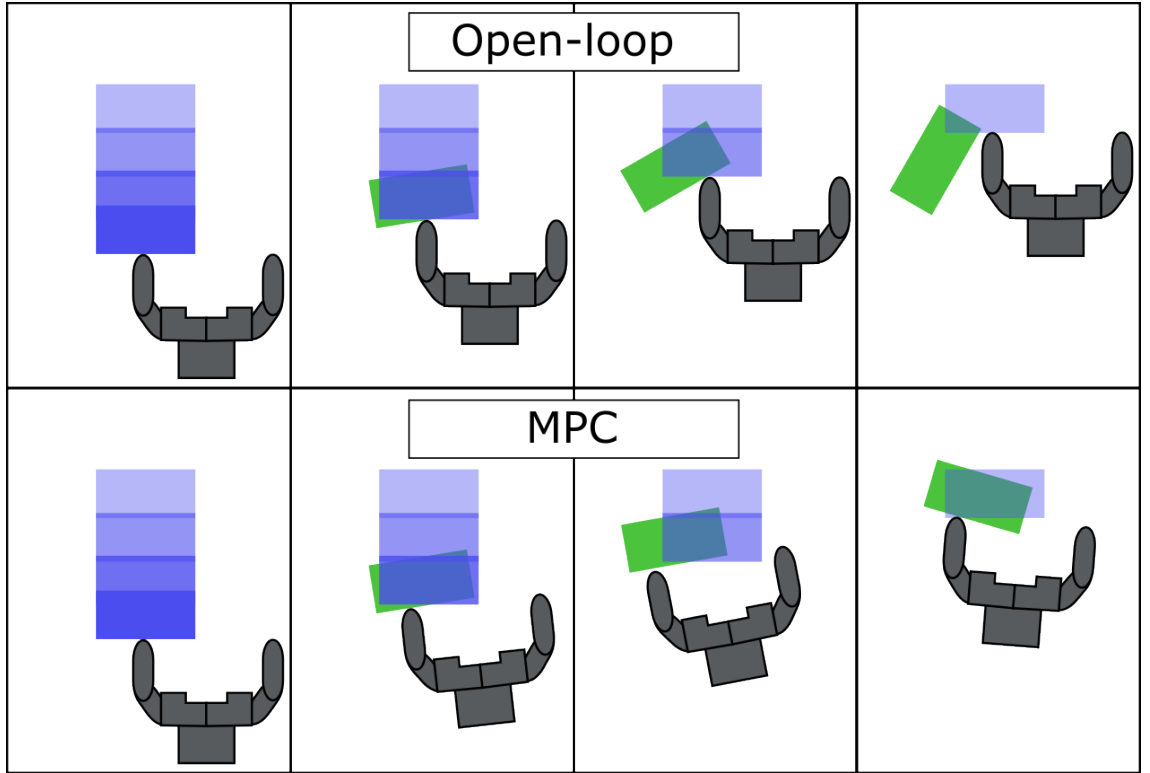


Figure 1.2: An example illustration of MPC being used for a contact-based manipulation task to correct for the deviations between a planned trajectory in simulation (blue rectangles) and the executed trajectory in the real world. The top row shows the real trajectory of the pushed object when plan is executed open-loop. The bottom row shows the path of pushed object when correct feedback is applied via MPC. The bottom trajectory is more successful at keeping the object on its planned path.

So why is it desirable for such a system to be able to take advantage of such manipulation skills? Firstly, these non-prehensile manipulation skills are often significantly more efficient than manipulating objects individually via grasping. Imagine a situation where several small objects need to be displaced to complete some task; the robot could either sequentially retrieve each object and transfer them elsewhere or it could sweep them all to some other location in one continuous action. The second reason why non-prehensile manipulation skills are valuable is situations where objects are too large or heavy to be directly grasped. In these situations, the robot can only manipulate such objects via non-prehensile manipulation.

Unfortunately, imbuing robots with such skills is challenging, which is one of the primary reasons why such systems are not commonplace in automated warehouses. These non-prehensile manipulation skills require imbuing our robots with *physical reasoning* skills. In other words, they need to understand how their actions will affect their environment and be able to plan accordingly. One popular method is “model-based” control, where a general-purpose physics simulator is used to predict how objects will move subject to the forces imposed by some robot. These physics simulators can then be used to plan non-prehensile manipulation actions to achieve

---

some goal.

Unfortunately, even with the abundance of accurate and fast physics simulators [129, 128, 77, 29, 86, 18] significant challenges remain with regard to planning and executing robust physics-based motion on real robotic hardware. The first challenge is that computing non-prehensile manipulation actions, even inside a simulator is a challenging problem due to them being an under-actuated problem. The second challenge is that even if computing an optimal plan inside a physics simulator can be guaranteed, trying to execute these plans on a real robotic system open-loop will often fail to complete the desired task. The reason for this is that no physics simulator can perfectly emulate the real world; there will be some inherent stochasticity between the predicted motion of objects and the real motion. This stochastic difference accumulates the longer the planned trajectory.

One powerful method to overcome this stochastic difference is called model predictive control (MPC). MPC is an advanced control strategy used in a variety of disciplines. The general idea of MPC is to have some mathematical model of a system that can predict its future behaviour subject to some control inputs. Using this model, an optimal control policy can be computed at every time-step to minimise some objective whilst obeying system constraints. When MPC is considered for robotic control, the model of the system is often a general purpose physics simulator. Consider Fig. 1.2, where a contact-based trajectory has been planned to push a rectangular object forwards. The blue rectangles indicate the planned trajectory for the object and the green rectangle indicates the rectangles current pose. On the top row is a sequence of snapshots illustrating how an object could move when executing the planned trajectory open-loop. As the trajectory progresses, the deviation of the object from its planned pose increases until the end-effector loses contact with it. The bottom row shows the same planned trajectory, but uses MPC to account for the deviation between the planned pose of the object and its actual pose. Due to this, the end-effector can correct its path to keep control of the rectangular object.

One of the key challenges of using MPC for non-prehensile manipulation in cluttered environments is the need to optimise trajectories quickly enough to keep pace with real-world dynamics. The time taken to re-optimize trajectories in simulation has a direct impact on real-world performance due to an effect known as *policy lag*. Ideally, the controller would re-optimize trajectories instantaneously, allowing it to immediately account for any deviations between the planned and actual trajectories. In practice, however, optimisation requires a finite amount of time, resulting in periods where the robot is executing a *stale* control policy. The longer this period, the greater the degradation in the robot’s performance. While policy lag can be mitigated by reducing the optimisation horizon, doing so excessively leads to *myopic* behaviour, where the robot focuses only on short-term goals at the expense of long-term effectiveness.

This thesis proposes several ideas to speed up the process of optimising trajectories via different methods of utilising approximations. The main goal of the proposed algorithms is to enable faster MPC algorithms to perform non-prehensile manipulation in cluttered environments.

## 1.1 Challenges of Manipulation in Clutter

There are a variety of challenges which need to be solved to create an effective contact-aware pick/place planner like the one described in Fig. 1.1. Some of these challenges are outside the scope of this thesis but deserve to be mentioned. The following challenges are addressed in this thesis:

- **Under-actuation:** Objects in the scene are under-actuated. To move them, contact forces need to be applied to the objects from the actuated robots to perform any required tasks. Under-actuated tasks are inherently more challenging to solve than fully-actuated problems.
- **Planning through multi-object contact:** Similarly to the previous point, manipulation in cluttered environments needs to consider a large variety of under-actuated objects, and the complex multi-object contact that is encountered between objects. This makes the problem even more challenging.
- **Curse of dimensionality:** Manipulation in cluttered environments are high-dimensional problems. As the dimensionality of a problem increases, so does the planning time, making motion planning in cluttered scenes computationally expensive.

The challenges that are worth mentioning but are not addressed in this thesis are:

- **Unknown object properties:** To create accurate physical predictions of how an object/objects will move due to non-prehensile manipulation actions, an accurate parametrisation of physical object properties is required. Examples of these physical properties include size, shape, mass, friction and others. Some of these properties can be deduced by visual observations, but others require estimates that can then be refined by physical interactions.
- **Occlusions:** Cluttered environments are inherently difficult to observe accurately. Fig. 1.1 partially shows this, the purple object is initially almost entirely occluded. To get a more accurate estimate of its pose, other objects need to be moved.



- **Task and motion planning:** Some tasks in manipulation in clutter will require multi-step planning to achieve. Perhaps this will be the requirement to move a large object to a location that will then enable a robot to reach past and grab some other object. Task and motion planning adds additional complexity to motion planning problems.

## 1.2 Main themes

The overall goal of this thesis is to work towards creating the system outlined in Fig. 1.1. Importantly, the full system outlined in Fig. 1.1 is not completed in this work, however, this thesis takes strides in addressing several of the challenges of creating such a system, namely: under-actuation, planning through multi-object contact and most importantly, the curse of dimensionality. In this thesis, a variety of methods/algorithms are proposed to use conventional trajectory optimisation techniques and make them computationally tractable when performing manipulation in cluttered environments.

Chapter 4 of this thesis is concerned with speeding up gradient-based trajectory optimisation. These methods have desirable convergence qualities when compared to zero-order methods, but the downside is their additional computational complexity. One of the bottlenecks of gradient-based trajectory optimisation methods is computing the dynamics derivatives for contact-based manipulation problems. The reason that computing dynamics derivatives is computationally expensive in these cases is because analytical derivatives are often not available, meaning that numerical methods, such as finite-differencing (FD) are often required. Chapter 4 proposes a general method of speeding up gradient-based trajectory optimisation by approximating dynamics derivatives. The general methodology is to limit the computation of dynamics derivatives using expensive methods like FD at *key-points* over the nominal trajectory. The remaining dynamics derivatives are then approximated via cheap linear interpolation.

In a similar vein, Chapter 5 is also concerned with speeding up gradient-based trajectory optimisation. The general method in this chapter is about limiting the number of degrees of freedoms (DoFs) considered in the optimisation state vector to only the most important DoFs to reduce the computational overhead of trajectory optimisation. Depending on the task and its current configuration, certain objects will be relevant to the task and others will not. Consider reaching through a cluttered shelf; at the start of the task you may need to consider the physics of certain objects that you push to the side to make space for your arm, but after this you can retrieve a desired object and stop considering the objects you have just displaced. The general method proposed in Chapter 5 is based on this philosophy; the method dynamically changes what DoFs are considered in the optimisation state vector during MPC.

Finally, Chapter 6 begins to consider creating the picking/packing system that was described in Fig. 1.1. This chapter shows the challenges of performing MPC on real robotic hardware. This challenge was that there was significant difference between the accelerations produced on the real robot compared to in simulation by the same commanded torques, due to an imperfect dynamics model. This chapter investigates the utility in using different low-level feedback controllers in tandem with MPC to enable more accurate tracking of a reference trajectory provided by the high-level MPC algorithm. In addition to this, a simplified packing in clutter task is formulated as an optimisation problem and evaluated on real robotic hardware.

## 1.3 Contributions

This thesis:

- Proposes a general method of speeding up gradient-based trajectory optimisation. The method only computes dynamics derivatives accurately at *key-points* over a trajectory and then approximates the remaining dynamics derivatives (Chapter 4).
- Proposes a variety of specific key-point selection methods to choose where dynamics derivatives are computed accurately (Chapter 4).
- Evaluates the proposed key-point selection methods on a wide-variety of trajectory optimisation tasks, with two different optimisation algorithms (iterative linear quadratic regulator (iLQR) and SCVX) and two popular methods of computing dynamics derivatives (FD and automatic differentiation (AD)) (Chapter 4).
- Proposes a general framework for online state vector reduction during MPC (Chapter 5).
- Proposes and evaluates a specific implementation of online state vector reduction during MPC for several high-dimensional tasks (Chapter 5).
- Performs an experimental analysis of the effectiveness of combining low-level feedback controllers with MPC for trajectory stabilisation for a contact-based manipulation task (Chapter 6).
- Formulates packing through contact as a optimisation problem (Chapter 6).

## 1.4 Structure

This Chapter has introduced the main themes of this thesis and roughly outlined the contributions that are made. Chapter 2 presents an in-depth literature review that is relevant to this thesis. Chapter 3 provides the reader with the necessary technical background to understand the contributions of this thesis. Chapters 4 through 6 each outline a specific technical contribution. Finally, Chapter 7 concludes this thesis and outlines possible directions for future work.

## 1.5 Publication Note

Some of the technical contributions in this thesis have already been published in various conferences and journals. The details of which chapters relate to which published work are listed below:

1. The content in Chapter 4 appears in:
  - (a) **Published conference paper:** International Conference on Robotics and Automation (ICRA) 2023 [113].
  - (b) **Source code:** <https://github.com/DMackRus/TrajOptKP>
  - (c) **Video:** <https://www.youtube.com/watch?v=rdRd2sgk8qY>
2. The content in Chapter 5 appears in:
  - (a) **Published conference paper:** Workshop on Algorithmic Foundations in Robotics (WAFR) 2024 [112].
  - (b) **Source code:** <https://github.com/DMackRus/iLQR-SVR>
  - (c) **Video:** [https://www.youtube.com/watch?v=8K\\_qfvb4lMI](https://www.youtube.com/watch?v=8K_qfvb4lMI)
3. The content in Chapter 6 appears in:
  - (a) **Controller code:** [https://github.com/roboticsleeds/panda\\_controllers](https://github.com/roboticsleeds/panda_controllers)
  - (b) **Video:** [youtube.com/watch?v=Q1mxb1CMIKg](https://www.youtube.com/watch?v=Q1mxb1CMIKg)
  - (c) **Cobot pump code:** [github.com/roboticsleeds/cobot\\_pump\\_ros](https://github.com/roboticsleeds/cobot_pump_ros)

# Chapter 2

## Literature Review

This chapter provides a comprehensive review of the literature in various areas which are relevant to this thesis. Although all works are discussed with an appropriate level of mathematical depth here, a dedicated background chapter (Chapter 3) provides the minimal necessary technical detail to understand the specific contributions of this thesis.

This chapter is structured as follows. Firstly, Section 2.1 discusses classical motion planning for robotics and its limitations. Next, the fundamental ideas for trajectory optimisation and some notable algorithms are discussed in Section 2.2. After this, contact-based trajectory optimisation is discussed in Section 2.3. Section 2.4 discusses non-optimisation based methods for contact-based manipulation. Finally, this chapter ends by discussing different methods of approximations that have been used in robotics to enable computationally tractable motion plans.

An in-depth review and background of trajectory optimisation is given in Sections 2.2 and 2.3. Both of these sections are relevant to all chapters of this thesis, as this thesis is primarily concerned with using trajectory optimisation to perform contact-based manipulation. Section 2.5 is relevant to Chapter 5 where dimensionality reduction is used to speed up trajectory optimisation. The remaining sections are included to give a holistic overview of the field of robotic manipulation and how it has evolved from classical motion planning.

### 2.1 Classical Motion Planning

The term “classical motion planning” has some flexibility and may be interpreted differently dependent on the researcher. In this thesis, the term “classical motion planning” refers to computing collision-free paths from a starting configuration to a goal configuration for some robot (whether the robot is a robotic manipulator, mobile robot or something else). With regards to robotic manipulators, simply moving from one configuration to another whilst avoiding collisions does not overly

achieve anything meaningful, as a robotic manipulator’s primary job is typically to interact with their environments.

In the early days of robotic manipulation, a lot of researches were interested in using robotic manipulators to create “pick-and-place” systems. These systems would use a combination of grasping objects and classical motion planning to interact with their environments. The robot could interact with an object using the following steps: (1) Compute and follow a collision-free path from the robot’s current configuration to a configuration that will place the robot’s end-effector in a pose ready to grasp some goal object. (2) Grasp the goal object so that it is dynamically stable within the robot’s end-effector. (3) Compute and follow another collision-free path (this time also considering the geometry of the grasped object) from the robot’s current configuration to a new configuration where the goal object will be placed.

It is important to note that this method of interacting with environments can be quite inefficient and sometimes impossible in a variety of situations. Imagine a situation where space on a table needs to be cleared and there are tens of small objects on the table. The pick-and-place approach would be to individually move every object sequentially from the table to some other location. A more efficient solution would be to use a sweeping action to move all of the objects to the new location at the same time. Imagine a second scenario, where the object that needs to be moved is a large box that is too large to be grasped; manipulating such an object can only be done by pushing it to the new desired location. These alternative manipulation strategies are referred to as “non-prehensile” manipulation. Non-prehensile manipulation is the opposite of prehensile manipulation. In prehensile manipulation, objects need to be rigidly grasped by the robotic manipulator and can be thought of as an extension of the robot. In non-prehensile manipulation, external forces need to be imposed on the object via the robotic manipulator, via actions like pushing, tilting, rolling, etc.

Returning back to the general pick-and-place problem; The two primary challenges that were present for creating a general pick-and-place system were: (1) How to compute stable grasps of an object, especially when the object has complicated or irregular geometry. (2) How to efficiently compute collision-free motion plans through complicated environments for the higher dimensional control problems present for robotic manipulators. This section only discusses the issue of computing collision-free motion plans as it sets the stage for how robotic motion planning has progressed from collision-free motion planning to contact-based motion planning.

### 2.1.1 Useful terminology

**Configuration space:** A configuration of a robot can be uniquely defined by a vector of joint angles/positions  $\mathbf{q} = [q_1, q_2, \dots, q_n]$ , where  $q_i$  represents the value of the  $i$ -th joint. The full set of configurations of the robot is defined as the configuration space  $C$ , otherwise known as the  $C$ -Space ( $\mathbf{q} \in C$ ). The  $C$ -Space is a useful

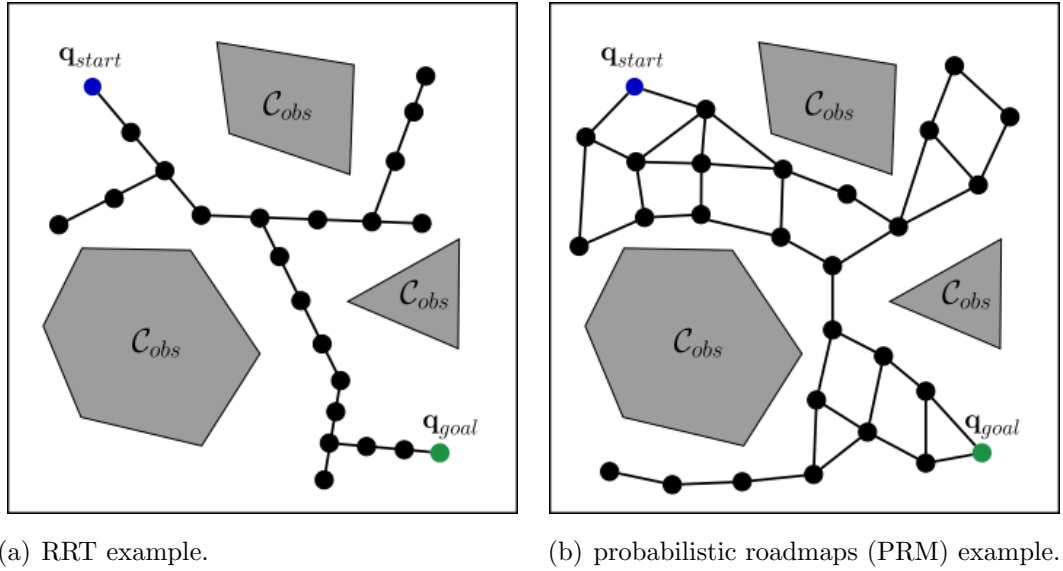


Figure 2.1: An example illustrated solution for the RRT [71] and PRM [61] algorithms for a 2D path planning problem.  $\mathbf{q}_{start}$  is the starting configuration and  $\mathbf{q}_{goal}$  is the goal configuration. Black dots are sampled configurations in  $C_{free}$  that were added to the graph/tree.

abstraction often used in geometric path planning. An important concept of the  $C$ -Space is the space in which the robot is in collision with either its environment or itself; this is denoted as  $C_{obs}$ . The inverse of this is  $C_{free}$  which is the space where the robot is not in collision with anything. As discussed earlier, the classical path planning problem was to find a path through  $C_{free}$  from some starting configuration  $\mathbf{q}_{start}$  to some goal configuration  $\mathbf{q}_{goal}$ .

**State space:** The state space is similar to the idea of configuration space, however it also considers dynamic attributes of the DoFs in a system. Commonly this means including velocities of DoFs, but can also include higher-order terms such as accelerations.

**Forward kinematics:** Forward kinematics refers to computing the end-effector pose (and poses of other coordinate frames attached to the robot body) given the joint positions of the robotic manipulator.

**Inverse Kinematics:** Inverse kinematics [47, 66] refers to computing a valid set of joint positions that will place the robotic end-effector at some desired pose. Inverse kinematics is a harder problem to solve than forward kinematics due to the fact that there can often be multiple solutions or no solutions to achieve a specific end-effector pose.

### 2.1.2 Sampling-based motion planning

A typical collision-free path planning method is to discretise a search space and create a graph to connect all adjacent nodes. Using this graph, a search algorithm such as the  $A^*$  [45] algorithm could be used to find a path from the start node to the end node. These discretisation methods worked well for mobile robots operating in a 2D plane where the dimensionality of the problem was fairly low, but they do not scale well to the higher-dimension systems present when controlling robotic manipulators; typical robotic manipulators may have 6 or 7 DoFs to enable them to operate effectively in their entire working environment. As such, a new branch of algorithms needed to be created that could scale better to these higher-dimensional path planning problems.

Sampling-based motion planning [62] offered a simple yet effective solution to this problem. The general idea of sampling-based motion planning was not to discretise the configuration space, but instead to sample from the configuration space and create a data structure (such as a graph or tree) of valid configurations and connect them to nearby configurations. Once sufficient configurations have been sampled, and the goal configuration is connected to the data structure, a traversal search can be performed to find a path from the starting configuration to the goal configuration. Every time a configuration is sampled, it needs to be determined whether this configuration is contained within  $C_{free}$ ; this is done by using a combination of forward kinematics and then geometric collision checking to see whether this configuration puts the robotic manipulator in collision with its environment or itself. These sampling-based planners can be probabilistically complete, that is to say that the more samples that are taken the more likely the planner will find a path if one exists.

Two of the most well known, and probabilistically complete, sampling-based planners are RRT [71] and PRM [61]. These two algorithms work similarly but have one fundamental difference, which is if they reuse the data structure they compute for subsequent path planning. Fig. 2.1 shows an illustration of the data structures that these two methods would create to solve a 2D path planning problem.

In RRT [71], for every path planning problem, a new tree structure is created to find a path between the start and goal configuration, even if the environment did not change. This tree is created by sampling the configuration space  $\mathbf{q}_{new}$ , checking if the sample is within  $C_{free}$ , and if it is, connecting it to the nearest node in the current tree  $\mathbf{q}_{near}$  by sampling points at intervals between  $\mathbf{q}_{near}$  and  $\mathbf{q}_{new}$ . The search is biased towards the goal configuration  $\mathbf{q}_{goal}$  by sampling the goal configuration as  $\mathbf{q}_{new}$  at set intervals. Once  $\mathbf{q}_{goal}$  has been added to the tree, the path can be determined by backtracking through the tree to the starting configuration.

In PRM [61], a graph structure is created that is a sparse representation of  $C_{free}$ , but more complete than the tree structure created by RRT. The graph is constructed

in two steps, Firstly, a user-defined number of points are sampled and are checked whether they are within  $C_{free}$ . All the points within  $C_{free}$  are added as vertices to the graph. Then, edges are created between each vertex and its  $k$  nearest neighbours, as long as they can be connected by a local path planner without entering  $C_{obs}$ . Once the graph has been created, it can be used to plan a path between a starting and goal configuration. This planning occurs by connecting  $\mathbf{q}_{start}$  and  $\mathbf{q}_{goal}$  to the graph and using a search algorithm like  $A^*$  to find the shortest path. This same graph can then be used for subsequent path planning queries, assuming that, the environment did not change drastically, making the existing graph invalid.

Both of these algorithms have inspired countless adaptations to adapt them to be more versatile in certain scenarios [125, 92, 67, 13]. Kuffner et al. [67] proposed changing the RRT algorithm to grow two trees instead of one (one starting at the start configuration and one starting at the goal configuration). Both of these trees would grow in parallel, extending them through the  $C_{free}$  space. At set intervals, both trees would try and connect themselves to the other. This bi-directional search strategy often decreases the time taken to find a collision-free path. Bohlin et al. [13] propose “lazy PRM”, a PRM adaptation that delays collision checking of states until necessary. Initially the graph is constructed and it is assumed that all edges are valid and no collision checking is required. When querying the graph to find a collision free path, only then are collision checks done. This approach reduces the amount of computation time required to build the initial graph.

There are two main issues with sampling-based algorithms. Firstly, the paths that are computed are often not dynamically smooth. Following these paths can often result in jerky movements of the robotic manipulator. This problem can be addressed by implementing an additional smoothing stage after a path has been found to try and make the path more dynamically smooth [98, 46]. Secondly, these algorithms are only probabilistically complete. This means that if a solution does *not* exist, these algorithms will not be able to report that. Therefore users must rely on some timeout mechanism to determine when to stop trying to plan a path and determine that a path is infeasible.

This section has described some of the classical elements of robotic motion planning. This was during a time when finding collision-free paths was the priority and contact between a robot and its environment was undesired. Moving towards the present day, it is quite common for researchers to try and make their robots complete tasks by deliberately making contact with objects in their environments and leveraging contact-based interactions. This new shift in thinking is largely what this thesis is interested in. The work in this thesis leverages contact-based interactions to enable robotic manipulators to achieve their goals using trajectory optimisation. Before discussing how trajectory optimisation can be used for contact-based manipulation, it is important to understand the fundamentals of trajectory optimisation



and its first intended uses.

## 2.2 Trajectory Optimisation Fundamentals

The purpose of this section is to give the interested reader a brief introductory overview into the field of trajectory optimisation. It discusses the general trajectory optimisation formulation, shooting methods of optimisation and some notable algorithms as well as direct methods of optimisation and some relevant algorithms. Finally, works are discussed that use trajectory optimisation for collision-free motion planning.

Trajectory optimisation is the field of study concerned with minimising the cost of trajectories subject to a set of constraints. A trajectory is a path that an object or dynamical system follows as it travels through both physical space and time. The cost function that is minimised is defined by a user to incite some desired behaviour, such as reaching some goal condition as fast as possible whilst minimising the cost of reaching said goal condition. One of the first use cases of trajectory optimisation was in the aerospace industry where it was used to optimise trajectories of rockets and satellites to compute optimal launch and re-entry trajectories to minimise fuel usage [10].

Following on from Sec. 2.1 where it was discussed how sampling-based planners have been used to compute collision-free motion plans, trajectory optimisation has also been employed to achieve the same goal. Two popular algorithms named STOMP [60] and CHOMP [110] both consider computing collision-free optimal motion plans to simultaneously compute a motion plan that avoids contact with obstacles as well as minimising some dynamical quantities like acceleration and jerk. One of the benefits that trajectory optimisation offers over sampling-based planners is that the optimal trajectories they compute can minimise dynamic quantities like acceleration or jerk automatically, by encoding this into the cost function. This means that there is no requirement for a post-processing step to smooth the trajectory as the optimal trajectory is already smooth.

### 2.2.1 General formulation

Consider a discrete-time dynamics system where the next state of the system is subject to the previous state and control vector:

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t), \quad (2.1)$$

where  $\mathbf{x}_t \in \mathbb{R}^{n_x}$  and  $\mathbf{u}_t \in \mathbb{R}^{n_u}$  are the state and control vector respectively at time-step  $t$ .

Given a discrete-time trajectory  $(\mathbf{X}, \mathbf{U})$  of length  $T$ , where  $\mathbf{X} \triangleq (\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_T)$  and  $\mathbf{U} \triangleq (\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_{T-1})$ , we want to minimise the total running cost of a trajectory  $J$ :

$$J(\mathbf{X}, \mathbf{U}) = l_f(\mathbf{x}_T) + \sum_{t=0}^{T-1} l(\mathbf{x}_t, \mathbf{u}_t), \quad (2.2)$$

where  $l(\mathbf{x}_t, \mathbf{u}_t)$  is some state and control vector dependent cost function and  $l_f(\mathbf{x}_T)$  is a terminal state dependent cost function.

The algorithms that solve this optimisation problem can be generally broken down into two main categories; *shooting* and *direct* methods. Shooting methods only consider the control vectors as the optimisation parameters and enforce the dynamics of the system by rolling-out (or “shooting”) the control sequence through the system dynamics. Direct methods consider *both* the control and state vectors as optimisation variables, and enforce the system dynamics through constraints in the optimisation problem. There is also the concept of *multiple-shooting* [10] which is a hybrid of these two categories. Multiple-shooting works by dividing the trajectory into segments, each segment is optimised in a shooting-like way where the dynamics are rolled-out, but the knot points between each segment are optimised more like a direct method where constraints are used to make sure that transitions between segments is dynamically feasible. This enables the trajectory to be optimised more in parallel which is one of the major shortcomings of shooting methods.

### 2.2.2 Shooting methods

Shooting methods optimise a trajectory by only considering the control vectors as decision variables and then enforce the dynamics of the system by rolling out new control trajectories through the system dynamics. More formally, they minimise the total running cost of a trajectory (Eq. 2.2):

$$\mathbf{U}^*, \mathbf{X}^* = \arg \min_{\mathbf{U}} [J(\mathbf{X}, \mathbf{U})], \quad (2.3a)$$

$$\text{subject to: } \mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) \quad \text{for } t = 0, \dots, T-1, \quad (2.3b)$$

where the state trajectory  $\mathbf{X}$  is computed by rolling out a control trajectory  $\mathbf{U}$  through the system dynamics. This thesis will refer to the *nominal* trajectory  $(\bar{\mathbf{X}}, \bar{\mathbf{U}})$  as the current best solution to the trajectory optimisation problem. After every successful optimisation iteration, the nominal trajectory is updated.

Some shooting methods require optimisation to be considered over the *full* control sequence (i.e.,  $\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_{T-1}$ ), whereas others are able to compress the control sequence into a lower dimensional object such as a time-dependent spline representation [50]. When compressing controls using splines, a set of  $M$  *knot points* are defined at times  $(t_0, t_1, \dots, t_{M-1})$  with associated control values  $(\mathbf{w}_0, \mathbf{w}_1, \dots, \mathbf{w}_{M-1})$ . The full control sequence can then be computed by interpolating between the knot

points subject to the interpolation scheme used. For a linear interpolation scheme, the control values between knot points  $i$  and  $i + 1$  would be computed via:

$$\mathbf{u}_t = \mathbf{w}_i + \frac{t - t_i}{t_{i+1} - t_i}(\mathbf{w}_{i+1} - \mathbf{w}_i) \quad \text{for } t_i \leq t \leq t_{i+1}. \quad (2.4)$$

There are a few advantages to using spline representations. Firstly, as the number of decision variables is reduced, the computational complexity of trajectory optimisation can be reduced, as well as this, sometimes having fewer decision variables can make it easier for optimisers to find and follow descent directions. Secondly, by using splines, some smoothness is automatically incorporated into the optimal trajectory, without smoothness even necessarily needing to be specified in the cost function. However, one disadvantage of using splines is that they can limit the robot's ability to execute highly precise and discontinuous actions that can sometimes be required dependent on the task.

### Pontryagin's maximum principle

Pontryagin's maximum principle [79] is a first-order trajectory optimisation method that uses gradient information about the system dynamics and cost function to compute improved trajectories.

Initially, a nominal state trajectory needs to be computed via a rollout of some initial control trajectory guess. Then, the costate  $\lambda$  sequence is computed using a backwards pass from the end of the trajectory to the beginning. This costate sequence is then used to compute optimal control modifications that are applied to the nominal control trajectory via a line-search.

The backwards pass stage to compute the costate variables from  $(T - 1)$  to 0 can be shown as the following:

$$\lambda_t = \frac{\delta l(\mathbf{x}_t, \mathbf{u}_t)}{\delta \mathbf{x}_t} + \frac{\delta f(\mathbf{x}_t, \mathbf{u}_t)}{\delta \mathbf{x}_t} \lambda_{t+1}. \quad (2.5)$$

These costate variables are then used to compute the descent direction of the total running cost with respect to the control vector

$$\frac{\delta J}{\delta \mathbf{u}_t} = \frac{\delta l(\mathbf{x}_t, \mathbf{u}_t)}{\delta \mathbf{u}_t} + \frac{\delta f(\mathbf{x}_t, \mathbf{u}_t)}{\delta \mathbf{u}_t} \lambda_{t+1}. \quad (2.6)$$

The descent direction of the total running cost can then be used with a line-search parameter  $\alpha \in [\alpha_{min}, \alpha_{max}]$  to ensure that a lower cost trajectory is found:

$$\mathbf{U} = \bar{\mathbf{U}} + \alpha \frac{\delta J}{\delta \bar{\mathbf{U}}}. \quad (2.7)$$

It is difficult to know what values are the best to use for the bounds on  $\alpha$ , this depends on how valid the linear approximation of the system dynamics is and how close to any local minima the nominal trajectory is. As such, using a log-scale of  $\alpha$  values is a common method to handle this issue.

### Differential dynamic programming and the iterative linear quadratic regulator

Differential dynamic programming (DDP) [84] and the iLQR [73, 126] are both trajectory optimisation algorithms that use dynamic programming to optimise a trajectory. They both consider the Bellman equation which colloquially states that the optimal action to take at the current state is irrespective of previous states and actions. Mathematically, this idea can be expressed through the use of the “Value” function.

Firstly, the *cost-to-go*  $J_i$  is defined as the running cost of a trajectory from a specific time-step  $i$  along the trajectory when the remainder of the trajectory controls  $\mathbf{U}_i$  are applied:

$$J_i(\mathbf{x}_i, \mathbf{U}_i) = l_f(\mathbf{x}_T) + \sum_{t=i}^{T-1} l(\mathbf{x}_t, \mathbf{u}_t). \quad (2.8)$$

The value at time-step  $i$  is then defined as the set of controls that minimise the cost-to-go from the current time-step and state:

$$V(\mathbf{x}, i) = \arg \min_{\mathbf{U}_i} [J_i(\mathbf{x}_i, \mathbf{U}_i)]. \quad (2.9)$$

By setting the value function at the final time-step of the trajectory to be equal to the terminal state cost  $V(\mathbf{x}, T) = l_f(\mathbf{x}_T)$ , dynamic programming can be used to reduce the minimisation of the entire control sequence to a series of individual control minimisation problems from the end of the trajectory to the beginning. This is done by propagating the value function using the Bellman principle:

$$V(\mathbf{x}, i) = \arg \min_{\mathbf{u}_i} [l(\mathbf{x}_i, \mathbf{u}_i) + V(\mathbf{x}, i + 1)]. \quad (2.10)$$

Colloquially, this equation means that the optimal control to take at any current state is the one which minimises the current state cost in addition to the value at the state which you end up in as a result of taking that action.

Using this dynamic programming principle, an optimal control policy can be computed at every time-step, which constitutes of an open-loop term  $\mathbf{k}_t$  and state feedback gain term  $\mathbf{K}_t$ . The details for how this optimal control policy is computed have been skipped, but the interested reader can find these details in [126].

This optimal control policy is then rolled out using the original system dynamics (this is referred to as the “Forwards Pass”).  $(\hat{\mathbf{X}}, \hat{\mathbf{U}})$  represent the new states and controls that are computed during the forwards pass.

$$\hat{\mathbf{x}}_0 = \bar{\mathbf{x}}_0, \quad (2.11a)$$

$$\hat{\mathbf{u}}_t = \bar{\mathbf{u}}_t + \alpha \mathbf{k}_t + \mathbf{K}_t(\hat{\mathbf{x}}_t - \bar{\mathbf{x}}_t), \quad (2.11b)$$

$$\hat{\mathbf{x}}_{t+1} = f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t), \quad (2.11c)$$

where  $\alpha$  is the line-search parameter  $0 \leq \alpha \leq 1$ .

To summarise, at every iteration of iLQR the following three steps need to be repeated. Firstly, the first order dynamics derivatives of the system about the nominal trajectory as well as the first and second order cost derivatives need to be computed about the nominal trajectory. Secondly, the optimal control feedback policy is computed through the use of dynamic programming. Finally, this control feedback policy is then rolled out using the original system dynamics using a line-search method to ensure that a lower cost trajectory is found.

iLQR has seen widespread use and adaptations. Some of these adaptations include combining iLQR with reinforcement learning (RL) [151] (to simultaneously aid in training a policy and using the policy to better inform the terminal state cost), a probabilistic implementation for iLQR to consider stochastic systems [72], modifying the iLQR algorithm into a multiple-shooting variant [38] and adding box constraints that are solved via quadratic programming (QP) to ensure that controls are kept within feasible limits [127].

In this thesis, iLQR is the primary trajectory optimisation algorithm used. This is due to its computational efficiency at finding solutions to difficult problems and its ease of use for highly non-linear dynamics. This work uses iLQR with the MuJoCo [129] physics simulator to simulate the non-linear dynamics of a wide variety of problems.

### Sampling-based approaches

Sampling-based trajectory optimisation algorithms do not consider gradient-based information about the system dynamics or cost function (they are sometimes referred to as zero-order methods) and instead rely on sampling new control trajectories and then evaluating their performance by rolling out the sampled control trajectories using the system dynamics.

Sampling-based methods do not enjoy the same convergence properties as more efficient gradient-based methods but can be effective irregardless of this. The primary reason for this is that the only computational bottleneck of these algorithms is rolling out the sampled control trajectories using the system dynamics; they do not require the computational overhead of computing derivatives or any form of back propagation.

They are particularly useful when combined with MPC. In MPC it is typically more important to compute an *acceptable* solution fast rather than an *optimal* solution slowly [50]. This is because of the ever changing cost landscape that occurs from using MPC on a real system and it is more important to stay within the basin of the optimal solution than exactly at the optimal solution.

Pinneri et al. [107] propose an adapted cross entropy method [111] which they name “iCEM” for sampling-based trajectory optimisation in MPC. The general

CEM represents candidate trajectories by some mean and covariance values ( $\mu_t, \sigma_t \in \mathbb{R}^{d \times h}$ ), where  $\mu$  is the mean of the sample,  $\sigma$  is the variance and  $d$  and  $h$  are the dimensionality of the action space and number of planning time-steps respectively. At every iteration, candidate trajectories are evaluated and an *elite* set of candidates (the top performing trajectories) are used to instantiate the next iteration of samples. Pinneri et al. [107] make two main improvements to the original algorithm which are: (1) Proposing a new sampling strategy to make their method more efficiently search the action space. (2) Maintaining some memory of previous elite trajectories to be instantiated at the next iteration, instead of just updating the mean and covariance values of the trajectory.

Model predictive path integral control (MPPI) [2, 140] works similarly to the CEM. Candidate trajectories are randomly sampled from some Gaussian distribution about the nominal trajectory. These candidate trajectories are evaluated by rolling them out using the system dynamics and a cost for each sampled trajectory is computed. The key difference in MPPI is that *all* candidate trajectories are then weighted and then used to compute an update to the nominal trajectory.

### 2.2.3 Direct methods

In direct optimisation, both the states and controls are treated as optimisation variables. As there are no rollouts of the system dynamics, dynamic feasibility needs to be enforced through the use of constraints. The direct optimisation problem can be written more formally as:

$$\mathbf{U}^*, \mathbf{X}^* = \arg \min_{\mathbf{U}, \mathbf{X}} [J(\mathbf{U}, \mathbf{X})]. \quad (2.12)$$

Direct methods of trajectory optimisation transform the trajectory optimisation problem into a nonlinear program (NLP) [8]. A NLP problem is an optimisation problem where the objective function and/or constraints are nonlinear (as most interesting problems are). A NLP can be written formally as

$$\min_{\mathbf{y}} f(\mathbf{y}), \quad (2.13a)$$

$$\text{subject to: } g_i(\mathbf{y}) \leq 0, \quad i = 1, \dots, m, \quad (2.13b)$$

$$h_j(\mathbf{y}) = 0, \quad j = 1, \dots, p, \quad (2.13c)$$

where:

- $\mathbf{y}$  is the decision variable vector,
- $f(\mathbf{y})$  is the nonlinear objective function,
- $g_i(\mathbf{y})$  are inequality constraints,
- $h_j(\mathbf{y})$  are equality constraints.

Finding the optimal solution to constrained NLP problems is challenging and there are a variety of methods that have been proposed to do this. These methods generally consider what is referred to as the Karush-Kuhn-Tucker (KKT) conditions. The KKT conditions outline a set of conditions for a solution to be optimal for such a constrained optimisation problem.

Firstly, the Lagrangian of the original constrained NLP can be constructed by combining the objective condition with its constraints through the use of Lagrange multipliers:

$$L(\mathbf{y}, \lambda, \mathbf{v}) = f(\mathbf{y}) + \sum_{i=1}^m \lambda_i g_i(\mathbf{y}) + \sum_{j=1}^p \mathbf{v}_j h_j(\mathbf{y}), \quad (2.14)$$

where  $\lambda_i$  and  $\mathbf{v}_j$  are the Lagrange multipliers for the inequality and equality constraints respectively.

There are four conditions which make up the KKT conditions, which are:

### 1. Stationarity

The gradient of the Lagrangian with respect to the decision variable must be zero:

$$\nabla f(\mathbf{y}^*) + \sum_{i=1}^m \lambda_i^* \nabla g_i(\mathbf{y}^*) + \sum_{j=1}^p \mathbf{v}_j^* \nabla h_j(\mathbf{y}^*) = 0. \quad (2.15)$$

### 2. Primal feasibility

The optimal solution  $\mathbf{y}^*$  must satisfy the original constraints:

$$g_i(\mathbf{y}^*) \leq 0, \quad i = 1, \dots, m, \quad (2.16a)$$

$$h_j(\mathbf{y}^*) = 0, \quad j = 1, \dots, p. \quad (2.16b)$$

### 3. Dual feasibility

The Lagrange multipliers for the inequality constraints must be non-negative:

$$\lambda_i^* \geq 0 \quad i = 1, \dots, m. \quad (2.17)$$

This requirement effectively means that violating the constraint should *not* improve the objective function.

### 4. Complementary Slackness

For each inequality constraint, the multiplication of the constraint and its Lagrange multiplier must be equal to zero:

$$\lambda_i^* g_i(\mathbf{y}^*) = 0 \quad i = 1, \dots, m. \quad (2.18)$$

This effectively means that either  $g_i(\mathbf{y}) = 0$  (i.e the constraint is active) or that  $\lambda_i^* = 0$  (the constraint is inactive). Informally this means that the constraint only affects the solution when the constraint is active.

One of the family of algorithms to solve the KKT conditions are sequential quadratic programming (SQP) methods [120]. The general SQP method is to iteratively approximate the NLP problem as a QP and solve the resulting QP sub

problem efficiently. This process can be repeated until convergence or satisfaction of the KKT conditions. A QP is a specific NLP where the objective function is quadratic and the constraints are linear, which is a problem that can be efficiently solved using a simple set of equations. In practice, with SQP the minimiser of the QP subproblem can sometimes overshoot and increase the original objective function. To address this, a descent direction is computed, and the solution is then updated along this direction using either a trust region or a line-search method. A well-known software library that implements this SQP method is SNOPT [39].

Interior point methods [54] are another popular family of methods to solve NLP problems. Interior point methods relax the fourth KKT condition (complementary slackness) by relaxing Eq. 2.18 to:

$$\lambda_i^* g_i(\mathbf{y}^*) = \mu \quad i = 1, \dots, m, \quad (2.19)$$

for some  $\mu \geq 0$ . This relaxation makes the problem easier to solve as it smooths any possible discontinuities caused by the complementarity requirement. To enforce this modification, the objective function is augmented with logarithmic barrier functions that penalise violations of the inequality constraints. The new objective function is written as:

$$\min_{\mathbf{y}} f(\mathbf{y}) - \mu \sum_{i=1}^m \ln(-g_i(\mathbf{y})). \quad (2.20)$$

This relaxed problem can then be solved efficiently using Newton's method. However, the solution to this is not the solution to the original problem. To address this issue, interior point methods iteratively solve this minimisation sub problem and reduce the size of  $\mu$ . For every new sub problem, the previous solution is used to warm start the optimisation, making Newton's method converge faster and more reliably. As  $\mu \rightarrow 0$ , the solution will approach the optimal solution to the original problem as the KKT conditions will be satisfied. A popular implementation of an interior point method solver is known as IPOPT [136] and is commonly used to solve such NLP problems.

NLP problems and general methods of solving them have been discussed. It is now important to briefly understand how this relates to our particular use-case, which is for trajectory optimisation. Direct optimisation tries to minimise Eq. 2.12 and so this is the objective function of the NLP. The inequality and equality constraints are generally problem specific. Some quite common constraints are keeping controls within certain limits:

$$\mathbf{u}_{min} \leq \mathbf{u}_t \leq \mathbf{u}_{max}, \quad (2.21)$$

keeping dynamical quantities (such as velocity or acceleration) between certain safety limits:

$$\mathbf{x}_{min} \leq \mathbf{x}_t \leq \mathbf{x}_{max}, \quad (2.22)$$



or maintaining dynamic feasibility of the system:

$$\mathbf{x}_{t+1} - f(\mathbf{x}_t, \mathbf{u}_t) = 0. \quad (2.23)$$

This final general dynamic feasibility constraint is the subject of a great amount of research, particularly for contact-based optimisation problems. Works that address this issue will be discussed in Sec. 2.3.

Direct optimisation methods offer several advantages over shooting-based approaches, such as improved global convergence properties and the ability to initialize trajectory states instead of just controls. However, these benefits come with significant challenges, particularly for contact-based optimisation problems. Some of these issues include:

- **Discontinuities:** Due to the presence of contact dynamics, it is common to have discontinuities in the objective and/or constraints of the NLP.
- **Hybrid dynamics:** Contact-based systems have hybrid dynamics with different modes. Modelling the switching between different modes leads to a combinatorial explosion in the NLP.
- **Sensitivity:** Solving NLP problems can require a good initial guess to the solution of the problem to converge to the global optima. This is especially true for problems with highly non-linear dynamics which is the case for contact-based systems.

Some of the works discussed in Sec. 2.3 will address some of these listed issues.

### 2.2.4 Collision-free trajectory optimisation

Trajectory optimisation has also been used for finding collision-free motion plans. These works aimed to solve that key issue that occurred from the sampling-based methods in classical motion planning. This issue was that the found collision-free paths would often be non-smooth and jerky. The solutions found by trajectory optimisation methods are naturally smooth, if smoothness of the trajectory was encoded as a requirement into the cost function.

Ratliff et al. [110] proposed CHOMP (covariant hamiltonian optimised motion planning). Their goal is to find a smooth collision-free path from a starting configuration to a goal configuration via a set of waypoints. They denote this trajectory of waypoints as  $\xi$ , where  $\xi = [\mathbf{q}_0, \mathbf{q}_1, \dots, \mathbf{q}_T]$ . They split their cost function into two parts: (1)  $f_{prior}(\xi)$  which is used to minimise dynamic quantities such as velocity, acceleration or any other higher order terms and (2)  $f_{obs}(\xi)$  which is used to penalise the robotic manipulator getting too close to obstacles in the environment. They use covariant gradient descent to update the trajectory iteratively to find an optimal smooth collision-free path.

Kalakrishnan et al. [60] propose STOMP (stochastic optimised motion planning). Their problem formulation is largely identical to the problem in CHOMP but they use stochastic sampling to compute descent directions to iteratively update the optimal trajectory. One advantage of their method is that it can enable the use of more complex and discontinuous cost functions as the direct need for gradients is not required. The authors also note that STOMP is sometimes able to escape small local minima due to the stochastic nature of its sampling.

It should be noted that both of these works (CHOMP and STOMP) do not consider the general dynamics of the system. They both treat all joints of the robotic manipulator as independent (which is not the case as the motion of linked joints can affect one another based on inertial effects).

TrajOpt [117] and GuSTo [14] are both examples of direct optimisation works that also consider the problem of finding a collision free motion plan. These works *do* consider the system dynamics through the use of constraints and employ either SQP or Sequential Convex Programming (SCP) to compute optimal motion plans.

GOMP [53] (grasp optimised motion planning) focuses on the problem of computing optimal motion plans in a pick-and-place scenario, where it is desired for a robot to maximise its “picks per hour” in an automated warehouse setting. They formulate their problem as a trajectory optimisation problem for finding efficient collision-free paths from one configuration to a goal configuration whilst transporting an object. Interestingly, they consider optimising the grasp that is used to hold objects while they are being transported. Certain grasps of an object will be more dynamically stable than other grasps, which is also trajectory dependent. Choosing an optimal grasp that enables the robotic manipulator to travel at higher speeds without dropping the desired object is important for efficiency.

### 2.2.5 Remarks

This section has outlined the general trajectory optimisation problem and discussed some different classes of algorithms that can be used to solve them. It has also briefly touched upon some algorithms that have been developed that use these techniques to solve collision-free optimised motion planning, the same type of problem that has been deemed as “classical motion planning”.

Everything discussed so far is how trajectory optimisation has been used for finding collision-free motion plans. However, the primary goal of this thesis is using trajectory optimisation to perform contact-based motion plans, a significantly harder challenge. The next section discusses how trajectory optimisation has been extended in the literature to optimise motion plans that use contact.

## 2.3 Contact-Based Trajectory Optimisation

This section discusses various works that consider using trajectory optimisation to perform contact-based tasks, either for locomotion or manipulation. A common theme that will be noticed throughout this section is the discussion of computational complexity, especially relating to two key ideas; computing dynamics derivatives and scaling issues with higher-dimensional problems like manipulation in clutter. These two issues directly relate to Chapter 4 for how the issue of computational expense computing dynamics derivatives can be reduced and Chapter 5 where methods of reducing the dimensionality of a problem online during MPC are discussed.

### 2.3.1 Shooting methods

Several works have considered the problem of reaching through clutter using shooting-based trajectory optimisation methods. The reaching through clutter problem consists of a robot that is trying to grasp and retrieve some goal object in some tight environment (such as a shelf or cupboard). In this environment are a variety of other distractor objects that sometimes the robot does not want to interact with (such as delicate objects) and other objects that the robot can interact with. Finding motion plans in these setting can be challenging as the robot needs to reason about multi robot-object and object-object interactions so that it can weave through the clutter to grasp the goal object.

Kitaev et al. [63] consider this reaching through clutter problem. In their problem, they are interested in environments with lots of thin and tall objects that are amenable to toppling and they encode not toppling objects in the environment into their cost function. They sample different straight line movement of the end-effector from different directions and choose the best trajectory as the initial input to their optimiser. In their work, they show optimisation time results for solving environments with different levels of clutter. They show that the optimisation times can become prohibitively expensive as the number of objects increases, noting that, the main bottleneck of computation is computing dynamics derivatives via FD.

Both Agboh et al. [4] and Papallas et al. [100] also both consider the problem of reaching through clutter using sampling-based shooting methods. Their tasks are more focused on dense clutter where the target object is more difficult to grasp but there are less concerned with what happens to the clutter. Importantly, both of these works are focused on executing on real robotic hardware. Agboh et al. [4] propose an approach call “Online Re-planning”, they note how executing trajectories that were computed in simulation open-loop is bound to fail in such a task due to the stochastic difference between physics simulators and the real world. Instead of using conventional MPC techniques (due to longer planning times) they propose executing trajectories open-loop and detecting the error between the planned state

and the real world state (i.e. differences in object positions). If this error becomes too large, they halt execution and re-optimize their trajectory. Papallas et al. [100] consider a human in the loop approach to reaching through clutter. They note, that often times directly reaching through the clutter for the goal object is not a viable strategy, and in fact some objects may need to be relocated before a successful grasp can occur. They leverage human input to provide directions to the robot to move objects from one position to another before then trying to re-grasp the goal object. In later work, Papallas et al. consider learning whether the robot could benefit from human intervention using a neural network [102].

Kurtz et al. [69] use iLQR to optimize contact rich trajectories for both locomotion of a quadruped as well as full arm manipulation. The full arm manipulation is particularly impressive, managing to synthesize trajectories where the robotic manipulator can squeeze a large foam ball against itself and lift it vertically upwards. This is achieved by the creation of a hydroelastic contact model that the authors implemented in Drake [128]. This hydroelastic contact model proved to be useful in softening the discontinuous effects of making and breaking contact and realistic enough that the trajectories could be executed open-loop on real robotic hardware. However, computing these trajectories was prohibitively expensive, taking tens of minutes to compute trajectories that were only tens of seconds long. The authors note that most of the optimization time was spent computing dynamics derivatives via AD.

There are a variety of locomotion based works that use either iLQR or DDP to synthesize complex contact-rich locomotion of legged robots. Chatzinikolaïdis et al. [21] use DDP for synthesizing dynamic motions for a single legged robot to jump up from the floor. Tassa et al. [126] use iLQR for controlling a humanoid in an asynchronous MPC framework in simulation. They note that the “real-world” environment required a 7x slowdown so that their humanoid could operate effectively. They also note that the largest computational bottleneck was computing dynamics derivatives. Erez et al. [35] also use iLQR for synthesizing a variety of complex contact-rich tasks such as making a humanoid robot stand up from a laying down position. Grandia et al. [42] use a DDP variant to control a quadruped using MPC. They show that despite the low update rate of the MPC algorithm (15Hz) they can employ the local state-based feedback policy of the algorithm to keep the robot stable.

Despite not having the strong convergence guarantees of gradient-based methods, sampling-based planners have been used in a wide array of tasks [3, 20, 50, 100, 106]. One of the advantageous aspects of sampling-based shooting methods compared to gradient-based methods is their ease of implementation and reduced computational load (generally due to not requiring the expensive computation of dynamics derivatives). Sampling-based planners are particularly amenable to multi-threading, and

with the rise of GPU parallelised physics simulation [77, 6], this offers new opportunities. Pezzato et al. [106] use MPPI combined with GPU parallelisable physics simulation to sample hundreds of possible trajectories for optimisation. Using hundreds of environments also enables the use of domain randomisation in trajectory optimisation, making policies more robust to inaccuracies in physical parameters.

### 2.3.2 Direct methods

The hybrid nature of contact dynamics is significantly challenging for direct optimisation. In situations where there is only a single continuous dynamics model, then direct optimisation is fairly simple, however this is not often the case for contact-based tasks. Original works often considered “mode scheduling” where the optimiser was given a set schedule for when the robot would switch between contact modes and its task was to optimise each segment individually [118]. However, it is much more desirable to let the optimiser itself decide when the robot should switch between contact modes as it allows greater flexibility in the final solution, as well as taking away a possibly complex decision making process away from the user. This second school of thinking is referred to as contact-implicit trajectory optimisation. Under this contact-implicit paradigm, Mordatch et al. [87, 88] generated complex behaviour for animated characters that would require complex mode scheduling.

Hogan and Rodriguez [49] consider such an approach. They consider the problem of the two dimensional push slider. They use the quasi-static assumption in this work and formulate the problem as a MIQP (a more difficult version of a QP problem). They use integer decision variables to handle the hybrid dynamics present in this problem (sticking, sliding up and sliding down), enabling their method to autonomously reason about switching between dynamics modes. However, as the controller can switch between any of the three contact modes during execution, this leads to a combinatorial explosion in the planning problem. They propose a “family of modes” approach that uses a fundamental intuition that any instance of their problem can be solved by the same sequence of four dynamics modes of different lengths. For contact rich tasks like in-hand manipulation, this approach would not be computationally tractable.

Moura et al. [89] look at a similar problem. They formulate this problem as an MPCC (mathematical program with complementarity constraints)<sup>1</sup>. They encode the hybrid dynamics of this task as complementarity conditions, as opposed to the mixed-integer formulation proposed by Hogan and Rodriguez [49]. Solving strict complementarity conditions can be challenging in NLP problems, as such, they add

---

<sup>1</sup>Complementarity constraints are ones where the multiplication of two variables must equal zero, enforcing that at least one of the two variables must be zero. A common use of this in contact handling is the idea of *not* applying force at a distance. Either the force a robot applies to an object must be zero, or the distance between the robot and the object must be zero.

a slack variable that allows their controller to violate complementarity slightly and the size of this slack variables is then penalised in the cost function. Posa et al. [108] also consider modelling problems as MPCC. They present a more general method for tasks that involve contact dynamics and show that their method scales well with dimensionality and can be applied to a variety of tasks, including both locomotion and manipulation.

Önol et al. [95] investigate the differences between different contact models and how well they facilitate contact-implicit trajectory optimisation when used with complementarity constraints. Building upon this work, Önol et al. [96] propose a method for performing contact-implicit trajectory optimisation for manipulation that requires no meaningful initial trajectory. Their method uses the successive convexification algorithm [80] and aims to penalise force applied at a distance until a realistic trajectory is found. Later, they propose a tuning-free algorithm that uses an outer-loop penalty method. This outer-loop iteratively increases the penalty of applying force at a distance so that successive trajectories become more dynamically feasible [97]. One issue with their method is that they need to specify contact points that can apply force at a distance before optimisation begins, requiring human intuition in that regard. Zhang et al. [145] propose one such solution to this problem which they name STOCS (simultaneous optimisation and contact selection). Their method interleaves contact point selection with trajectory optimisation.

Most trajectory optimisation methods that have been discussed so far consider forward dynamics to create optimal trajectories. However, it is also possible to use inverse dynamics to optimise trajectories which can be advantageous. Kurtz et al. [70] propose an inverse dynamics trajectory optimisation approach, where they formulate the decision variables as only the generalised positions of the scene. Whilst inverse dynamics trajectory optimisation is not new [87, 88, 34], Kurtz et al. show an efficient implementation with results on a wide variety of tasks. In their formulation they create a custom point contact model that only requires their generalised positions to compute which is accurate enough for MPC. Inverse dynamics trajectory optimisation can be faster than forward dynamics methods due to inverse dynamics typically being faster to evaluate [36].

Graesdal et al. [41] use an approach named graphs of convex sets [81] for manipulating a single T-block on a 2D planar surface. The general idea of graphs of convex sets is to solve a graph search problem where the nodes in the graph are actually continuous convex sets. The nodes that are travelled to in the problem can then be solved independently and efficiently as convex problems are easy to solve. Graesdal et al. [41] formulate their problem using this idea where each convex set was either free motion of the robotic manipulator or an individual contact mode of the system (sticking, sliding etc.). This approach has the advantage of providing global optimality. However, it should be noted that this approach is not easily extended

to multiple objects, as the authors formulate the problem from an object-centric perspective.

## 2.4 Other Methods for Contact-based Manipulation / Locomotion

This section provides an overview of non-optimisation based methods that have been used for contact-based manipulation/locomotion. The largest category is learning-based methods (such as RL and imitation learning (IL)), but before those methods are discussed, non learning-based works will be discussed first.

One of the first seminal pieces of work that considered enabling robots to interact with their environments using non-prehensile manipulation was conducted by Mason [82]. In this work, Mason derives the fundamental mechanics of pushing. Building on top of this, Dogar et al. [33] proposed push grasping, a mechanism for the robot to use non-prehensile manipulation pushing actions to assist in grasping objects. They note how pushing actions can actually be used as a method to funnel objects into the manipulators end-effector and reduce uncertainty in the object's position.

Dogar et al. [31] also consider the problem of grasping an object in clutter. They propose a physics-based grasp planner that enables contact between a robotic end-effector and multiple objects in an environment to grasp an object. Their method enables the robot to push other objects out of the way to successfully close the fingers around the desired object. To speed up planning, they pre-compute possible contact interactions between the robot's end-effector and objects to be used for planning. Importantly, whilst they enable multiple robot-object interactions in their planner, they do not allow object-object interactions so that the problem is computationally tractable. As well as this, Dogar et al. [32] propose a general planning framework for non-prehensile manipulation in clutter. They consider the problem of trying to retrieve some goal object from clutter where it is impossible to initially grasp the object, without first relocating other objects out of the way of the robot. Their method is to initially create a plan whilst ignoring contact to grasp the target object. They then evaluate this plan and check what objects the robot would collide with to achieve it and add them a list of objects that need to be relocated. This methodology is then applied recursively, trying to relocate objects to new areas using non-prehensile manipulation, evaluating if prior plans will require prior rearrangements of other objects.

Saxena et al. [116, 115] consider a similar problem of retrieving a desired object from a cluttered environment. They also consider situations where the robot will first have to relocate objects before being able to grasp an object. They decompose the planning problem into two steps, firstly they use a similar concept to the work from Dogar et al. [32] and they plan a path of the robot whilst ignoring collision to

the desired goal object. They then compute all the objects that would be in collision over this path. To rearrange the scene they first treat all the objects that need to be moved as “agents” which can actuate themselves in the scene. They compute paths for all these agents and then achieve these paths by computing pushing plans for the robot to manipulate them.

Papallas et al. also consider the problem of reaching through clutter [101]. They use a kinodynamic sampling planner to compute trajectories for reaching through clutter to retrieve some target object. Notably, they consider a human in the loop approach where a human can be queried to provide sub-goals for the planner (moving an object to a new location) which are intended to make the planning problem easier.

The works discussed so far in this section have all employed some form of vision to observe the state of the real world and use this information for planning trajectories. While vision is undoubtedly an important aspect of robotic motion planning, other perception tools, such as tactile and force feedback, become relevant when discussing contact-based manipulation. Tactile sensing enables humans to perform a variety of dexterous tasks and operate in environments when we can not see. A variety of tactile sensors have been designed and suggested by numerous researchers, significant numbers of these can be prohibitively expensive. Ward-cherrier et al. [138] present the TacTip family of tactile sensors, a 3D printable set of tactile sensors that are cheap to manufacture. Brouwer et al. [17] use an end-effector covered in soft tactile sensors to reach through dense clutter where vision is limited. Jain et al. [55] also consider reaching through dense clutter but instead use whole-arm tactile sensing to minimise the forces felt on the robotic manipulator.

Both Zito et al. [150] and Pang et al. [99] consider using RRT for performing contact-based manipulation. Zito et al. [150] consider a two-level approach for pushing manipulation. They consider the problem of manipulating an “L-flap” object, in terms of its position and its orientation. The first stage of planning is to use a global RRT planner in the objects space to find a sequence of poses that the object could follow from one pose to the goal pose. Then a secondary local push planner plans what actions the robot must take to manipulate the object through the sequence of nodes as specified by the global planner. Pang et al. [99] propose a global planning method for contact-rich tasks. They use a method called “randomised smoothing” to smooth out discontinuous dynamics derivatives about contact locations that are then used with an RRT based planner to plan contact-rich manipulations.

### 2.4.1 Learning-based approaches

Learning-based contact manipulation approaches refers to methods that use some form of machine learning to assist in the process of creating contact-based motion



plans. This is quite a generic description, some examples of what this can refer to are:

- “Model-free” approaches where an optimal robot policy is learned from data that enables a robot to choose an action given its current state. There are two main methods for learning these policies, either by RL where a robot can experiment in a world to see which actions give the best cumulative reward through trial and error [37, 143, 141], or by IL where a robot aims to copy an expert demonstrator who shows what the best action is to take given a current state observation [15, 146, 11].
- Methods that learn some form of a dynamics model of the system [5, 149, 56, 37, 65]. This dynamics model can then be used afterwards for planning actions.
- Learning other useful and hard to predict components for other control algorithms. One such example is learning optimal value functions [9, 148, 135] that can then be used in trajectory optimisation for MPC. Another example is learning better initial trajectories that can be used in trajectory optimisation, enabling optimisation to converge to a solution in fewer iterations. [19, 7].

This thesis does not use any learning-based approaches, however, it is important to discuss some of the most relevant and cutting edge works in this area to show examples of what these methods are capable of. Model-free approaches will be the primary focus, starting with RL methods and then IL methods.

### Reinforcement learning

RL is an unsupervised machine learning method where an agent<sup>1</sup> learns to interact with its environment with the goal of maximising the rewards it receives [124]. The standard framework for RL is that a user will setup some task and define some reward function (the reward function is meant to be large when the robot does something that is desired) and then the robot will attempt to learn an optimal policy through experimentation in the environment. An important aspect to note, is that for RL for robotics, the environments the robots are trained in is usually some form of a physics simulator. This is not always the case and there are works where robot policies are directly trained on real robotic hardware [75, 30], however, it is most common to train policies in physics simulators for the following reasons: (1) RL training is data intensive. This process can be sped up via physics simulators as they can run faster than real time. (2) It is easier to reset the system back to an initial state in a physics simulator as it can be done autonomously, rather than in the real world where human intervention would be needed and, (3) robots trained via

---

<sup>1</sup>For the purpose of this thesis, an agent refers to a robot or set of robots.

RL can break easily in the real world in the training stage, however it is impossible to “break” a simulated robot.

It should be noted, there are a lot of parallels between the mathematical formulations of both RL and trajectory optimisation. As discussed earlier, trajectory optimisation is concerned with *minimising cost*, while RL is about *maximising reward* - two sides of the same coin. On occasion, a footnote will be added about these similarities for the interested reader. This section will respect the typical RL mathematical formulation; to begin with, this means that the state of the system at some time-step  $t$  is denoted by  $\mathbf{s}_t$  and the action is denoted by  $\mathbf{a}_t$ <sup>1</sup>.

Most RL methods consider the decision-making problem as a *Markov Decision Process*, defined by the tuple  $\langle S, A, P, R, \gamma \rangle$ , where:

- $S$ : is a set of states representing the environment.
- $A$ : is a set of actions that the agent can take.
- $P(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ : is the transition probability distribution, specifying the likelihood that the agent ends up in state  $\mathbf{s}'$  when currently in state  $\mathbf{s}$ , taking action  $\mathbf{a}$ .
- $R(\mathbf{s}, \mathbf{a})$ : represents the reward function, a function that turns the current state and action into a scalar reward for the agent.
- $\gamma \in [0, 1)$ : is the discount factor, determining the importance of rewards gained in the future as opposed to rewards gained now.

The formal goal of RL is to find a policy  $\pi(\mathbf{a}|\mathbf{s})$  (usually encoded as a neural network) that enables the agent to maximise the cumulative reward over time:

$$G = \sum_{k=0}^{\infty} \gamma^k R(\mathbf{s}_{t+k}, \mathbf{a}_{t+k}). \quad (2.24)$$

It should be noted, that in the typical RL formulation an infinite horizon is considered, rather than the finite-horizon that is considered in trajectory optimisation. As such, without the introduction of the discount factor which drives rewards to zero as time progresses, the agent would learn that it is best to never complete its task in the hope of gaining future rewards.

Most RL methods consider either the Value or action-value function (sometimes referred to as the Q-function) to optimise their agents policy:. The value function is defined as:

$$V^\pi(\mathbf{s}) = E_\pi[G|S = \mathbf{s}], \quad (2.25)$$

---

<sup>1</sup>Recall, that in trajectory optimisation, the state vector was denoted by  $\mathbf{x}_t$  and the control vector as  $\mathbf{u}_t$  (analogous for action).

where  $E_\pi$  denotes the expectation under policy  $\pi$ . The value function effectively tells the agent how useful it is to be in a certain state. The action-value function, on the other hand tells the robot how useful it is to take a specific action dependent on the specific state the agent is currently in:

$$Q^\pi(\mathbf{s}, \mathbf{a}) = E_\pi[G|S = \mathbf{s}, A = \mathbf{a}]. \quad (2.26)$$

Then, using Bellmans equation<sup>1</sup>, recursive definitions of these functions can be written. These recursive equations are the cornerstone of a variety of RL algorithms. The mathematical formulation of RL will stop here as they are outside the scope of this thesis, however, the interested reader is directed to a more comprehensive introduction to RL provided by Sutton [124].

RL has shown great success in locomotion problems for walkers and quadrupeds [43, 103, 64, 104], often enabling robots to find more efficient gait patterns, as well as traversing complex and uneven terrain.

Zeng et al. [144] and Deng et al. [30] both consider situations where the robot is required to grasp some object but is unable to due to the presence of surrounding objects. They use RL to choose pushing actions to rearrange the objects so that a specific item can then be grasped. Yuan et al. [143] learn end-to-end rearrangement planning using pushing manipulation actions in simulation and then consider how they can effectively transfer the policy their agent learned in simulation onto a real robotic system effectively. Finn and Levine [37] propose a model-based approach that can learn how to push objects in the robots environment through self experimentation. The robot, during a training phase, trials pushing actions in its environment to create a dataset of how its actions affect that environment. Their data is performed at a pixel level, so their model learns to predict how the pixels of a camera image will change subject to its actions. The user can then specify the robot to move an object by specifying a set of pixels to move to a new location in the image. They show generalisation to unseen objects. The previous works primarily considered 2D object pushing in a plane. Xu et al. [141] consider the problem of pushing tall objects without toppling them. They use deep RL Q-learning and demonstrate that their policy can be executed on real robotic hardware.

Some researchers have been interested in teaching high-dimensional anthropomorphic hands to complete a wide range of tasks using RL. Charlesworth et al. [20] teach a robotic hand to perform a variety of complex tasks such as handing objects between robots and spinning a pen in hand using a combination of trajectory optimisation and RL. They use a trajectory optimisation algorithm approach to speed up the training of a RL agent which increases overall sample efficiency. The resulting RL agent can outperform the trajectory optimisation method due to the trajectory optimisation algorithm struggling to run in real-time for meaningful optimisation

---

<sup>1</sup>Similarly to how it is used in the iLQR optimisation algorithm.

horizons. Huang et al. [52] also consider RL for controlling an anthropomorphic hand. They consider the concept of handling delicate objects, in such situations it is desired for the robot hand to minimise impact forces on the object when manipulating it. They add this information to the reward function of the agent, making the agent attempt to minimise the forces it applies to the object.

One common theme of these works is for the requirement that the user needs to specify and probably experiment with a reward function for the agent to elicit some desired behaviour. A different approach is inverse reinforcement learning where the user provides a demonstration to the robot and then the robot infers what the goal of the task is from the user demonstration [1].

Matas et al. [83] use RL for deformable object manipulation for the purpose of manipulating 2D deformable objects, such as cloths. They show that their method can achieve tasks such as diagonal folding and hanging a cloth over a wire frame. Importantly, they show that their method can be deployed and executed on real robotic hardware through the use of *domain randomisation*. Domain randomisation is a common technique in RL to assist with transferring policies learned in simulation to the real world, to overcome the sim-2-real gap. Basically, domain randomisation randomises physical parameters of the world in simulation (e.g. friction values, object weight, joint friction etc.) and trains the policy over a variety of different environments. The policy that is trained is then more robust to different physical parameters, making it easier to transfer to the real world.

## Imitation learning

Imitation learning is an alternate learning approach in robotics where demonstrations as to how to solve a task are provided by an expert, and the robot attempts to use these demonstrations to help it solve the same task. Many of the works that will be discussed in this section are more specifically classed as “behaviour cloning”, a subset of imitation learning which has gained significant popularity in recent years [15, 16, 146, 25, 149]. As opposed to RL, behaviour cloning is a supervised machine learning technique, where a policy  $\pi$  is trained to replicate the behaviour from expert demonstrations.

This process initially requires an expert<sup>1</sup> to create a dataset of demonstrations  $D_E = [(\mathbf{s}_i, \mathbf{a}_i)]_{i=1}^N$ , where  $\mathbf{s}_i \in S$  is the state of the system,  $\mathbf{a}_i \in A$  is the corresponding action taken by the expert and  $N$  is the number of labelled data points. The goal of behaviour cloning is then to find a policy  $\pi_\theta(a|s)$  that minimises the discrepancy between the actions it takes and the actions provided by the expert, given a specific state. This training can then be done using classical machine learning methods

---

<sup>1</sup>The expert is usually a human, but in some works, trajectory optimisation or other algorithms can be used to provide expert demonstrations.

by defining some loss function and then training a neural network using gradient-descent algorithms.

One main reason why behaviour cloning has become so popular in recent years is the advancements made in neural network architectures, particularly transformer-based and diffusion-based models. Chi et al. [25] propose a diffusion policy method that uses an iterative de-noising process to plan trajectories. Their method is shown to perform 15 different tasks, such as T-block pushing, T-shirt folding and flipping rigid objects, but importantly *each task* required a separate policy.

Brohan et al. [15, 16] consider using a transformer-based architecture for imitation learning for completing tasks in a kitchen environment. They show a wide variety of tasks can be learned using their architecture and importantly only train a single model for all tasks. The task that the robot should complete is then encoded as a text input with commands such as “retrieve the coke can from the fridge”. They also show generalisation to some unseen tasks for which demonstrations were not explicitly given. Zhao et al. [146] also consider transformer-based models, they show how they can be used for fine-grained manipulation tasks (inserting batteries into a TV remote, opening a zip-lock bag).

Several of these works realise that to achieve good control, temporal consistency between control steps is a key requirement [25, 146, 15, 16]. Instead of planning a single action for every observed state, these policies generally plan a sequence of actions in the future, even if they are not all executed. This enables temporal consistency between action steps and ensures the policy is not going to try and alternate between different methods of solving a task<sup>1</sup>.

One of the major limitations of such behaviour cloning techniques is the data-intensive requirement to be able to show generalisation over a wide variety of tasks. O’Neill et al. [94] propose a general dataset they name “Open X-embodiment” which collects datasets from research labs around the world for a variety of robot morphologies and tasks. The idea being that this data-set can be used by researches to train foundational models for robotic control. Black et al. [11] shows impressive generalisation to a wide variety of tasks in one model which they name  $\pi_0$ . They use some of the data provided by the Open X-embodiment data-set to train their model consisting of a pre-trained visual language model that gives their robot semantic reasoning skills along with an “action expert” which generates sequences of actions.

Whilst these large datasets are indeed useful, there is no possible way that these datasets will ever be able to hold data on all tasks that humans could ever want a robot to do. If these models can not generalise to all tasks then some element of retraining will be required to make a specific robot do a specific task that a human wants. Some researchers have instead focused on what robots can achieve when very

---

<sup>1</sup>Imagine trying to go around a tree, you can either go left or right, but you need to stick to one direction.

few demonstration are given. Johns et al. [58] shows how contact-based tasks can be performed when only giving a single demonstration to the robot. They provide a demonstration at the “bottleneck” of the task (typically the point when contact is made between the robot and object). The robot then learns under a self-supervised setting how to approach the object to reach a desired configuration and then the demonstration can simply be replayed exactly to accomplish the challenging part of the task.

Providing demonstrations can also be used to assist with training RL policies, to give them a foundational policy that can be improved upon. This idea has been used in several RL works [109, 91, 48]. Hu et al. [51] proposed “Imitation Bootstrapped Reinforcement Learning”, a framework that uses expert demonstrations to provide a RL agent with a starting policy for how to solve a task. In the RL training phase, the agent can either sample an action that an expert would take, or take a random action; Using this methodology the agent can improve the policy provided by the expert through unsupervised learning. They show that this methodology can greatly increase the training speed of their agent.

One issue with imitation learning techniques is that if you want to teach the robot to perform a task in a specific location, such as outdoors in a park, you need to manually transport the robot to this location. This is so you can teleoperate the robot in the location where it will be performing some task. Chi et al. [26] address this issue and present UMI (the Universal Manipulation Interface). They create a custom end-effector that can be both attached to a robot as well as held by a human. The core idea is that the human can take just this end-effector to any desired location and collect data themselves. This data can then be transferred to the robot seamlessly as from the robots perspective, it looks like the robot was executing the task.

## 2.5 Approximations for Speeding up Robotic Motion Planning

The work presented in this thesis broadly focuses on using approximations to speed up robotic motion planning. This is quite a common idea in the robotics community and has been used in a wide variety of different ways.

One common approach is dimensionality reduction, where high-dimensional problems are simplified by identifying and focusing only on the most important dimensions or by discovering a lower dimensional representation of the problem which captures most of the relevant information of the higher-dimensional problem.

### 2.5.1 Dimensionality reduction

Dimensionality reduction has been used extensively in various different ways in robotics. Zheng et al. [147] consider applying RRT to very high-dimensional problems. Even though RRT was designed to operate in high-dimensional spaces compared to classical methods, eventually, increased dimensionality can harm computation time. They consider accelerating the RRT algorithm by reducing the dimensionality of the state space.

Vernaza et al. [133, 134] propose “learning dimensional-descent”. They consider locomotion problems for large abstract robots trying to find a collision-free path through a heavily cluttered environment. Their method learns the optimal dimensions to plan in whilst leaving the path in other dimensions fixed.

A different form of dimensionality reduction is explored by Hogan and Rodriguez [49]. They formulate a mixed-integer direct trajectory optimisation problem for a 2D push-slider system. Due to the integer decision variables that are used to determine contact modes of the system, solving this problem is difficult due to the combinatorial explosion in the possible sequences of contact modes. They consider a “Family of Modes” which reduces this combinatorial explosion. Their main observation was that the problem they aimed to solve could always be solved by the same sequence of 4 contact modes, making this problem computationally tractable.

Manipulation of deformable objects is inherently very highly-dimensional. Wang et al. [137] 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. [76] 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 [121] propose a reduced order iLQR implementation similar to the method proposed in Chapter 5, where the roll-outs are still performed using the full system dynamics. They evaluate their methods for optimisation until convergence for high-dimensional partial differential equations. The work presented in Chapter 5 is different in that it focuses on robotic manipulation problems where analytical formulations of the partial differential equations are not readily available. Crucially, it also focuses on dynamically changing the size of the state vector used during online task execution, as the optimal reduced order model changes during task execution.

Dexterous hands are another common high-dimensional problem routinely investigated in the robotics community. Robotic hands often have significantly more DoFs than is required to solve certain routine manipulations, and as such, are good candidates for dimensionality reduction. Ciorcarlie et al. [27] 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 of grasping a wide variety of complex objects. Jin et al. [57] consider learning a reduced-order hybrid model for complex three finger manipulation, enabling real-time closed loop MPC.

Locomotion is one such area where reduced order models have found great performance in enabling real time closed-loop control of high-dimensional robots. A common simplification that is used for legged locomotion is approximating complex robotic legs as simple inverted pendulums [12, 59]. Chen et al. [22] take these ideas further and consider the idea 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.

### 2.5.2 Other approximation methods

There are also a variety of works that have used other types of approximations that would not fit into the category of dimensionality reduction. However these methods have been used to speed up various parts of robotic motion planning.

One of the most common approximations/simplifications in contact-planning is the quasi-static assumption. This is where, at any given time the problem can be thought of as static (that is to say that all inertial effects, such as acceleration are ignored and considered negligible). Using this assumption can simplify the system dynamics [49, 23].

Another common approximation is to approximate the friction cone within contact-based planning [131, 123]. In direct optimisation methods, it enables exchanging a single non-linear inequality constraint with four linear inequality constraints (by approximating the friction cone as a pyramid). This approximation makes solvers much more efficient as it is significantly easier to enforce four linear constraints as opposed to one inequality constraint.

Saleem and Likachev [114] considered a planning approach that only used a computationally expensive physics simulator when necessary. They relied on a cheaper geometric model for actions when a robot was far away from obstacles and only used the general physics simulator when contact was possible.

Following on from this, collision checking is an important aspect in many robotic motion planning algorithms. A common approach to speed up collision checking is to approximate robotic models with simple 3D objects (spheres, cylinder, cuboids) as they are much easier to perform collision checks on [110, 90]. Kitaev et al. [63] use a



simplified mesh of their robotic end-effector to speed up their optimisation algorithm in the early iterations. As their algorithm begins to converge they use a more detailed mesh of the end-effector. In locomotion, Perrin et al. [105] consider swept volume approximations for fast collision checking so they can control a humanoid robot to walk over uneven ground online.

In trajectory optimisation, one of the primary parameters that affects computation time is the optimisation horizon. Longer optimisation horizons naturally result in longer computation times of the optimal trajectory. This is particularly important in MPC applications, where it is critical for optimisation to occur fast so that *policy-lag* does not have a detrimental effect on the real robotic system. However reducing the optimisation horizon too significantly can result in *myopic* behaviour. One way to address this is to add additional information into the terminal state cost function to overcome this myopic behaviour. Some works use learning to try and create more informative terminal state cost functions that can be used online during MPC so that the optimisation horizon can be reduced [148, 135, 74].

As was discussed previously in Sec. 2.2, iLQR is a version of DDP that does not consider the second order dynamics derivatives. Whilst using the second order dynamics derivatives aids convergence, the cost of computing them can be prohibitively expensive. Due to this computational expense, many researchers use iLQR instead. Manchester and Kuindersma [78] and Nganga and Wensing [93] both consider approaches to perform variations of DDP at roughly the computational cost of iLQR. They acquire the advantages of the quadratic convergence from DDP whilst not paying the full expensive cost of computing the second order dynamics derivatives exactly.

Work by Cheng et al. [24] is relevant to the methods proposed in Chapter 4. They consider a learning approach for approximating first order dynamics derivatives of a system. They train a neural network per trajectory optimisation task, that given a state and control vector, returns a prediction of the first order dynamics derivatives of the system. The work proposed in Chapter 4 takes a different approach, it computes key-points over a trajectory where the dynamics derivatives will be computed accurately. The remainder of the dynamics derivatives are approximated using cheap linear interpolation. In addition, the methods proposed in Chapter 4 are tested on more complex tasks than Cheng et al. (contact-rich manipulation/locomotion). Furthermore, the methods proposed in Chapter 4 do not require training a neural network for every new trajectory optimisation task.

Finally, Han et al. [44] investigated methods for speeding up trajectory optimisation for humanoid control. One of the techniques they investigate is referred to as “derivative skipping”. Derivative skipping in their work is effectively the same as the naive “Set Interval” method proposed in Chapter 4. Chapter 4 also proposes

a variety of heuristic informed methods that can change the spacing between key-points, allowing them to reduce the computational overhead of computing dynamics derivatives more significantly. Furthermore, Chapter 4 performs a thorough investigation of sparsely computing dynamics derivatives for a wide variety of tasks, two optimisation algorithms and two different methods of computing dynamics derivatives.

## 2.6 Remarks

This literature review has covered a wide variety of topics, including: classical robot motion planning, trajectory optimisation and its uses in contact-based planning, other methods for contact-based planning and how dimensionality reduction and approximations have been used to speed up robotic motion planning.

Even though work on physics-based manipulation has improved in recent years, there are still a variety of problems and research questions that need to be addressed. One such issue in trajectory optimisation is the limiting factor of computing dynamics derivatives through general purpose physics simulators. Several works have been discussed which have noted that computing dynamics derivatives is a computational bottleneck in gradient-based trajectory optimisation, limiting what tasks can be solved by these methods. The work presented in Chapter 4 aims to address this issue through the use of approximations.

Another limiting factor for trajectory optimisation is the dimensionality of the task. As dimensionality increases, it makes performing trajectory optimisation difficult due to the long optimisation times. This makes applying trajectory optimisation to situations like manipulation in clutter and also manipulation of deformable objects difficult. Whilst some works have been discussed that use dimensionality reduction to simplify high-dimensional spaces, these works often consider reducing the dimensionality offline and using the reduced order model online. Whilst this is effective for situations like locomotion, where the robot is self contained and there is a theoretic optimal reduced order model, this logic does not extend to manipulation in clutter. In manipulation in clutter scenarios, there is no one optimal reduced order model and it in fact depends on the distribution of objects at any particular instance. As such, Chapter 5 investigates a method of *online* dimensionality reduction.

This chapter has given a broad overview of the relevant literature surrounding this work. In the next chapter, additional technical details for a small subset of concepts that are necessary to understand the technical contributions of this thesis will be provided.

# Chapter 3

## Background

The purpose of this chapter is to give the interested reader the minimal necessary knowledge to understand the contributions of this thesis. Naturally, there will be some overlap of certain discussion points from Chapter 2, however this section will go into slightly more depth.

The topics that will be covered in this section will be: (1) The concept of an optimisation state vector and what this typically includes. (2) Enhanced details about specific trajectory optimisation algorithms used in this work, namely iLQR and SCVX. (3) Different methods of computing dynamics derivatives and their relevance to the work in this thesis.

### 3.1 The Optimisation State Vector

Trajectory optimisation aims to compute optimal motion plans for a system to achieve some desired goal. This process relies on two key concepts, the state vector  $\mathbf{x}$  and the control vector  $\mathbf{u}$ .

The control vector is a collection of control inputs that are applied to the actuated joints of a robot or multiple robots. Its dimensionality is equal to the total number of actuated DoFs in the system.

The optimisation state vector is slightly more complex. It describes the instantaneous state of the system by consisting of positional and velocity elements for all DoFs in a system. This includes:

- All the joints of any robots in the scene (for non-grounded robots, a free joint to represent the robots' pose in the world frame is also added).
- Contributions from all rigid bodies in a scene, with every rigid body contributing 6 DoFs  $(x, y, z, \phi, \beta, \psi)$ .
- Contributions from all deformable objects in a scene. These deformable objects are generally modelled as a set of particles all connected to their neighbours

by spring-damper models. Every particle contributes three DoFs to the state vector  $(x, y, z)$ .

Chapter 5 explores methods to reduce the number of DoFs that are considered in the optimisation state vector by focusing on task-relevant DoFs. This reduction in the size of the optimisation vector reduces the computational burden of trajectory optimisation.

## 3.2 Trajectory Optimisation (iLQR and SCVX)

Sec. 2.2 has already outlined the general trajectory optimisation problem along with a brief overview of both iLQR and SCVX. This section goes into more detail about these algorithms and how they relate to the contributions proposed in this thesis. To ensure this chapter is self-contained, the general trajectory optimisation problem will be re-introduced.

Consider a discrete-time dynamics system where the next state of the system is subject to the previous state and control vector:

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t), \quad (3.1)$$

where  $\mathbf{x}_t \in \mathbb{R}^{n_x}$  and  $\mathbf{u}_t \in \mathbb{R}^{n_u}$  are the state and control respectively vector at time-step  $t$ . Given a discrete-time trajectory  $(\mathbf{X}, \mathbf{U})$  of length  $T$ , where  $\mathbf{X} \triangleq (\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_T)$  and  $\mathbf{U} \triangleq (\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_{T-1})$ , we want to minimise the total running cost of a trajectory  $J$ :

$$J = l_f(\mathbf{x}_T) + \sum_{t=0}^{T-1} l(\mathbf{x}_t, \mathbf{u}_t), \quad (3.2)$$

where  $l(\mathbf{x}_t, \mathbf{u}_t)$  is some state and control vector dependent cost function and  $l_f(\mathbf{x}_T)$  is a terminal state dependent cost function.

Both trajectory optimisation algorithms considered in this work (iLQR and SCVX) require a first-order linearisation of the system dynamics about the previous trajectory. This first-order approximation is given by:

$$\delta \mathbf{x}_{t+1} = \mathbf{A}_t \delta \mathbf{x}_t + \mathbf{B}_t \delta \mathbf{u}_t, \quad (3.3)$$

where  $\mathbf{A}_t = \partial f(\mathbf{x}_t, \mathbf{u}_t) / \partial \mathbf{x}_t$  and  $\mathbf{B}_t = \partial f(\mathbf{x}_t, \mathbf{u}_t) / \partial \mathbf{u}_t$  are the first-order partial derivatives of the dynamics function with respect to the state and control vector respectively.

### 3.2.1 iLQR

At every iteration of iLQR [73, 126] the first order dynamics derivatives  $\mathbf{A}, \mathbf{B}$  and the first and second order cost derivatives ( $l_{\mathbf{x}}, l_{\mathbf{xx}}, l_{\mathbf{u}}, l_{\mathbf{uu}}$ ) are computed around the nominal trajectory. Please note that all these derivatives are specific to the time-step along the trajectory, (i.e.,  $\mathbf{A} \triangleq (\mathbf{A}_0, \mathbf{A}_1, \dots, \mathbf{A}_{T-1})$ ).

iLQR uses dynamic programming to reduce the minimisation over all controls in the trajectory to a sequence of individual minimisation problems. This is done using the concept of the *value* function. Consider the *cost-to-go*  $J_i$  as the cost of a trajectory starting at some time-step along the trajectory  $i$ :

$$J_i(\mathbf{x}_i, \mathbf{U}_i) = l_f(\mathbf{x}_T) + \sum_{t=i}^{T-1} l(\mathbf{x}_t, \mathbf{u}_t). \quad (3.4)$$

The value  $V$  at time-step  $i$  is the sequence of controls from time-step  $i$  that minimises the *cost-to-go*:

$$V(\mathbf{x}, i) = \arg \min_{\mathbf{U}_i} [J_i(\mathbf{x}_i, \mathbf{U}_i)]. \quad (3.5)$$

By setting the value function equal to the terminal state cost  $V(\mathbf{x}, T) = l_f(\mathbf{x}_T)$ , the dynamic programming principle can be used to compute optimal controls from the end of the trajectory to the beginning:

$$V(\mathbf{x}, i) = \arg \min_{\mathbf{u}} [l(\mathbf{x}, \mathbf{u}) + V(f(\mathbf{x}, \mathbf{u}), i + 1)]. \quad (3.6)$$

By defining the minimum of Eq. 3.6 as the perturbation of the pair  $(\mathbf{x}, \mathbf{u})$ :

$$\begin{aligned} Q(\delta \mathbf{x}, \delta \mathbf{u}) &= l(\mathbf{x} + \delta \mathbf{x}, \mathbf{u} + \delta \mathbf{u}, i) - l(\mathbf{x}, \mathbf{u}, i) \\ &\quad + V(f(\mathbf{x} + \delta \mathbf{x}, \mathbf{u} + \delta \mathbf{u}), i + 1) - V(f(\mathbf{x}, \mathbf{u}), i + 1). \end{aligned} \quad (3.7)$$

The perturbation in Eq. 3.7 will be referred to as the “Q-function”. The second order Taylor expansion coefficients of this Q-function can be shown as the following:

$$Q_{\mathbf{x}} = l_{\mathbf{x}} + (A)^{\top} V'_{\mathbf{x}}, \quad (3.8a)$$

$$Q_{\mathbf{u}} = l_{\mathbf{u}} + (B)^{\top} V'_{\mathbf{x}}, \quad (3.8b)$$

$$Q_{\mathbf{xx}} = l_{\mathbf{xx}} + (A)^{\top} V'_{\mathbf{xx}} A, \quad (3.8c)$$

$$Q_{\mathbf{uu}} = l_{\mathbf{uu}} + (B)^{\top} V'_{\mathbf{xx}} B, \quad (3.8d)$$

$$Q_{\mathbf{ux}} = l_{\mathbf{ux}} + (B)^{\top} V'_{\mathbf{xx}} A. \quad (3.8e)$$

These Q-function expansions can then be used to compute an optimal control policy at every time-step:

---

**Algorithm 1** Simplified iLQR algorithm
 

---

**Require:** Prediction horizon  $T$

Convergence criteria

Starting state  $\mathbf{x}_0$  and initial control sequence  $\mathbf{U}$

1: **while** not converged **do**

**Step 1: (Get Derivatives)**

2:     **for**  $t$  in  $(0, 1, \dots, T)$  **do**

3:         Calculate  $\mathbf{A}_t$   $\mathbf{B}_t$   $l_{\mathbf{x},t}$   $l_{\mathbf{xx},t}$   $l_{\mathbf{u},t}$   $l_{\mathbf{uu},t}$

**Step 2: (Backwards Pass)**

4:     **for**  $t$  in  $(T, \dots, 1, 0)$  **do**

5:         Calculate  $\mathbf{k}_t$  and  $\mathbf{K}_t$  solving Riccati-like equations

**Step 3: (Forwards Pass with line-search)**

6:     Update  $\mathbf{X}$  using the new control law  $\mathbf{k}_t$  and  $\mathbf{K}_t$

**return**  $\mathbf{U}$

---

$$\mathbf{k}_t = -Q_{\mathbf{uu}}^{-1}Q_{\mathbf{u}}, \quad (3.9a)$$

$$\mathbf{K}_t = -Q_{\mathbf{uu}}^{-1}Q_{\mathbf{ux}}. \quad (3.9b)$$

where  $\mathbf{k}_t$  is an open-loop control term and  $\mathbf{K}_t$  is the state-dependent closed-loop feedback term.

The value function is then updated:

$$V'_{\mathbf{x}} = Q_{\mathbf{x}} - Q_{\mathbf{u}}Q_{\mathbf{uu}}^{-1}Q_{\mathbf{ux}}, \quad (3.10a)$$

$$V'_{\mathbf{xx}} = Q_{\mathbf{xx}} - Q_{\mathbf{u}}Q_{\mathbf{uu}}^{-1}Q_{\mathbf{ux}}. \quad (3.10b)$$

The backwards pass stage of iLQR recursively computes Eqs. 3.8 - 3.10 starting at the end of the trajectory ( $t = T - 1$ ) to the beginning of the trajectory ( $t = 0$ ).

This optimal closed-loop control policy is then rolled-out using the original system dynamics within a line-search to find a lower cost trajectory:

$$\hat{\mathbf{x}}_0 = \bar{\mathbf{x}}_0, \quad (3.11a)$$

$$\hat{\mathbf{u}}_t = \bar{\mathbf{u}}_t + \alpha \mathbf{k}_t + \mathbf{K}_t(\hat{\mathbf{x}}_t - \bar{\mathbf{x}}_t), \quad (3.11b)$$

$$\hat{\mathbf{x}}_{t+1} = f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t), \quad (3.11c)$$

where  $(\hat{\mathbf{x}}, \hat{\mathbf{u}})$  represents the new rolled-out trajectory and  $(\bar{\mathbf{x}}, \bar{\mathbf{u}})$  represents the nominal trajectory which the derivatives were computed around. A simplified overview of iLQR can be seen in Alg. 1.

The majority of the computational expense of iLQR is in computing the first order dynamics derivatives, followed by performing the backwards pass. Computing

dynamics derivatives for the problems this thesis is concerned with (contact-based manipulation) requires the use of either finite-differencing or automatic differentiation (please see Sec. 3.3). Even though the computation of these dynamics derivatives can be parallelised, it still ends up being the computational bottleneck. This problem directly relates to the work outlined in Chapter 4, which is about reducing this computational expense through the use of approximations.

The backwards pass can also become computationally expensive as the size of the state and control vector grows. Since the backward pass involves matrix-matrix multiplication, its computational complexity scales cubically with the size of the state vector (i.e,  $O(n_x^3)$ ). Chapter 5 addresses this challenge by reducing the dimensionality of high-dimensional problems, making them computationally feasible for MPC.

### 3.2.2 SCVX

The SCvx (successive convexification) algorithm was originally proposed by Mao et al. [80]. This algorithm broadly works by linearising the non-convex constraints in a direct optimisation problem about the current nominal trajectory. This reduced problem can then be solved efficiently and the nominal trajectory can be updated. This process is repeated iteratively.

Önol et al. propose SCVX [96, 97], an adaptation of the original SCvx algorithm. At every iteration of SCVX, a QP sub problem is created by linearising the non-linear constraints. The primary constraint that is non-linear is dynamic feasibility. The dynamic feasibility constraint is linearised by computing the first order approximation of the dynamics derivatives ( $\mathbf{A}, \mathbf{B}$ ) about the current nominal trajectory ( $\bar{\mathbf{X}}, \bar{\mathbf{U}}$ ). The following Quadratic Program (QP) is solved at each SCVX iteration:

$$\min_{\delta \mathbf{X}, \delta \mathbf{U}} \quad l_f(\bar{\mathbf{x}}_T + \delta \mathbf{x}_T) + \sum_{t=0}^{T-1} l(\bar{\mathbf{x}}_t + \delta \mathbf{x}_t, \bar{\mathbf{u}}_t + \delta \mathbf{u}_t), \quad (3.12a)$$

$$\text{subject to: } \delta \mathbf{x}_{t+1} = \mathbf{A}_t \delta \mathbf{x}_t + \mathbf{B}_t \delta \mathbf{x}_t \text{ for } t = 1, \dots, T-1, \quad (3.12b)$$

$$\mathbf{x}_L \leq \bar{\mathbf{x}}_t + \delta \mathbf{x}_t \leq \mathbf{x}_U \text{ for } t = 1, \dots, T, \quad (3.12c)$$

$$\mathbf{u}_L \leq \bar{\mathbf{u}}_t + \delta \mathbf{u}_t \leq \mathbf{u}_U \text{ for } t = 1, \dots, T-1, \quad (3.12d)$$

$$\|\delta \mathbf{X}\|_1 + \|\delta \mathbf{U}\|_1 \leq r, \quad (3.12e)$$

solved subject to dynamics, box constraints, and an  $l_1$ -norm trust region constraint that bounds the deviation from the nominal trajectory. The state cost  $l(\cdot)$  and terminal state cost  $l_f(\cdot)$  both need to be quadratic functions to ensure that the subproblem remains a QP.

This problem is solved within a trust radius instead of the global optima of the QP, this is due to the fact that the global minima of the QP sub problem is not

necessarily a descent direction for the original problem due to the linearisation of the dynamics.

Interestingly, SCVX does not directly update the nominal state trajectory directly from the solution to the QP. The updated nominal control trajectory will be referred to as  $\mathbf{U}_{new} = \bar{\mathbf{U}} + \delta\mathbf{U}$  and the discarded updated state trajectory as  $\mathbf{X}_{lin} = \bar{\mathbf{X}} + \delta\mathbf{X}$ . SCVX applies the updated control trajectory and performs a rollout using the original non-linear dynamics:

$$\mathbf{x}_{new,t+1} = f(\mathbf{x}_{new,t}, \bar{\mathbf{u}}_t + \delta\mathbf{u}_t) \quad \text{for } t = 0, 1, \dots, T-1, \quad (3.13)$$

making this method a sort of hybrid between direct and shooting-based methods.

Finally, the size of the trust region is updated based on the accuracy of the reduction in cost between the linearised states  $\mathbf{X}_{lin}$  and non-linearised states  $\mathbf{X}_{new}$ . The cost of the current nominal trajectory is:

$$P = l_f(\bar{\mathbf{x}}_T) + \sum_{t=0}^{T-1} l(\bar{\mathbf{x}}_t, \bar{\mathbf{u}}_t). \quad (3.14)$$

The cost of the new linearised trajectory  $L$  and non-linearised trajectory  $C$  are then computed as follows:

$$C = l_f(\mathbf{x}_{new,T}) + \sum_{t=0}^{T-1} l(\mathbf{x}_{new,t}, \mathbf{u}_{new,t}), \quad (3.15a)$$

$$L = l_f(\mathbf{x}_{lin,T}) + \sum_{t=0}^{T-1} l(\mathbf{x}_{lin,t}, \mathbf{u}_{new,t}). \quad (3.15b)$$

The similarity between the cost improvements is then computed as  $\varphi$ :

$$\varphi = \frac{P - C}{P - L}. \quad (3.16)$$

Subject to the value of  $\varphi$ , the newly computed trajectory can then either be accepted or rejected and the trust region can be shrunk or expanded. These steps are then repeated until convergence is reached. A simplified overview of the SCVX algorithm can be seen in Alg. 2.

In terms of computational complexity, computing the dynamics derivatives of the nominal trajectory is also a computational bottleneck for the SCVX algorithm. However, this bottleneck is not as severe as in iLQR as solving the QP subproblems and performing a rollout with the non-linear dynamics also takes considerable time. As well as this, there are generally a number of optimisation iterations where the solution to the QP does not decrease the solution using the original non-linear dynamics. In such situations, the trust region is decreased and a new QP is solved. In these situations, since the nominal trajectory is not updated, no time is spent computing dynamics derivatives.



---

**Algorithm 2** Simplified SCVX algorithm

---

**Require:** Prediction horizon  $T$

Convergence criteria

Starting state  $\mathbf{x}_0$  and initial control sequence  $\mathbf{U}$

- 1: **while** not converged **do**
  - Step 1: (Get Derivatives)**
  - 2:   **for**  $t$  in  $(0, 1, \dots T)$  **do**
  - 3:     Calculate  $\mathbf{A}_t$   $\mathbf{B}_t$  about  $\bar{\mathbf{X}}, \bar{\mathbf{U}}$
  - Step 2: (Solve the Convex Subproblem)**
  - 4:    $\mathbf{U}_{new}, \mathbf{X}_{lin} = \text{SolveQP}(\mathbf{A}, \mathbf{B})$
  - Step 3: (Roll-out Non-linear Dynamics)**
  - 5:   **for**  $t = 0, 1, \dots T - 1$  **do**
  - 6:      $\mathbf{x}_{new.t+1} = f(\mathbf{x}_{new.t}, \mathbf{u}_{new.t})$
  - Step 4: Update Trajectory**
  - 7:    $P = l_f(\bar{\mathbf{x}}_T) + \sum l(\bar{\mathbf{x}}_t, \bar{\mathbf{u}}_t)$
  - 8:    $C = l_f(\mathbf{x}_{lin.T}, \mathbf{u}_{new.T}) + \sum l(\mathbf{x}_{lin.t}, \mathbf{u}_{new.t})$
  - 9:    $L = l_f(\mathbf{x}_{new.T}, \mathbf{u}_{new.T}) + \sum l(\mathbf{x}_{new.t}, \mathbf{u}_{new.t})$
  - 10:    $\varphi = (P - C)/(P - L)$
  - 11:   **if**  $\varphi > \text{threshold}$  **then**
  - 12:      $\bar{\mathbf{X}}, \bar{\mathbf{U}} = \mathbf{X}_{new}, \mathbf{U}_{new}$
  - 13:     Update trust region
  - 14:   **else**
  - 15:     Shrink trust region
- return**  $\mathbf{U}$

---

### 3.3 Differentiation Methods

Computing the first-order dynamics derivatives ( $\mathbf{A}, \mathbf{B}$ ) are an important aspect of the trajectory optimisation algorithms that have been discussed. Generally, there are four different methods for computing the derivatives of a function, these are:

- **Symbolic differentiation:** Keeping track of how variables in a function are updated symbolically and using these to evaluate the function derivatives.
- **Analytical differentiation:** This method relies on the user writing analytical expressions for the derivatives of their function manually.
- **Finite-differencing:** Finite-differencing can be used to estimate the derivatives of a function at a particular point by applying a small perturbation and observing how the value of the function is affected.
- **Automatic differentiation:** Similarly to symbolic differentiation, automatic differentiation can be used to keep track of the variables in a function as well as its derivatives numerically.

For the problems this thesis is concerned with (contact-based manipulation) the first two methods are infeasible. Symbolic differentiation becomes computationally intractable for moderately difficult problems as the lengths of the derivatives can explode combinatorially. Writing analytical derivatives for contact-based manipulation is infeasible, especially for situations where there is contact between multiple bodies in 3D space.

This leaves two practical methods for computing the derivatives that are required for trajectory optimisation.

#### 3.3.1 Finite-differencing

---

**Algorithm 3** Finite-differencing

---

**Require:** State vector  $\mathbf{x}_t$ ; Control vector  $\mathbf{u}_t$ ;

Column index  $i$

- 1:  $\mathbf{x}_{inc,t}[i] = \mathbf{x}_t[i] + \epsilon$
  - 2:  $\mathbf{x}_{dec,t}[i] = \mathbf{x}_t[i] - \epsilon$
  - 3:  $\mathbf{A}_t[:, i] = \frac{f(\mathbf{x}_{inc,t}, \mathbf{u}_t) - f(\mathbf{x}_{dec,t}, \mathbf{u}_t)}{2\epsilon}$
- 

Finite-differencing is a common method of approximating derivative values of a function by observing how small changes in input values affect the output of a function.

The first-order dynamics derivatives ( $\mathbf{A}, \mathbf{B}$ ) are matrices with size relative to the size of the state and control vector. A state vector of size  $n_x$  and a control vector of

size  $n_u$  will result in matrices  $\mathbf{A}, \mathbf{B}$  of size  $n_x \times n_x$  and  $n_x \times n_u$  respectively. Square brackets are used to refer to an individual element within these matrices; so  $\mathbf{A}[i, j]$  will represent an element at the  $i^{\text{th}}$  row and  $j^{\text{th}}$  column.  $\mathbf{A}[:, j]$  will refer to an entire column of this matrix at the  $j^{\text{th}}$  position.

When using central differences, every computation of a column in the  $\mathbf{A}$  and  $\mathbf{B}$  matrices requires two evaluations of the dynamics function. Given a specific state and control vector  $(\mathbf{x}_t, \mathbf{u}_t)$ , a small perturbation  $\epsilon$  is added/subtracted to an individual element of the state vector. These perturbed states are then used as the input to the dynamics function and the resulting states are saved. The difference between these states is compared and divided by the size of the perturbation to compute one column of the  $\mathbf{A}$  matrix. An example algorithm block that shows finite-differencing for an individual column in the  $\mathbf{A}$  matrix can be seen in Alg. 3. The computation for the  $\mathbf{B}$  matrix is equivalent, except that the control vector is perturbed instead.

When using a physics simulator, evaluating contact-interaction dynamics can be expensive, especially when contacts are plentiful, since solving for contacts often requires inner optimisation routines. Moreover, computing dynamics derivatives over a trajectory becomes prohibitively more expensive as the size of the system increases compared to simulating roll-outs of a trajectory. A single roll-out of a trajectory requires  $T$  computations of  $f(\mathbf{x}_t, \mathbf{u}_t)$ , whereas computing the first-order dynamics derivatives over the trajectory would require  $2n_x T + 2n_u T$  computations. The computational expense of computing dynamics derivatives can be lowered by parallelisation, however, this still becomes the significant time bottleneck in optimisation as the size of the state and control vector increases.

### 3.3.2 Automatic-differentiation

Automatic differentiation is a tool used to track the first or higher-order derivatives of programmatic functions automatically. There are two main approaches to automatic differentiation: source code transformation [85] or operator overloading [128].

Source code transformation takes a code base and converts it into a graph network of simple analytically differentiable expressions. This pre-computed graph can then be used to compute derivatives. Operator overloading is a more common approach used in certain physics simulators, such as Drake [128]. All operations simultaneously keep track of the primal values (values stored in variables) as well as their associated tangent values (derivatives with respect to other variables in the system). When any operation is performed, both the primal and tangent values are updated simultaneously.

# Chapter 4

## Speeding up Trajectory Optimisation Via Approximated Dynamics Derivatives

### Chapter Deliverables

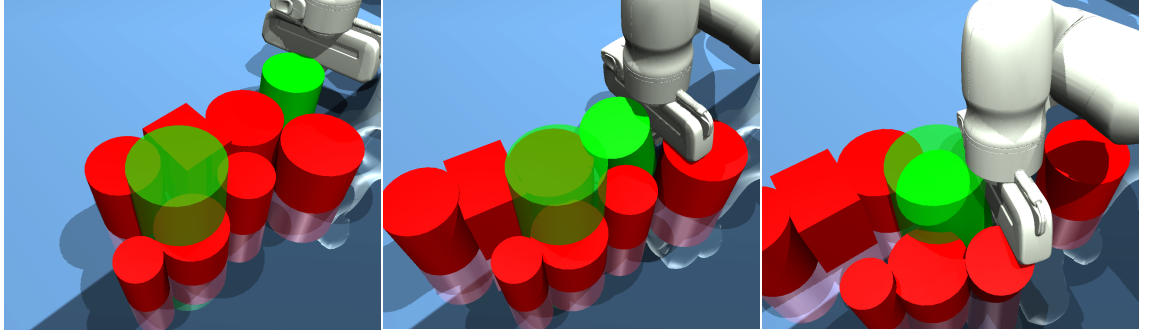
**Video:** <https://www.youtube.com/watch?v=rdRd2sgk8qY>

**Source Code:** <https://github.com/DMackRus/TrajOptKP>

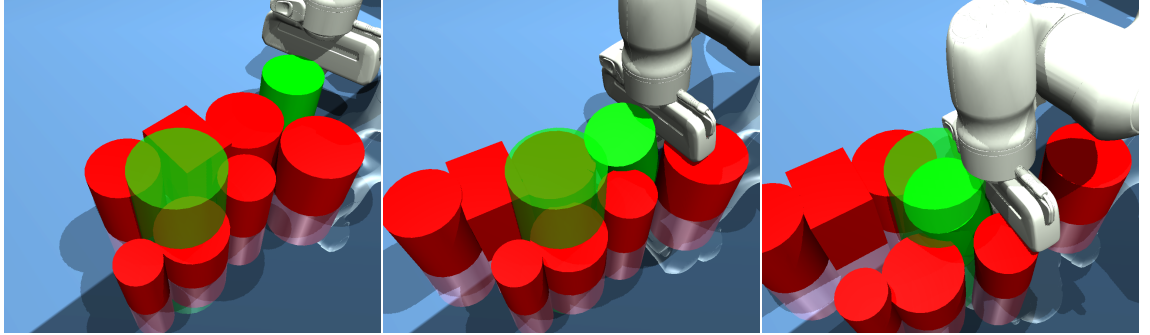
### 4.1 Introduction

This chapter is about how approximating the dynamics derivatives of a system can be used as a method of speeding up trajectory optimisation algorithms, particularly for contact-based trajectory optimisation. In trajectory optimisation, the derivatives of the dynamics are often required to compute optimal trajectories to solve a given task. The computation of these dynamics derivatives is often the computational bottleneck of these algorithms. This is particularly true for contact-based trajectory optimisation. Due to the complexity of contact-interactions, formulating analytical derivatives is typically infeasible for manipulation (especially when considering 3D manipulation amongst multiple objects). Therefore, methods such as FD [63, 96, 126] or AD [69] are often used to compute the dynamics derivatives in works that consider contact. As a result, a large portion of optimisation time is spent computing these dynamics derivatives.

The work in this chapter aims to address this issue, and make trajectory optimisation a more computationally tractable approach for solving contact-rich tasks. The core insight is fundamentally quite simple; between certain time-steps on a trajectory, dynamics derivatives do not change significantly, making them amenable to interpolation. This chapter proposes a general method for speeding up trajectory op-



Baseline - Optimisation time: 9.48s, Initial cost: 5.045, Final cost: 2.02



*TrajOptKP* - Optimisation time: 2.66s, Initial cost: 5.045, Final cost: 1.84

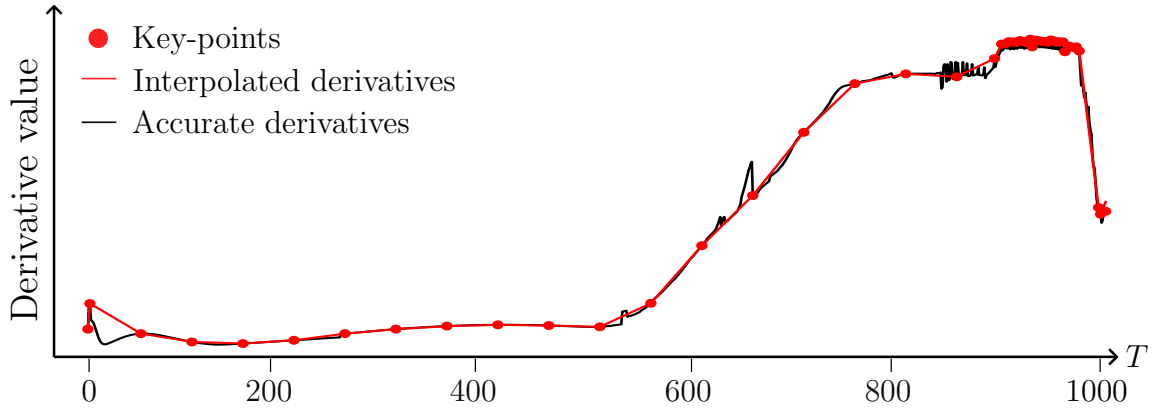


Figure 4.1: Two finalised trajectories for pushing a green cylinder to a target location (green silhouette) through clutter. The top row shows the baseline method taking 9.48 s of optimisation time, whereas the middle row shows a trajectory produced from one of the methods proposed in this chapter which only took 2.66 s of optimisation time, whilst achieving a similar solution. The bottom graph shows a derivative value (one of many) over the course of the trajectory along with its approximation. Through key-points, the number of derivative evaluations was reduced from 1000 to 48.

timisation which is named *TrajOptKP* (Trajectory Optimisation with Key-Points). The approach is to compute the dynamics derivatives using computationally expensive methods (FD or AD) only at certain time-steps along a trajectory, which are referred to as *key-points*. The dynamics derivatives computed at these key-points by AD or FD will be referred to as being computed *accurately*. In between these key-points, the dynamics derivatives are approximated using interpolation. The result of this, is the computation of a full (albeit partially approximated) set of dynamics derivatives, that were computed significantly faster than computing all the dynamics derivatives *accurately*.

Fig. 4.1 shows an example of a manipulation in clutter task (the most computationally expensive task in this chapter). The goal of this task is to push the green cylinder to a target location (green silhouette) whilst minimally disturbing some clutter (red objects). The top row of snapshots shows the final computed trajectory produced in the baseline case (where all dynamics derivatives are computed accurately over the entire trajectory). The middle row of Fig. 4.1 shows the final trajectory computed by the general *TrajOptKP* method. At the bottom of Fig. 4.1 is a sample dynamics derivative value in this task over the course of the trajectory as well as the key-points and approximated dynamic derivative values. The two sets of snapshots show similar final trajectories that both complete the task effectively, however, the *TrajOptKP* method significantly reduces the overall computation time from 9.66 s to 2.66 s. This reduction in optimisation time is due to the reduced number of expensive FD computations. The bottom row of Fig. 4.1 shows that the number of dynamics derivatives that were computed accurately in this example were reduced from 1000 to only 48.

The main challenge of the proposed approach is how to choose key-points intelligently, without directly knowing the shape of the underlying dynamics derivatives. This chapter investigates a variety of key-point selection methods. First, a naive method named *Set Interval* is proposed to serve as a baseline. Three more sophisticated key-point selection methods are also proposed, which are named *Adaptive Jerk*, *Velocity Change* and *Iterative Error*. Adaptive Jerk and Velocity Change both consider some dynamics information about the current nominal trajectory to decide where to place key-points. Iterative Error iteratively refines its dynamics derivative approximation until some error threshold is satisfied. It begins with a greedy approximation of the dynamics derivatives and performs evaluations on how accurate they are, refining its approximation as necessary.

This chapter evaluates the effects of using approximated dynamics derivatives on two different optimisation algorithms. The two optimisation algorithms are: a shooting method, named iLQR [73], and a sort of hybrid between a direct and shooting-based method named SCVX [96]. The performance of these optimisation algorithms is evaluated in terms of the reduction in optimisation time, as well as

the reduction of the cost (i.e., the quality of the final solution). The results indicate that, for both trajectory optimisation algorithms, there are significant time gains to be made without degrading the quality of the final solution significantly.

The performance of the key-point selection methods is evaluated on a variety of trajectory optimisation tasks, including existing contact-rich manipulation and locomotion tasks from Kurtz et al. [69] and Önel et al. [95]. Additionally, a variety of other tasks (Table 4.1) have been implemented, including under-actuated yet dynamic tasks like the acrobot, contact-based manipulation through clutter (similar to tasks from Kitaev et al. [63]), and contact-based locomotion of a 9-DoF walker.

The methods proposed in this chapter are particularly applicable to MPC approaches. When controlling real robotic hardware with MPC, optimal trajectories that were computed can quickly become obsolete due to the simulation gap between physics simulators and the real world. As such, it is more important to converge to an acceptable solution quickly, rather than an optimal solution slowly [50]. This chapter demonstrates MPC results in simulation for controlling a 9 DoF walker and a mini-cheetah to perform locomotion whilst remaining upright. The optimisation time gains are also demonstrated on a real-world task (manipulating objects in clutter using a Franka Panda manipulator).

### 4.1.1 Contributions

This chapter:

- Proposes a general framework to reduce the computational burden in trajectory optimisation that comes from expensively computing dynamics derivatives. The general framework only computes dynamics derivatives *accurately* at *key-points* and approximates the remainder;
- Proposes three specific “intelligent” key-point selection methods, namely *Adaptive Jerk*, *Velocity Change* and *Iterative Error*;
- Presents a formal proof that one of the proposed key-point methods (*Velocity Change*) can bound the approximation error subject to its parametrisation for the single pendulum dynamics;
- Performs an empirical case study regarding a toy contact-based optimisation task and shows the importance of modelling contact discontinuities;
- Integrates and evaluates the performance of the proposed methods on two separate optimisation algorithms, namely iLQR and SCVX;
- Investigates the effects of using different methods to compute the dynamics derivatives (FD and AD) and

- Evaluates the performance of the proposed key-point selection methods on ten tasks in simulation as well as one on real hardware. These tasks include dynamic motion, contact-based manipulation in clutter and locomotion.

### 4.1.2 Organisation

The structure of this chapter is as follows: Sec. 4.2 specifies the general problem formulation for this chapter and then Sec. 4.3 outlines the proposed key-point selection methods. A small mathematical proof is provided in Sec. 4.4 that shows how one of the proposed key-point selection methods can bound the approximation error between the accurate derivatives and the approximated derivatives. Sec. 4.5 describes and gives the mathematical details for all the tasks that are evaluated in this work, as well as introduces abbreviations for the optimisers that are used. Then, Sec. 4.6 presents a variety of results: relating to approximation accuracy, how the proposed key-point selection methods work around contact dynamics, general results for long- and short-horizon optimisation and finally, some results performed on real robotic hardware. The results are then discussed in Sec. 4.7, and finally, this chapter is concluded in Sec. 4.8.

## 4.2 Problem formulation

Chapter 3 defined the first-order linearisation of the system dynamics matrices as  $\mathbf{A}$  and  $\mathbf{B}$ , recall that  $\mathbf{A}_t = \frac{\partial f(\mathbf{x}_t, \mathbf{u}_t)}{\partial \mathbf{x}_t}$  and  $\mathbf{B}_t = \frac{\partial f(\mathbf{x}_t, \mathbf{u}_t)}{\partial \mathbf{u}_t}$ . Computing these dynamics derivatives over an entire trajectory is computationally expensive, especially as dimensionality grows and multi-object contacts are plentiful. The methods proposed in this chapter consider only performing these expensive computations at certain time-steps over a trajectory, which are referred to as key-points. The remainder of the derivatives are then approximated via linear interpolation. The dynamics derivatives that are computed at key-points using FD or AD will be referred to as the *accurate* derivatives and denoted by  $\hat{\mathbf{A}}_t, \hat{\mathbf{B}}_t$ .

An important aspect of this chapter is how key-points are considered. In the earlier version of this work [113], the key-points were considered as a single list for the entire system. For example, imagine the optimisation horizon  $T$  was 10 time-steps long and the key-points over the trajectory were located at  $t = [0, 3, 7, 10]$ . In the earlier versions of this work, the method was to compute the *full*  $\mathbf{A}$  and  $\mathbf{B}$  matrices at these time-steps, i.e  $[\hat{\mathbf{A}}_0, \hat{\mathbf{A}}_3, \hat{\mathbf{A}}_7, \hat{\mathbf{A}}_{10}]$  and  $[\hat{\mathbf{B}}_0, \hat{\mathbf{B}}_3, \hat{\mathbf{B}}_7, \hat{\mathbf{B}}_{10}]$ . The remainder of the dynamics derivatives would then be computed via linear interpolation.

This concept can be taken further. When using finite-differencing, the computation of dynamics derivatives is done *column-by-column*. For instance, focusing on the  $\mathbf{A}$  matrices, every column of this matrix is computed by applying a perturbation to an individual element (a positional or velocity component of an individual DoF)



in the the state vector and then observing how this perturbation affects the general integration of the system dynamics, i.e.  $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$ . Perturbing an individual element in the state vector will result in the computation of the respective column in the  $\mathbf{A}$  matrix, i.e. perturbing the first element in the state vector  $\mathbf{x}[0]$ , would yield:

$$\mathbf{A}_t = \begin{bmatrix} \frac{\delta f[0]}{\delta \mathbf{x}[0]} & \cdot & \cdots & \cdot \\ \frac{\delta f[1]}{\delta \mathbf{x}[0]} & \cdot & \cdots & \cdot \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\delta f[n_x-1]}{\delta \mathbf{x}[0]} & \cdot & \cdots & \cdot \end{bmatrix} \quad (4.1)$$

where  $f[0], f[1], \dots, f[n_x]$  represents the output an individual component in the state vector of the function  $f(\mathbf{x}_t, \mathbf{u}_t)$ . This concept is the same for  $\mathbf{B}$  matrices, except the control vector is perturbed instead.

Therefore, it is more computationally efficient to have individual specific lists of key-points for every DoF in the system. This is because, for any segment over the optimisation horizon, one DoF might have quite complex dynamics derivatives that are changing rapidly over time whilst another DoF may have fairly static dynamics derivatives<sup>1</sup>.

Chapter 3 discussed two gradient-based optimisation algorithms (iLQR [73] and SCVX [95]) which both require first-order dynamics derivatives over the trajectory. Simplified algorithm blocks for these two optimisation algorithms can be seen in Chapter 3. The high-level idea of the proposed method is to re-implement **Step 1: Get Derivatives** of these algorithms using Alg. 4. The algorithm works as follows: firstly, a set of key-points for every DoF are computed using the SELECTKEYPOINTS function. Once a list of key-points has been computed, the specific columns inside the  $\mathbf{A}$  and  $\mathbf{B}$  matrix that relate to that DoF at the specified time-step are computed *accurately*; For DoF  $i$ , this refers to columns  $\mathbf{A}[:, i]$ ,  $\mathbf{A}[:, i + \frac{n_x}{2}]$  and  $\mathbf{B}[:, i]$ <sup>2</sup>. These columns of derivatives are then interpolated as specified by the list of key-points for that DoF. This work only considers linear interpolation methods. Whilst higher-order methods could potentially be useful depending on the task, it was empirically determined that the additional computational expense in using higher-order interpolation methods was not worth any potential accuracy improvements.

<sup>1</sup>Consider the case of manipulation in clutter, when contact is made or broken between two objects, this results in an often discontinuous change in the dynamics derivatives of those objects, whereas other objects that are not in contact with anything and are not moving will have no change in their dynamics derivatives.

<sup>2</sup>There may not be a corresponding  $\mathbf{B}$  component for every  $i$ , as the systems of interest are typically under-actuated.

## 4.3 Key-point Selection Methods

This section proposes a variety of key-point selection methods to implement the function `SELECTKEYPOINTS` from Alg. 4. The general aim of these methods are to compute a sufficient set of key-points (to reduce computational load) whilst still accurately approximating the underlying accurate dynamics derivatives.

Four different key-point methods are introduced; *Set Interval*, *Velocity Change*, *Adaptive Jerk*, and *Iterative Error*.

### The Set Interval method

This method is the simplest key-point method to implement. It equally spaces key-points over the trajectory and does not consider information about the nominal trajectory. The only parameter for this method is  $N$ , the number of time steps between consecutive key-points. This method implements the function `SELECTKEYPOINTS` from Alg. 4 as:  $[0, N, 2N, \dots, T] \leftarrow \text{SELECTKEYPOINTS}$ .

### The Velocity Change method

The parameters for this method are:

- $N_{min}$ : Minimum number of time-steps between key-points.
- $N_{max}$ : Maximum number of time-steps between key-points.
- $\Delta \bar{\mathbf{v}}$ : Array of velocity change threshold values for each DoF in the system.

This Velocity Change method monitors the velocity profile of all DoFs in the system over the trajectory. It enforces that key-points must be within  $N_{min}$  and  $N_{max}$  of one another subject to the velocity profile. This method monitors the absolute sum of velocity change since its last key-point, if this absolute change is greater than  $\Delta \bar{\mathbf{v}}$ , then a new key-point will be placed. This method also detects when the velocity of a DoF changes direction and places a key-point in these situations also. This algorithm implements the `SELECTKEYPOINTS` function from Alg. 4 as Alg. 5. An illustrative example of this method can be seen in Fig. 4.2.

---

#### Algorithm 4 TrajOptKP implementation of GET DERIVATIVES

---

```

1: for  $i$  in  $(1, 2, \dots, n_x/2)$  do
2:    $kp \leftarrow \text{SELECTKEYPOINTS}(i)$ 
3:   for  $j$  in  $(1, 2, \dots, \text{len}(kp))$  do
4:      $\hat{\mathbf{A}}_{kp[j]}[:, i], \hat{\mathbf{A}}_{kp[j]}[:, i + \frac{n_x}{2}],$ 
        $\hat{\mathbf{B}}_{kp[j]}[:, i] = \text{COMPUTEDERIVS}()$ 
5:  $\mathbf{A}, \mathbf{B} = \text{INTERPOLATE}(\hat{\mathbf{A}}, \hat{\mathbf{B}} kp)$ 

```

---

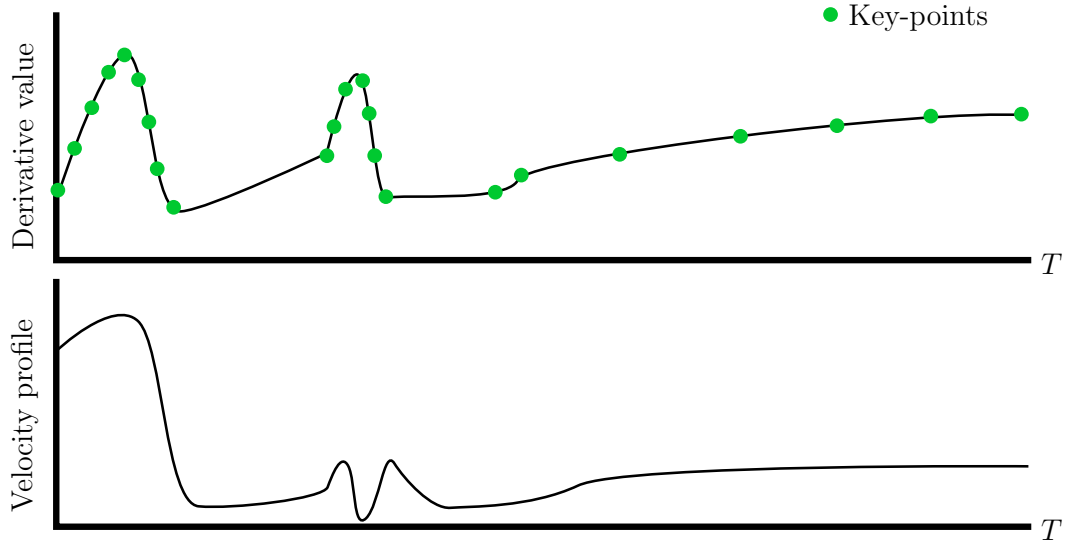


Figure 4.2: Illustrative one-dimensional example of the Velocity Change method. The top plot shows the underlying true derivative value (black curve) with a set of key-points (green points) placed as determined by the Velocity change method. The bottom curve shows the velocity profile over the trajectory for some DoF. When velocities are large (at the start of the trajectory), or the velocity changes direction often (near the middle of the trajectory), key-points are placed more densely, and when the velocity is small or more constant, key-points are placed more sparsely.

---

**Algorithm 5** Velocity Change implementation of SELECTKEYPOINTS
 

---

**Require:**  $N_{min}$ ,  $N_{max}$ ,  $\Delta\bar{v}$  // keypoint parameters

DoF index  $i$ , Nominal trajectory  $\bar{\mathbf{X}}$

```

1:  $v_{sum} = 0$ 
2: for  $t$  in  $(0, 1, \dots, T)$  do
3:    $interval = \text{INTERVALSINCELASTKEYPOINT}()$ 
4:    $v_{sum} = v_{sum} + |\bar{\mathbf{x}}_t[i + n_x/2]|$  // accumulate velocity
5:   if  $interval > N_{min}$  then
6:     if  $\bar{\mathbf{x}}_t[i + n_x/2] \times \bar{\mathbf{x}}_{t-1}[i + n_x/2] < 0$  then //velocity direction changes
7:        $keypoints[i].append(t)$ 
8:        $v_{sum} = 0$ 
9:     else if  $v_{sum} > \Delta\bar{v}$  then
10:       $keypoints[i].append(t)$ 
11:       $v_{sum} = 0$ 
12:   if  $interval > N_{max}$  then
13:      $keypoints[i].append(t)$ 
14:      $v_{sum} = 0$ 
15: return  $keypoints[i]$ 
    
```

---

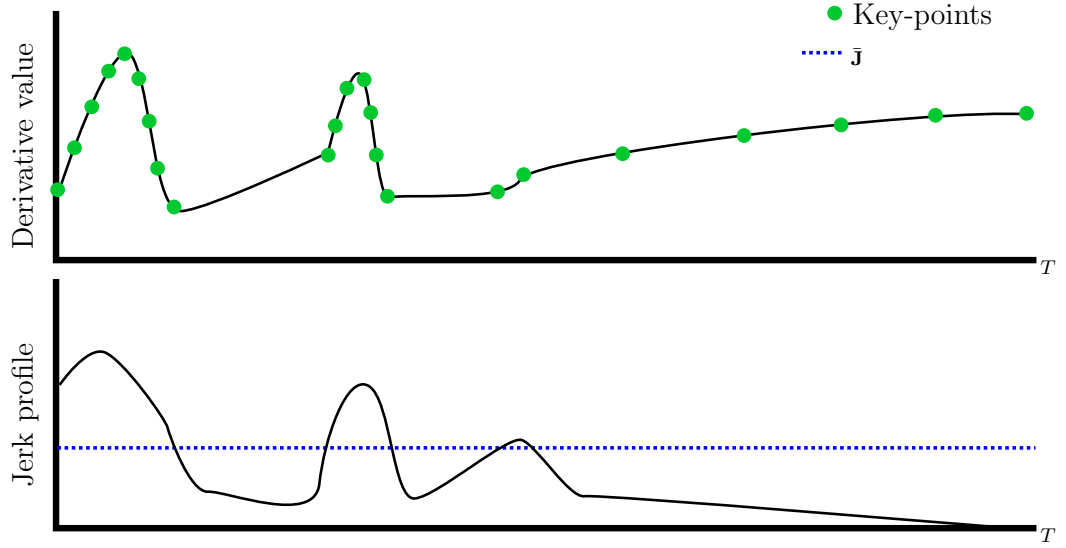


Figure 4.3: Illustrative one-dimensional example of the Adaptive Jerk method. The top plot shows the underlying true derivative value (black curve) with a set of key-points (green points) placed as determined by the Adaptive Jerk method. The bottom curve shows the jerk profile over the trajectory for some DoF. When the jerk exceeds the jerk threshold ( $\bar{J}$ ), key-points are placed more densely, and when jerk is below the threshold, key-points are placed more sparsely.

### The Adaptive Jerk method

The parameters for this method are:

- $N_{min}$ : Minimum number of time-steps between key-points.
- $N_{max}$ : Maximum number of time-steps between key-points.
- $\bar{J}$ : Array of jerk threshold values for each DoF in the system.

The Adaptive Jerk method changes the interval size between key-points for each DoF dependent on the magnitude of jerk (derivative of acceleration with respect to time). When DoFs have high levels of jerk, it translates to periods in the trajectory where the dynamics derivatives are changing rapidly. Due to the impulse effect of detecting jerk on different DoFs, it enables this algorithm to detect the making and breaking of contact, which has a noticeable impact on the dynamics derivatives.

This method places key-points from the start of the trajectory to the end by enforcing that key-points are placed within  $N_{min}$  and  $N_{max}$  of one another depending on the jerk of that DoF in the trajectory. This algorithm implements the `SELECTKEYPOINTS` function from Alg. 4 as Alg. 6. An illustrative example of this method can be seen in Fig. 4.3. The function `COMPUTEJERKPROFILE()` estimates the jerk profile for a given DoF over the trajectory horizon by applying backward

---

**Algorithm 6** Adaptive Jerk implementation of SELECTKEYPOINTS

---

**Require:**  $N_{min}$ ,  $N_{max}$ ,  $\bar{\mathbf{J}}$  // keypoint parameters

DoF index  $i$ , Nominal trajectory  $\bar{\mathbf{X}}$

```

1:  $jerk = \text{COMPUTEJERKPROFILE}(\bar{\mathbf{X}})$ 
2: for  $t$  in  $(0, 1, \dots, T)$  do
3:    $interval = \text{INTERVALSINCELASTKEYPOINT}()$ 
4:   if  $interval > N_{min}$  then
5:     if  $jerk[i][t] > \bar{\mathbf{J}}[i]$  then
6:        $keypoints[i].append(t)$ 
7:   else if  $interval > N_{max}$  then
8:      $keypoints[i].append(t)$ 
9: return  $keypoints[i]$ 
```

---

Euler finite differences to the velocity profile twice — first to obtain acceleration, and then again to obtain jerk. As this method relies on discrete differentiation, it can be sensitive to noise. Anecdotally, noisy estimates of the jerk did not seem to affect the performance of this key-point method.

### The Iterative Error method

The parameters for this method are:

- $N_{min}$ : Minimum number of time-steps between key-points.
- $\bar{\epsilon}$ : Error threshold to determine when a line fitting approximation is accurate enough or whether the algorithm needs to insert additional key-points for a more accurate approximation.

The Iterative Error method is significantly different from Adaptive Jerk and Velocity Change methods. Iterative Error begins with a greedy (and probably poor) approximation of the dynamics derivatives. It then performs an error check between its approximation and the accurate derivatives (i.e., as determined by FD or AD) and determines whether it needs to refine its approximation by inserting additional key-points. An illustrative example of this method can be seen in Fig. 4.4.

This algorithm implements the SELECTKEYPOINTS function from Alg. 4 as Alg. 7. Notably, the function APPROXIMATIONERROR can be implemented in a variety of ways. The implementation in this chapter simply checks the mid-point between every pair of key-points and compares the linear approximation against the accurate derivatives. If the error is below the error threshold ( $\bar{\epsilon}$ ) then the approximation was acceptable and that segment of the trajectory is finished. If the error is above  $\bar{\epsilon}$ , the algorithm would further subdivide that section and repeat the process. The Iterative Error algorithm is an example of a “divide-and-conquer” strategy [28].

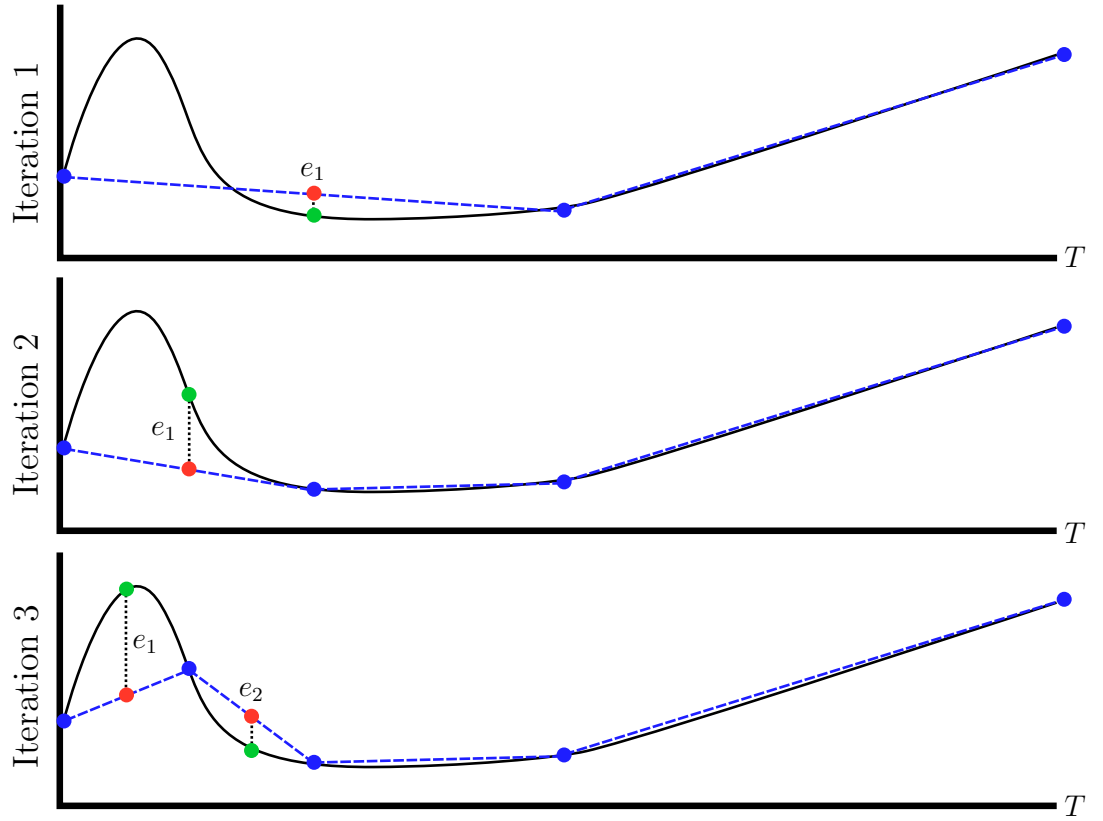


Figure 4.4: Illustrative one-dimensional example of the Iterative Error method. The black curve is the true underlying derivative, the blue dashed line is the current linear approximation. Three iterations of the approximation being refined are shown. The top image is the first iteration where it selects only three key-points (blue). Halfway between the key-points, the approximate derivative value (red) is compared against the accurate derivative (green). If the error between these values is above a threshold, the segment is subdivided. This process happens recursively until all segments are complete.

---

**Algorithm 7** Iterative Error implementation of SELECTKEYPOINTS
 

---

**Require:**  $N_{min}, \bar{\tau}$  // keypoint parameters

DoF index  $i$

```

1:  $stack = [(0, T)]$ 
2: while  $stack$  not empty do
3:    $indices = stack.pop()$ 
4:    $interval = \text{INTERVALBETWEENINDICES}(indices)$ 
5:   if  $interval < N_{min}$  then
6:      $keypoints[i].append(indices)$ 
7:   else
8:      $error = \text{APPROXIMATIONERROR}(indices)$ 
9:     if  $error > \bar{\tau}$  then
10:       $newIndices = \text{SUBDIVIDE}(indices)$ 
11:       $stack.push(newIndices)$ 
12:   else
13:      $keypoints[i].append(indices)$ 
14: return  $keypoints[i]$ 
    
```

---

## 4.4 Approximation Error Bound for the Pendulum System

This section performs a short proof that shows how one of the proposed key-point selection methods, the Velocity Change method, can bound the approximation error between the true dynamics derivatives and the linear approximations. Due to the fact that this proof needs explicit dynamics derivatives, a specific dynamic system is considered, as such, this proof is performed for the single pendulum. Fig. 4.5 shows the free body diagram for the single pendulum. The equation of motion for

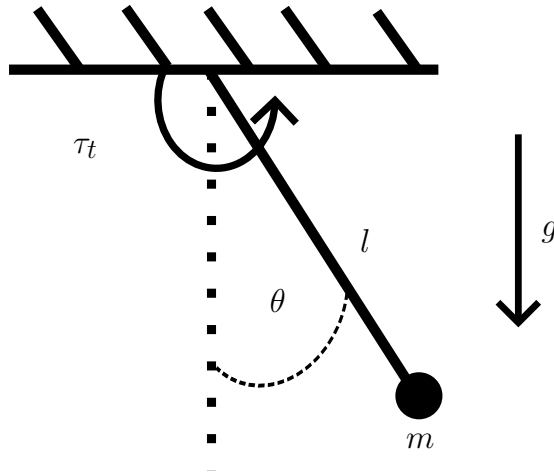


Figure 4.5: Free body diagram of the pendulum.

this system is:

$$ml^2\ddot{\theta}_t + mgl\sin(\theta_t) = -b\dot{\theta}_t + \tau_t, \quad (4.2)$$

where  $m$  is the pendulum mass,  $l$  is the pendulum length,  $\theta_t$  is the angle of the pendulum from its resting position at time-step  $t$ ,  $b$  is some damping coefficient,  $g$  is gravity and finally  $\tau_t$  is a control torque applied at time-step  $t$ .

By rearranging Eq. 4.2 in terms of acceleration, and assuming discrete-time Euler integration of the system dynamics, how the state of the pendulum system updates with time can be written as:

$$\theta_{t+1} = \theta_t + \dot{\theta}_t \Delta t, \quad (4.3)$$

$$\dot{\theta}_{t+1} = \dot{\theta}_t + \left( \frac{\tau_t}{ml^2} - \frac{b\dot{\theta}_t}{ml^2} - \frac{g\sin\theta_t}{l} \right) \Delta t, \quad (4.4)$$

where  $\Delta t$  is the system integration time-step. Differentiating Eq. 4.3 and Eq. 4.4 with respect to the state and control vector yields the first order linearisation of the system dynamics:

$$\mathbf{A}_t = \begin{bmatrix} 1 & \Delta t \\ \frac{-g\cos(\theta_t)\Delta t}{l} & 1 - \frac{b}{ml^2}\Delta t \end{bmatrix}, \quad (4.5)$$

$$\mathbf{B}_t = \begin{bmatrix} 0 \\ \frac{1}{ml^2}\Delta t \end{bmatrix}. \quad (4.6)$$

The aim of this proof is to show that the Velocity Change method can bound the approximation error of  $\mathbf{A}_t$  and  $\mathbf{B}_t$ . Only one term in  $\mathbf{A}_t$  and  $\mathbf{B}_t$  changes with respect to the state or control vector. This term is  $\frac{\delta\dot{\theta}_{t+1}}{\delta\theta_t} = \frac{-g\cos(\theta_t)\Delta t}{l}$ . The remainder of this proof focuses on this term, as the other elements of  $\mathbf{A}_t$  and  $\mathbf{B}_t$  are constants.

During optimisation, there is some nominal trajectory  $(\bar{\mathbf{X}}, \bar{\mathbf{U}})$ . Without loss of generality, assume there are two placed key-points, such that the first key-point is at  $t = 0$  and the second is at  $t = n$ . The placement of the second key-point will be subject to the parametrisation of the Velocity Change key-point selection method. The Velocity Change method (Alg. 5) has three parameters:  $N_{min}$  (minimum interval between key-points);  $N_{max}$  (maximum interval between key-points); and  $\Delta\bar{\mathbf{v}}$  (maximum absolute velocity gain between key-points). The algorithm enforces the following constraints:

$$\sum_{m=0}^n |\dot{\theta}_m| \leq \Delta\bar{\mathbf{v}}, \quad (4.7a)$$

$$N_{min} \leq n \leq N_{max}. \quad (4.7b)$$

At any time-index  $i$  between the two key-points, the key-point method's approximated value will be a linear interpolation between the true values at the two key-points:

$$\frac{i}{n} \left( -\frac{g\cos(\theta_n)\Delta t}{l} + \frac{g\cos(\theta_0)\Delta t}{l} \right) - \frac{g\cos(\theta_0)\Delta t}{l}. \quad (4.8)$$

Therefore, the absolute approximation error at time-step  $i$  can be written using the difference between Eq. 4.8 and the true value  $\frac{-g\cos(\theta_i)\Delta t}{l}$ :



$$Error_i = \frac{g\Delta t}{l} \left| -\cos(\theta_i) + \frac{i}{n} \cos(\theta_n) - \frac{i}{n} \cos(\theta_0) + \cos(\theta_0) \right|. \quad (4.9)$$

As the velocity profile of the trajectory is known and using Euler integration of the system dynamics, the general expression for any joint angle in the system is:

$$\theta_a = \theta_0 + \sum_{m=0}^a \dot{\theta}_m \Delta t. \quad (4.10)$$

Substituting Eq. 4.10 into Eq. 4.9 yields;

$$Error_i = \frac{g\Delta t}{l} \left| -\cos \left( \theta_0 + \sum_{m=0}^i \dot{\theta}_m \Delta t \right) + \frac{i}{n} \cos \left( \theta_0 + \sum_{m=0}^n \dot{\theta}_m \Delta t \right) - \frac{i}{n} \cos(\theta_0) + \cos(\theta_0) \right|. \quad (4.11)$$

The goal is to bound the total approximation error between the true dynamics derivatives and the linear approximation:

$$TotalError = \sum_{i=0}^n Error_i. \quad (4.12)$$

The following terms are introduced as simplifications:

$$C_i = \cos(\theta_0 + \sum_{m=0}^i \dot{\theta}_m \Delta t), \quad (4.13a)$$

$$C_n = \cos(\theta_0 + \sum_{m=0}^n \dot{\theta}_m \Delta t), \quad (4.13b)$$

$$C_0 = \cos(\theta_0). \quad (4.13c)$$

Therefore, the total error equation can be written as:

$$TotalError = \frac{g\Delta t}{l} \sum_{i=0}^n \left| -C_i + \frac{i}{n} C_n - \frac{i}{n} C_0 + C_0 \right|. \quad (4.14)$$

The following identity is true for any real values of  $x, y$ :

$$|\cos(x) - \cos(y)| \leq |x - y|. \quad (4.15)$$

Eq. 4.15 can be derived from the Mean Value Theorem. For the function  $\cos(x)$ , which has the derivative  $-\sin(x)$ , the Mean Value Theorem states that

$$\cos(x) - \cos(y) = -\sin(c)(x - y), \quad (4.16)$$

for some value  $c$  between  $x$  and  $y$ . Taking the absolute value of both sides and using  $\sin(c) \leq 1$ , Eq. 4.15 is achieved.

Using Eq. 4.15, the following expression can be written:

$$|C_0 - C_i| \leq \left| \sum_{m=0}^i \dot{\theta}_m \Delta t \right|. \quad (4.17)$$

Using the triangle inequality:

$$\Delta t \left| \sum_{m=0}^i \dot{\theta}_m \right| \leq \Delta t \sum_{m=0}^i |\dot{\theta}_m|. \quad (4.18)$$

Hence, the difference between  $C_0$  and  $C_i$  can be bounded:

$$|C_0 - C_i| \leq \Delta t \sum_{m=0}^i |\dot{\theta}_m| \leq \Delta t \Delta \bar{v}, \quad (4.19)$$

where Eq. 4.7a is used.

Similarly, it can be shown that:

$$|C_n - C_0| \leq \Delta t \Delta \bar{v}. \quad (4.20)$$

Each term can be rewritten in the summation in Eq. 4.14 as:

$$|C_0 - C_i + \frac{i}{n}(C_n - C_0)| \leq |C_0 - C_i| + \left| \frac{i}{n}(C_n - C_0) \right|. \quad (4.21)$$

Then, using Eqs. 4.19 and 4.20, the total error can be bounded by the velocity change threshold and the length of the interval between key-points:

$$TotalError \leq \frac{g\Delta t}{l} \sum_{i=0}^n \Delta t (\Delta \bar{v} + \frac{i}{n} \Delta \bar{v}), \quad (4.22)$$

$$TotalError \leq \frac{g(\Delta t)^2}{l} (\Delta \bar{v} n + \frac{(n+1)}{2} \Delta \bar{v}). \quad (4.23)$$

Eq. 4.23 clearly shows that the total error is bounded, linearly by  $\Delta \bar{v}$  and linearly by  $n$ . Both of these are parameters that can be controlled by the Velocity change key-point method.  $\Delta \bar{v}$  can be adjusted as desired to linearly reduce the error and  $n$  has to be between the parameters  $N_{min}$  and  $N_{max}$ .

## 4.5 Task Specifications and Testing Setup

This chapter evaluates the proposed key-point methods on a wide variety of tasks, ranging from dynamic locomotion, full body robotic manipulation and manipulation in clutter. This section outlines the objective of each task and the specific costs used in optimisation, short hand notation for three different optimisation implementations and also the specific key-point parametrisations that were used to generate results in Sec. 4.6.

All experiments were performed on a 16 virtual core CPU (11h Gen Intel(R) Core(TM) i9-11900@2.50GHz) with 128 GB RAM.

### 4.5.1 Optimisers

Three different optimisers were used to perform simulation experiments, which were:

- **iLQR-FD**: An implementation of iLQR in C++ using MuJoCo as the physics simulator. Derivative computation is performed using FD and is parallelised.
- **iLQR-AD**: An implementation provided by Kurtz and Lin [69] which is augmented with the proposed key-point methods. Written in Python using Drake as the physics simulator. Derivative computation is performed using AD and is *not* parallelised.<sup>1</sup>
- **SCVX**: An implementation of SCVX provided by Önl et al. [95] which is augmented with the proposed key-point methods. Written in C++ using MuJoCo as the physics simulator. Derivative computation is performed using FD and is *not* parallelised.

For instances where a specific key-point algorithm for an optimiser is discussed, square brackets will be used to denote this. For example **iLQR-FD [Adaptive Jerk]** refers to the implementation of iLQR using FD for dynamics derivative computation, using the Adaptive Jerk key-point method. When the method in square brackets is **Baseline**, this means that all the dynamics derivatives are computed and no interpolation is used. Finally, throughout the results section, when a specific Set Interval method is referred to, the abbreviation *SIN* is used, where  $N$  is the size of the interval.

### 4.5.2 Tasks

Table 4.1 presents the tasks used in this chapter. In this table, the left column shows an example image of the task underneath the task name. The right column gives a brief description of the task and also shows the specific costs that were used during optimisation. For brevity, some cost function abbreviations will be denoted here. The cost function for all tasks can be written as:

$$l(\mathbf{x}_t, \mathbf{u}_t) = (\mathbf{x}_t - \tilde{\mathbf{x}}_t) \mathbf{W} (\mathbf{x}_t - \tilde{\mathbf{x}}_t)^T + \mathbf{u}_t \mathbf{R} \mathbf{u}_t^T, \quad (4.24)$$

as well as some final cost function:

$$l_f(\mathbf{x}_T) = (\mathbf{x}_T - \tilde{\mathbf{x}}_T) \mathbf{W}_f (\mathbf{x}_T - \tilde{\mathbf{x}}_T)^T, \quad (4.25)$$

where  $\mathbf{R}$  is a positive semi-definite matrix that penalises control costs and  $\mathbf{W}$  and  $\mathbf{W}_f$  are similarly positive semi-definite matrices that penalise the difference from the

<sup>1</sup>Key-points were used for the whole system in this implementation, rather than using a unique set of key-points per DoF. This is due to the fact that there was **no** time save for only computing certain columns of the dynamics derivatives matrices when using Drake's AD.


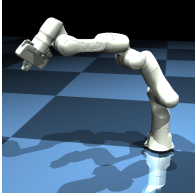
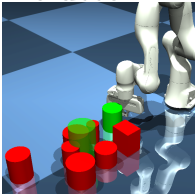
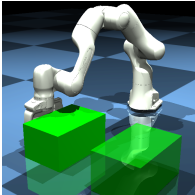

desired state  $\tilde{\mathbf{x}}_t$ , with  $\mathbf{W}_f$  only being used for the final state in the trajectory. The costs inside the  $\mathbf{W}$  matrix used for each task are separated into multiple segments that refer to different aspects of the state vector, such as position and velocity components for robots and bodies, as shown in the list below:

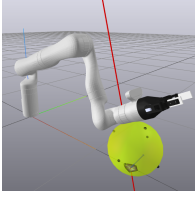
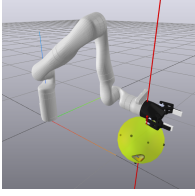
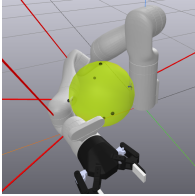
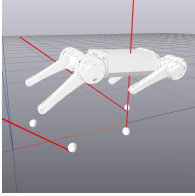

- $\mathbf{W}_{pr}$  : Positional cost of a robot joint.
- $\mathbf{W}_{vr}$  : Velocity cost of a robot joint.
- $\mathbf{W}_{pb}$  : Positional cost of an un-actuated body.
- $\mathbf{W}_{vb}$  : Velocity cost of an un-actuated body.
- $\mathbf{W}_{p\theta b}$  : Orientation cost of an un-actuated body.
- $\mathbf{W}_{pbg}$  : Positional cost of manipuland.
- $\mathbf{W}_{vbg}$  : Velocity cost of manipuland.
- $\mathbf{W}_{pbj}$  : Positional cost of clutter objects.
- $\mathbf{W}_{vbj}$  : Velocity components of clutter objects.

The  $f$  subscript is used to denote a final cost. For example,  $\mathbf{W}_{fpr}$  refers to the final position costs for a robot. Robot cost elements refer to actuated joints. Free-floating bodies are un-actuated DoFs in the system which are sometimes split up into manipulands (objects that need to be pushed to some goal position) as well as clutter objects (objects that hinder the motion of the manipuland).

The cost parameters used in all experiments were determined empirically. They were selected to elicit desired optimisation behaviour when using the baseline method. The time-step parameter,  $\Delta t$ , was chosen to be as large as possible without compromising simulation stability or optimisation performance, again based on observations from the baseline method of optimisation.

Table 4.1: Tasks used for simulation experiments. The left column shows the task name and an image of the task. Right column gives a brief description of the task objective followed by the optimiser used for the task. The costs used for optimisation are shown below the task description with cost notation as defined previously.  $\Delta t$  is the model time-step.

Task	Description
<b>Acrobot</b> 	Under-actuated double pendulum. Needs to build momentum to swing up to unstable position. Optimiser: <b>iLQR_FD</b> . $\mathbf{x} = (q_1, q_2, \dot{q}_1, \dot{q}_2) \quad \mathbf{u} = (\tau_1) \quad \Delta t = 0.01$ $\mathbf{W}_{pr} = [0], \mathbf{W}_{fpr} = [10], \mathbf{W}_{pb} = [0], \mathbf{W}_{fpb} = [10]$ $\mathbf{W}_{vr} = [0.001], \mathbf{W}_{fvr} = [1], \mathbf{W}_{vb} = [0.001], \mathbf{W}_{fvb} = [1]$ $\mathbf{R} = [0.0001]$
<b>Panda reaching</b> 	Franka Panda robot, moving to a desired joint configuration whilst minimising joint velocities. Optimiser: <b>iLQR_FD</b> . $\mathbf{x} = (q_1, q_2, \dots, q_7, \dot{q}_1, \dot{q}_2, \dots, \dot{q}_7) \quad \mathbf{u} = (\tau_1, \tau_2, \dots, \tau_7)$ $\Delta t = 0.008$ $\mathbf{W}_{pr} = [0], \mathbf{W}_{fpr} = [100],$ $\mathbf{W}_{vr} = [1], \mathbf{W}_{fvr} = [10], \mathbf{R} = [0]$
<b>Push (n/l/h) clutter</b> 	Franka Panda robot, pushing a green cylinder to target location (green silhouette), in different levels of clutter (red objects). (no light heavy) = (0, 3, 7) objects. Optimiser: <b>iLQR_FD</b> . $\mathbf{x} = (q_1, q_2, \dots, q_7, x_g, y_g, x_j, y_j, \dot{q}_1, \dot{q}_2, \dots, \dot{q}_7, \dot{x}_g, \dot{y}_g, \dot{x}_j, \dot{y}_j)$ $\mathbf{u} = (\tau_1, \tau_2, \dots, \tau_7) \quad \Delta t = 0.008$ $\mathbf{W}_{pr} = [0], \mathbf{W}_{fpr} = [0], \mathbf{W}_{pbg} = [0], \mathbf{W}_{fpg} = [1000]$ $\mathbf{W}_{pbj} = [0], \mathbf{W}_{fpbj} = [10], \mathbf{W}_{vr} = [0], \mathbf{W}_{fvr} = [0]$ $\mathbf{W}_{vbg} = [0.1], \mathbf{W}_{fvbg} = [0], \mathbf{W}_{vbj} = [0], \mathbf{W}_{fvbj} = [0], \mathbf{R} = [0]$
<b>Box sweep</b> 	Franka Panda robot sweeping a large and heavy green box to target location (transparent). Optimiser: <b>iLQR_FD</b> . $\mathbf{x} = (q_1, q_2, \dots, q_7, x_g, y_g, \dot{q}_1, \dot{q}_2, \dots, \dot{q}_7, \dot{x}_g, \dot{y}_g)$ $\mathbf{u} = (\tau_1, \tau_2, \dots, \tau_7) \quad \Delta t = 0.008$ $\mathbf{W}_{pr} = [0], \mathbf{W}_{fpr} = [100], \mathbf{W}_{pg} = [0], \mathbf{W}_{fpg} = [100]$ $\mathbf{W}_{vr} = [1], \mathbf{W}_{fvr} = [10], \mathbf{W}_{vg} = [0.1],$ $\mathbf{W}_{fvg} = [0], \mathbf{R} = [0]$
<b>Walker</b> 	9 DoF walker, walk forwards at desired velocity as well as maintain a certain height and upright orientation of the body. Optimiser: <b>iLQR_FD</b> . $\mathbf{x} = (q_1, q_2, \dots, q_6, x, z, \theta, \dot{q}_1, \dot{q}_2, \dots, \dot{q}_6, \dot{x}, \dot{z}, \dot{\theta})$ $\mathbf{u} = (\tau_1, \tau_2, \dots, \tau_6) \quad \Delta t = 0.005$ $\mathbf{W}_{pr} = [0], \mathbf{W}_{fpr} = [0], \mathbf{W}_{vr} = [0], \mathbf{W}_{fvr} = [0]$ $\mathbf{W}_{pb} = [1], \mathbf{W}_{fpb} = [1], \mathbf{W}_{vb} = [0.1], \mathbf{W}_{fvb} = [0]$ $\mathbf{W}_{p\theta} = [1], \mathbf{W}_{v\theta} = [0], \mathbf{R} = [1e - 5]$

Task	Description
<b>Ball push forwards</b> 	Kinova manipulator pushing a large ball to a desired position by rolling it forwards. Optimiser: <b>iLQR_AD</b> . $\mathbf{x} = (q_1, q_2, \dots, q_7, x_g, y_g, z_g, qx_g, qy_g, qz_g, w_g, \dot{q}_1, \dot{q}_2, \dots, \dot{q}_7, \dot{x}_g, \dot{y}_g, \dot{z}_g, \dot{\psi}_g, \dot{\theta}_g, \dot{\phi}_g) \quad \mathbf{u} = (\tau_1, \tau_2, \dots, \tau_7) \quad \Delta t = 0.01$ $\mathbf{W}_{pr} = [0], \mathbf{W}_{fpr} = [0], \mathbf{W}_{vr} = [0.1], \mathbf{W}_{fvr} = [0.1]$ $\mathbf{W}_{pb} = [100], \mathbf{W}_{fpb} = [100], \mathbf{W}_{vb} = [0.1], \mathbf{W}_{fvb} = [10],$ $\mathbf{W}_{pb\theta} = [0], \mathbf{W}_{fpb\theta} = [0], \mathbf{R} = [0.01]$
<b>Ball push side</b> 	Kinova manipulator sliding a large ball to a desired position by rolling it sideways. Optimiser: <b>iLQR_AD</b> . $\mathbf{x} = (q_1, q_2, \dots, q_7, x_g, y_g, z_g, qx_g, qy_g, qz_g, w_g, \dot{q}_1, \dot{q}_2, \dots, \dot{q}_7, \dot{x}_g, \dot{y}_g, \dot{z}_g, \dot{\psi}_g, \dot{\theta}_g, \dot{\phi}_g) \quad \mathbf{u} = (\tau_1, \tau_2, \dots, \tau_7) \quad \Delta t = 0.01$ $\mathbf{W}_{pr} = [0], \mathbf{W}_{fpr} = [0], \mathbf{W}_{vr} = [0.1], \mathbf{W}_{fvr} = [0.1]$ $\mathbf{W}_{pb} = [100], \mathbf{W}_{fpb} = [100], \mathbf{W}_{vb} = [0.1], \mathbf{W}_{fvb} = [10],$ $\mathbf{W}_{pb\theta} = [0], \mathbf{W}_{fpb\theta} = [0], \mathbf{R} = [0.01]$
<b>Ball lift</b> 	Kinova manipulator lifting a ball against itself to a desired height. Optimiser: <b>iLQR_AD</b> . $\mathbf{x} = (q_1, q_2, \dots, q_7, x_g, y_g, z_g, qx_g, qy_g, qz_g, w_g, \dot{q}_1, \dot{q}_2, \dots, \dot{q}_7, \dot{x}_g, \dot{y}_g, \dot{z}_g, \dot{\psi}_g, \dot{\theta}_g, \dot{\phi}_g) \quad \mathbf{u} = (\tau_1, \tau_2, \dots, \tau_7) \quad \Delta t = 0.01$ $\mathbf{W}_{pr} = [0], \mathbf{W}_{fpr} = [0], \mathbf{W}_{vr} = [0.1], \mathbf{W}_{fvr} = [0.1]$ $\mathbf{W}_{pb} = [100], \mathbf{W}_{fpb} = [100], \mathbf{W}_{vb} = [0.1], \mathbf{W}_{fvb} = [10],$ $\mathbf{W}_{pb\theta} = [0], \mathbf{W}_{fpb\theta} = [0], \mathbf{R} = [0.01]$
<b>Mini cheetah</b> 	Mini cheetah moving forwards whilst remaining upright trying to maintain a target velocity. Optimiser: <b>iLQR_AD</b> . $\mathbf{x} = (q_1, q_2, \dots, q_{12}, x_g, y_g, z_g, qx_g, qy_g, qz_g, w_g, \dot{q}_1, \dot{q}_2, \dots, \dot{q}_{12}, \dot{x}_g, \dot{y}_g, \dot{z}_g, \dot{\psi}_g, \dot{\theta}_g, \dot{\phi}_g) \quad \mathbf{u} = (\tau_1, \tau_2, \dots, \tau_{12}) \quad \Delta t = 0.01$ $\mathbf{W}_{pr} = [0.01], \mathbf{W}_{fpr} = [1.1], \mathbf{W}_{vr} = [0.01], \mathbf{W}_{fvr} = [0.01]$ $\mathbf{W}_{pb} = [2], \mathbf{W}_{fpb} = [10], \mathbf{W}_{vb} = [1], \mathbf{W}_{fvb} = [10],$ $\mathbf{W}_{pb\theta} = [2], \mathbf{W}_{fpb\theta} = [10], \mathbf{R} = [0.01]$
<b>Box slide</b> 	Sawyer robot arm pushing a box to a desired location. Optimiser: <b>SCVX</b> . $\mathbf{x} = (q_1, q_2, \dots, q_7, x_g, y_g, z_g, qx_g, qy_g, qz_g, w_g, \dot{q}_1, \dot{q}_2, \dots, \dot{q}_7, \dot{x}_g, \dot{y}_g, \dot{z}_g, \dot{\psi}_g, \dot{\theta}_g, \dot{\phi}_g) \quad \mathbf{u} = (\tau_1, \tau_2, \dots, \tau_7) \quad \Delta t = 0.01$ $\mathbf{W}_{pr} = [0], \mathbf{W}_{fpr} = [0], \mathbf{W}_{vr} = [0], \mathbf{W}_{fvr} = [0]$ $\mathbf{W}_{pb} = [10000], \mathbf{W}_{fpb} = [10000], \mathbf{W}_{vb} = [0], \mathbf{W}_{fvb} = [0],$ $\mathbf{W}_{pb\theta} = [1], \mathbf{W}_{fpb\theta} = [1], \mathbf{R} = [0]$

### 4.5.3 Key-point parametrisations

The following key-point parametrisations are used for all the outlined tasks:

- Acrobot - Adaptive Jerk:**  $N_{min}=1$ ,  $N_{max}=50$ ,  $\bar{\mathbf{J}} = 150$  for both joints. **Velocity Change:**  $N_{min}=1$ ,  $N_{max}=50$ ,  $\Delta \bar{\mathbf{v}} = 1$  for both joints. **Iterative Error:**  $N_{min}=1$ ,  $\bar{v}=0.0001$ .
- Reaching - Adaptive Jerk:**  $N_{min}=5$ ,  $N_{max}=100$ ,  $\bar{\mathbf{J}}$  of 10 used. **Velocity**

**Change:**  $N_{min}$ : 5,  $N_{max}$ : 50,  $\Delta\bar{\mathbf{v}}$ : [1, 1, 1, 1, 0.1, 0.1, 0.1]. **Iterative Error:**  $N_{min}$ : 1,  $\bar{\epsilon}$ : 0.001.

- **Push no clutter - Adaptive Jerk:**  $N_{min}=5$ ,  $N_{max}=100$ ,  $\bar{\mathbf{J}}$  of 1 used for robotic arm and 1 used for goal object. **Velocity Change:**  $N_{min}=5$ ,  $N_{max}=50$ ,  $\Delta\bar{\mathbf{v}}$  of 1 used for robotic arm joints and goal object. **Iterative Error:**  $N_{min}=1$ ,  $\bar{\epsilon}=0.01$ .
- **Push low clutter - Adaptive Jerk:**  $N_{min}=5$ ,  $N_{max}=100$ ,  $\bar{\mathbf{J}}$  of 1 used for robotic arm and 0.1 used for all objects in scene. **Velocity Change:**  $N_{min}=1$ ,  $N_{max}=100$ ,  $\Delta\bar{\mathbf{v}}$  of 0.1 used for robotic arm joints and all objects in scene. **Iterative Error:**  $N_{min}=1$ ,  $\bar{\epsilon}=0.001$ .
- **Push heavy clutter - Adaptive Jerk:**  $N_{min}=2$ ,  $N_{max}=100$ ,  $\bar{\mathbf{J}}$  of 1 used for robotic arm and 0.1 used for all objects in scene. **Velocity Change:**  $N_{min}=2$ ,  $N_{max}=100$ ,  $\Delta\bar{\mathbf{v}}$  of 0.05 used for robotic arm joints and 0.01 for all objects in scene. **Iterative Error:**  $N_{min}=1$ ,  $\bar{\epsilon}=0.0001$ .
- **Box sweep - Adaptive Jerk:**  $N_{min}=1$ ,  $N_{max}=1000$ ,  $\bar{\mathbf{J}}$  of 1000 used for robotic arm and for goal object. **Velocity Change:**  $N_{min}=1$ ,  $N_{max}=100$ ,  $\Delta\bar{\mathbf{v}}$  of 0.1 used for robotic arm joints and 0.05 for goal object. **Iterative Error:**  $N_{min}=1$ ,  $\bar{\epsilon}=0.00001$ .
- **Box slide - Adaptive Jerk:**  $N_{min}=1$ ,  $N_{max}=5$ ,  $\bar{\mathbf{J}}$  of 500 used for robotic arm and for goal object. **Velocity Change:**  $N_{min}=1$ ,  $N_{max}=5$ ,  $\Delta\bar{\mathbf{v}}$  of 0.1 used for robotic arm joints and goal object. **Iterative Error:**  $N_{min}=1$ ,  $\bar{\epsilon}=0.1$ .
- **Ball tasks - Adaptive Jerk:**  $N_{min}=2$ ,  $N_{max}=20$ ,  $\bar{\mathbf{J}}$  of 5000 used for robotic arm and for goal object. **Velocity Change:**  $N_{min}=2$ ,  $N_{max}=20$ ,  $\Delta\bar{\mathbf{v}}$  of 0.5 used for robotic arm joints and goal object. **Iterative Error:**  $N_{min}=2$ ,  $\bar{\epsilon}=800$ .

## 4.6 Results

This section showcases a wide variety of simulation and hardware results. Initially, Sec. 4.6.1 investigates how the different key-point methods perform in terms of approximation accuracy and the number of key-points required when comparing the approximated dynamics derivatives with the accurate dynamics derivatives for the Acrobot task. Next, Sec. 4.6.2 focuses on a simplistic toy contact problem (shown in Fig. 4.8) to understand how optimisation via the proposed key-points methods works around the introduction of contact-interactions. Sec. 4.6.3 performs long horizon optimisation performance tests (shown in Table 4.2) on a wide variety of tasks. The key-point methods are evaluated compared to the baseline in terms of optimisation time reduction and the quality of the final computed solution. Following on from

this, Sec. 4.6.4 evaluates how the proposed key-point methods perform when using shorter optimisation horizons (such as the ones used when performing MPC) for two locomotion tasks in simulation. Finally, Sec. 4.6.5 performs some experiments on real robotic hardware for a complex manipulation in clutter task.

### 4.6.1 Evaluating interpolation accuracy

The key-point methods outlined in Section 4.3 aim to match the accurate derivatives over the trajectory as closely as possible with as few key-points as possible. To evaluate the approximation accuracy of an interpolated set of dynamics derivatives, a scalar value representing overall approximation accuracy is defined by comparing the absolute difference of the values between the approximated and accurate dynamics derivatives over a trajectory:

$$MAE = \sum_{i=0, j=0}^{n_x, n_x} \frac{\sum_{t=0}^T |\hat{\mathbf{A}}_t[i, j] - \mathbf{A}_t[i, j]|}{T} / (n_x * n_x) + \sum_{i=0, j=0}^{n_u, n_x} \frac{\sum_{t=0}^T |\hat{\mathbf{B}}_t[i, j] - \mathbf{B}_t[i, j]|}{T} / (n_x * n_u). \quad (4.26)$$

To evaluate the approximation error of the proposed key-point methods, 100 test trajectories are generated and the  $\mathbf{A}$  and  $\mathbf{B}$  matrices are computed accurately at every time-step, either by FD or AD. To evaluate a particular key-point method and parametrisation, the MAE is computed for each of the 100 trajectories and averaged.

A variety of key-point parametrisations are evaluated to determine what methods worked best to obtain a cheap, yet accurate approximation. The results are shown in Fig. 4.6 for the Acrobot task where the horizontal axis is the percentage of derivatives that were computed accurately.

Ideally, one would choose a key-point method and its associating parameters to minimise both the error as well as the percentage of derivatives. Naturally, there is a trade-off between accuracy and the percentage of derivatives making this a multi-objective optimisation problem. Therefore, a suitable level of MAE is empirically defined above which the optimisation performance begins to degrade (denoted as MAE threshold and a dashed red line in Fig. 4.6). Everything below this line is defined as a *good* approximation. For each method, three sets of parameters are highlighted, the **star** values are what are defined as optimal, which are values below the MAE threshold with the lowest percentage of derivatives. The **cross symbol** values have the lowest percentage of derivatives and the **diamond** markers have the lowest MAE.

For any Set Interval method that is plotted (black markers), there is always an adaptive method that outperforms it by having both a lower MAE as well as a lower percentage of derivatives. That is to say no Set Interval methods would



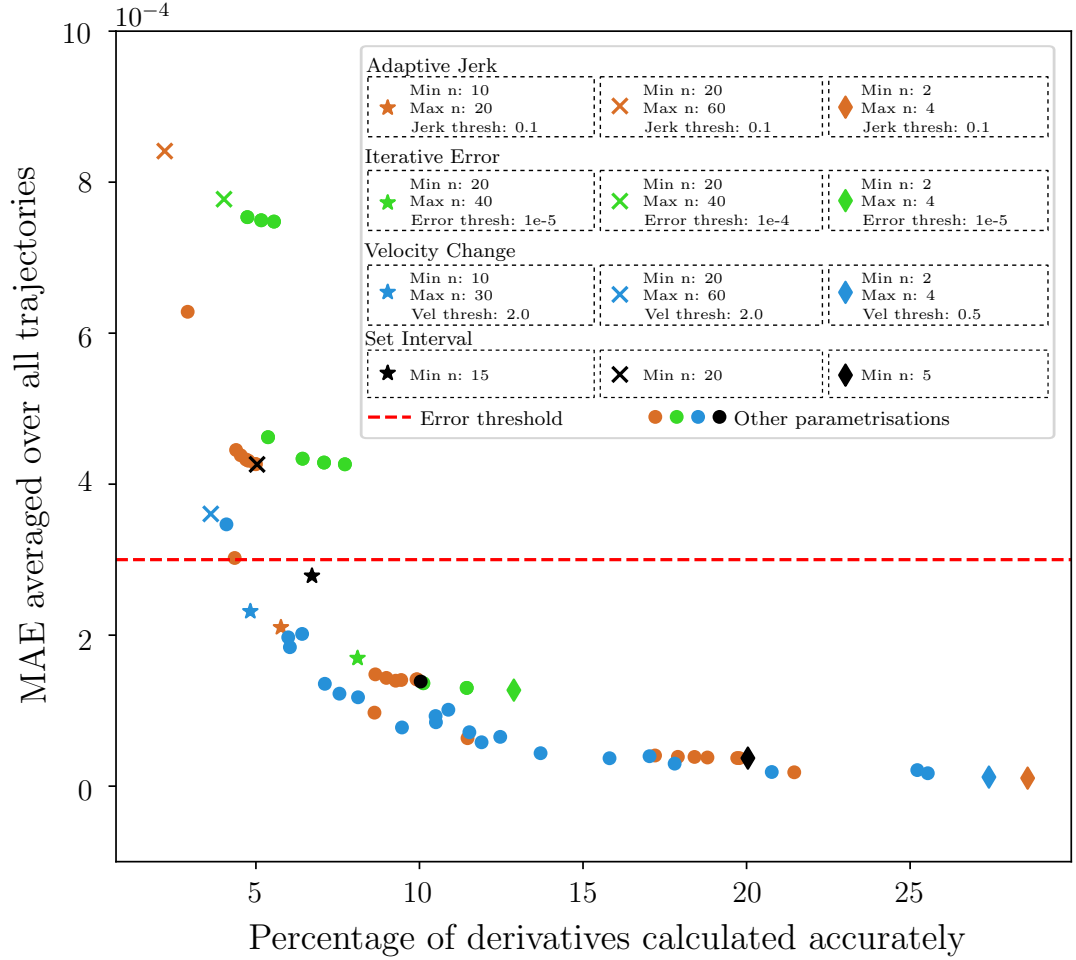


Figure 4.6: Approximation accuracy for different key-point methods and different key-point parametrisations values, for the Acrobot task. The colour of each data-point represents the specific key-point selection method used and each data-point is a different parameterisation for that method. Three parameterisations for each key-point method are explicitly shown by a **star**, **cross** and **diamond** symbol and labelled in the top right of the figure.

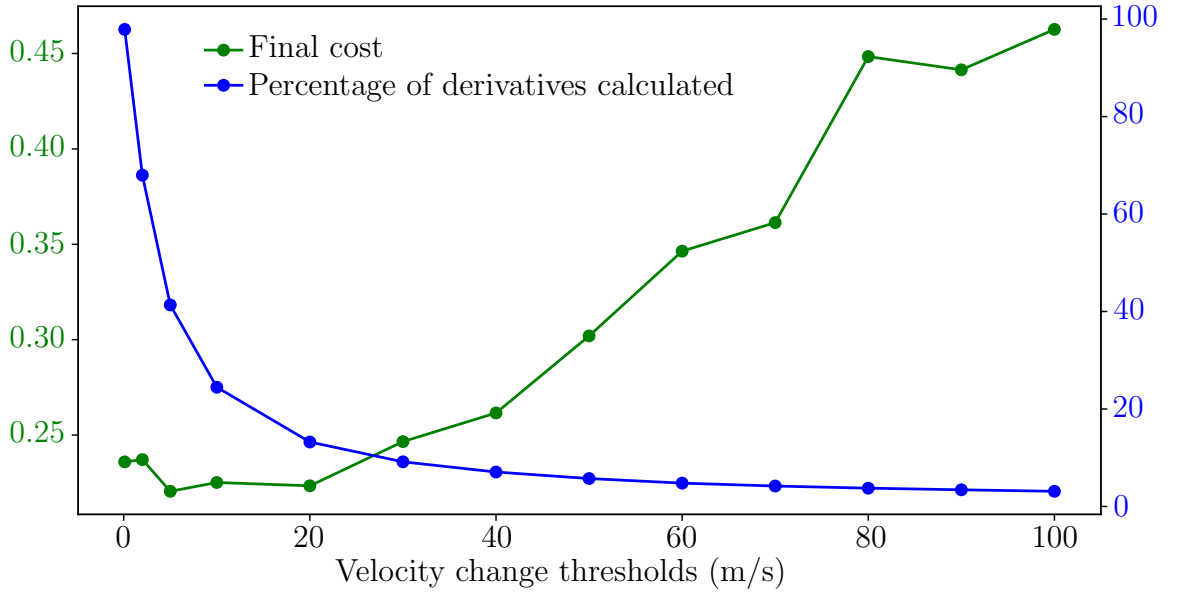


Figure 4.7: Optimisation performance for 100 instantiations of the Acrobot task versus scaling values for the velocity change threshold. Green line shows normalised final cost of the trajectory, blue line shows the average percentage of derivatives computed accurately.

be included on the pareto-front boundary. Generally, for this Acrobot task, the Velocity Change method works exceptionally well. This is due to the fact that the dynamics derivatives change in a way that are very closely related to the velocity of both the acrobot joints.

To understand what level of error was acceptable, the optimisation performance for the Velocity Change key-point method is evaluated on the Acrobot task. Optimisation until convergence for 100 instantiations of the Acrobot task (different start and goal states) was performed with scaling thresholds for the Velocity Change key-point method.

The results of this can be seen in Fig. 4.7. As the velocity change threshold was increased, the percentage of derivatives computed accurately reduces and the final cost achieved by optimisation increases, which is of course the expected outcome. It can be seen that there is a region (around the velocity change threshold value of 20) where the percentage of derivatives calculated is significantly reduced, without any loss in terms of the final cost performance. This can be thought of as the sort of “sweet spot” where the optimisation time reductions are gained with no significant degradation in the quality of the final trajectory.

#### 4.6.2 Impact of contact on optimisation and key-point selection

An important aspect to consider is how the proposed key-point methods perform in situations that involve contact. When contact is made or broken, this generally

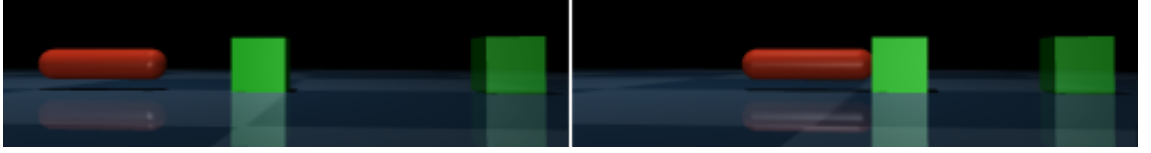


Figure 4.8: Two snapshots from the 1D toy contact task. The red piston is actuated and limited to 1D of motion. The green object is an un-actuated cube. The goal is to push the green cube to the goal location (green silhouette).

creates a discontinuity in the dynamics derivatives due to the contact mode being changed. Even though physics simulators typically circumvent this issue through the use of “soft contact” models [129, 128], there is still a rapid change in the dynamics derivatives when a contact is made or broken.

This section focuses on two particular questions: (1) How important is it for optimisation performance that key-points are accurately located at time-steps when contact is made or broken? (2) How capable are the proposed key-point methods at detecting these contact discontinuities?

To investigate these questions in-depth, this section focuses on a 1D toy contact task. This task was implemented in MuJoCo and consists of a 1 DoF robot piston and a green un-actuated cube (Fig. 4.8). The objective is to minimise the cube’s distance to the goal at the end of the trajectory, whilst also minimising the cube’s velocity. The state vector of this system is four dimensional: the piston’s joint position and joint velocity, as well as the cube’s position and velocity (all in 1D).

**(1) Impact of contact discontinuities on optimisation performance:** In Fig. 4.9, a subset of the values in the  $\mathbf{A}$  matrix (4 of 16) are shown and how they change over an initial trajectory. The initial trajectory moves the piston forward and makes contact with the green cube at time-step  $t_c$ . This time-step is shown on the figure as the vertical green-dashed line. As can be clearly seen in the figure, this contact event creates a discontinuity in the dynamics derivatives.

To analyse how important it is to accurately map this contact discontinuity for optimisation performance, the following experiment is performed. The contact making time-step  $t_c$  is identified and the derivative values inside the  $\mathbf{A}$  matrix are deliberately “smoothed” around this time-step. This smoothing is done by linearly interpolating the dynamics derivatives between  $t_c - p$  and  $t_c + p$ , where  $p$  is the number of time-steps of smoothing. A single optimisation iteration is then performed about the same nominal trajectory using increasing values of  $p$ . As the value of  $p$  increases, the contact discontinuity is smoothed more.

The result of this experiment can be seen in Fig. 4.10. The vertical axis shows the cost of the trajectory after *one* optimisation iteration and the horizontal axis shows the value of  $p$ . As the value of  $p$  increases the general trend is that the optimisation performance decreases. Some level of blending is acceptable (up to

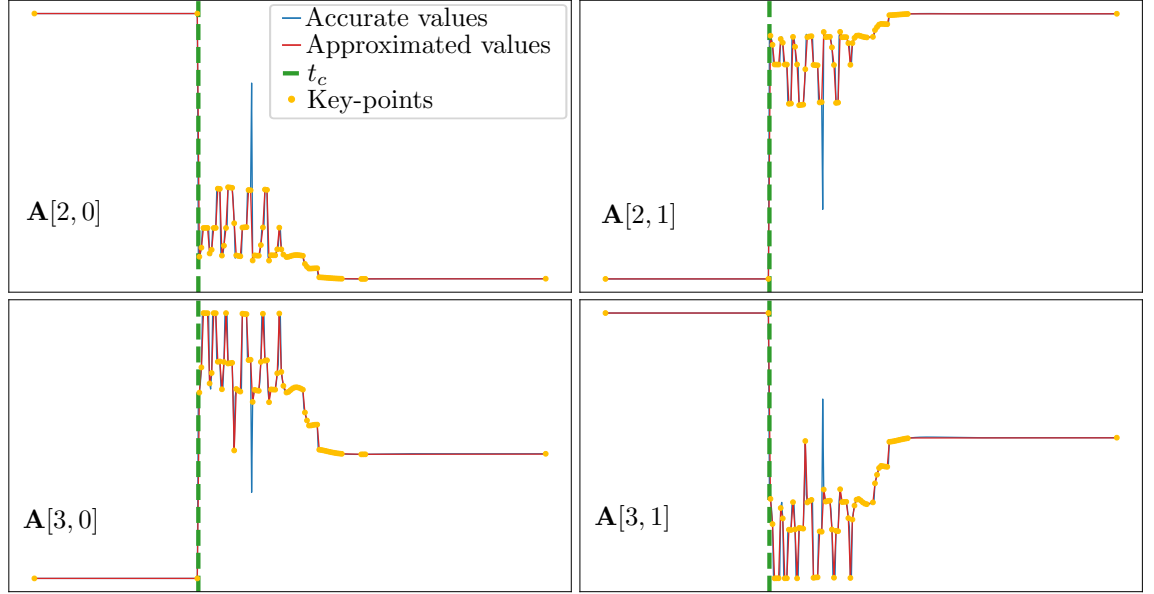


Figure 4.9: A 2x2 subset of graphs that show how values inside the  $\mathbf{A}$  matrix for the toy contact task change over the initial trajectory. The subset shown are how the cube’s position and velocity are affected by the piston’s position and velocity. The blue line shows the *accurate* values, the red line shows the approximation subject to the computed key-points (yellow dots). The green dashed vertical line denotes moment of contact  $t_c$ .

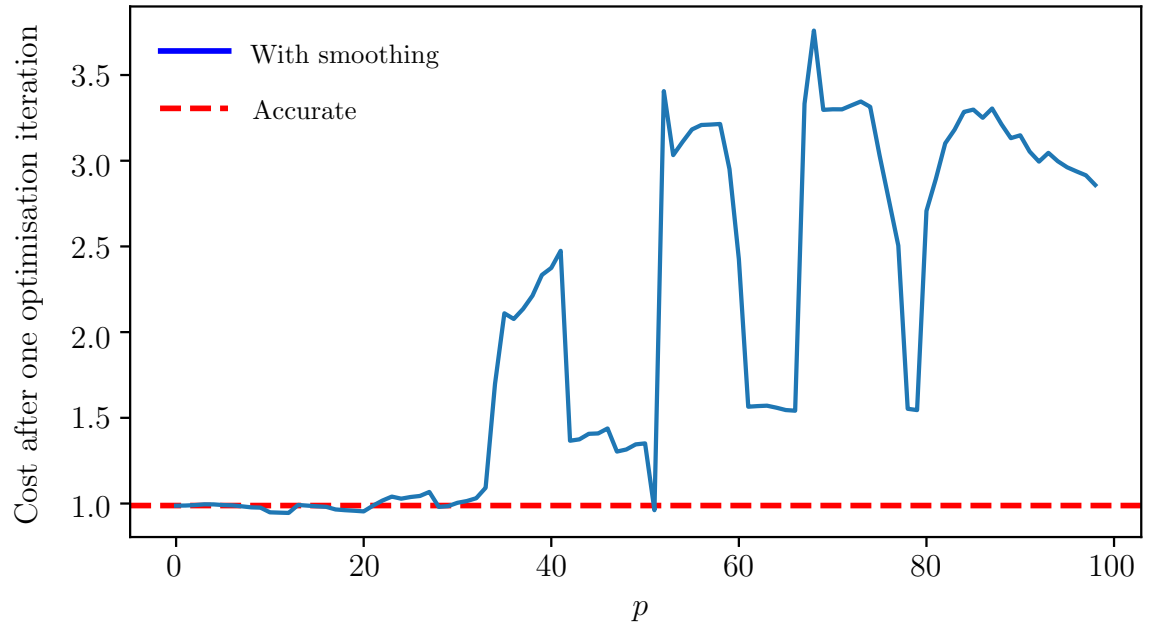


Figure 4.10: Optimisation performance versus smoothing effect after one iteration of optimisation.

approximately 30 time-steps for this scene). This result implies that in contact-based optimisation, it is important to generally model the contact discontinuity, but some level of blending can be acceptable.

**Identifying contact discontinuities with key-point methods:** Fig. 4.9 shows that the Adaptive Jerk key-point method can accurately map the contact discontinuity when appropriate jerk threshold values are selected (jerk thresholds of  $1 \text{ ms}^{-3}$  for the piston and un-actuated cube are used). The yellow dots are the placement of key-points, as decided by the Adaptive Jerk method. The plots clearly show that the Adaptive Jerk method places a significant number of key-points where contact is made and the dynamics derivatives are very noisy. Around the start and end of the trajectory, where the derivative values are static, Adaptive Jerk places minimal key-points.

### 4.6.3 Long horizon optimisation performance

In this section, the performance of the key-point methods is investigated on what is referred to as “long horizon” optimisation. Long horizon optimisation refers to optimisation on a task where the horizon is sufficient to complete the task and optimisation is performed until convergence or a maximum number of optimisation iterations. These experiments are performed to validate that the key-point methods can significantly reduce optimisation time whilst retaining a similar level of performance in terms of the quality of the final converged trajectory.

To evaluate the performance of the final converged trajectories, a metric called *cost reduction* (CR) is defined. Cost reduction simply measures the proportion that the cost has been reduced by comparing the converged trajectory cost to the initial trajectory cost. A value of zero means the converged trajectory has the same cost as the initial trajectory, and a value of one means that the converged trajectory had a cost of zero (generally not feasible with how most optimisation problems are formulated). This is shown in Eq. 4.27:

$$\text{CR} = 1 - \frac{\text{Final cost}}{\text{Initial cost}}. \quad (4.27)$$

In Table 4.2, various optimisation metrics are presented for four key-point selection methods across a variety of tasks. These metrics include overall optimisation time (OT), cost reduction (CR) (defined in Eq. 4.27), and number of optimisation iterations (NI). For the **iLQR-FD** and **SCVX** experiments, 100 start/goal states were randomly generated, and the mean values are reported (the standard deviations were all small and are not shown). For the **iLQR-AD** experiments, only a single experiment was conducted for each task, as the comparison follows the whole-body ball manipulation tasks from Kurtz & Lin [69], with no changes other than the use of key-point methods. However, average results for all three types of ball manipulation using the *same* key-point parametrisations are shown in the “Ball average” row.

The results in Table 4.2 clearly show that, across all tasks and optimisers, optimisation times can be significantly reduced by all key-point methods when compared to the baseline, at the cost of some small degradation in performance in terms of CR. This naturally poses an interesting question regarding the trade-off between these two metrics: How much reduction in CR performance is acceptable for the level of optimisation time reduction achieved? This trade-off largely depends on the specific use-case. In certain contexts, such as MPC, it is more important to optimise to an acceptable solution quickly, rather than a perfect solution slowly, due to the detrimental effects that policy lag can have on execution performance. This concept is explored further in Sec. 4.6.4.

On average, the key-point methods perform marginally fewer optimisation iterations than the baseline. This contributes to the slight degradation observed in terms of CR. Importantly, the achieved optimisation time reductions are *not* primarily due to this reduction in the number of optimisation iterations, but are instead mainly attributed to the reduced optimisation iteration time. The average optimisation iteration time can be computed by dividing the OT by the NI. This result suggests that approximated derivatives are fundamentally effective at optimising trajectories close to the global minima, but may struggle to reach the global minima directly.

Table 4.2: Summary of long-horizon optimisation performance. OT is the total optimisation time in seconds, CR is cost reduction and NI is the number of optimisation iterations. For **iLQR-FD** and **SCVX**, the values reported in the tables are mean values averaged over 100 start/goal states. For **iLQR-AD** the values were single runs.

iLQR-FD																	
Baseline			SI5			SI1000			Adaptive Jerk			Velocity Change			Iterative Error		
	OT	CR	NI	OT	CR	NI	OT	CR	NI	OT	CR	NI	OT	CR	NI		
Acrobot	0.041	0.77	7.49	0.021	0.76	7.06	0.010	0.36	4.15	0.018	0.73	6.08	0.024	0.77	7.62		
	0.91	0.94	7.64	0.51	0.94	7.56	0.39	0.90	7.11	0.43	0.94	7.63	0.46	0.94	7.65		
Push n clutter	0.99	0.25	4.45	0.41	0.29	4.38	0.25	0.20	4.24	0.31	0.24	4.50	0.48	0.25	4.35		
Push l clutter	4.08	0.28	4.78	1.23	0.27	4.61	0.58	0.26	4.45	0.91	0.22	4.35	1.25	0.24	4.54		
Push h clutter	9.26	0.47	5.09	2.74	0.42	5.00	1.09	0.34	4.37	2.85	0.42	4.83	3.36	0.43	4.90		
Box sweep	1.88	0.60	4.87	0.53	0.38	4.18	0.38	0.46	4.50	0.39	0.54	4.37	0.73	0.45	4.42		
SCVX																	
Baseline			SI2			SI20			Adaptive Jerk			Velocity Change			Iterative Error		
	OT	CR	NI	OT	CR	NI	OT	CR	NI	OT	CR	NI	OT	CR	NI		
Box slide	8.87	0.989	30.69	6.14	0.985	27.48	7.62	0.937	35.65	6.13	0.985	30.43	6.81	0.983	30.85		
iLQR-AD																	
Baseline			SI2			SI20			Adaptive Jerk			Velocity Change			Iterative Error		
	OT	CR	NI	OT	CR	NI	OT	CR	NI	OT	CR	NI	OT	CR	NI		
Ball forward	128.71	0.78	35	49.20	0.79	24	4.34	0.77	13	4.12	0.75	11	46.00	0.77	20		
Ball side	65.24	0.74	27	24.77	0.70	20	2.73	0.54	8	15.83	0.65	32	21.43	0.70	20		
Ball lift	261.52	0.92	41	38.35	0.88	17	26.46	0.86	20	34.32	0.90	22	16.00	0.86	9		
Ball average	151.82	0.81	34.33	37.44	0.79	20.30	11.18	0.72	13.67	18.09	0.77	21.67	27.81	0.78	16.3		

It is important to note that the key-point selection methods require some form of tuning to perform well. The general approach for tuning was to start with a non-greedy key-point parametrisation for any particular method, and then to iteratively make the key-point parametrisation more greedy until an unsatisfactory reduction in CR was observed.

By averaging the results in Table 4.2, it can be seen that Adaptive Jerk and Velocity Change both reduce the optimisation time more than the smallest interval Set Interval method, whilst achieving identical performance in terms of CR. They do not reduce the optimisation time as much as the largest interval Set Interval method, however they achieve a noticeably higher CR. Iterative Error performs well in terms of CR for the **iLQR-FD** tasks, however does not achieve as substantial optimisation time reductions as the other key-point methods. The reason for this is that the Iterative Error method does not take advantage of the ability to compute dynamics derivatives in parallel as much as the other key-point methods do. There will be a more detailed discussion about the Iterative Error method and its shortcomings in Sec. 4.7.

For the **iLQR-AD** tasks, the optimisation times were significantly longer than the **iLQR-FD** tasks, despite having similar dimensionality. There are three main reasons for this: (i) Python is a slower language than C++, (ii) the computation of dynamics derivatives was **not** parallelised over CPU cores, and (iii) the hydro-elastic contact model is more expensive than the one used in iLQR-FD. However, any improvements made to these three attributes would also speed up the key-point methods performance.

Finally, when using **SCVX**, the optimisation time reductions were not as substantial as they were in the iLQR based methods. The fundamental reason behind this was that computing dynamics derivatives was not as severe a bottleneck as it was for iLQR. In SCVX, the two parts of the algorithm that take substantial amounts of time were computing dynamics derivatives and solving quadratic programs (QPs) using SQOPT. Even though computing dynamics derivatives was more computationally expensive than solving QPs, there were a significant number of iterations (even in the baseline case) where the solver had to reduce the size of the trust region to find a lower cost trajectory when using the non-linear dynamics. When the solver reduced the size of the trust region, it was unnecessary to recompute dynamics derivatives as the nominal trajectory had remained the same.

#### 4.6.4 Short horizon optimisation performance

The previous section examined the performance of the key-point methods in long-horizon optimisation. While long-horizon optimisation planning offers significant utility, it also has its drawbacks. For instance, executing such plans on real robotic hardware in an open-loop manner can be problematic, particularly when contact



---

**Algorithm 8** *Execution with MPC.*

---

**Require:** *controls\_before\_replan, T, max\_iterations*

```

1: U = Initialise controls for  $T$  horizon
2: controls_executed = 0
3: while not timeout do
4:   if controls_executed > controls_before_replan then
5:     controls_executed = 0
6:     U = Optimise(U,  $T$ , max_iterations)
7:   Execute U0 in environment
8:   Remove U0 from U
9:   Expand U with U $T-1$ 
10:  controls_executed++

```

---

dynamics are involved. Despite best efforts, no physics simulator can perfectly replicate the real world. Consequently, executing a long-horizon trajectory computed in simulation often leads to different outcomes when transferred to the real world.

There are a variety of ways to overcome this, optimisation can be done using domain randomisation to create probabilistically robust actions [106], execution of the optimised trajectory can be executed open-loop until a significant enough deviation has occurred from the plan in simulation and what has happened in reality [4] or finally, MPC can be used to optimise closed-loop to account for the deviations between the physics simulator and the real world. This section investigates how the key-point methods can aid in an MPC setting, under optimisation with shorter horizons.

MPC results are provided for two locomotion tasks in simulation (Walker and Mini cheetah from Table 4.1). The context in which MPC is performed is important to understand the results. In this MPC implementation the execution (in a simulator) and optimisation is performed *sequentially*, meaning that the execution is not integrated with time until the optimisation has converged and the next optimal control is ready to be applied. This means that there is no *policy lag* in the system as the system is frozen until optimisation converges. This means that slower optimisation processes are not punished (i.e. the baseline). The MPC approach is shown in Alg. 8.

The Mini cheetah locomotion model used the **iLQR-AD** optimiser and used MPC from Alg. 8 with the following parameters (4, 50, 10). This controller executed 400 controls before terminating, with 100 receding horizon optimisations (as 4 controls were executed before the next re-optimisation step). For the baseline, the whole task took approximately 18 minutes of computation time, where 76 % of the planning time was spent computing dynamics derivatives.

The key-point methods significantly reduce the time taken to perform the same number of controls, whilst importantly still enabling the mini-cheetah to remain up-

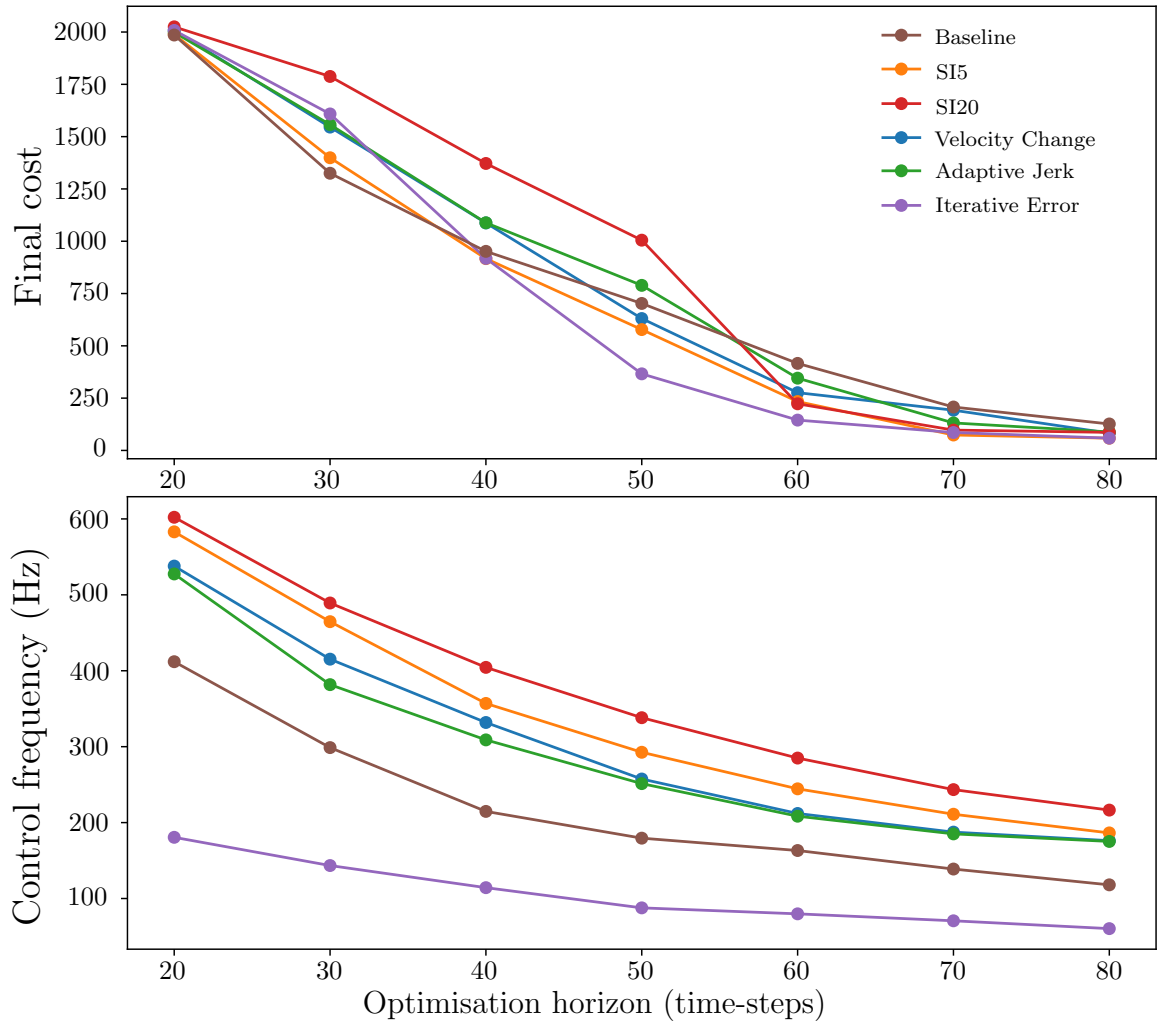


Figure 4.11: MPC results for the Walker model over a variety of receding horizon lengths for the different key-point methods. The top plot is the performance of the executed trajectory when evaluated with the cost function. The bottom plot shows the effective control frequency that could be attained for the various methods.

right whilst performing locomotion. The results are shown in Table 4.3. The overall optimisation time is the summation of all the optimisations until task termination (i.e., 100 optimisations). The cost function used in the optimiser is also used to evaluate the final executed trajectory. Finally, if a method caused the mini cheetah to topple, the simulation would crash and the percentage through the task at which the simulation crashed is recorded. All three of the adaptive key-point methods managed to complete the task, with substantial reductions in optimisation time<sup>1</sup>.

The MPC implementation for the Walker was Alg. 8 with the following parameters (1, (20 - 80), 1). This controller executed 1500 controls before terminating. Notably, experiments are provided for a range of horizons for this task. For each

<sup>1</sup>The optimisation times were still prohibitively expensive for this task, making traditional MPC infeasible. This is due to a variety of factors, including: contact model, python implementation and lack of parallelisation. The purpose of these experiments are to show generality of the proposed methods in this chapter in terms of reducing optimisation times.

Table 4.3: MPC results for mini-cheetah locomotion using different key-point methods.

	Cost	Optimisation time (s)	Percentage of task
Baseline	45.9	1001	100
SI2	43.5	545	100
SI5	-	-	95
Adaptive Jerk	39.0	313	100
Iterative error	50.7	300	100
Velocity change	50.1	502	100

horizon length, the task was performed via MPC 100 times for each of the key-point methods. The 100 different trajectories all had different target velocities for the Walker between 0.9 and 1.3 m/s. The results are then averaged over the 100 runs and the data is presented the data in Fig. 4.11.

The top plot of Fig. 4.11 shows the final cost of the generated trajectories from the different key-point methods. The bottom plot shows the average control frequency that was achieved by each method. This control frequency was computed by averaging the optimisation iteration times over the task execution.

The top plot of Fig. 4.11 shows two main ideas. Firstly, increasing the optimisation horizon of the Walker model is important to achieve good performance in terms of final cost. This is due to the fact that the optimised controls are myopic when the optimisation horizon is low, resulting in poor overall performance. Secondly, as the optimisation horizon is increased, all key-point methods achieve similar performance in terms of final cost as the baseline method. As can be seen in the bottom plot of Fig. 4.11, at an optimisation horizon of 80 time-steps, the control frequency of the Walker model is increased between 33% and 66% when using the key-point methods (except for Iterative Error). Increasing the control frequency of MPC iterations is desirable for real world execution, as it enables the robot to make adjustments to its trajectory to account for model inaccuracies between the real world and the physics simulator.

#### 4.6.5 Execution performance on hardware

Finally, the proposed key-point methods were validated on real robotic hardware, namely the Franka Panda, and show the significant speed improvements that can be achieved on robotic manipulation in cluttered environments.

A similar task to the “push heavy clutter” task from the simulation experiments was created, with a few notable changes which can be seen in Fig. 4.12. Firstly, static immovable walls to constrain the scene were added. The goal location for the pushed

object is then placed near the static walls, making the push substantially more challenging as clutter objects can get stuck. A cost penalty was added to penalise disturbing the clutter obstacles in the scene for the terminal state. Meaning that whilst pushing the goal object to its target location is desired, it was also necessary to push through the clutter intelligently. Fig. 4.12 shows the starting configuration for scenes 1 and 2 for the real-world experiments.

Three different methods were evaluated for this task on the real robot. Simple straight line pushes (**Initialisation**), **iLQR-FD [Baseline]** as well as **iLQR-FD [Adaptive Jerk]**. In these experiments, to control real robotic hardware, an error-based re-planning approach was used. Long horizon optimisation plans were computed and then executed open-loop. These long-horizon plans were executed until completion, or until significant deviation between the planned and actual state of the system. The Euclidean distance of all objects in the scene between their planned positions and their actual was used to determine whether significant deviation had occurred or not.

Execution on the robot was stopper after either: (i) the time-out (24 seconds of executing controls) occurred, (ii) four optimised pushes have been executed, or (iii) the target object was within a 3-cm radius of the goal location.

## Results

Experiments were ran 10 times for both scenes, for each of three methods discussed. The average of these results can be seen in Table 4.4. Execution time is the time taken for the task to be completed or to time-out (including both planning and execution time). Planning time includes the time spent initialising and optimising controls in simulation. Finally, final cost measures how well the task was performed by replaying the *real* states from execution through the cost function.

In both scenes, using optimisation enabled the robot to find a more effective final trajectory for completing the task, as shown by the final cost. The Adaptive Jerk

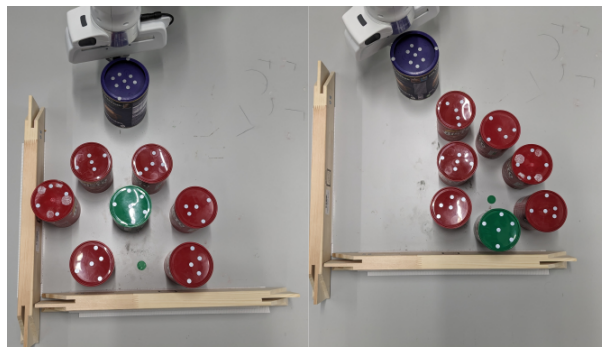


Figure 4.12: Real robot scene setup, scene 1 (left) and scene 2 (right). Objective is to push purple object to goal location (small green sticker) whilst minimising the disturbance to the clutter obstacles (red and green cylinders).

Table 4.4: Results from real robot experiments.

<b>Scene 1</b>			
	<b>Initialisation</b>	<b>[Baseline]</b>	<b>[Adaptive Jerk]</b>
Execution time (s)	25.2	134.2	36.6
Planning time (s)	1.3	110.5	12.7
Final cost	15.2	11.3	11.4
<b>Scene 2</b>			
	<b>Initialisation</b>	<b>[Baseline]</b>	<b>[Adaptive Jerk]</b>
Execution time (s)	25.0	159.2	36.3
Planning time (s)	1.2	135.0	13.5
Final cost	11.3	9.7	9.2

key-point method reduced the optimisation time by approximately 75 % whilst still achieving a near-identical final cost as the baseline method.

## 4.7 Discussion

This chapter has proposed and evaluated three “intelligent” key-point selection methods (i.e., Adaptive Jerk, Velocity Change, and Iterative Error) using two optimisation algorithms (i.e., iLQR and SCVX) for a variety of contact-interaction tasks. The Adaptive Jerk method was designed in part to detect contact mode changes. Whereas, the Velocity Change method was designed for dynamic tasks that require large velocities, such as the Acrobot or Walker tasks. However, from the data that has been collected, both methods seem to work equally well over most tasks (both contact-based and dynamic). These proposed methods optimised trajectories faster on average than the smallest-interval Set Interval methods, whilst obtaining similar performance. Compared to the largest-interval Set Interval methods, they optimised slightly slower but yielded higher-quality trajectories on average.

The Iterative Error method worked fundamentally differently to the Adaptive Jerk and Velocity Change methods. Adaptive Jerk and Velocity Change considered some dynamic attributes of the nominal trajectory to compute a set of key-points. The dynamics derivatives were then computed at these key-points in parallel over CPU cores. The Iterative Error method on the other hand interleaved choosing key-points with dynamic derivative computations. A negative consequence of this was that this imposed a certain level of sequential computation into the process. That is, after each iteration of computing dynamics derivatives, there was a break whilst the key-point method checked whether the approximation was adequate or not. Notably, in the iLQR-FD tasks, the Iterative Error method obtained the highest CR of all the key-point methods. However, the optimisation time reductions were

not as significant as one would expect from the percentage of derivatives computed, due to this formulation issue. There were improvements that could theoretically be made to the Iterative Error method, but fundamentally, some level of sequential computation is required by this key-point method.

Another potential limitation of the Iterative Error method is its susceptibility to being misled by certain function behaviours. For example, consider approximating a sinusoidal function using only the start and end points of a full cycle as key-points. The midpoint, where the function returns to zero, may appear to yield a low approximation error when compared to the linear segment. However, this result can be deceptive: despite the low local error, a straight line is clearly an inadequate representation of the underlying oscillatory behaviour. This highlights a potential blind spot in the algorithm, where mid-point evaluations may fail to capture significant curvature or higher-order variations in the trajectory. It should be noted that, despite this possible limitation, the Iterative Error method generally performed quite well in terms of CR for the tasks in this chapter. The main issue was the lack of reduction in optimisation time due to the natural introduction of sequential computation.

The methods proposed in this chapter are particularly applicable to MPC. In MPC it is typically more important to optimise to an acceptable solution quickly, as opposed to an optimal solution slowly. This is due to the fact that the cost landscape is ever changing during MPC and by the time an optimisation iteration is complete, the system has fundamentally moved on from where it was when optimisation began. This chapter has shown that the proposed key-point methods work in a short-horizon optimisation setting on both the Walker and Mini-cheetah tasks. Whilst these tasks were completed in a synchronous environment, the intended result was to demonstrate that optimisation remains successful when using the key-point methods instead of the baseline, and that the possible control frequency is significantly increased.

One limitation of the proposed key-point methods is that the parameters of these methods need to be tuned per task which can sometimes be time-consuming, especially when trying to find an ideal trade-off between optimisation time and performance. The general method of tuning parameters for tasks in this work is shown in Fig. 4.7. The parametrisation of the key-point method was selected to be non-greedy initially, and then it was iteratively made increasingly greedy until a notable degradation in performance was observed (in terms of CR). This process was performed on small subset of tasks before being evaluated over the full set of tasks start/goal states to show generalisation.

## 4.8 Conclusion

This chapter has outlined a general method for how trajectory optimisation can be performed faster by not paying the *full* computational cost of computing dynamics derivatives. The methods proposed involve only computing dynamics derivatives *accurately* at certain time-steps over a trajectory (which are referred to as key-points), the remainder of the dynamics derivatives are then approximated via simple linear interpolation. Three “intelligent” key-point selection methods have been proposed and their performance evaluated on a wide variety of tasks. Depending on the task, this chapter has shown that the proposed key-point methods can greatly reduce the optimisation time required without significant loss in the quality of the final computed trajectory. Even in situations where the quality of the computed trajectory may be worse, this trade-off may still be beneficial, especially when one considers controlling robots with MPC, where fast optimisation times are more valuable than converging to optimal solutions due to the ever changing cost landscape of executing controls on real robotic hardware.

One of the strengths of this work is the demonstration that the proposed methods can generalise to a variety of trajectory optimisation tasks, as well as to two different trajectory optimisation algorithms (iLQR and SCVX), and two different methods for computing dynamics derivatives in physics simulators (FD and AD). Although no experiments were conducted on trajectory optimisation methods that utilise higher-order dynamics information (such as DDP), the methods presented here are expected to be extendable to such approaches and may yield similar optimisation time reductions.

There are several limitations of this work which could lead to future research areas:

- Firstly, the proposed key-point methods require some level of parameter tuning to perform well. This may be a natural consequence of these methods but it would be advantageous if key-point methods could be tuning-free.
- On occasion, these methods can fail achieve satisfactory levels of CR when compared to the baseline method. This can sometime happen when the key-point methods miss some key information in the dynamics derivatives due to a unlucky poor approximation. A possible method around this would be to create some mechanism that if optimisation fails too early, the key-point methods can be *relaxed* allowing more dynamics derivatives to be computed accurately, to hopefully overcome this issue.
- Hardware experiments using a conventional MPC implementation are not performed. The reason for this was due to issues in controlling the real robotic hardware using direct torque control. The simulated robot did not seem to

be a good digital clone of the real robot when using direct torque control. As a result, a position controller that used some form of PID control to follow a desired trajectory was used instead. It is difficult to perform short-horizon MPC with a position controller due to small differences between the target and current joint positions do not convert to torques that can overcome joint friction.

A primary future research direction could be in regards to the key-point parameter tuning issues. During this work, a concept for automatically adjusting key-point parametrisation during MPC was experimented with. The core idea was that during successful MPC optimisation iterations, the key-point parametrisation could be made more “greedy” (i.e. use fewer key-points) whereas when optimisation fails to improve the trajectory, the key-point parametrisation would be made less “greedy”. This general idea could be a good method to address the issues of tuning key-point parameters automatically.



# Chapter 5

## Online State Vector Reduction during Model Predictive Control

### Chapter Deliverables

**Video:** [https://www.youtube.com/watch?v=8K\\_qfvb4lMI](https://www.youtube.com/watch?v=8K_qfvb4lMI)

**Source Code:** <https://github.com/DMackRus/iLQR-SVR>

### 5.1 Introduction

Trajectory optimisation algorithms are capable of synthesising complex motion to solve a variety of challenging robotic manipulation tasks [63, 69, 126, 96, 102, 119]. 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 dimensionality make it difficult to perform closed-loop MPC in these high-dimensional systems.

This chapter is concerned with trying to reduce the dimensionality of these systems dynamically so that closed-loop MPC can be achieved. The key insight is that, in a variety of these high-dimensional tasks, a large number of DoFs are not relevant to the problem at all times during task execution. This intuition makes sense on 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. 5.1 shows a trajectory generated by the method proposed in this chapter 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

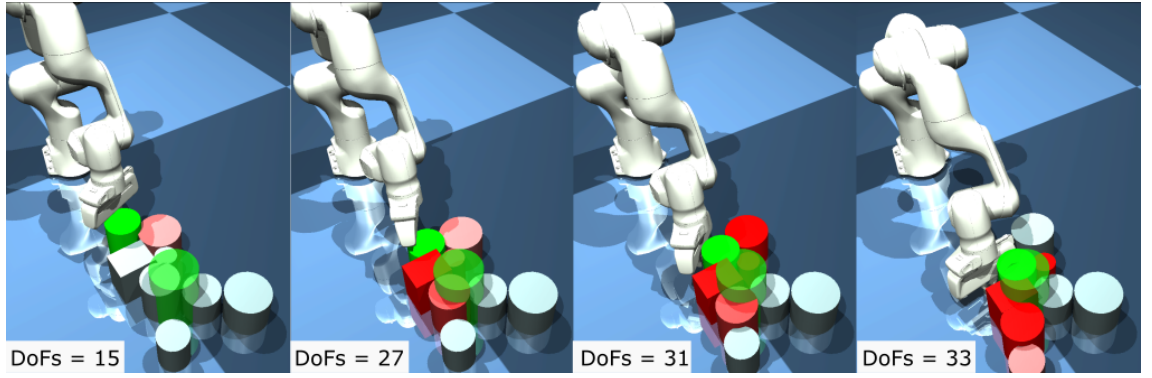


Figure 5.1: A sequence of snapshots showing an example MPC trajectory generated by the method introduced in this chapter. The task is to push the green cylinder to a goal region (the green transparent cylindrical region) whilst minimally disturbing some clutter objects. The full number of DoFs in this system is 55. The 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.

considered in the system state, in addition to the robot DoFs. Instead, this chapter proposes 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 [12, 59]. These can be pre-defined, such as inverted-pendulum models, or learned models given a task description [22]. 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. 5.1, since the full model can include an arbitrary number, shape, and configuration of objects, which are unknown beforehand. That is why this work considers a method of reducing dimensionality *online* during task execution, given a task instance. Furthermore, the reduced state can be different for different stages of the task, as shown in the different snapshots in Fig. 5.1.

An important aspect of this work is that a reduced *dynamics* model of the system is not extracted; rather, only a reduced state vector to be used in optimisation is extracted. The *full* dynamics model is still used for system integration, whereas the reduced state vector is what is considered for how to optimise a trajectory. This chapter shows 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., iLQR [73]. There are two parts of the iLQR algorithm that can benefit from a reduced optimisation state vector; these parts are computing dynamics derivatives and performing the backwards pass. As discussed in Chapter 4, computing dynamics derivatives is computationally costly. That chapter shows how the computational cost could be reduced by only paying the expensive cost of FD or AD at *key-points*, and then approximating the remainder of dynamics derivatives via linear interpolation. This chapter shows how reducing the dimensionality can reduce this computational cost as well, by computing *no* dynamics derivatives for *certain* DoFs in the system. The backwards pass can also become computationally expensive for high-dimensional systems, due to it involving many matrix-matrix multiplication computations. By reducing the optimisation state vector, the size of these matrices is smaller, making the backwards pass take less computation time as well. This chapter presents modifications to the iLQR algorithm, which enables it to be used with a reduced optimisation state vector and make significant time gains, while the forward/dynamics model of the system still uses the full state inside a physics simulator, namely MuJoCo [129].

This chapter proposes and compares different methods to identify the relevant DoFs of the system. The first proposed method is a *naive* method, which considers a DoF relevant if and only if it directly 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, more intelligent methods that try to identify the relevant DoFs for the current state and around the current nominal trajectory are investigated. These methods use the LQR gain matrix,  $\mathbf{K}$ , which relates how changes in the current state should result in changes in current controls, for optimal behaviour. Two methods are proposed that use the  $\mathbf{K}$  matrix: The first method directly uses the values in the columns of  $\mathbf{K}$  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  $\mathbf{K}$  to identify the principal axes and use them to identify the most important state elements. These three methods are compared to two baselines: A vanilla iLQR implementation that uses the full state vector at all times, and finally a method that *randomly* chooses subsets of the state vector during optimisation. The results of this work show that the  $\mathbf{K}$ -informed methods can identify the relevant DoFs of the task at different points during task execution (as can also be seen in Fig. 5.1).

It is important to clarify why the proposed methods are particularly useful for MPC. Since optimisation is performed with a reduced state vector, each individ-

ual optimisation iteration is expected to perform slightly worse when compared to the baseline. However, the proposed methods offer advantages for two key reasons. Firstly, due to the significantly reduced optimisation time, more optimisation iterations can be completed within a given time period, compounding the benefits of previous optimisation iterations. Secondly, the methods exhibit reduced *policy-lag*. In MPC, every time optimisation occurs from a specific state, by the time optimisation has finished, the system will have moved on past that state. This means that the optimal control trajectory that has just been computed is stale. Depending on how large the policy-lag is, this can have significant detrimental effects on the real control performance of the robot.

### 5.1.1 Contributions

This chapter:

- Proposes a general MPC approach that changes the elements inside the state vector *online* to reduce optimisation times, and therefore policy-lag.
- Proposes and compares different methods to identify the relevant DoFs inside the state vector for a given task.
- Presents modifications to iLQR, to perform trajectory optimisation with a reduced state vector.
- Evaluates the proposed methods on three high-dimensional non-prehensile manipulation tasks.

### 5.1.2 Organisation

The structure of this chapter is as follows: The general problem formulation is specified in Sec. 5.2. Sec. 5.3 presents the general methodology for how to perform optimisation with a reduced state vector, as well as how to determine what DoFs to keep in the reduced state vector. Results for the proposed methods are shown in Sec. 5.4 and finally Sec. 5.5 concludes this chapter.

## 5.2 Problem Formulation

As discussed in Chapter 3, a discrete time dynamics system is considered:

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t), \quad (5.1)$$

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

There is some running cost function:

$$l(\mathbf{x}_t, \mathbf{u}_t) = (\mathbf{x}_t - \tilde{\mathbf{x}}_t)^\top \mathbf{W}(\mathbf{x}_t - \tilde{\mathbf{x}}_t) + \mathbf{u}_t^\top \mathbf{R} \mathbf{u}_t, \quad (5.2)$$

where  $\mathbf{W}$  and  $\mathbf{R}$  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  $\mathbf{U} \triangleq (\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_{T-1})$  is applied from some initial starting state  $\mathbf{x}_0$ :

$$J(\mathbf{x}_0, \mathbf{U}) = l_f(\mathbf{x}_T) + \sum_{t=0}^{T-1} l(\mathbf{x}_t, \mathbf{u}_t), \quad (5.3)$$

where  $l_f$  is the terminal cost function of the same form as Eq. 5.2 but with a different final state matrix  $\mathbf{W}_f$  and no control cost penalisation as this is the final state.

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:

$$\mathbf{U}^*(\mathbf{x}_0) = \arg \min_{\mathbf{U}} J(\mathbf{x}_0, \mathbf{U}). \quad (5.4)$$

This work is interested in using trajectory optimisation within an MPC framework. During each MPC iteration, the optimisation problem from Eq. 5.4 is solved, and the optimised controls are executed on the *real*<sup>1</sup> 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. 5.4 is solved, and it has a significant effect on the performance of the real robot.

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<sup>2</sup>. `MPC_Cost` is used to quantify method performance under an MPC framework:

$$\text{MPC\_Cost} = l_f(\underline{\mathbf{x}}_Y) + \sum_{t=0}^{Y-1} l(\underline{\mathbf{x}}_t, \mathbf{u}_t). \quad (5.5)$$

The expression above is similar to Eq. 5.3 and uses the same cost functions  $l$  and  $l_f$ . The difference is that, while Eq. 5.3 is evaluated over the states  $\mathbf{x}_t$  as predicted by the dynamics model, Eq. 5.5 is evaluated over the real states achieved by the system. Furthermore, while Eq. 5.3 sums over the optimisation horizon  $T$ , Eq. 5.5 sums over the complete duration of the execution,  $Y$ .

In this work, the aim is to reduce the `MPC_Cost` by solving Eq. 5.4 faster, and therefore achieving a higher control frequency during MPC.

<sup>1</sup>This work uses a simulator to also represent the *real* system.

<sup>2</sup>The controller is stopped if a *Task timeout* is reached, or if some success condition is achieved.

### 5.2.1 Definitions

Chapter 3 discussed what elements are traditionally included inside the optimisation state vector. These definitions will now be briefly formalised.

This work considers manipulation tasks consisting of a robot with  $n_q$  joints and multiple objects. There are  $N_O$  rigid objects and  $N_S$  deformable/soft objects. Rigid objects 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.  $\mathcal{F}$  refers to the *set* of all DoFs in the system. Therefore,

$$|\mathcal{F}| = n_q + 6N_O + 3 \sum_{i=1}^{N_S} N_{S_i}. \quad (5.6)$$

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

The optimisation state vector considered both positional and velocity elements for each DoF. 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  $n_u$ ,  $\mathbf{u}_t \in \mathbb{R}^{n_u}$ , and does not change size. The set of DoFs *not* currently in the reduced set of DoFs is denoted as  $\mathcal{L}$ , i.e.,  $\mathcal{L} = \mathcal{F} \setminus \mathcal{C}$ .

## 5.3 Method

A general asynchronous MPC scheme (similar to Howell et al. [50]) can be seen in Alg. 9. 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)<sup>1</sup>. Depending on how long the optimiser takes to optimise the control sequence  $\mathbf{U}$ , 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*.

This work augments 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. The reduced set of DoFs is initialised in line 1: this work simply sets  $\mathcal{C} = \mathcal{F}$ , but if a better heuristic is available that can be used instead. Before every optimisation call, a number of unused DoFs in  $\mathcal{L}$  are identified to be reintroduced to the reduced set of DoFs  $\mathcal{C}$  on line 6. A control sequence is then optimised using this reduced set of DoFs and after computing the control sequence, a set of DoFs that were unimportant to the previous trajectory optimisation problem are identified on line 10. These DoFs are then removed from the reduced set of DoFs.

<sup>1</sup>In MPC, only one iteration of optimisation is performed, not optimisation until convergence.

There are three important components of this general approach. Firstly, a method of optimising a trajectory using only a reduced set of DoFs is needed (Alg. 9, line 9). Sec. 5.3.1 discusses this and formulates performing iLQR [73, 126] on the subset of reduced DoFs. Secondly, a method of determining which DoFs can be removed is required (Alg. 9, line 10).

Sec. 5.3.2 outlines different methods to identify the DoFs to be removed. Finally, a method for reintroducing DoFs into the reduced set of DoFs is required (Alg. 9, line 6). This work proposes 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 this line of work was not pursued.

### 5.3.1 Optimise

This section discusses the specific augmentations made to the iLQR [73, 126] algorithm to operate on the reduced set of DoFs  $\mathcal{C}$ . The high level overview is that derivative computation and backwards pass calculations are only performed for the reduced set of DoFs. The MuJoCo physics simulator is used to model the full system. When performing the forwards-rollouts, the full set of DoFs are updated using

---

#### Algorithm 9 MPC with state vector reduction (asynchronous)

---

```

1:  $\mathcal{C} \leftarrow \text{INITIALISESUBSET}(\mathcal{F})$ 
2:  $\mathcal{L} \leftarrow \mathcal{F} \setminus \mathcal{C}$ 
3:  $\mathbf{U} \leftarrow \text{INITIALISECONTROLS}()$ 
   Optimiser Asynchronous (Planning)
4:   while task not complete do
5:      $\mathbf{x}_0 \leftarrow \text{GETCURRENTSTATE}()$ 
6:      $\text{dofs\_to\_add} \leftarrow \text{IDENTIFYDOFSTOADD}(\mathcal{L})$ 
7:      $\mathcal{C} \leftarrow \mathcal{C} \cup \text{dofs\_to\_add}$ 
8:      $\mathcal{L} \leftarrow \mathcal{L} \setminus \text{dofs\_to\_add}$ 
9:      $\mathbf{U}, \mathbf{K}^{\mathcal{C}} \leftarrow \text{OPTIMISE}(\mathbf{x}_0, \mathbf{U}, \mathcal{C})$ 
10:     $\text{dofs\_to\_remove} \leftarrow \text{IDENTIFYDOFSTOREMOVE}(\mathbf{K}^{\mathcal{C}})$ 
11:     $\mathcal{C} \leftarrow \mathcal{C} \setminus \text{dofs\_to\_remove}$ 
12:     $\mathcal{L} \leftarrow \mathcal{L} \cup \text{dofs\_to\_remove}$ 
   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

```

---

MuJoCo, as well as computing the total running cost of the trajectory. This iLQR adaptation is referred to as *iLQR with state vector reduction* (iLQR-SVR).

iLQR-SVR uses a first order approximation of the system dynamics where Eq. 5.1 is written as Eq. 5.7.

$$\mathbf{x}_{t+1}^c = \mathbf{A}_t^c \mathbf{x}_t^c + \mathbf{B}_t^c \mathbf{u}_t, \quad (5.7)$$

where  $\mathbf{A}_t^c = \partial f(\mathbf{x}_t^c, \mathbf{u}_t) / \partial \mathbf{x}_t^c$  and  $\mathbf{B}_t^c = \partial f(\mathbf{x}_t^c, \mathbf{u}_t) / \partial \mathbf{u}_t$ . iLQR-SVR also requires a first and second order approximation of the cost derivatives with respect to the state and control vector  $(l_{\mathbf{x}}^c, l_{\mathbf{xx}}^c, l_{\mathbf{u}}, l_{\mathbf{uu}})$ .

The computation of the dynamics derivatives  $\mathbf{A} \triangleq (\mathbf{A}_0, \mathbf{A}_1, \dots, \mathbf{A}_{T-1})$  and  $\mathbf{B} \triangleq (\mathbf{B}_0, \mathbf{B}_1, \dots, \mathbf{B}_{T-1})$  is often the bottleneck in gradient-based trajectory optimisation. These derivatives usually need to be computed via computationally costly *finite-differencing*. FD requires evaluating the system dynamics for every DoF in the system as well as any control inputs. By only computing dynamics derivatives for the reduced state ( $\mathbf{A}^c$  and  $\mathbf{B}^c$ ), the number of dynamics evaluations per time-step is reduced to  $2|\mathcal{C}| + n_u$  instead of  $2|\mathcal{F}| + n_u$ .

Using these approximations, optimal control modifications can be computed recursively using the dynamic programming principle [73, 126]. 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 5.8 - 5.10 from  $t = T$  to  $t = 0$ . The following equations are rewritten to operate on the reduced set of DoFs:

$$Q_{\mathbf{x}}^c = l_{\mathbf{x}}^c + (\mathbf{A}^c)^\top V_{\mathbf{x}}'^c, \quad (5.8a)$$

$$Q_{\mathbf{u}} = l_{\mathbf{u}} + (\mathbf{B}^c)^\top V_{\mathbf{x}}'^c, \quad (5.8b)$$

$$Q_{\mathbf{xx}}^c = l_{\mathbf{xx}}^c + (\mathbf{A}^c)^\top V_{\mathbf{xx}}'^c \mathbf{A}^c, \quad (5.8c)$$

$$Q_{\mathbf{uu}} = l_{\mathbf{uu}} + (\mathbf{B}^c)^\top V_{\mathbf{xx}}'^c \mathbf{B}^c, \quad (5.8d)$$

$$Q_{\mathbf{ux}}^c = l_{\mathbf{ux}}^c + (\mathbf{B}^c)^\top V_{\mathbf{xx}}'^c \mathbf{A}^c. \quad (5.8e)$$

At every time-step, these Q matrices can be used to compute an open-loop feedback term  $\mathbf{k}$  as well as a closed-loop state feedback gain  $\mathbf{K}$ :

$$\mathbf{k}_t = -Q_{\mathbf{uu}}^{-1} Q_{\mathbf{u}}, \quad (5.9a)$$

$$\mathbf{K}_t^c = -Q_{\mathbf{uu}}^{-1} Q_{\mathbf{ux}}^c. \quad (5.9b)$$

Finally the value function needs to be updated.

$$V_{\mathbf{x}}^c = Q_{\mathbf{x}}^c - Q_{\mathbf{u}} Q_{\mathbf{uu}}^{-1} Q_{\mathbf{ux}}^c, \quad (5.10a)$$

$$V_{\mathbf{xx}}^c = Q_{\mathbf{xx}}^c - Q_{\mathbf{u}} Q_{\mathbf{uu}}^{-1} Q_{\mathbf{ux}}^c. \quad (5.10b)$$



iLQR-SVR performs a forwards roll-out using the MuJoCo model, i.e., using the full non-linear system dynamics. The state vector is updated for the full set of DoFs whilst only computing control modifications based on the reduced set of DoFs, according to:

$$\hat{\mathbf{x}}_0 = \mathbf{x}_0, \quad (5.11)$$

$$\hat{\mathbf{u}}_t = \bar{\mathbf{u}}_t + \alpha \mathbf{k}_t + \mathbf{K}_t^c(\hat{\mathbf{x}}_t^c - \bar{\mathbf{x}}_t^c), \quad (5.12)$$

$$\hat{\mathbf{x}}_{t+1} = f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t). \quad (5.13)$$

Above,  $\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t$  denotes the new computed trajectory states and controls and  $\alpha$  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  $\mathcal{F}$ .

### 5.3.2 Reducing dimensionality

This section outlines how DoFs are identified to be removed from the current state vector. The core idea of these methods is leveraging information from the LQR gain matrices,  $\mathbf{K}$ , which 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  $\mathbf{K} \triangleq (\mathbf{K}_0, \mathbf{K}_1 \dots, \mathbf{K}_{T-1})$  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  $\mathbf{K}$ ) matrix. This is shown explicitly for a single time-step in Eq. 5.14.

$$\begin{bmatrix} \mathbf{u}(0) \\ \mathbf{u}(1) \\ \vdots \\ \mathbf{u}(n_u) \end{bmatrix} = \begin{bmatrix} \mathbf{K}^c(0,0) & \mathbf{K}^c(0,1) & \dots & \mathbf{K}^c(0,2|\mathcal{C}|) \\ \mathbf{K}^c(1,0) & \mathbf{K}^c(1,1) & \dots & \mathbf{K}^c(1,2|\mathcal{C}|) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{K}^c(n_u,0) & \mathbf{K}^c(n_u,1) & \dots & \mathbf{K}^c(n_u,2|\mathcal{C}|) \end{bmatrix} \begin{bmatrix} \hat{\mathbf{x}}^c(0) - \bar{\mathbf{x}}^c(0) \\ \hat{\mathbf{x}}^c(1) - \bar{\mathbf{x}}^c(1) \\ \vdots \\ \hat{\mathbf{x}}^c(2|\mathcal{C}|) - \bar{\mathbf{x}}^c(2|\mathcal{C}|) \end{bmatrix} \quad (5.14)$$

Parentheses notation is used to denote indexing inside the matrix, i.e.,  $\mathbf{K}(.,.)$ . The **columns** inside the  $\mathbf{K}$  matrix correspond to computing entire control vector modifications based on the deviation of a specific DoF from the nominal. This chapter reasons 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.

Two approaches of leveraging this information are considered, 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  $\mathcal{C}$ , then, a threshold parameter  $\rho$  is used 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  $\mathbf{K}$  matrix for each DoF and how much of an impact they have on the control vector modification. If the values inside the  $\mathbf{K}$  matrix are large for a particular DoF over the entire optimisation horizon, then that DoF is likely important to the trajectory optimisation problem currently. A `dof_importance` value for each DoF is computed using the following formula:

$$\text{dof\_importance}[j] = \sum_{t=0}^T \sum_{p=0}^m \left( |\mathbf{K}_t^c(p, j)| + |\mathbf{K}_t^c(p, j + |\mathcal{C}|)| \right) / T. \quad (5.15)$$

The summation is carried out over the number of controls in the control vector  $n_u$  as well as summing over all matrices over the optimisation horizon. The summation is also carried out over both columns corresponding to the positional and velocity element for the DoF  $j$ .

### SVD

The SVD method aims to identify the most influential combinations of DoFs based on the structure of the state-feedback gain matrices. For each time step  $t$ , the matrix  $\mathbf{K}_t \in \mathbb{R}^{n_u \times 2|\mathcal{C}|}$  is decomposed using singular value decomposition as follows:

$$\mathbf{K}_t = U_t \Sigma_t V_t^T, \quad (5.16)$$

where  $U_t \in \mathbb{R}^{n_u \times n_u}$  and  $V_t \in \mathbb{R}^{2|\mathcal{C}| \times 2|\mathcal{C}|}$  are orthogonal matrices, and  $\Sigma_t$  is a diagonal matrix containing the singular values  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_{n_u} \geq 0$ . The columns of  $V_t$  represent orthonormal directions in the state space (i.e., combinations of position and velocity terms) that contribute most to the control adjustments.

To measure the influence of each individual DoF, the method examines how strongly its associated state variables project onto the most dominant singular directions. Specifically, the first  $g$  singular values ( $g = 3$  in this work) and corresponding right-singular vectors are used to compute a `dof_importance` score as follows:

$$\text{dof\_importance}[j] = \sum_{t=0}^T \sum_{n=1}^g \left( (|V_t(j, n)| + |V_t(j + |\mathcal{C}|, n)|) \sigma_n \right) / T. \quad (5.17)$$

Again, parentheses  $V(.,.)$  are used to denote indexing inside the matrix. Here,  $V_t(j, n)$  denotes the  $j$ -th DoF's positional contribution to the  $n$ -th principal direction, and  $V_t(j + |\mathcal{C}|, n)$  the corresponding velocity component. The use of singular values  $\sigma_n$  as weights ensures that more important directions (as judged by their influence on control effort) contribute more to the importance score. Notably, both methods of computing `dof_importance` values normalise via the optimisation horizon so that  $\rho$  does not need to change dependent on the optimisation horizon used.

Both of the proposed methods use the `dof_importance` values to determine which DoFs to remove for Alg. 9 line 10. Any DoFs that have an importance value that is below  $\rho$  are removed from the current reduced set of DoFs  $\mathcal{C}$ . One exception to this is that the robot DoFs are not allowed to be removed by any of the methods; i.e., the robot DoFs always appear in the state vector.

## 5.4 Results

Simulation experiments are performed under an asynchronous MPC framework as described in Alg. 9. Results are provided for a variety of methods, there are two baseline methods:

- *iLQR-Baseline*: A baseline method, where iLQR is performed as normal on the full set of DoFs in the system.
- *iLQR-Rand*: A baseline method where iLQR is performed on a random reduced set of DoFs. Instead of using any intelligent method to determine which DoFs, this method randomly samples  $\theta$  DoFs to use (in addition to the robot DoFs) in every optimisation iteration<sup>1</sup>.

This chapter proposes three methods to evaluate against these baselines:

- *iLQR-Naive*: This method reduces the set of DoFs in the reduced set to only the DoFs that are directly considered in the cost function for the task.
- *iLQR-SVR-SVD*: This method dynamically changes the number of DoFs as described in Sec. 5.3.2-SVD.
- *iLQR-SVR-Sum*: This method dynamically changes the number of DoFs as described in Sec. 5.3.2-Summing.

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

<sup>1</sup>The purpose of this baseline is to show that simply reducing the number of DoFs is not enough to perform well, there needs to be some intelligent thought behind it.

### 5.4.1 Task definition

Three high-dimensional non-prehensile manipulation tasks were selected. The main idea behind these tasks was to create tasks that were not too mechanically challenging but were high-dimensional to see if the proposed 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, it was 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).

The general description of the three tasks as well as their task parameters are outlined below.  $T$  is the optimisation horizon,  $\Delta t$  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. For all tasks, the initial trajectory that was used to warm-start the first optimisation iteration of MPC was a straight line trajectory of the robotic end-effector to the manipulated object.

**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. 5.1). The task 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. 5.2). The task 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. 5.3). The task parameters were as follows;  $T = 100$ ,  $\Delta t = 0.004$ ,  $Y = 1000$ , Slowdown factor = 5.

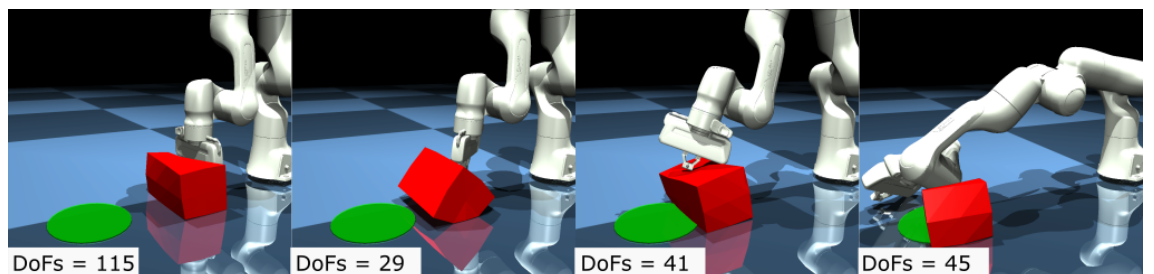


Figure 5.2: A sequence of snapshots showing an example trajectory for the **soft** task. The objective is to push the red deformable object to the green flat circle on the floor. The full number of DoFs in this system is 115.

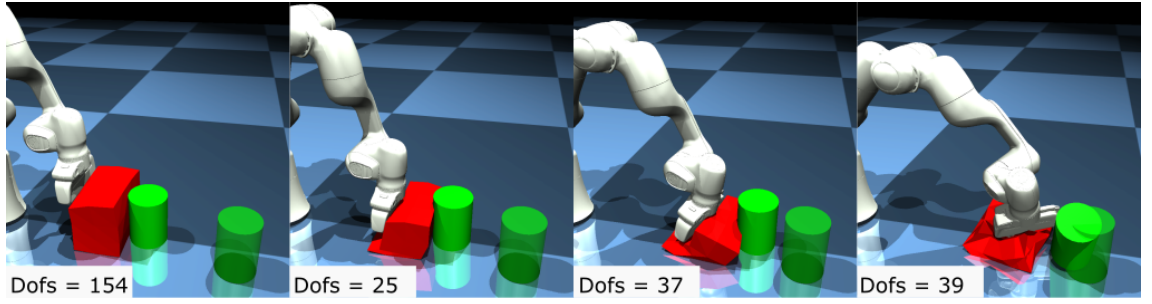


Figure 5.3: A sequence of snapshots showing an example trajectory for the **soft rigid** task. The objective is to move the green cylinder to the goal region (the transparent cylindrical region) with a high-dimensional soft body being placed between the robot end-effector and the green cylinder. The full number of DoFs in this system is 154.

### 5.4.2 Asynchronous MPC results

This section evaluates the performance of the proposed methods on the three outlined tasks. Earlier, Eq. 5.5 defined the **MPC\_cost** as the cost of the *actual* trajectory executed in the real system. This is used as the main metric of success in this section. This section shows that the proposed 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 5.1 shows the results from the asynchronous MPC experiments. 100 runs were performed for the clutter task and 20 runs for the soft and soft rigid tasks. The first values in the table are the mean values and the second values are 90% confidence intervals. It should be noted that outliers were removed 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, 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 these outliers were removed. Finally, cost values were normalised with respect to the *iLQR Baseline* method.

The results show that, for all three tasks, the proposed 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 the dynamic methods, they manage to lower the **MPC cost** by 14%. Importantly, it is shown that simply limiting the size of the state vector does not achieve the same level of performance as the intelligent methods as *iLQR-Rand* only reduces the **MPC cost** by 6%.

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 **clut-**

Table 5.1: 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	$1.00 \pm 0.034$	$1.00 \pm 0.080$	$1.00 \pm 0.096$
	Opt time (ms)	$485.71 \pm 5.16$	$620.97 \pm 7.35$	$4322.39 \pm 366.65$
	Num DoFs (preset)	55	115	154
iLQR-Rand $\theta = 5$	MPC cost	$0.94 \pm 0.033$	$1.14 \pm 0.104$	$0.74 \pm 0.071$
	Opt time (ms)	$148.91 \pm 1.69$	$79.12 \pm 4.13$	$881.54 \pm 7.67$
	Num DoFs (preset)	12	12	12
iLQR-Naive	MPC cost	$0.89 \pm 0.035$	$0.92 \pm 0.067$	<b><math>0.38 \pm 0.066</math></b>
	Opt time (ms)	$219.07 \pm 2.11$	$395.52 \pm 7.71$	$583.06 \pm 85.72$
	Num DoFs (preset)	23	79	9
iLQR-SVR-SVD $\theta = 10, \rho = 1$	MPC cost	<b><math>0.86 \pm 0.033</math></b>	<b><math>0.90 \pm 0.065</math></b>	$0.88 \pm 0.071$
	Opt time (ms)	$335.16 \pm 13.61$	$367.18 \pm 49.60$	$2320.80 \pm 392.91$
	Num DoFs	$31.85 \pm 1.12$	$72.55 \pm 9.37$	$85.19 \pm 18.32$
iLQR-SVR-SVD $\theta = 10, \rho = 500$	MPC cost	$0.91 \pm 0.036$	$0.91 \pm 0.069$	$0.66 \pm 0.063$
	Opt time (ms)	$237.79 \pm 5.76$	$116.20 \pm 4.73$	$1051.20 \pm 32.55$
	Num DoFs	$21.06 \pm 0.33$	$23.32 \pm 0.77$	$24.72 \pm 0.17$
iLQR-SVR-SVD $\theta = 5, \rho = 1$	MPC cost	$0.91 \pm 0.036$	$0.90 \pm 0.061$	$0.89 \pm 0.068$
	Opt time (ms)	$271.21 \pm 11.41$	$251.90 \pm 38.30$	$1296.29 \pm 184.13$
	Num DoFs	$24.38 \pm 1.00$	$49.59 \pm 7.34$	$38.69 \pm 8.66$
iLQR-SVR-Sum $\theta = 10, \rho = 1$	MPC cost	$0.86 \pm 0.032$	$0.94 \pm 0.073$	$0.77 \pm 0.074$
	Opt time (ms)	$287.05 \pm 10.91$	$136.18 \pm 10.92$	$1130.83 \pm 54.68$
	Num DoFs	$29.82 \pm 1.01$	$26.68 \pm 2.08$	$29.10 \pm 2.92$
iLQR-SVR-Sum $\theta = 10, \rho = 500$	MPC cost	$0.87 \pm 0.029$	$1.00 \pm 0.081$	$0.70 \pm 0.071$
	Opt time (ms)	$192.59 \pm 2.08$	$101.15 \pm 4.73$	$997.71 \pm 32.55$
	Num DoFs	$18.69 \pm 0.10$	$18.20 \pm 0.11$	$24.09 \pm 0.46$
iLQR-SVR-Sum $\theta = 5, \rho = 1$	MPC cost	$0.94 \pm 0.042$	$0.97 \pm 0.079$	$0.75 \pm 0.066$
	Opt time (ms)	$218.45 \pm 8.38$	$114.74 \pm 13.76$	$935.45 \pm 37.03$
	Num DoFs	$21.91 \pm 0.86$	$19.94 \pm 1.87$	$20.21 \pm 0.91$

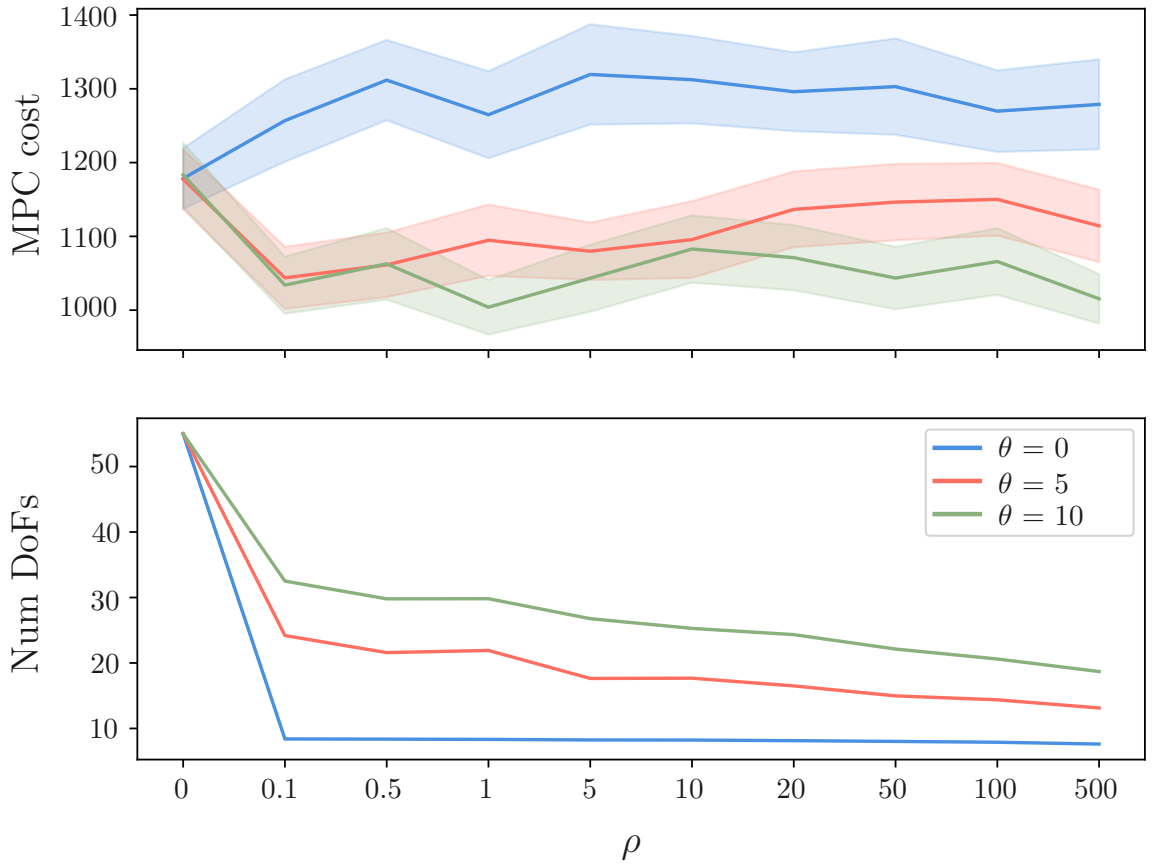


Figure 5.4: Top plot shows the `MPC cost` averaged over 100 runs for the clutter task. The lightly shaded area shows the 90% confidence interval range. Bottom plot shows the average number of DoFs in the state vector. Three different parameterisations of  $\theta$  were used with scaling values for  $\rho$ .

**ter** 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 the 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. The K-informed methods are capable of deducing this automatically.

It is important to recognise that only three parameterisations of  $\theta$  and  $\rho$  are shown in these results and are kept the same for all three tasks to test the generalisability of the proposed methods. Better results could have been achieved if these values were tuned to each task specifically.

This section also experimented with the effects of modifying the values for  $\theta$  and  $\rho$  for the clutter task, the results of this can be seen in Fig. 5.4. Three different values for  $\theta$  are used and the value of  $\rho$  is scaled between 0 and 500 (please note the values are not spaced equally). Firstly, when  $\rho$  is set to zero, *iLQR-SVR* operates identically to *iLQR-Baseline*. This can be seen clearly in Fig. 5.4 as all three data points have nearly identical `MPC Costs`. As the value for  $\rho$  is increased, it makes

the proposed 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  $\rho$  increases.

When  $\theta$  is set to zero, no DoFs are ever re-introduced to the 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 the proposed methodology which is that DoFs importance can change over time during a task, noticeably in Fig. 5.4, this value of  $\theta$  performs noticeably worse compared to the other two parameterisations of  $\theta$ .

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

## 5.5 Conclusion

This chapter has discussed how dimensionality reduction can be used *online* as a method of reducing the computational cost of trajectory optimisation for high-dimensional tasks, such as manipulation in clutter or manipulation of deformable objects. Several contributions have been made in this work: Firstly, this chapter has outlined a general method for changing the size of a state vector that is used within MPC *online*. Secondly, a heuristic method of reducing the number of DoFs considered in trajectory optimisation has been outlined and shown that the proposed methods can increase the performance of a robot under an asynchronous MPC scheme.

Importantly, in this work, the methods that have been outlined have been *task agnostic*. They do not use any specific task related information to determine how to remove or add DoFs from/into the optimisation state vector. As such, it would be fairly trivial to take the general approach and apply it to a specific robotic system where a more informed heuristic could be used to inform how important certain DoFs are.

The general methods proposed in this chapter could be very important in using trajectory optimisation for robotic manipulation in high-dimensional scenes, particularly for manipulation in clutter where there are lots of objects, but not all of these objects are relevant to the task at all times. In such situations, it could be advantageous to consider a more informed heuristic for how to consider what DoFs are included in the state vector, particularly with regards to adding DoFs back into the state vector.

This work has some limitations, some of the most notable of these are:

- Using a thresholding scheme to determine what DoFs should be kept or removed can be problematic and requires some level of manual tuning. These



parameters can also be somewhat task dependent.

- The methods considered in this chapter are axis-aligned forms of dimensionality reduction, where each degree of freedom (DoF) is either fully considered or not considered at all. While simple and interpretable, these methods cannot capture lower-dimensional correlations across multiple DoFs, which may be useful for certain manipulation tasks.
- The proposed methods are limited to deliberately not remove robot joints from the state vector. This limitation was a design choice for the tasks investigated in this chapter as most of the robot joints should be considered at most times during task execution. However for tasks like locomotion, the proposed methods would not be useful with this limitation.

Finally, regarding future avenues of work, the primary focus should be on addressing the previously stated limitations. First, eliminating the need for manual tuning of a threshold parameter would be advantageous. One possible approach is to determine an optimal fixed size for the state vector. Rather than removing DoFs that fall below a threshold, the most important DoFs could be retained until the desired state vector size is reached. Although this still introduces a tunable parameter—the optimal size of the state vector—it is likely to be easier to tune than a threshold value.

There are certain tasks for which the current methods may not be well suited. For example, in tasks involving the scooping of many small objects from one location to another using a funnel, it is unlikely that any specific DoFs are significantly more important than others. However, individually considering all DoFs remains computationally inefficient. This issue could potentially be addressed by applying a dimensionality reduction technique that combines the relevant object states into a lower-dimensional representation.

Finally, an interesting direction for future work would be to remove the current limitation of excluding robot joints from state vector reduction and to apply the proposed methods to high-dimensional humanoid robots. Depending on the task, a humanoid robot may not need to explicitly consider all of its joints. For instance, when walking from one location to another, it may be unnecessary to account for the joints of the fingers—or even the arms. Conversely, during a manipulation task after reaching a goal location, considering leg joints might offer little benefit, and assuming a static stance could be advantageous. Similar ideas have been investigated previously under standard control paradigms [40] for locomotive tasks for highly redundant robots.

## Chapter 6

# Challenges of MPC on Real Robotic Hardware

### Chapter Deliverables

**Video:** [youtube.com/watch?v=Q1mxb1CMIKg](https://youtube.com/watch?v=Q1mxb1CMIKg)

**Controllers Code:** [github.com/roboticsleeds/panda\\_controllers](https://github.com/roboticsleeds/panda_controllers)

**Cobot Pump Code:** [github.com/roboticsleeds/cobot\\_pump\\_ros](https://github.com/roboticsleeds/cobot_pump_ros)

### 6.1 Introduction

The motivation of this chapter is to begin building the general packing/picking system that was outlined in Chapter 1. Specifically, creating such a system to operate on real robotic hardware, namely the Franka-Emika Panda.

To recap, the system that was motivated at the start of this thesis was one where a robotic manipulator could leverage contact-based interactions for efficient picking and packing operations. Let us consider a picking example where the desired object to be picked is occluded by other objects. A traditional method to handle this would be to move objects out of the way of the desired object by sequentially grasping and moving each object to a new location. A more efficient approach would be to use non-prehensile manipulation to move objects out of the way to clear space so that the desired object can be picked. Similarly, let us consider a packing example where the robot wants to pack an object at a location that is obstructed by other objects. Again, a traditional approach would be to sequentially grasp each obstructing object and move them out of the way, before grasping the desired object and placing it at the now cleared location. A more efficient approach would be to grasp the desired object and use non-prehensile manipulation to clear space by using the grasped object to push other objects out of the way.

In this chapter, to enable a robot to operate in such a way, trajectory opti-

misation is used with a general purpose physics simulator (namely MuJoCo [129]) to simulate contact-interactions MPC is used to account for the fidelity issues between the physics simulator and the real world. The previous two chapters have discussed how trajectory optimisation can be sped up for contact-based manipulation of robotic systems for gradient-based methods. This has either been done via using approximations (Chapter 4) or through the use of dimensionality reduction (Chapter 5). One of the purposes of these methods was to make trajectory optimisation faster so that MPC can be performed at higher control frequencies so that better control of real robotic systems can be achieved.

This thesis has already presented experiments on asynchronous MPC in simulation in Chapter 5. In theory, translating the MPC code from simulation to the real robot should be straightforward. Instead of sending controls to a simulated robot and relying on the physics simulator to integrate the system dynamics, the controls are sent to the real robot instead and sensors are queried to obtain the real-world state. However, this transition was not as straightforward. Despite using a popular dynamics model of the Franka Panda arm [68], no system identification is perfect, and there were significant discrepancies between the real and simulated accelerations induced by the same torques.

While MPC is designed to compensate for differences between a model and a real system, if the model deviates too much from reality, MPC alone cannot fully account for these discrepancies. Although better system identification could potentially improve the model, it is unclear how accurate the model would need to be for MPC alone to be sufficient. Therefore, the primary focus of this chapter is to explore the design of a robust control method that enables effective MPC-based control despite imperfect system identification.

Chapter 4 performed some real world experiments and faced the same issue. That chapter addresses this issue by using a position controller to enable better tracking of the optimised trajectory. However, this lead to a new issue that made controlling the robot using conventional short-horizon MPC problematic. The issue was that the commanded positions being sent to the robot were close to the robots current configuration; the resulting torques produced by the proportional controller were then not enough to overcome joint friction. Instead, an online-replanning method [4] was used, where longer-horizon trajectories were optimised and executed on the real robotic hardware. These longer horizon trajectories were executed until completion or until significant divergence was detected between the planned system state and the actual real world state. Significant divergence was defined by measuring the Euclidean distance between the planned and actual positions of all objects in a scene. This chapter performs more conventional MPC with a shorter optimisation horizon and the optimisation thread is constantly re-optimising the trajectory from the current state. To achieve this, two low level feedback controllers are implemented and evaluated: an *inverse dynamics controller* and a *torque proportional derivative* (*torque PD*) controller. As well as testing these two different controllers, a variety

of controller parameters and some MPC strategies are also evaluated to observe how they affect the control performance of the real robot.

Finally, this chapter concludes by using the best performing controller setup with MPC to control the robot to perform a packing-based task. This general task involves placing some object at a desired location, where the location is obstructed by other objects. The robotic manipulator needs to move the obstructing obstacles out of the way using non-prehensile manipulation so that the grasped object can be placed at the correct location. Some brief analysis is performed on how effective this method is in terms of time to complete the task and placement accuracy.

### 6.1.1 Contributions

This chapter:

- Frames the packing in clutter problem as an optimisation problem, where the robot wants to place a desired object at some desired location that is obscured by clutter. The proposed method enables the robot to use contact-interactions to clear the desired area to place the object efficiently.
- Investigates the effects of various controllers and MPC parameters on the control performance of the real Franka-Emika Panda robot for a contact-based manipulation task.

### 6.1.2 Organisation

The organisation of this chapter is as follows: Sec. 6.2 investigates two separate controllers for the Franka Panda as well as a variety of controller and MPC parameters and how they affect the control performance of the real robot. Then, Sec. 6.3 frames the packing problem as an optimisation problem and shows some results of the efficiency of this method. Results are shown for how well the proposed method can perform in terms of placement accuracy and amount of disturbance applied to the scene. Finally, Sec. 6.4 concludes this chapter.

## 6.2 MPC Performance

As discussed previously, the focus here is on developing a system capable of reliably controlling the real Franka Panda robot using MPC, despite the imperfect system identification present in the current model of the arm. Two different controllers are implemented and tested to help address the fidelity gap between the simulated and the real robot. In addition to evaluating these controllers, various controller-dependent parameters and MPC parameters are also tested in an effort to improve the real robot's control performance.

The general asynchronous MPC framework used to control the real robot can be seen in Alg. 10. There are two threads running in parallel, the first thread (*the Optimiser*) is constantly optimising (using iLQR) a short horizon trajectory from the current real world state. In Line 3, the current real world state is stored in  $\mathbf{x}_0$ , then a short horizon trajectory is optimised in Line 4, from the current state  $\mathbf{x}_0$  and the current warm-start guess for the optimal trajectory. It returns a newly optimised trajectory of controls  $\mathbf{U}$  and states  $\mathbf{X}$ , as well as a state-dependent feedback control law  $\mathbf{K}$  (as the iLQR optimisation algorithm is used). A flag is set in Line 5 to communicate with the other thread to update the control sequence to send to the robot. Importantly, in line 6, the algorithm chooses what will be the first control index to send to the robot. Due to the fact that the OPTIMISE function will take a non-insignificant amount of time, the first control in the trajectory will have been optimised for a state that the robot is no longer in. A general GETSTARTINDEX function is proposed with three different implementations of this function being evaluated, detailed in Sec. 6.2.2.

The second thread (*the Executor*) is constantly sending controls provided by the optimiser to the real robot, under whatever control scheme is currently being used. In line 8, whenever the new controls flag is set to true, the executor thread will update the current control index. In line 11, the robot state is extracted from the system state. This simply involves retrieving the positional and velocity values of the robot from the correct indices in the state vector. The acceleration term is computed using finite-differencing of the current robot velocities and the last robot velocities. Similarly, in line 12, the torques to send to the robot are also retrieved from the

---

**Algorithm 10** MPC real robot

---

```

1:  $\mathbf{U} \leftarrow \text{INITIALISECONTROLS}()$ 
   The Optimiser Thread
2:   while task not complete do
3:      $\mathbf{x}_0 \leftarrow \text{GETREALWORLDSTATE}()$ 
4:      $\mathbf{U}, \mathbf{X}, \mathbf{K} \leftarrow \text{OPTIMISE}(\mathbf{x}_0, \mathbf{U})$ 
5:     New_controls_flag = true
6:      $t_s = \text{GETSTARTINDEX}()$ 
   The Executor Thread
7:   while task not complete do
8:     if New_controls_flag then
9:        $t_c = t_s$ 
10:      New_controls_flag = false
11:       $\mathbf{q}_{desired}, \dot{\mathbf{q}}_{desired}, \ddot{\mathbf{q}}_{feedforward} = \text{GETROBOTSTATE}(\mathbf{X}_{t_c})$ 
12:       $\tau_{feedforward} = \text{GETROBOTTORQUES}(\mathbf{U}_{t_c})$ 
13:       $\text{SENDCOMMANDTOROBOT}(\tau_{feedforward}, \mathbf{q}_{desired}, \dot{\mathbf{q}}_{desired}, \ddot{\mathbf{q}}_{feedforward})$ 
14:       $t_c = t_c + 1$ 

```

---

control vector. In simulation, the torques that were computed by the optimiser include gravity compensation terms of the Franka Panda. The real Franka Panda automatically compensates for gravity, and as such, the gravity compensation needs to be subtracted from the control sequence. Finally, in line 13, the current control command is sent to the robot using the generic function `SENDCOMMANDTOROBOT`. This work investigates different methods off sending the commands to the robot, the details of this will be discussed in Sec. 6.2.1.

### 6.2.1 Controllers

This section will discuss two different controllers that were implemented for the Franka Panda, as well as other control modifications that are evaluated. The combination of the controller and any small adaptations make up a unique implementation of the function `SENDCOMMANTTOROBOT` from Alg. 10.

#### Inverse dynamics controller

The first controller is an inverse dynamics controller, this controller computes torques to apply to the real robot in the following two parts. Firstly a desired acceleration term  $\ddot{\mathbf{q}}_{desired}$  is computed:

$$\ddot{\mathbf{q}}_{desired} = \ddot{\mathbf{q}}_{feedforward} + \mathbf{K}_p(\mathbf{q}_{desired} - \mathbf{q}_{actual}) + \mathbf{K}_d(\dot{\mathbf{q}}_{desired} - \dot{\mathbf{q}}_{actual}), \quad (6.1)$$

where  $\ddot{\mathbf{q}}_{feedforward}$ ,  $\mathbf{q}_{desired}$  and  $\dot{\mathbf{q}}_{desired}$  are the commanded acceleration, position and velocity sent to the robot.  $\mathbf{q}_{actual}$  and  $\dot{\mathbf{q}}_{actual}$  are the actual position and velocity terms of the real robot. Finally,  $\mathbf{K}_p$  and  $\mathbf{K}_d$  are gain terms.

The desired acceleration is then processed through the inverse dynamics formula to compute the torques to apply to the robot:

$$\tau = M(\mathbf{q}_{actual})\ddot{\mathbf{q}}_{desired} + C(\mathbf{q}_{actual}, \dot{\mathbf{q}}_{actual})\dot{\mathbf{q}}_{actual} + G(\mathbf{q}_{actual}), \quad (6.2)$$

where  $M$ ,  $C$  and  $G$  are the mass, coriolis and gravity matrices respectively.

There are two alterations to this controller which are evaluated. Firstly, the effects of sending or ignoring the feed forward acceleration term. When the feed forward term is included, this is denoted as FF in Table 6.1. Secondly, when the feedforward acceleration term is sent to the robot, a low-pass filter is also tested to reduce high-frequency noise that can cause jerky motion in the real system. Before sending the acceleration to the real robot, a damped version is computed via the following:

$$\ddot{\mathbf{q}}_{feedforward} = \alpha \ddot{\mathbf{q}}_{feedforward} + (1 - \alpha) \ddot{\mathbf{q}}_{feedforward\_last}, \quad (6.3)$$

where  $\ddot{\mathbf{q}}_{feedforward\_last}$  is the last feed forward acceleration sent to the robot and  $0 \leq \alpha \leq 1$  is a scalar (this work uses  $\alpha = 0.2$ ). When a low pass filter is used, it is denoted as LP in Table 6.1.

### Torque PD controller

The second controller that is implemented was a torque PD controller. This controller is similar to the inverse dynamics controller in the sense that it also uses feedback based on the desired and actual positions and velocities. However, the commanded torques are computed in one step:

$$\tau = \tau_{feedforward} + \mathbf{K}_p(\mathbf{q}_{desired} - \mathbf{q}_{actual}) + \mathbf{K}_d(\dot{\mathbf{q}}_{desired} - \dot{\mathbf{q}}_{actual}), \quad (6.4)$$

where  $\tau_{feedforward}$  is the feed forward torque computed from the optimiser. This controller does not consider the dynamics of the arm, but instead relies on gain terms for the positions and velocity components to correct for deviations between the simulated and real robot.

There are three alterations that are tested for this controller. Similarly to the inverse dynamics controller, the effects of sending or omitting the feedforward term  $\tau_{feedforward}$  are evaluated. When  $\tau_{feedforward}$  is included, it is denoted by FF in Table 6.1. The impact of applying a low-pass filter is also examined for this controller. However, unlike the inverse dynamics controller, the low-pass filter is applied on the controller side, meaning that the actual torques sent to the robot are filtered as opposed to the feed forward torques, i.e:

$$\tau = \alpha\tau + (1 - \alpha)\tau_{last}, \quad (6.5)$$

where  $\tau_{last}$  are the last torques sent to the real robot. Again, when a low pass filter is applied, this is denoted as LP in Table 6.1.

Finally, the effects of using a torque rate limiter are also tested, again implemented on the controller side. A torque rate limiter ensures that the torques sent to the robot can not change faster than some preset limit:

$$\tau = \tau_{last} + \text{clamp}(\tau - \tau_{last}, -\dot{\tau}_{max}, \dot{\tau}_{max}), \quad (6.6)$$

where  $\dot{\tau}_{max}$  is the maximum allowed rate of change of the commanded torques. When the torque rate limiter is used, this is denoted as TRL in Table 6.1.

### 6.2.2 Time indexing methods

This section describes different implementations of the function `GETSTARTINDEX` from Alg. 10. To reiterate why the choice of the starting control index is important, for contact-based manipulation, the `OPTIMISE` function from Alg. 10 takes a non-negligible amount of time. As a result, by the time a new optimal control sequence has been computed, the initial control was optimised for a state that no longer represents the state of the scene. This effect (commonly referred to as policy-lag) worsens as the optimisation time increases, which scales with both the dimensionality of the optimisation problem and the optimisation horizon.

This chapter proposes three methods for selecting the starting control index when applying controls from the optimiser to the real robot:

- **T0:** Simply select the first control index and ignore the effects of policy lag.
- **Opt Time:** Measure the duration of the OPTIMISE function and convert that time into the corresponding control index.
- **Error:** Iterate through the nominal state trajectory produced by the optimiser and compute the error between each state and the current real robot state. Select the control index corresponding to the state that is closest to the real robot's current state.

### 6.2.3 Results

This section evaluates the control performance of the different controllers, controller adaptations and MPC methods. A unique combination of these three concepts is referred to as a *control parametrisation*.

For the controller performance evaluations, a packing task is set up where the objective is to carry a grasped object using a vacuum gripper and place it at a desired location. These initial evaluations are conducted with no obstacles obstructing the goal region, in order to isolate and assess the control performance of the arm under collision-free motion.

For each control parametrisation, the task is executed five times from the same initial robot arm configuration and goal position of the grasped object. Execution continues until one of the following conditions is met: a task timeout is reached (2000 control steps applied), the task is successfully completed (determined by the distance between the grasped object and its desired position falling below a pre-defined threshold), or the robot is stopped prematurely (either manually due to observed unsafe behaviour, or automatically by the internal Franka Panda firmware under various fault conditions). Under these testing conditions, a variety of data and performance metrics are collected for each control parametrisation. The results are presented in Table 6.1. The collected data and metrics include the following:

- **Pos Error:** The positional error between commanded positions  $\mathbf{q}_{desired}$  and the actual positions of the robot  $\mathbf{q}_{actual}$  over the task execution, normalised by the length of the trial.

$$\text{Pos Error} = \frac{1}{N} \sum_{t=0}^N |\mathbf{q}_{desired,t} - \mathbf{q}_{actual,t}|, \quad (6.7)$$

where  $N$  is the length of the individual task.

- **Vel Error:** The velocity error between commanded velocities  $\dot{\mathbf{q}}_{desired}$  and the actual velocities of the robot  $\dot{\mathbf{q}}_{actual}$  over the task execution, normalised by the



length of the trial.

$$\text{Vel Error} = \sum_{t=0}^N |\dot{\mathbf{q}}_{desired,t} - \dot{\mathbf{q}}_{actual,t}|/N. \quad (6.8)$$

- **Torque Grad:** The summation of the gradient of the actual torques sent to the robot, normalised by the length of the trial.

$$\text{Torque Grad} = \sum_{t=1}^N |\tau_t - \tau_{t-1}|/N. \quad (6.9)$$

- **Cost:** The cost of the actual trajectory of states and controls from the real robot, when ran through the cost function used for optimisation.

$$\text{Cost} = \sum_{t=0}^N l(\mathbf{x}_t, \mathbf{u}_t). \quad (6.10)$$

- **Task Length:** The length of the individual task.

$$\text{Task Length} = N. \quad (6.11)$$

- **TC (Task Complete):** Whether the individual task run was successful or not.
- **RS (Robot Stopped):** Whether the robot was stopped early or not, either manually when unsafe motion was observed or when the internal Franka Panda firmware stopped the robot for safety reasons.

These metrics were averaged over the 5 runs and mean and standard deviation values were computed and shown in Table 6.1. For the TC and RS metrics, the number of times these events occurred is shown compared to the total number of runs.

Table 6.1: Summary of real robot control performance under different control parametrisations. LP is whether a low pass filter was used, TRL was whether a torque rate limiter was used and FF is whether the feed forward term was used. The values reported in the table are mean values plus/minus the standard deviation, averaged over 5 runs for each control parametrisation. TC is whether the task was completed and RS is whether the robot was stopped early, with the values showing how many times this event occurred.

Controller Name	Time Indexing	LP	TRL	FF	Pos Error ( $\times 10^{-3}$ )	Vel Error ( $\times 10^{-3}$ )	Torque Grad ( $\times 10^{-3}$ )	Cost	Task Length	TC	RS
Inverse-Dynamics-1	T0				0.40 $\pm$ 0.57	14.42 $\pm$ 21.36	2.63 $\pm$ 0.15	<b>203.0 <math>\pm</math> 42.7</b>	1,929.8 $\pm$ 97.3	2 / 5	0 / 5
Inverse-Dynamics-2	T0			✓	0.40 $\pm$ 0.23	11.64 $\pm$ 6.91	5.10 $\pm$ 1.57	46.2 $\pm$ 12.1	578.4 $\pm$ 289.8	4 / 5	1 / 5
Inverse-Dynamics-3	T0	✓		✓	0.24 $\pm$ 0.24	7.71 $\pm$ 7.63	3.14 $\pm$ 0.49	105.3 $\pm$ 84.3	977.6 $\pm$ 442.8	3 / 5	2 / 5
Inverse-Dynamics-4	Error				0.86 $\pm$ 0.20	6.23 $\pm$ 1.04	4.21 $\pm$ 0.41	124.9 $\pm$ 8.3	994.2 $\pm$ 134.9	0 / 5	5 / 5
Inverse-Dynamics-5	Error			✓	0.36 $\pm$ 0.21	7.65 $\pm$ 5.57	6.77 $\pm$ 2.99	90.4 $\pm$ 114.3	589.6 $\pm$ 823.8	0 / 5	4 / 5
Inverse-Dynamics-6	Error	✓		✓	0.19 $\pm$ 0.26	<b>2.69 <math>\pm</math> 1.46</b>	3.26 $\pm$ 1.02	110.0 $\pm$ 89.5	783.4 $\pm$ 655.9	0 / 5	5 / 5
Inverse-Dynamics-7	Opt Time				0.34 $\pm$ 0.17	7.64 $\pm$ 4.94	<b>2.54 <math>\pm</math> 0.10</b>	95.5 $\pm$ 89.7	1,391.0 $\pm$ 834.0	2 / 5	0 / 5
Inverse-Dynamics-8	Opt Time			✓	0.15 $\pm$ 0.11	2.92 $\pm$ 2.38	4.20 $\pm$ 1.67	106.8 $\pm$ 105.6	725.2 $\pm$ 715.7	1 / 5	3 / 5
Inverse-Dynamics-9	Opt Time	✓		✓	0.21 $\pm$ 0.09	5.25 $\pm$ 2.95	2.65 $\pm$ 0.15	63.5 $\pm$ 5.8	1,396.2 $\pm$ 827.3	0 / 5	2 / 5
Torque-PD-1	T0				0.87 $\pm$ 0.47	22.67 $\pm$ 10.37	4.51 $\pm$ 0.76	120.2 $\pm$ 31.8	1,287.2 $\pm$ 548.9	4 / 5	0 / 5
Torque-PD-2	T0			✓	0.61 $\pm$ 0.25	14.78 $\pm$ 5.69	6.37 $\pm$ 0.81	85.0 $\pm$ 46.1	131.0 $\pm$ 22.4	0 / 5	5 / 5
Torque-PD-3	T0		✓	✓	0.66 $\pm$ 0.21	16.27 $\pm$ 4.26	5.55 $\pm$ 0.71	84.1 $\pm$ 55.4	202.0 $\pm$ 120.6	0 / 5	5 / 5
Torque-PD-4	T0	✓		✓	4.43 $\pm$ 5.25	<b>136.37 <math>\pm</math> 169.33</b>	9.20 $\pm$ 6.15	154.3 $\pm$ 110.1	336.0 $\pm$ 327.1	0 / 5	5 / 5
Torque-PD-5	T0	✓	✓	✓	2.06 $\pm$ 3.10	95.22 $\pm$ 172.94	<b>9.71 <math>\pm</math> 5.34</b>	156.4 $\pm$ 214.7	320.2 $\pm$ 438.8	0 / 5	5 / 5
Torque-PD-6	Error				1.56 $\pm$ 0.90	41.02 $\pm$ 66.00	2.91 $\pm$ 1.69	89.4 $\pm$ 62.2	1,655.6 $\pm$ 770.1	0 / 5	0 / 5
Torque-PD-7	Error			✓	<b>18.34 <math>\pm</math> 32.24</b>	83.28 $\pm$ 87.52	8.44 $\pm$ 2.00	153.5 $\pm$ 158.0	1,143.0 $\pm$ 687.8	3 / 5	1 / 5
Torque-PD-8	Error		✓	✓	0.88 $\pm$ 0.64	4.46 $\pm$ 1.39	5.96 $\pm$ 2.68	88.8 $\pm$ 46.9	471.2 $\pm$ 221.3	1 / 5	4 / 5
Torque-PD-9	Error	✓		✓	0.92 $\pm$ 0.45	12.91 $\pm$ 12.22	6.81 $\pm$ 1.77	71.4 $\pm$ 32.6	774.0 $\pm$ 416.1	4 / 5	1 / 5
Torque-PD-10	Error	✓	✓	✓	0.72 $\pm$ 0.28	5.74 $\pm$ 3.32	4.97 $\pm$ 1.37	62.9 $\pm$ 7.5	490.8 $\pm$ 247.1	3 / 5	2 / 5
Torque-PD-11	Opt Time				0.82 $\pm$ 0.12	15.09 $\pm$ 1.65	2.84 $\pm$ 0.11	49.5 $\pm$ 2.2	<b>2,000.0 <math>\pm</math> 0.0</b>	0 / 5	0 / 5
Torque-PD-12	Opt Time			✓	<b>0.13 <math>\pm</math> 0.08</b>	2.91 $\pm$ 1.96	2.94 $\pm$ 0.21	48.9 $\pm$ 9.0	<b>2,000.0 <math>\pm</math> 0.0</b>	0 / 5	0 / 5
Torque-PD-13	Opt Time		✓	✓	1.87 $\pm$ 1.90	65.69 $\pm$ 66.43	5.58 $\pm$ 2.61	42.6 $\pm$ 20.0	610.2 $\pm$ 785.7	3 / 5	0 / 5
<b>Torque-PD-14</b>	Opt Time	✓		✓	0.19 $\pm$ 0.05	3.39 $\pm$ 0.82	3.52 $\pm$ 0.46	43.3 $\pm$ 6.0	367.6 $\pm$ 78.3	5 / 5	0 / 5
Torque-PD-15	Opt Time	✓	✓	✓	1.16 $\pm$ 2.11	38.42 $\pm$ 75.92	5.38 $\pm$ 2.98	<b>38.0 <math>\pm</math> 3.3</b>	<b>309.4 <math>\pm</math> 132.0</b>	5 / 5	0 / 5

Regarding Table 6.1, the best result for each metric is highlighted in bold, and the worst result is marked in red. An exception is made for the task length metric. For this particular metric, the smallest task length was observed for a controller that did not perform well overall; this controller triggered an early robot stopping condition in all five trials, thereby artificially reducing the task length. Consequently, the lowest task length that also included successful task completions is highlighted instead.

Due to the number of different metrics considered, no single controller outperformed across all of them. Determining the “best” controller involves a degree of subjectivity. The selection criteria used prioritised strong position and velocity tracking, consistent task completion, and a low overall task cost. The controller name believed to be the most suitable for these objectives is highlighted in bold in Table 6.1. This controller (Torque-PD-14) demonstrated strong position tracking (2nd overall), strong velocity tracking (3rd overall), completed the task in all five trials, and achieved a low task cost (3rd overall). Additionally, it maintained a relatively low torque gradient (9th overall). Based on these results, this control parametrisation was selected for further experiments in Sec. 6.3.

Before proceeding, some context is provided for the numerical levels of position and velocity tracking errors. In general, accurately tracking commanded velocities is more challenging than tracking positions; therefore, results are presented for velocity tracking only. Fig. 6.1 illustrates a single trial on the real Franka Panda robot arm, highlighting the differences between commanded and actual joint velocities. The top set of plots corresponds to the best-performing control parametrisation for velocity tracking (Inverse-Dynamics-6), while the bottom set depicts the worst-performing parametrisation (Torque-PD-6). The seventh joint was omitted from the plots to improve visual clarity, as its commanded velocities were minimal.

As shown in Fig. 6.1, the top plots demonstrate close adherence to the commanded velocity profiles, whereas the bottom plots exhibit significantly more oscillatory behaviour. The poor performance of the torque controller is likely attributable to the combination of the controller itself and the naive T0 time indexing method. This method appears particularly ineffective when used with the torque controller, possibly because it does not account for the robot’s true dynamics (as described by the inverse dynamics equation). Consequently, torques computed for a prior robot state receive less corrective adjustment than those computed via the inverse dynamics controller, negatively impacting this control parametrisation.

To examine the effects of individual control parametrisation design choices (controller, time indexing etc.), the data from Table 6.1 is grouped by different controller parameters in Table 6.2. This grouping of the data reveals several noteworthy trends that align with observations made during the experimental process. Firstly, the torque controller generally demonstrated better task completion rates and lower overall task costs compared to the inverse dynamics controller. However, it tended to perform worse in tracking the commanded positions and velocities. One likely

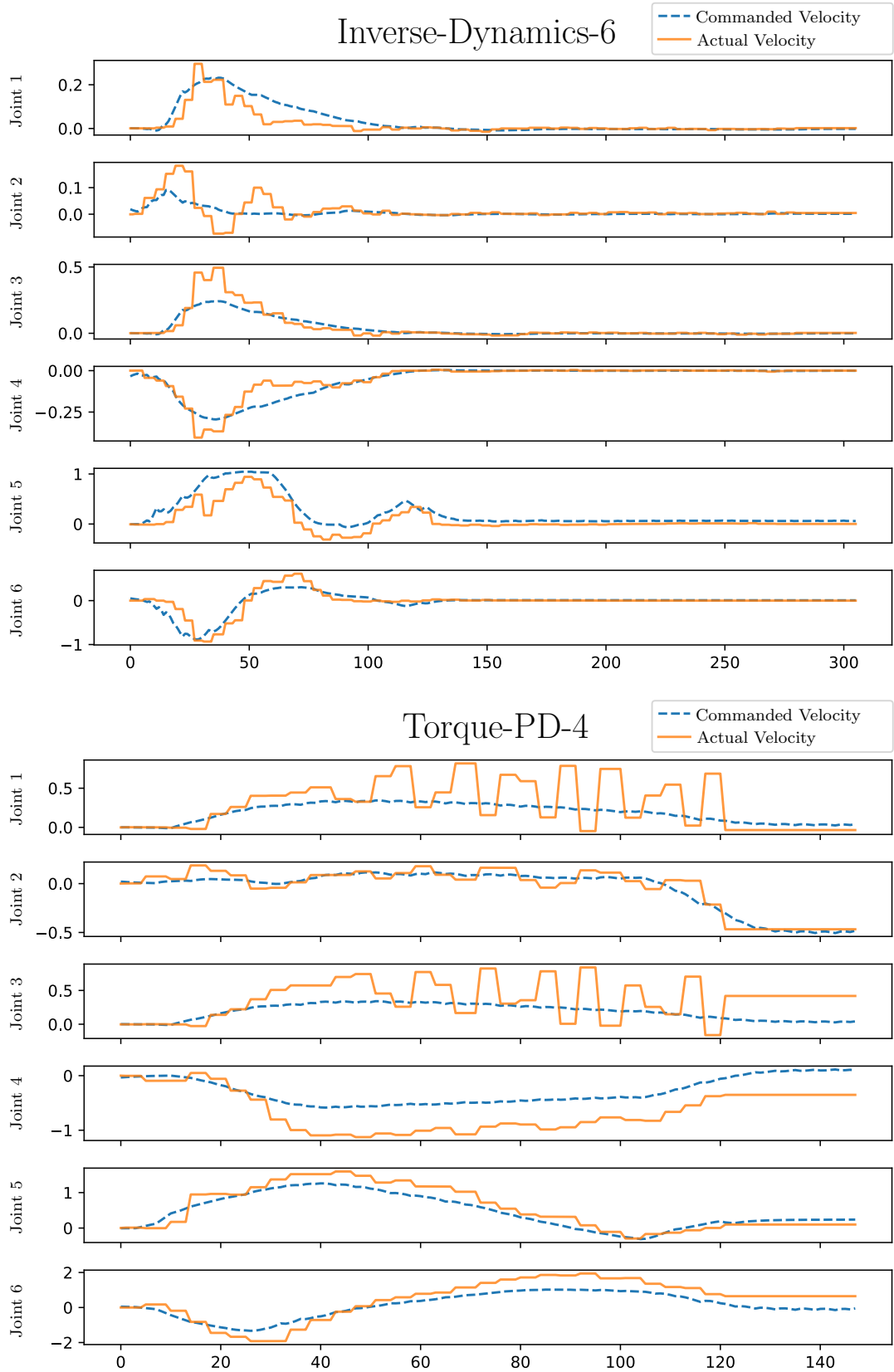


Figure 6.1: A set of plots showing the deviations between commanded velocity (blue) and actual velocity (orange) of the real Franka Panda joints. The top plots shows the best example (Inverse-Dynamics-6) of velocity tracking from Table 6.1, whereas the bottom sets of plots shows the worst (Torque-PD-4). 7th joint omitted for the sake of space.

explanation for this is that the torque controller frequently overshoot the commanded trajectory, while the inverse dynamics controller typically undershot. As a result, the inverse dynamics controller often became trapped in local minima, where the optimiser was unable to compute commands that would improve the cost of the current trajectory. In several trials using the inverse dynamics controller, this led to the robot ceasing movement and failing to complete the task. Additionally, in some trials, a substantial portion of the execution time was spent commanding small or zero velocities, which may have made trajectory tracking appear more accurate. This issue is addressed in Sec. 6.4 as one of the limitations of the work presented in this chapter.

One of the most notable performance differences is by the time indexing method that was used. The results clearly imply that the Opt Time method for choosing an initial control index proved to be the most effective in almost all metrics. It enabled the task to be completed slightly more than the other methods, and significantly reduced the number of times the robot needed to be stopped early, generally implying that the robot operated in a more safe manner when this time indexing method was used. The Opt Time indexing method had the lowest positional and velocity tracking error, as well as the smallest torque gradient and overall task cost. This method is able to consistently account for the policy-lag of the optimiser by converting the optimisation time into a corresponding control index. Whilst the robot is not guaranteed to be at the state specific by the time-indexing method, it does help reduce state mismatch which lead to better MPC performance.

Generally, sending the feedforward term for either controller improved the robot's ability to complete the task, while also resulting in a higher frequency of early stop events due to safety concerns. This outcome is likely related to the previously discussed issue, where controllers that undershot the desired trajectory became stuck in local minima. The feedforward term played a critical role in preventing undershooting of commanded positions and velocities; however, it was also a primary factor contributing to unsafe robot movements. Lastly, the use of either a torque rate limiter or a low-pass filter appeared to have negligible overall effects, although there was some indication of improved position and velocity tracking performance.

Table 6.2: Summary of real robot performance data from Table 6.1 grouped by different categories (Controller used, time indexing method, whether the feed forward term was used or not and finally whether a low pass filter or torque rate limiter were used). The values reported in this table are mean values plus/minus the standard deviation, except for TC and RS columns which report the number of times these events occurred.

Grouped by Controller											
Controller Name	Time Indexing	LP	TRL	FF	Pos Error ( $\times 10^{-3}$ )	Vel Error ( $\times 10^{-3}$ )	Torque Grad ( $\times 10^{-3}$ )	Cost	Task Length	TC	RS
Inverse Dynamics Torque_PD	-	-	-	-	$0.35 \pm 0.32$	$7.35 \pm 8.42$	$3.83 \pm 1.80$	$105.1 \pm 79.1$	$1,040.6 \pm 697.1$	$12 / 45$	$22 / 45$
	-	-	-	-	$2.35 \pm 8.85$	$37.22 \pm 76.95$	$5.65 \pm 3.19$	$85.9 \pm 83.2$	$806.5 \pm 733.8$	$28 / 75$	$28 / 75$
Grouped by Time Indexing											
Controller Name	Time Indexing	LP	TRL	FF	Pos Error ( $\times 10^{-3}$ )	Vel Error ( $\times 10^{-3}$ )	Torque Grad ( $\times 10^{-3}$ )	Cost	Task Length	TC	RS
-	T0	-	-	-	$1.21 \pm 2.39$	$39.89 \pm 90.42$	$5.78 \pm 3.64$	$119.3 \pm 98.9$	$720.3 \pm 671.6$	$13 / 40$	$23 / 40$
-	Error	-	-	-	$2.98 \pm 11.89$	$20.50 \pm 44.36$	$5.42 \pm 2.51$	$98.9 \pm 79.0$	$862.7 \pm 629.0$	$11 / 40$	$22 / 40$
-	Opt Time	-	-	-	$0.61 \pm 1.09$	$17.67 \pm 38.92$	$3.71 \pm 1.81$	$61.0 \pm 51.4$	$1,100.0 \pm 829.0$	$16 / 40$	$5 / 40$
Grouped by Send Feedforward Term											
Controller Name	Time Indexing	LP	TRL	FF	Pos Error ( $\times 10^{-3}$ )	Vel Error ( $\times 10^{-3}$ )	Torque Grad ( $\times 10^{-3}$ )	Cost	Task Length	TC	RS
-	-	-	-	-	$0.81 \pm 0.60$	$17.85 \pm 28.70$	$3.27 \pm 1.07$	$113.8 \pm 65.7$	$1,543.0 \pm 595.2$	$8 / 30$	$5 / 30$
-	-	-	-	✓	$1.86 \pm 8.12$	$28.74 \pm 70.30$	$5.53 \pm 3.07$	$86.2 \pm 85.9$	$678.1 \pm 633.0$	$32 / 90$	$45 / 90$
Grouped by LP OR TRL											
Controller Name	Time Indexing	LP	TRL	FF	Pos Error ( $\times 10^{-3}$ )	Vel Error ( $\times 10^{-3}$ )	Torque Grad ( $\times 10^{-3}$ )	Cost	Task Length	TC	RS
-	-	-	-	-	$3.72 \pm 13.71$	$29.96 \pm 49.01$	$4.67 \pm 2.39$	$91.1 \pm 76.5$	$1,369.5 \pm 783.8$	$7 / 30$	$6 / 30$
-	-	✓	✓	-	$0.89 \pm 1.77$	$24.70 \pm 66.72$	$5.07 \pm 3.03$	$93.7 \pm 84.0$	$735.9 \pm 635.3$	$33 / 90$	$44 / 90$

## 6.3 Packing an Object as an Optimisation Problem

Building upon the best performing control parametrisation from Sec. 6.2 (Torque-PD-6), this chapter now tackles performing a simplified version of the packing task that has previously been outlined. Fig. 6.2 shows three examples of this packing task with different starting configurations and goal positions, showing that the control method is capable of using non-prehensile manipulation actions to clear the goal region so that a grasped object can be placed efficiently.

A cost formulation setup similar to Howell et al. [50] was used:

$$\text{cost} = \sum_{i=1}^M w_i \cdot n_i(r_i(\mathbf{x}, \mathbf{u})), \quad (6.12)$$

where  $w_i$  is a scalar weight,  $n_i$  is some twice differentiable norm function (a simple squared function is used in this work),  $r_i$  is a residual term that is “small when the task is solved” and  $M$  is the number of residuals. The five residuals specified for this task were:

1. **X position:** The difference between the grasped object’s current x position and the goal x position. The cost weight scalar was 2.0.
2. **Y position:** The difference between the grasped object’s current y position and the goal y position. The cost weight scalar was 2.0.
3. **Z position:** The difference between the grasped object’s current z position and the goal z position. The cost weight scalar was 1.0.
4. **Object upright:** A residual that was minimised when the grasped object was facing upright. The cost weight scalar was 0.4.
5. **Object velocity:** The object’s Euclidean velocity. The cost weight scalar was 0.6.

Importantly, the cost term for minimising the object’s position to the goal was split into three separate residuals, instead of also using the Euclidean object position. The reason for this was to penalise the separate dimensions differently, allowing the robot to demonstrate behaviour where the robot would prioritise moving the grasped object over the goal position first and then moving vertically downwards.

To implement the function `GETREALWORLDSTATE` from Alg. 10, two sources of sensor information were used. Firstly, to track the poses of the obstacle objects in the scene, an optitrack system was used and reflective markers were placed on these objects for accurate pose estimation. Secondly, joint state information from the franka ROS library was used to access the current positional and velocity information of the robot arm. Importantly, this work always assumed that the grasped object

was grasped by the vacuum gripper of the robot throughout task execution. To this end, a model of the grasped object was added to the robot model used in MuJoCo. This simplification was necessary due to the fact that the OptiTrack<sup>1</sup> system struggled to track the grasp object due to occlusion.

The performance of the control parametrisation and optimiser was evaluated over 30 trials, each with different start and goal configurations. On average, the method was able to place the target object within 3.6 cm of the goal location, with an average displacement of surrounding objects of 7.5 cm. The task was successfully completed in 26 out of the 30 trials, with an average completion time of 6.5 seconds. Although no direct baseline methods are available for comparison, these results demonstrate the potential efficiency of non-prehensile manipulation for this type of task. Unlike prehensile manipulation strategies, which would require the robot to individually grasp and relocate each obstructing object before transporting the target, this approach achieves the objective in a single coordinated motion.

---

<sup>1</sup>OptiTrack website: <https://optitrack.com/>



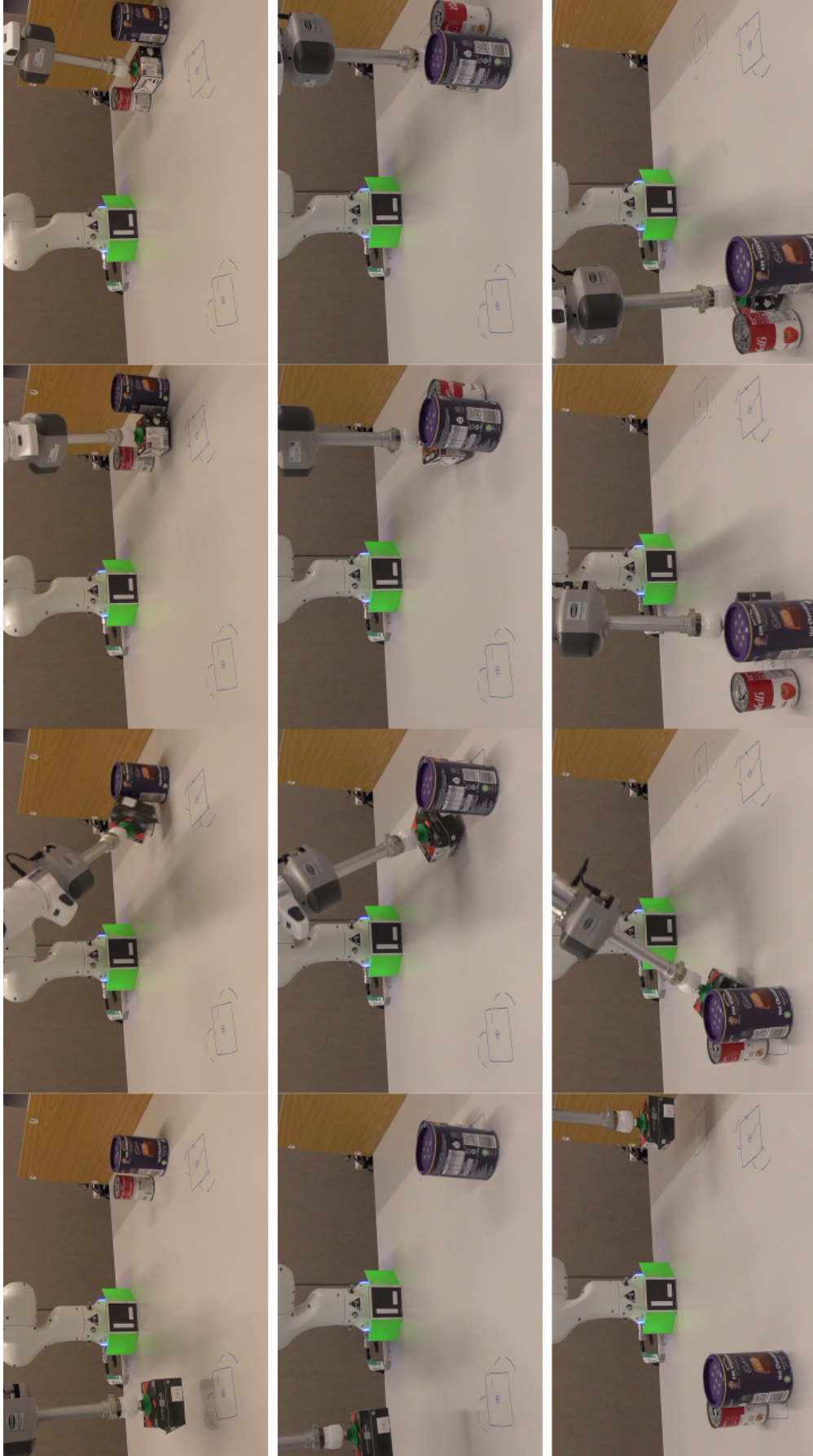


Figure 6.2: Three example sequences (from left to right) of packing a grasped object (tomato sauce) to a goal position obstructed by two obstacles (tomato soup and hot chocolate powder) on real robotic hardware.

## 6.4 Conclusion

This chapter has investigated some of the issues of controlling real robotic hardware using MPC despite imperfect system identification. Two well known feedback controllers were implemented for the Franka Panda robotic manipulator through a ROS interface, as well as a variety of time-indexing methods and control alterations. These control parametrisations have been evaluated under an MPC framework to choose a “best” control parametrisation for real robot MPC.

The “best” control parametrisation was then used to complete a simplified packing through clutter task, similar to the general one motivated in Chapter 1. This chapter shows that the proposed control parametrisation and trajectory optimisation formulation is capable of completing the task efficiently on real robotic hardware. Despite the number of obstructing objects being small in the case that was tested, these methods could be extended to work in more complicated scenes with higher levels of clutter. As the levels of clutter increases, the optimisation time would also increase, consequently increasing the policy lag of the execution thread. As such, the methods outlined in Chapters 4 and 5 would prove useful in making the problem computationally tractable.

There are a few limitations to the work in this chapter. These are:

- Issues in the initial analysis of controller performance whilst using MPC. As mentioned in Sec. 6.2.3, some controllers got stuck in certain robot configurations due to the optimiser getting stuck in a local minima. This generally occurred from controllers that undershot the commanded velocity. This had two negative effects, firstly, this increased the overall task cost of these methods and secondly, it decreased the tracking errors for these methods as the commanded positions and velocities were easier to achieve.
- The initial testing performed for each control parametrisation only used 5 trials per control parametrisation, due to the large number of control parametrisations and limited time.
- The final tests were performed in a simplified packing scene with a low level of clutter, whereas the problem that was motivated in Chapter 1 had higher levels of clutter. In this vein, it would have been desired to show that increasing the levels of clutter would then necessitate using the methods proposed in Chapters 4 and 5 to improve task performance.

In terms of future work, it would be valuable to address the limitations outlined above. Although the initial tests provided insights into the performance of various control parametrisations under MPC, tracking performance could be more effectively evaluated using an alternative testing procedure. In particular, assessing the controllers on a pre-programmed, collision-free trajectory would isolate their tracking capabilities by removing the influence of the optimiser. This approach would enable

a clearer analysis of each controller’s ability to follow a desired path. Naturally, this test would not apply to the time indexing methods, as they are only relevant within the context of MPC. Additionally, increasing the number of trials conducted would help to improve the statistical reliability of the results.

Finally, a key objective for future work is to integrate the methods outlined in this chapter and throughout the thesis into a complete packing system. The envisioned system would be capable of packing a diverse set of rigid and deformable objects, stored in an unstructured manner, into a designated container. Crucially, the system would leverage non-prehensile manipulation actions to achieve efficient and robust packing performance.

# Chapter 7

## Conclusions & Future Work

This concluding chapter will summarise and reflect on the key contributions of this thesis, alongside its limitation. Content from Chapter 1 will be referenced to evaluate the progress made toward the general picking/packing system that has been proposed. The remaining research problems that need to be addressed to fully create such a system will also be discussed.

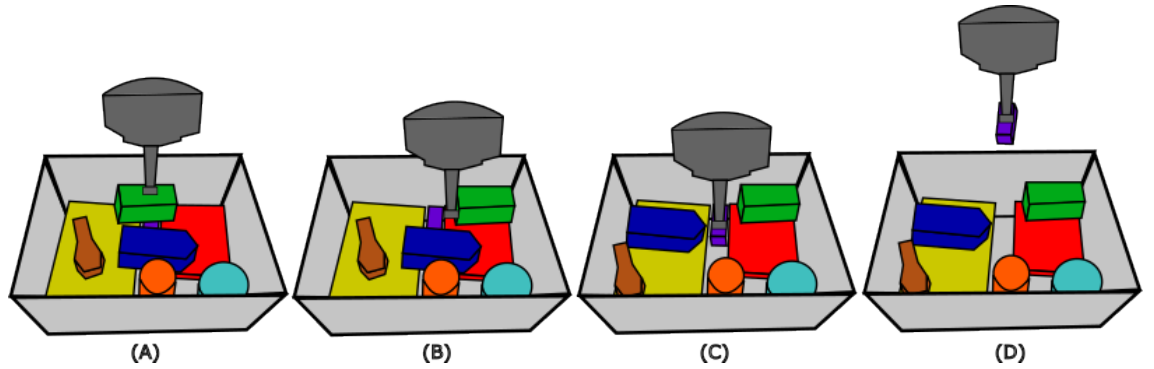


Figure 7.1: The contact-aware picking/packing system that was discussed in Chapter 1. The dark gray shapes represent a pump end-effector attachment for a robot. To recap: In this example, the high level objective is to retrieve a purple object from a bin. Frame A shows the initial scene, where the purple object is obscured by a green and dark blue object. In frame B, the robot slides the green object into the corner of the bin. Then, in frame C, the robot pushes the dark blue object out of the way, subsequently moving the brown object also. These actions have then cleared enough space so that the vacuum pump tip can reach in and grasp the purple object. Finally, frame D shows the robot removing the purple object from the bin.

### 7.1 Conclusions

This thesis has begun working toward creating the system outlined in Chapter 1. This system, shown in Fig. 7.1, employs non-prehensile manipulation to efficiently pick or pack goods in a cluttered bin. However, there were several challenges to

create such a system, one of these challenges is the difficulty of executing non-prehensile manipulation actions reliably in the real world. The method used in this thesis to overcome this was MPC, where an optimiser is constantly re-planning trajectories from the current real world state to account for differences between a physics simulator and the real world. For MPC to be effective, optimisation needs to occur fast, which traditionally has been a challenge in contact-based manipulation, particularly in clutter. The methods proposed in this thesis aim to tackle this challenge.

Chapter 4 introduced a method of using approximations to speed up trajectory optimisation. This idea was to only compute dynamics derivatives at *key-points* over a trajectory and approximate the remainder through the use of simple linear interpolation. It was shown that this methodology can significantly decrease optimisation times with only small degradations in the quality of the computed trajectories. Chapter 4 showed that the proposed method generalised to two different gradient-based optimisation algorithms (iLQR and SCVX), ten different tasks (most of which included contact-based manipulation) as well as two methods of computing dynamics derivatives (FD and AD). The methods proposed in that chapter have multiple uses: They can speed up long horizon trajectory optimisation to compute a trajectory to solve a given task much faster or they can be used with shorter horizon MPC to increase the control frequency, enabling robots to account for deviations between a physics simulator and the real world faster.

In a similar vein, the methods proposed in Chapter 5 aimed to speed up trajectory optimisation for MPC. This work focused on robotic manipulation in high-dimensional scenes, where not all of the DoFs are important to the task at all times during execution. Chapter 5 reasoned that, in such situations, the trajectory optimisation algorithm should only consider the DoFs most relevant to the current state of the problem. By reducing the number of DoFs considered by the optimiser, optimisation times are decreased, resulting in tighter closed-loop control of the real system.

While the work presented in Chapters 4 and 5 produced promising results in speeding up trajectory optimisation, both share a common limitation: reliance on tuneable parameters for optimal performance. In Chapter 4, these parameters involved the parametrisation of the proposed key-point methods, whereas in Chapter 5, they pertained to the thresholding criteria for determining whether a particular DoF should be considered in optimisation. Anecdotally, it was observed that although these parameters were not highly specific, they still required some degree of tuning to achieve the best performance. In the conclusion sections of both chapters, potential strategies for mitigating this tuning issue were outlined, which represent promising avenues for future work.

Finally, Chapter 6 investigated the issues of using MPC to control real robotic hardware, namely the Franka-Emika Panda. The issue was that the torques computed in simulation induced a much larger acceleration on the real robot than they

did in simulation. This issue caused motion plans to become quickly invalid. As such, Chapter 6 investigates using an inverse dynamics controller and also a torque proportional derivative controller to help account for this deviation between simulated and real robot accelerations. This chapter also investigates a variety of other control parameters to improve the control performance of the real robot. In addition to this, results were shown for a simplified packing task through low levels of clutter on real robotic hardware using MPC.

## 7.2 Remaining Problems / Limitations

All of the chapters of this thesis build towards creating the efficient picking/packing system that has previously been discussed. One of the largest pieces of work that still need to be implemented is some form of a high-level task planner which can use trajectory optimisation primitives to achieve the overall goal. One of these trajectory optimisation primitives would be the one showed in Chapter 6. Other primitives might include: Moving an object to a new location to clear space, grouping objects closer together, and reorientating an object. The methods of speeding up trajectory optimisation in chapters 4 and 5 would prove useful in enabling MPC to run efficiently when the number of items in the bin is increased.

Another issue that remains to be addressed in this system represents an additional limitation of this thesis: perception. This thesis has relied on an OptiTrack system to provide accurate information about the poses of all objects in the simulated environment. OptiTrack is a system that uses infra-red cameras attached in known locations and relies on a unique combination of reflective stickers to be placed on objects so each object can be uniquely identified. Using OptiTrack in the real world for such a system would be highly impractical. For instance, imagine a warehouse that processes tens of thousands of items per day, there would be no feasible way to individually attach reflective markers in a unique pattern to identify each object. However, it is possible to use QR codes for pose tracking of objects which is already used in some warehouse environments. For unconventional objects however, perception may need to rely on more conventional stereo camera systems to obtain pose information for any objects in the environment. One major challenge, particularly in warehouse scenarios, is high levels of occlusion, where objects are partially or fully hidden. This gives rise to the intriguing question: How can our system instantiate the scene within the physics simulator without having full knowledge of the poses of all objects? One potential solution could involve combining the physical knowledge of partially visible objects with a physics simulator to create valid approximations of the objects' locations [142].

The work done in this thesis has always assumed some level of complete knowledge of the environment. For real robotic experiments, this has involved creating scenes to be used in a simulator which included all of the relevant objects in the environment. In reality, a system would be needed to create a model of the scene

purely from the information provided by any sensors. In addition to this is the issue of object’s physical parameters. Before conducting physical experiments, objects were weighed and estimates of their friction coefficients were used as inputs to a physics simulator. In reality, a robot will need to discover and adapt these physical object parameters during task execution, as they can not necessarily be known a priori.

Finally, another limitation of this work is that it does not address the issue of absent gradients in under-actuated trajectory optimisation tasks. This problem is often overlooked in non-prehensile manipulation scenarios. Essentially, when an actuated robot is tasked with moving an under-actuated object to a target position, the robot must apply force directly to the object. In situations where the robot and object are not in contact, the optimiser lacks the gradient information needed to improve the nominal trajectory. There are several workarounds to this issue. This work typically initialises a trajectory that ensures contact between the robot and the object. This approach provides the optimiser with the necessary gradient information to improve the nominal trajectory, although it does require additional implementation considerations for the user, dependent on the task. Another approach involves adding a term to the cost function that encourages the robot to make contact with the object. For instance, one can include a term that minimises the distance between the robot’s end-effector and the object, thereby supplying gradient information that directs the optimiser to move the end-effector toward the object. However, this method also presents implementation challenges and limits the robot’s ability to manipulate the object solely using its end-effector.

A more sophisticated method is demonstrated by Önal et al. [96]. They employ a direct optimisation approach that allows a robot to apply “virtual” forces on objects despite not being in contact with them. These “virtual” forces are then penalised in the cost function dependant on the distance between the robot and the object. This method is similar to the previous approach but is perhaps more robust, however, the disadvantage remains the same which is the need to specify contact points between the robot and the object a priori. Nonetheless, some work [145] has explored ways to overcome this challenge.

## 7.3 Future Work

This section will briefly discuss some broad research areas for possible future avenues of work, some of which extend ideas presented in this thesis.

### 7.3.1 Efficient picking/packing

The efficient contact-based picking/packing system that was introduced at the beginning of this thesis still has several components that need to be implemented. Firstly, a high-level planner capable of invoking trajectory optimisation tasks us-

ing a library of contact-based trajectory optimisation primitives is required. These primitives would include placing an object through clutter, relocating an object to a new position using non-prehensile manipulation, and reorienting objects, among others. The contributions made in Chapters 4 and 5 can be used to improve optimisation times for these trajectory optimisation primitives, enabling the system to operate effectively in highly cluttered environments.

Beyond requiring a high-level task planner, such a system will also need a robust pose detection method that does not rely on physical modifications to the appearance of objects. Placing multiple cameras around the scene appears to be the most logical approach, and various neural network libraries exist for estimating the 3D pose of objects from 2D RGB images [130, 139]. However, these methods can only detect objects that are visible to the cameras. In a bin picking/packing scenario, objects will frequently be occluded due to the movement of other items. To address this issue, tracking systems that integrate sensor data with physics simulations to predict object motion could prove useful [142].

### 7.3.2 Change of basis online state vector reduction

As a direction for future work, it may be valuable to extend the methods discussed in Chapter 5 beyond purely axis-aligned dimensionality reduction, in which each degree of freedom (DoF) is either fully considered or not considered at all. A natural progression would involve exploring change-of-basis approaches that enable feature compression, allowing multiple DoFs to be represented within a lower-dimensional subspace. Such methods could offer improved generalisation and coordination for tasks involving complex interactions, such as scooping small objects or manipulating deformable materials.

### 7.3.3 Combining learning with trajectory optimisation

There are various works that explore using learning in conjunction with trajectory optimisation for improved efficiency. This can be achieved by learning better warm-start trajectories for the optimiser [7, 19], reducing the number of iterations required to converge to an optimal solution, or learning better-informed terminal value conditions, enabling the use of shorter optimisation horizons [9, 135, 148].

These methods have not been extended to or demonstrated in contact-based manipulation tasks. They could be highly beneficial in such settings, particularly in learning improved warm-start trajectory initialisations. As discussed earlier, the absent gradients problem in under-actuated manipulation tasks necessitates some workaround. A learning-based warm-start trajectory could offer an effective way to initialise a trajectory so that the robot makes meaningful contact with the object being manipulated.



### 7.3.4 Using GPU-based physics simulation

Finally, the recent advancements in GPU-based physics simulation [77, 6] open up promising avenues for future work. These advancements enable hundreds or even thousands of simulated environments to run in parallel, significantly increasing the number of sampled actions per second. This is particularly beneficial to the RL community, where it accelerates the training of neural network policies by generating more simulated data. However, it can also be applied to trajectory optimisation, allowing for a greater number of sampled trajectories in short-horizon MPC. As graphics cards become more affordable, it becomes more feasible that future robotic systems will have access to them for motion planning.

GPU-based physics simulators naturally extend to traditional sampling-based algorithms [2, 111, 50]. Additionally, for gradient-based optimisers such as iLQR, computing the dynamics derivatives is a highly parallelisable task. Therefore, it would be interesting to investigate the effectiveness of performing derivative computations on the GPU.

One important consideration when using GPU-based physics simulators for optimisation is the trade-off between optimisation horizon and the number of sampled trajectories. Intuitively, CPU-based trajectory optimisation can be thought of as a “depth-first” search, where fewer trajectories are sampled, but they extend further in time due to the efficient optimisation of CPU cores. In contrast, GPU-based trajectory optimisation resembles a “breadth-first” search, where many more trajectories can be trialled, but with shorter horizons. This distinction arises from the architectural differences: While a single physics simulation step is typically faster on a CPU, the GPU accelerates overall throughput by performing hundreds or thousands of physics simulation steps in parallel. Some tasks may benefit from a longer optimisation horizon, whereas others may be better suited to evaluating a larger number of short-horizon trajectories.

## 7.4 Final Remarks

This thesis has explored some of the challenges of robotic manipulation, particularly focusing on speeding up trajectory optimisation for contact-based manipulation. By addressing computational bottlenecks and integrating efficient motion planning strategies, this work has contributed to the broader goal of making robotic systems more capable of interacting with their environments using more efficient actions that humans take for granted.

Fundamentally, the tasks considered in this thesis such as pushing, sliding, and dynamically repositioning objects are ones that a human would solve effortlessly. Yet, for a robot, these actions require intricate modelling, careful control, and computational efficiency to execute in real time. This discrepancy highlights the complexity of robotic manipulation in unstructured environments and shows the need for continued research in this field.

The general motivation for this work extends beyond the realm of academic inquiry. The world as a whole is experiencing an ageing workforce [132] and increasing labour shortages, a problem, that is bound to worsen in the future. As such, there is a growing need for increased automation that extends beyond traditional controlled factory settings. The robots of the future must go beyond traditional rigid pick-and-place operations and develop the abilities to manipulate their surroundings as efficiently as humans.

# Bibliography

- [1] Pieter Abbeel and Andrew Y Ng. “Apprenticeship learning via inverse reinforcement learning”. In: *Proceedings of the twenty-first international conference on Machine learning*. 2004, p. 1.
- [2] Ian Abraham et al. “Model-based generalization under parameter uncertainty using path integral control”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 2864–2871. ISSN: 2377-3766.
- [3] Wisdom C Agboh and Mehmet R Dogar. “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*. Springer, 2018, pp. 160–176. ISBN: 3030440508.
- [4] Wisdom C Agboh and Mehmet R Dogar. “Real-time online re-planning for grasping under clutter and uncertainty”. In: *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2018, pp. 1–8. ISBN: 1538672839.
- [5] Pulkit Agrawal et al. “Learning to poke by poking: Experiential learning of intuitive physics”. In: *Advances in neural information processing systems* 29 (2016).
- [6] Genesis Authors. *Genesis: A Universal and Generative Physics Engine for Robotics and Beyond*. 2024. URL: <https://github.com/Genesis-Embodied-AI/Genesis>.
- [7] Somrita Banerjee et al. “Learning-based warm-starting for fast sequential convex programming and trajectory optimization”. In: *2020 IEEE Aerospace Conference*. IEEE, 2020, pp. 1–8. ISBN: 1728127343.
- [8] Mokhtar S Bazaraa, Hanif D Sherali, and Chitharanjan M Shetty. *Nonlinear programming: theory and algorithms*. John wiley & sons, 2006.
- [9] Wissam Bejjani, Mehmet R Dogar, and Matteo Leonetti. “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 (IROS)*. IEEE, 2019, pp. 6562–6569. ISBN: 1728140048.

- [10] John T Betts. *Practical methods for optimal control and estimation using nonlinear programming*. SIAM, 2010.
- [11] Kevin Black et al. “ $\pi_0$ : A Vision-Language-Action Flow Model for General Robot Control”. In: *arXiv preprint arXiv:2410.24164* (2024).
- [12] Reinhard Blickhan. “The spring-mass model for running and hopping”. In: *Journal of biomechanics* 22.11-12 (1989), pp. 1217–1227. ISSN: 0021-9290.
- [13] Robert Bohlin and Lydia E Kavraki. “Path planning using lazy PRM”. In: *Proceedings 2000 ICRA. Millennium conference. IEEE international conference on robotics and automation. Symposia proceedings (Cat. No. 00CH37065)*. Vol. 1. IEEE. 2000, pp. 521–528.
- [14] Riccardo Bonalli et al. “Gusto: Guaranteed sequential trajectory optimization via sequential convex programming”. In: *2019 International conference on robotics and automation (ICRA)*. IEEE. 2019, pp. 6741–6747.
- [15] Anthony Brohan et al. “Rt-1: Robotics transformer for real-world control at scale”. In: *arXiv preprint arXiv:2212.06817* (2022).
- [16] Anthony Brohan et al. “Rt-2: Vision-language-action models transfer web knowledge to robotic control”. In: *arXiv preprint arXiv:2307.15818* (2023).
- [17] Dane Brouwer et al. “Tactile-informed action primitives mitigate jamming in dense clutter”. In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 7991–7997.
- [18] Justin Carpentier, Quentin Le Lidec, and Louis Montaut. “From Compliant to Rigid Contact Simulation: a Unified and Efficient Approach”. In: *20th edition of the “Robotics: Science and Systems”(RSS) Conference*. 2024.
- [19] Davide Celestini et al. “Transformer-based model predictive control: Trajectory optimization via sequence modeling”. In: *IEEE Robotics and Automation Letters* (2024).
- [20] Henry J Charlesworth and Giovanni Montana. “Solving challenging dexterous manipulation tasks with trajectory optimisation and reinforcement learning”. In: *International Conference on Machine Learning*. PMLR, 2021, pp. 1496–1506. ISBN: 2640-3498.
- [21] Iordanis Chatzinikolaïdis and Zhibin Li. “Trajectory optimization of contact-rich motions using implicit differential dynamic programming”. In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 2626–2633. ISSN: 2377-3766.
- [22] Yu-Ming Chen and Michael Posa. “Optimal reduced-order modeling of bipedal locomotion”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 8753–8760. ISBN: 1728173957.
- [23] Xianyi Cheng et al. “Contact mode guided motion planning for quasidynamic dexterous manipulation in 3d”. In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 2730–2736.

- [24] Z Cheng et al. “Neural network iLQR: A reinforcement learning architecture for trajectory optimization”. In: *arXiv preprint arXiv:2011.10737* (2020).
- [25] Cheng Chi et al. “Diffusion policy: Visuomotor policy learning via action diffusion”. In: *The International Journal of Robotics Research* (2023), p. 02783649241273668.
- [26] Cheng Chi et al. “Universal Manipulation Interface: In-The-Wild Robot Teaching Without In-The-Wild Robots”. In: *Proceedings of Robotics: Science and Systems (RSS)*. 2024.
- [27] Matei Ciocarlie, Corey Goldfeder, and Peter Allen. “Dimensionality reduction for hand-independent dexterous robotic grasping”. In: *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2007, pp. 3270–3275. ISBN: 142440911X.
- [28] Thomas H Cormen et al. “Introduction to Algorithms (3-rd edition)”. In: *MIT Press and McGraw-Hill* (2009).
- [29] Erwin Coumans and Yunfei Bai. “Pybullet, a python module for physics simulation for games, robotics and machine learning”. In: (2016).
- [30] Yuhong Deng et al. “Deep reinforcement learning for robotic pushing and picking in cluttered environment”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ieee. 2019, pp. 619–626.
- [31] M Dogar et al. “Physics-based grasp planning through clutter”. In: (2012).
- [32] Mehmet R Dogar and Siddhartha S Srinivasa. “A planning framework for non-prehensile manipulation under clutter and uncertainty”. In: *Autonomous Robots* 33.3 (2012), pp. 217–236. ISSN: 1573-7527.
- [33] Mehmet R Dogar and Siddhartha S Srinivasa. “Push-grasping with dexterous hands: Mechanics and a method”. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2010, pp. 2123–2130.
- [34] Tom Erez and Emanuel Todorov. “Trajectory optimization for domains with contacts using inverse dynamics”. In: *2012 IEEE/RSJ International conference on intelligent robots and systems*. IEEE. 2012, pp. 4914–4919.
- [35] Tom Erez et al. “An integrated system for real-time model predictive control of humanoid robots”. In: *2013 13th IEEE-RAS International conference on humanoid robots (Humanoids)*. IEEE, 2013, pp. 292–299. ISBN: 1479926191.
- [36] Henrique Ferrolho et al. “Inverse dynamics vs. forward dynamics in direct transcription formulations for trajectory optimization”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 12752–12758.
- [37] Chelsea Finn and Sergey Levine. “Deep visual foresight for planning robot motion”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 2786–2793. ISBN: 150904633X.

- [38] Markus Gifftthaler et al. “A family of iterative gauss-newton shooting methods for nonlinear optimal control”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 1–9.
- [39] Philip E Gill, Walter Murray, and Michael A Saunders. “SNOPT: An SQP algorithm for large-scale constrained optimization”. In: *SIAM review* 47.1 (2005), pp. 99–131. ISSN: 0036-1445.
- [40] Vinícius Mariano Gonçalves et al. “Parsimonious kinematic control of highly redundant robots”. In: *IEEE Robotics and Automation Letters* 1.1 (2015), pp. 65–72.
- [41] Bernhard P Graesdal et al. “Towards Tight Convex Relaxations for Contact-Rich Manipulation”. In: *Proceedings of Robotics: Science and Systems (RSS)* (2024).
- [42] Ruben Grandia et al. “Feedback mpc for torque-controlled legged robots”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2013, pp. 4730–4737. ISBN: 1728140048.
- [43] Tuomas Haarnoja et al. “Learning to Walk Via Deep Reinforcement Learning”. In: *Robotics: Science and Systems XV* (2019).
- [44] Daseong Han et al. “Data-guided model predictive control based on smoothed contact dynamics”. In: *Computer Graphics Forum*. Vol. 35. Wiley Online Library, 2016, pp. 533–543. ISBN: 0167-7055.
- [45] Peter E Hart, Nils J Nilsson, and Bertram Raphael. “A formal basis for the heuristic determination of minimum cost paths”. In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.
- [46] Kris Hauser and Victor Ng-Thow-Hing. “Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts”. In: *2010 IEEE international conference on robotics and automation*. IEEE. 2010, pp. 2493–2498.
- [47] Yanhao He and Steven Liu. “Analytical Inverse Kinematics for Franka Emika Panda – a Geometrical Solver for 7-DOF Manipulators with Unconventional Design”. In: *2021 9th International Conference on Control, Mechatronics and Automation (ICCMA2021)*. IEEE, Nov. 2021. DOI: 10.1109/ICCMA54375.2021.9646185.
- [48] Todd Hester et al. “Deep q-learning from demonstrations”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.
- [49] François Robert Hogan and Alberto Rodriguez. “Feedback control of the pusher-slider system: A story of hybrid and underactuated contact dynamics”. In: *Algorithmic Foundations of Robotics XII: Proceedings of the Twelfth Workshop on the Algorithmic Foundations of Robotics*. Springer. 2020, pp. 800–815.

- [50] Taylor Howell et al. “Predictive sampling: Real-time behaviour synthesis with mujoco”. In: *arXiv preprint arXiv:2212.00541* (2022).
- [51] Hengyuan Hu, Suvir Mirchandani, and Dorsa Sadigh. “Imitation Bootstrapped Reinforcement Learning”. In: *arXiv preprint arXiv:2311.02198* (2023).
- [52] Sandy H Huang et al. “Learning gentle object manipulation with curiosity-driven deep reinforcement learning”. In: *arXiv preprint arXiv:1903.08542* (2019).
- [53] Jeffrey Ichnowski et al. “GOMP: Grasp-optimized motion planning for bin picking”. In: *2020 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2020, pp. 5270–5277.
- [54] “Interior-Point Methods for Nonlinear Programming”. In: *Numerical Optimization*. New York, NY: Springer New York, 2006, pp. 563–597. ISBN: 978-0-387-40065-5. DOI: 10.1007/978-0-387-40065-5\_19. URL: [https://doi.org/10.1007/978-0-387-40065-5\\_19](https://doi.org/10.1007/978-0-387-40065-5_19).
- [55] Advait Jain et al. “Reaching in clutter with whole-arm tactile sensing”. In: *The International Journal of Robotics Research* 32.4 (2013), pp. 458–482. ISSN: 0278-3649.
- [56] Michael Janner et al. “Planning with Diffusion for Flexible Behavior Synthesis”. In: *International Conference on Machine Learning*. PMLR, 2022, pp. 9902–9915.
- [57] Wanxin Jin and Michael Posa. “Task-driven hybrid model reduction for dexterous manipulation”. In: *IEEE Transactions on Robotics* (2024). ISSN: 1552-3098.
- [58] Edward Johns. “Coarse-to-fine imitation learning: Robot manipulation from a single demonstration”. In: *2021 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2021, pp. 4613–4619. ISBN: 1728190770.
- [59] Shuuji Kajita and Kazuo Tani. “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*. IEEE Computer Society, 1991, pp. 1405, 1406, 1407, 1408, 1409, 1410, 1411–1405, 1406, 1407, 1408, 1409, 1410, 1411.
- [60] Mrinal Kalakrishnan et al. “STOMP: Stochastic trajectory optimization for motion planning”. In: *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 4569–4574. ISBN: 1612843859.
- [61] Lydia E Kavraki et al. “Probabilistic roadmaps for path planning in high-dimensional configuration spaces”. In: *IEEE transactions on Robotics and Automation* 12.4 (1996), pp. 566–580. ISSN: 1042-296X.

- [62] Zachary Kingston, Mark Moll, and Lydia E Kavraki. “Sampling-based methods for motion planning with constraints”. In: *Annual review of control, robotics, and autonomous systems* 1.1 (2018), pp. 159–185.
- [63] Nikita Kitaev et al. “Physics-based trajectory optimization for grasping in cluttered environments”. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 3102–3109. ISBN: 1479969230.
- [64] Nate Kohl and Peter Stone. “Policy gradient reinforcement learning for fast quadrupedal locomotion”. In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA’04. 2004*. Vol. 3. IEEE, 2004, pp. 2619–2624.
- [65] Marek Kopicki et al. “Learning to predict how rigid objects behave under simple manipulation”. In: *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 5722–5729. ISBN: 1612843859.
- [66] Serdar Kucuk and Zafer Bingul. *Robot kinematics: Forward and inverse kinematics*. INTECH Open Access Publisher London, UK, 2006.
- [67] James J Kuffner and Steven M LaValle. “RRT-connect: An efficient approach to single-query path planning”. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. Vol. 2. IEEE, 2000, pp. 995–1001. ISBN: 0780358864.
- [68] Vikash Kumar. *Franka Sim - A MuJoCo Model of the Franka Panda Robot*. GitHub repository. 2018. URL: [https://github.com/vikashplus/franka\\_sim](https://github.com/vikashplus/franka_sim).
- [69] Vince Kurtz and Hai Lin. “Contact-implicit trajectory optimization with hydroelastic contact and ilqr”. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 8829–8834. ISBN: 1665479272.
- [70] Vince Kurtz et al. “Inverse Dynamics Trajectory Optimization for Contact-Implicit Model Predictive Control”. In: *arXiv preprint arXiv:2309.01813* (2023).
- [71] Steven M LaValle. “Rapidly-exploring random trees: A new tool for path planning”. In: (1998).
- [72] Teguh Santoso Lembono and Sylvain Calinon. “Probabilistic iterative LQR for short time horizon MPC”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 579–585.
- [73] Weiwei Li and Emanuel Todorov. “Iterative linear quadratic regulator design for nonlinear biological movement systems”. In: *ICINCO (1)*. Citeseer, 2004, pp. 222–229.



- [74] Kendall Lowrey et al. “Plan online, learn offline: Efficient learning and exploration via model-based control”. In: *arXiv preprint arXiv:1811.01848* (2018).
- [75] Jianlan Luo et al. “Precise and Dexterous Robotic Manipulation via Human-in-the-Loop Reinforcement Learning”. In: *arXiv preprint arXiv:2410.21845* (2024).
- [76] Arthur Mahoney, Joshua Bross, and David Johnson. “Deformable robot motion planning in a reduced-dimension configuration space”. In: *2010 IEEE International Conference on Robotics and Automation*. IEEE, 2010, pp. 5133–5138. ISBN: 1424450381.
- [77] Viktor Makoviychuk et al. “Isaac gym: High performance gpu-based physics simulation for robot learning”. In: *arXiv preprint arXiv:2108.10470* (2021).
- [78] Zachary Manchester and Scott Kuindersma. “Derivative-free trajectory optimization with unscented dynamic programming”. In: *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, 2016, pp. 3642–3647. ISBN: 1509018379.
- [79] Olvi L Mangasarian. “Sufficient conditions for the optimal control of nonlinear systems”. In: *SIAM Journal on control* 4.1 (1966), pp. 139–152. ISSN: 0036-1402.
- [80] Yuanqi Mao et al. “Successive convexification: A superlinearly convergent algorithm for non-convex optimal control problems”. In: *arXiv preprint arXiv:1804.06539* (2018).
- [81] Tobia Marcucci et al. “Shortest paths in graphs of convex sets”. In: *SIAM Journal on Optimization* 34.1 (2024), pp. 507–532.
- [82] Matthew T Mason. “Mechanics and planning of manipulator pushing operations”. In: *The International Journal of Robotics Research* 5.3 (1986), pp. 53–71.
- [83] Jan Matas, Stephen James, and Andrew J Davison. “Sim-to-real reinforcement learning for deformable object manipulation”. In: *Conference on Robot Learning*. PMLR. 2018, pp. 734–743.
- [84] David Mayne. “A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems”. In: *International Journal of Control* 3.1 (1966), pp. 85–95. ISSN: 0020-7179.
- [85] Bart van Merriënboer, Alexander B Wiltchko, and Dan Moldovan. “Tangent: Automatic differentiation using source code transformation in Python”. In: *arXiv preprint arXiv:1711.02712* (2017).
- [86] Mayank Mittal et al. “Orbit: A unified simulation framework for interactive robot learning environments”. In: *IEEE Robotics and Automation Letters* 8.6 (2023), pp. 3740–3747. ISSN: 2377-3766.

- [87] Igor Mordatch, Zoran Popović, and Emanuel Todorov. “Contact-invariant optimization for hand manipulation”. In: *Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation*. 2012, pp. 137–144.
- [88] Igor Mordatch, Emanuel Todorov, and Zoran Popović. “Discovery of complex behaviors through contact-invariant optimization”. In: *ACM Transactions on Graphics (ToG)* 31.4 (2012), pp. 1–8. ISSN: 0730-0301.
- [89] João Moura, Theodoros Stouraitis, and Sethu Vijayakumar. “Non-prehensile planar manipulation via trajectory optimization with complementarity constraints”. In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 970–976. ISBN: 1728196817.
- [90] Mustafa Mukadam et al. “Continuous-time Gaussian process motion planning via probabilistic inference”. In: *The International Journal of Robotics Research* 37.11 (2018), pp. 1319–1340.
- [91] Ashvin Nair et al. “Overcoming exploration in reinforcement learning with demonstrations”. In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 6292–6299.
- [92] Jauwairia Nasir et al. “RRT\*-SMART: A rapid convergence implementation of RRT”. In: *International Journal of Advanced Robotic Systems* 10.7 (2013), p. 299. ISSN: 1729-8814.
- [93] John N Nganga and Patrick M Wensing. “Accelerating second-order differential dynamic programming for rigid-body systems”. In: *IEEE Robotics and Automation Letters* 6.4 (2021), pp. 7659–7666.
- [94] Abby O’Neill et al. “Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0”. In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 6892–6903. ISBN: 9798350384574.
- [95] Aykut Ozgun Onol, Philip Long, and Taskin Padlr. “A comparative analysis of contact models in trajectory optimization for manipulation”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1–9. ISBN: 1538680947.
- [96] Aykut Özgün Önoł, Philip Long, and Taşkın Padır. “Contact-implicit trajectory optimization based on a variable smooth contact model and successive convexification”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 2447–2453. ISBN: 153866027X.
- [97] Aykut Özgün Önoł et al. “Tuning-free contact-implicit trajectory optimization”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 1183–1189. ISBN: 1728173957.

- [98] Jia Pan, Liangjun Zhang, and Dinesh Manocha. “Collision-free and smooth trajectory computation in cluttered environments”. In: *The International Journal of Robotics Research* 31.10 (2012), pp. 1155–1175.
- [99] Tao Pang et al. “Global planning for contact-rich manipulation via local smoothing of quasi-dynamic contact models”. In: *IEEE Transactions on robotics* (2023). ISSN: 1552-3098.
- [100] Rafael Papallas, Anthony G Cohn, and Mehmet R Dogar. “Online replanning with human-in-the-loop for non-prehensile manipulation in clutter—a trajectory optimization based approach”. In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 5377–5384. ISSN: 2377-3766.
- [101] Rafael Papallas and Mehmet R Dogar. “Non-prehensile manipulation in clutter with human-in-the-loop”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 6723–6729.
- [102] Rafael Papallas and Mehmet R Dogar. “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)*. IEEE, 2022, pp. 649–656. ISBN: 1665479272.
- [103] Xue Bin Peng, Glen Berseth, and Michiel Van de Panne. “Terrain-adaptive locomotion skills using deep reinforcement learning”. In: *ACM Transactions on Graphics (TOG)* 35.4 (2016), pp. 1–12.
- [104] Xue Bin Peng et al. “Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning”. In: *Acm transactions on graphics (tog)* 36.4 (2017), pp. 1–13.
- [105] Nicolas Perrin et al. “Fast humanoid robot collision-free footstep planning using swept volume approximations”. In: *IEEE Transactions on Robotics* 28.2 (2011), pp. 427–439. ISSN: 1552-3098.
- [106] Corrado Pezzato et al. “Sampling-based model predictive control leveraging parallelizable physics simulations”. In: *IEEE Robotics and Automation Letters* (2025).
- [107] Cristina Pinneri et al. “Sample-efficient cross-entropy method for real-time planning”. In: *Conference on Robot Learning*. PMLR, 2020, pp. 1049–1065. ISBN: 2640-3498.
- [108] Michael Posa, Cecilia Cantu, and Russ Tedrake. “A direct method for trajectory optimization of rigid bodies through contact”. In: *The International Journal of Robotics Research* 33.1 (2014), pp. 69–81. ISSN: 0278-3649.
- [109] Aravind Rajeswaran et al. “Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations”. In: *Robotics: Science and Systems XIV* (2018).

- [110] Nathan Ratliff et al. “CHOMP: Gradient optimization techniques for efficient motion planning”. In: *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 489–494. ISBN: 1424427886.
- [111] Reuven Rubinstein. “The cross-entropy method for combinatorial and continuous optimization”. In: *Methodology and computing in applied probability* 1 (1999), pp. 127–190.
- [112] D Russell, R Papallas, and M Dogar. “Online state vector reduction during model predictive control with gradient-based trajectory optimisation”. In: *Workshop on Algorithmic Foundations in Robotics, Springer Proceedings in Advanced Robotics (SPAR)*. Springer. 2024.
- [113] David Russell, Rafael Papallas, and Mehmet Dogar. “Adaptive approximation of dynamics gradients via interpolation to speed up trajectory optimisation”. In: *Proceedings of the 2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023. ISBN: 1728196825.
- [114] Muhammad Suhail Saleem and Maxim Likhachev. “Planning with selective physics-based simulation for manipulation among movable objects”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 6752–6758.
- [115] Dhruv Mauria Saxena and Maxim Likhachev. “Planning for complex non-prehensile manipulation among movable objects by interleaving multi-agent pathfinding and physics-based simulation”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 8141–8147.
- [116] Dhruv Mauria Saxena, Muhammad Suhail Saleem, and Maxim Likhachev. “Manipulation Planning Among Movable Obstacles Using Physics-Based Adaptive Motion Primitives”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 6570–6576. ISBN: 1728190770.
- [117] John Schulman et al. “Motion planning with sequential convex optimization and convex collision checking”. In: *The International Journal of Robotics Research* 33.9 (2014), pp. 1251–1270.
- [118] Gerrit Schultz and Katja Mombaur. “Modeling and optimal control of human-like running”. In: *IEEE/ASME Transactions on mechatronics* 15.5 (2009), pp. 783–792.
- [119] Mario Selvaggio et al. “Non-prehensile object transportation via model predictive non-sliding manipulation control”. In: *IEEE Transactions on Control Systems Technology* (2023).
- [120] “Sequential Quadratic Programming”. In: *Numerical Optimization*. New York, NY: Springer New York, 2006, pp. 529–562. ISBN: 978-0-387-40065-5. DOI: 10.1007/978-0-387-40065-5\_18. URL: [https://doi.org/10.1007/978-0-387-40065-5\\_18](https://doi.org/10.1007/978-0-387-40065-5_18).

- [121] Aayushman Sharma and Suman Chakravorty. “A Reduced Order Iterative Linear Quadratic Regulator (ILQR) Technique for the Optimal Control of Nonlinear Partial Differential Equations”. In: *2023 American Control Conference (ACC)*. IEEE, 2023, pp. 3389–3394. ISBN: 9798350328066.
- [122] Bruno Siciliano and Oussama Khatib. *Springer handbook of robotics*. springer, 2016. ISBN: 3319325523.
- [123] David Stewart and Jeffrey C Trinkle. “An implicit time-stepping scheme for rigid body dynamics with coulomb friction”. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. Vol. 1. IEEE, 2000, pp. 162–169. ISBN: 0780358864.
- [124] Richard S Sutton. “Reinforcement learning: An introduction”. In: *A Bradford Book* (2018).
- [125] Zaid Tahir et al. “Potentially guided bidirectionalized RRT\* for fast optimal path planning in cluttered environments”. In: *Robotics and Autonomous Systems* 108 (2018), pp. 13–27. ISSN: 0921-8890.
- [126] Yuval Tassa, Tom Erez, and Emanuel Todorov. “Synthesis and stabilization of complex behaviors through online trajectory optimization”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 4906–4913. ISBN: 1467317365.
- [127] Yuval Tassa, Nicolas Mansard, and Emo Todorov. “Control-limited differential dynamic programming”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 1168–1175.
- [128] R Tedrake. “Drake: Model-based design and verification for robotics”. In: *URL <https://drake.mit.edu>* (2019).
- [129] Emanuel Todorov, Tom Erez, and Yuval Tassa. “Mujoco: A physics engine for model-based control”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033. ISBN: 1467317365.
- [130] Jonathan Tremblay et al. “Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects”. In: *Conference on Robot Learning*. PMLR. 2018, pp. 306–316.
- [131] Tokuo Tsuji, Kensuke Harada, and Kenji Kaneko. “Easy and fast evaluation of grasp stability by using ellipsoidal approximation of friction cone”. In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2009, pp. 1830–1837. ISBN: 1424438039.
- [132] United Nations, Department of Economic and Social Affairs, Population Division. *World Population Prospects 2024: Summary of Results*. Accessed: 2025-02-25. July 2024. URL: <https://desapublications.un.org/publications/world-population-prospects-2024-summary-results>.

- [133] Paul Vernaza and Daniel Lee. “Learning dimensional descent for optimal motion planning in high-dimensional spaces”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 25. 2011. ISBN: 2374-3468.
- [134] Paul Vernaza and Daniel D Lee. “Learning and exploiting low-dimensional structure for efficient holonomic motion planning in high-dimensional spaces”. In: *The International Journal of Robotics Research* 31.14 (2012), pp. 1739–1760. ISSN: 0278-3649.
- [135] Julian Viereck, Avadesh Meduri, and Ludovic Righetti. “ValueNetQP: Learned one-step optimal control for legged locomotion”. In: *Learning for Dynamics and Control Conference*. PMLR, 2022, pp. 931–942. ISBN: 2640-3498.
- [136] Andreas Wächter and Lorenz T Biegler. “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming”. In: *Mathematical programming* 106 (2006), pp. 25–57. ISSN: 0025-5610.
- [137] Shengyin Wang et al. “Goal-conditioned action space reduction for deformable object manipulation”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 3623–3630. ISBN: 9798350323658.
- [138] Benjamin Ward-Cherrier et al. “The tactip family: Soft optical tactile sensors with 3d-printed biomimetic morphologies”. In: *Soft robotics* 5.2 (2018), pp. 216–227.
- [139] Bowen Wen et al. “Foundationpose: Unified 6d pose estimation and tracking of novel objects”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 17868–17879.
- [140] Grady Williams, Andrew Aldrich, and Evangelos Theodorou. “Model predictive path integral control using covariance variable importance sampling”. In: *arXiv preprint arXiv:1509.01149* (2015).
- [141] Zhuo Xu et al. “Cocoi: Contact-aware online context inference for generalizable non-planar pushing”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 176–182. ISBN: 1665417145.
- [142] Zisong Xu, Rafael Papallas, and Mehmet R Dogar. “Physics-Based Object 6D-Pose Estimation during Non-Prehensile Manipulation”. In: *International Symposium on Experimental Robotics*. Springer. 2023, pp. 181–191.
- [143] Weihao Yuan et al. “End-to-end nonprehensile rearrangement with deep reinforcement learning and simulation-to-reality transfer”. In: *Robotics and Autonomous Systems* 119 (2019), pp. 119–134.

- [144] Andy Zeng et al. “Learning synergies between pushing and grasping with self-supervised deep reinforcement learning”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 4238–4245.
- [145] Mengchao Zhang et al. “Simultaneous Trajectory Optimization and Contact Selection for Multi-Modal Manipulation Planning”. In: *arXiv preprint arXiv:2306.06465* (2023).
- [146] Tony Zhao et al. “Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware”. In: *Robotics: Science and Systems XIX* (2023).
- [147] Dongliang Zheng and Panagiotis Tsiotras. “Accelerating kinodynamic RRT\* through dimensionality reduction”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 3674–3680. ISBN: 1665417145.
- [148] Mingyuan Zhong et al. “Value function approximation and model predictive control”. In: *2013 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL)*. IEEE, 2013, pp. 100–107. ISBN: 1467359254.
- [149] Guangyao Zhou et al. “Diffusion model predictive control”. In: *arXiv preprint arXiv:2410.05364* (2024).
- [150] Claudio Zito et al. “Two-level RRT planning for robotic push manipulation”. In: *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2012, pp. 678–685. ISBN: 1467317365.
- [151] Tongyu Zong, Liyang Sun, and Yong Liu. “Reinforced iLQR: A sample-efficient robot locomotion learning”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 5906–5913.