An Online Framework for Changing-Contact Robot Manipulation

Saif Sidhik

School of Computer Science
University of Birmingham, UK

This dissertation is submitted for the degree of
Doctor of Philosophy

November 2021

Abstract

A robot performing a manipulation task encounters high degrees of non-linearities in its interaction dynamics as it interacts with other objects in its workspace. Often the interaction dynamics is highly discontinuous when the robot makes/breaks contact with other object(s) and smooth at other times, making it a piecewise-continuous (PWC) dynamical system. The interaction effects that the robot might experience during a task is also highly dependent on the nature and physical properties of the objects involved. Furthermore, the high discontinuities when a robot makes contact with other objects can be harmful for the robot and/or object if not performed in a controlled manner. This unpredictability and inherently discontinuous nature of interactions during task execution makes the design of a control framework for changing-contact manipulation extremely challenging.

Solving any manipulation task requires developing a plan and then executing it accurately and efficiently. There have been many attempts in literature to solve the former problem: developing planners that build task plans for specific manipulation tasks subjected to different types of constraints in motion such as collision-avoidance and contact locations. This thesis investigates the less-explored *plan execution* part of robot manipulation, by developing an adaptive and online control framework for efficient, safe, smooth and accurate execution of a provided task plan in the presence of continuous and discontinuous interaction dynamics.

The first main contribution of this thesis explores the problem of controlling a robot in a smoothly changing environment. For navigating such continuous environment, we propose an adaptive variable impedance controller (AVIC) which can incrementally model, predict, and compensate for the end-effector forces and torques that the robot experiences as it moves. The second part of the thesis expands the AVIC for hybrid dynamical systems, where the interaction dynamics can be discontinuous due to change in the contact mode of the robot. The contact mode of the robot changes due to making/breaking of contacts or due to discontinuities in the environment which makes the interaction dynamics piecewise-continuous, and hence difficult to capture using a single model. Here, we develop a hierarchical hybrid framework that has an incrementally updating mode-detection model that can learn to identify contact modes online, and choose appropriate dynamics models for the robot to use for navigating the mode. The final contribution focuses on developing a contact-change-handling

module that learns to deal with contact changes by attempting to reduce the discontinuities in motion (spikes in force, acceleration, jerk, etc.) during mode transitions. The module has an iteratively improving contact-anticipation model that incrementally improves estimates of where contact changes occur in the task-space. The module also has the capability to smoothly switch to special 'transition-phase' controllers that can automatically adapt their own parameters so as to reduce the discontinuities in the regions of anticipated contacts.

Each part of the framework is evaluated experimentally in simulation and/or real-world, demonstrating the effectiveness and importance of each component for building an incremental, adaptive framework that can learn to follow a task plan accurately, efficiently and safely. Our AVIC framework is compared with other standards in adaptive control literature, and we also demonstrate the need for having an adaptive forward model for handling changing environment dynamics. We experimentally evaluate the ability of our hybrid framework in identifying and modelling the dynamics of different contact modes, and then justify the choices made in the development of the framework by making incremental changes to a baseline framework in a simulated setting. The ability of our overall framework in predicting and handling (anticipated as well as unanticipated) contact changes is also evaluated in the context of a physical robot performing manipulation tasks that involves multiple contact changes.

Table of contents

Chapter 1

Introduction

Manipulation tasks in general are under-actuated systems, since addition of each inanimate object to the environment increases the number of DoFs that *cannot* be controlled using the fixed number of actuators that are available. The states of these objects can be altered only by changing the state of the robot to one where it can manipulate them, resulting in a complicated and almost unpredictable system dynamics. Moreover, in multi-object manipulation tasks which require the robot to use one object to change the configuration of another (eg. tooling tasks such as screwing, polishing etc.), the under-actuated dynamics become even more complex. This inherent complex nature of manipulation tasks makes it essential to study the basic, underlying properties and characteristics that are general to manipulation tasks.

Diverse manipulation tasks are addressed using different representations, control choices, learning strategies, planning algorithms, and sensor modalities. The choices for these depend on the type of task being attempted and the overall goal that is to be achieved. For ballistic actions such as tennis racket swinging, it is important to have control over position, velocities and sometimes accelerations of the end-effector [74], while in dexterous manipulation tasks such as industrial assembly or in-hand manipulation, it becomes important to directly model, predict and control the contacts and interaction forces.

Many of the research commonly seen in intelligent robot manipulation involve using large training data [12, 157, 117] to obtain very hardware-specific and task-specific strategies to solve a task. Moreover, solutions for multi-object manipulation tasks and tasks involving multiple contact changes are usually shown only in simulations [157], mainly due to the fact that data collection and training using hardware is impractical in most cases. In most works which demonstrate real-robot tasks, the tasks are either ballistic (involving only one impact contact with another object) [103], or make use of manual 'safe' interactions – for e.g., using passively compliant hardware [117, 142] or modelled objects [186, 12].

A solution to any manipulation task involves two sub-problems: (i) generating a task plan (desired trajectory, contact locations, contact wrenches, etc.), and (ii) executing the plan, which involves tracking the reference signals while respecting task-defined and/or user-defined constraints (such as overall energy cost, motion smoothness, vibrations, impact forces, etc.). Many research dealing with the planning aspect of multi-object manipulation tasks can be found in literature, but very few deal with the control strategies for accurate and safe trajectory tracking in interaction tasks. Control theory provides a vast repository of strategies for controlling a system and tracking a target, but these often do not take into account the discontinuous nature of changing-contact manipulation tasks and ignore the effects of collisions and contact changes.

This thesis explores this second problem of executing a plan for a changing-contact manipulation task efficiently and smoothly, when the physical and interaction properties of the object(s) involved may be unknown. This requires the robot, in minimum number of trials, to be able to handle continuous and discontinuous interaction dynamics due to the nature of contacts that the robot has to make during the task, as well as handle impacts and contact changes that may or may not be known beforehand, while ensuring that the overall motion is smooth and that the provided plan is followed as accurately as possible.

1.1 Motivation and problem formulation

Typical manipulation tasks involve making and breaking contacts with different objects in the work space. Consider the robot manipulator in Fig. 1.1 that has to move its end-effector along a motion pattern that requires it to make and break contact with objects and surfaces. This task's dynamics, i.e., the relationship between the forces acting on the robot and the resultant accelerations, vary markedly before and after contact. They also vary based on other factors such as type of contact, surface friction, and applied force. Many industrial assembly tasks, e.g., peg insertion, screwing, and stacking, and many human manipulation tasks are instances of such '*changing-contact*' tasks. The interaction dynamics of the robot performing these tasks are discontinuous when a contact is made or broken and continuous elsewhere, making it difficult to construct a single dynamics model. It is possible to construct a *hybrid* model with separate continuous dynamics and distinct control laws within each of a number of discrete *dynamic modes*; the overall dynamics are then *piece-wise continuous*, with the robot transitioning between modes as needed [99]. Even then, the highly non-linear nature of such interactions results in large discontinuities in the overall motion dynamics of the robot at the regions of transition, in the form of high forces, jerk, and vibrations. These sudden and uncontrolled system disturbances during contact change instances such as collisions can

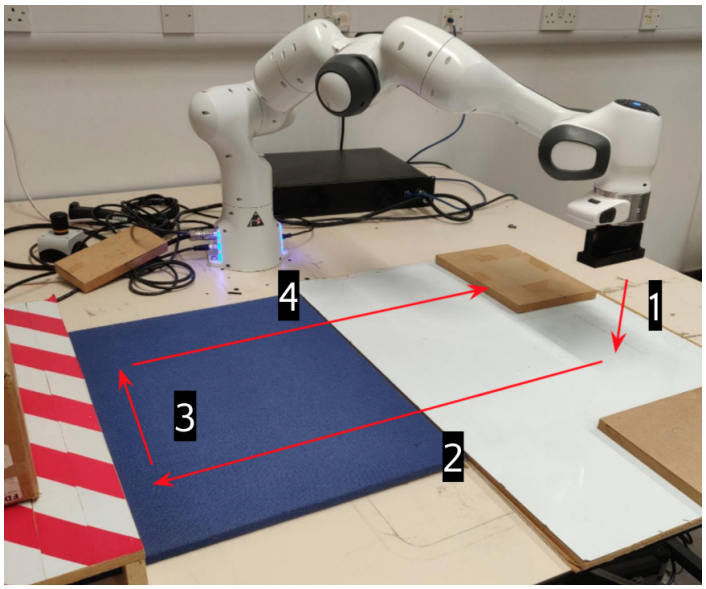


Fig. 1.1 A sliding task that involves the robot making and breaking contacts with different surfaces. The interaction dynamics is piecewise continuous, where the dynamics is discontinuous when contact changes occur, but is continuous otherwise. Contact changes can be due to collisions with objects, or due to impact-less mode transitions (e.g. when the robot slides between the surfaces (white and blue), the difference in their friction causes a discontinuity in the interaction forces) – see Chapter 6.

damage the robot and/or objects involved. Therefore, the robot also has to learn to handle these discontinuities such that the overall motion is smooth and safe.

This thesis explores these problems and proposes an incremental adaptive framework for changing-contact manipulation tasks. The framework includes (i) an adaptive variable impedance control (AVIC) strategy with an incrementally updating forward model for handling and tracking a trajectory in continuously changing dynamical environments (Chapter 4), which is then adapted for piecewise continuous systems using (ii) a hierarchical formulation with a high-level online mode-detection component and separate low-level dynamics models for different contact modes (Chapter 5); as well as (iii) a contact-change-handling module for anticipating contact changes and reducing impact forces and jerk during contact changes (Chapter 6).

Problem statement

Given a motion plan for a changing-contact manipulation task, the robot has to be able to execute the task such that:

1. it follows the planned trajectory with minimum deviation while using minimum stiffness (and thereby expending lower energy) whenever possible,

2. it can handle continuously changing interaction dynamics that arise when the robot is in continuous contact with an object/environment as part of the task,
3. it can handle the piecewise continuous nature of interaction dynamics caused due to discrete changes in the interacting environment or due to making/breaking of contacts,
4. it can quickly learn to deal with previously unseen environments (interaction dynamics) and use an appropriate adaptive controller for navigating it,
5. it can learn to anticipate collisions and contact changes in the task, and be able to learn and switch to a control strategy that produce reduced impact forces and vibrations at these transitions,
6. the robot should require as few trials as possible to learn to perform the task efficiently (where 'efficiency' is defined in terms of trajectory-tracking accuracy, controller stiffness, delays in task-completion, low impact forces, and overall smoothness of motion).

Primary assumptions

For exploring the addressed research problems in a contained set-up, several assumptions are made regarding other components involved in a general changing-contact manipulation task. The primary assumptions made are:

- The robot is an open chain manipulator with the tool/object being manipulated fixed at the end-effector. It has at least six torque-controllable revolute joints. We used 7 DoF robots for our experiments in this thesis (except in certain simulation setups).
- The motion plan for the task is provided as a task-space trajectory by an external planner or is obtained from demonstrations. The task plan is assumed to be a smooth (jerk-free) time-indexed sequence of end-effector poses in the task-space of the robot which the robot has to follow. In case the robot has to track a force target, this is also assumed to be provided.
- Except for the object(s) being manipulated by the robot, the environment is static and other objects involved are fixed in the workspace with respect to the base frame of the robot. This assumption is valid in most industrial machining and tooling tasks, and also allows for testing the repeatability of the developed framework.

- The object being manipulated is assumed to be firmly grasped by the robot providing stable grasp through all interactions in the task. This assumption avoids grasp planning and associated challenges relating to grasp instabilities that could interfere with the results of the developed framework.
- All involved objects are assumed to be rigid and inelastic. This is a valid assumption for most industrial tasks, and allows for building reliable models for collision anticipation and handling contact changes.
- The kinematic and dynamics models of the robot is assumed to be available and accurate. These are available for most robot platforms and are factory-defined.
- The robot has access to measurements of joint positions, joint velocities, joint torques, as well as forces and torques at the end-effector of the robot at all times.

1.2 Contributions

The thesis presents a unified incremental and adaptive framework for changing-contact manipulation, which can be roughly divided into three parts (each of which correspond to a chapter in this thesis):

1. An adaptive variable impedance controller with an incrementally updating feed-forward model for continuous contact tasks (Chapter 4).
2. A hybrid hierarchical framework that can automatically and efficiently detect known contact modes, recognise and model new modes, and incrementally learn a dynamics model for variable impedance control in each mode, invariant to the direction of motion and the magnitude of applied forces (Chapter 5).
3. A contact-change-handling module for dealing with discontinuities during contact changes, comprising a Kalman filter-based contact anticipation model as well as a transition-phase control strategy that ensures that the overall motion is C^{∞} smooth and help achieve a desired force on impact (Chapter 6).

Publications

Chapters 4 to 6 are mostly expansions of our published work. The relevant peer-reviewed publications are:

I: M. J. Mathew[1], S. Sidhik[1], M. Sridharan, M. Azad, A. Hayashi and J. Wyatt, *"Online Learning of Feed-Forward Models for Task-Space Variable Impedance Control,"* 2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids), 2019, pp. 222-229.

II: S. Sidhik, M. Sridharan, D. Ruiken. *"Learning Hybrid Models for Variable Impedance Control of Changing-Contact Manipulation Tasks,"* Eighth Annual Conference on Advances in Cognitive Systems (ACS 2020), August 2020, (Virtual). [2]

III: S. Sidhik, M. Sridharan, D. Ruiken. *"Towards a Framework for Changing-Contact Robot Manipulation,"* International Conference on Intelligent Robots and Systems (IROS), 27 September – 1 October 2021, Prague, Czech Republic (Virtual).

Chapter 4 is adapted from Publication I, and is modified based on part of the formulation presented in Publication II as well as additional experiments demonstrating framework validity. Chapter 5 is an expansion of Publication II with additional experiments to analyse the need for incremental hybrid models. Chapter 6 is an extension of Publication III with a more generalisable formulation and more experimental analyses.

Technical contributions

Several software interfaces and simulators had to be created during the research. A critical piece of software that was used in almost all the experiments with the physical robot (Franka Emika Panda) was the *Franka ROS Interface* software which was developed mainly to provide low-level torque control of the robot using ROS; the low-level control interface was needed for implementing the hybrid force-motion impedance controller which is the basis for the overall control framework presented in this thesis. The *Franka ROS Interface* library also provides Python (2 and 3) interfaces to control and monitor the robot in real-time, which was used to write most of the code for this thesis. The library also provides the functionality to interface multiple machines in a network to the same robot. This was needed for synchronising independent control loop, learning process, and data recording process running in different machines (see Appendix C). A unified Python library called *PandaRobot* was built over this interface library to provide the base API for most of the code that ran the experiments in this thesis.

During the COVID '19 pandemic and the resulting lockdown, there was a need for a good physics simulator to proceed with the research. This led to testing and developing several

[1]Equal contribution authors.

[2]Expanded version accepted in *Advances in Cognitive Systems* journal.

simulation platforms using the model of the Franka Emika Panda robot in physics engines such as Bullet [33], Mujoco [194], DART [109], and ODE [179]. A ROS-based Gazebo simulator called *Panda Simulator* was specifically developed for continuing the experiments with minimum change in the code used for the real robot. The simulator was written such that it responded similar to the real robot when the *Franka ROS Interface* or *PandaRobot* libraries were used. The results and experiments from the simulators were mostly not reliable due to the vast difference in the dynamics in simulators compared to real world in changing-contact tasks, and most experiments are not included in the thesis (see Appendix B.2).

All the above software are published online and are used widely for research around the world. More information and links to the repositories can be found in Appendix D. Other software that were developed during the research are also mentioned.

1.3 Thesis structure

The thesis begins with a primer on some of the basic concepts and background knowledge required for understanding the remainder of the thesis (Chapter 2). Chapter 3 discusses several related work from literature that are relevant to the research problem and to the different components of the proposed framework.

In Chapter 4, we present Adaptive Variable Impedance Control (AVIC), our incremental learning and control framework for continuously changing interaction dynamics, where we describe its mathematical formulation and experimentally demonstrate its capability in different continuously changing environments. The chapter also evaluates the need for having incrementally updating models for handling continuously changing environments.

Chapter 5 then expands the AVIC framework for piecewise continuous systems, and presents the experiments and results of using the proposed hybrid framework in different piecewise continuous interactions. The chapter also experimentally justifies the choices made in the development of the hybrid framework by using a long-term predictive model as a baseline, and demonstrating the results of making incremental modifications to it.

The final components of the overall framework are presented in Chapter 6, where we describe our incremental contact anticipation module and the 'transition-phase' control strategies for producing smooth motion dynamics and minimising impact forces and jerk during contact transitions. We also experimentally evaluate the overall framework in tasks involving multiple collisions and mode transitions, as well as analyse its capability in handling unforeseen contact changes.

Finally, the thesis is concluded in Chapter 7 by critically analysing the overall framework, discussing its limitations, major challenges that were tackled, and possible directions for future research.

Chapter 2

Background

This chapter will briefly introduce basic concepts that can help better understand the remainder of the thesis. The sections are meant to refresh the basics of some relevant topics that are explored in this thesis, and are in no way a complete guide to the subjects discussed. Wherever possible, references are provided for interested readers to learn more about the topics.

2.1 Manipulator dynamics and task-space impedance control

Kinematics is the study of the geometric relations in a moving system without considering the effects of forces and acceleration, whereas *dynamics* focus on understanding and modelling the relationship between forces and the resultant accelerations (and vice versa). For realistic robotic systems, kinematic relations are important for obtaining mappings between joint and task spaces of the robot in static (or quasi-static settings), while modelling and compensating the motion and interaction dynamics is critical for reliable control of the robot.

For a robot manipulator interacting with the environment, it is more useful to consider dynamics in the task/operational space of the robot instead of the generalised coordinate space (joint space). A common operational space representation used in robot manipulators is the pose of the robot's end-effector ($\boldsymbol{x}$) in the base frame of the robot.

Without loss of generality, consider an n degree-of-freedom robot whose joint positions are denoted by the n-dimensional vector $\boldsymbol{q}$, and its corresponding end-effector pose in the task-space denoted by the 6-dimensional minimal vector $\boldsymbol{x}$ (see example Fig. 2.1). The task-space dynamics of this robot can then be described by the following equation [199]:

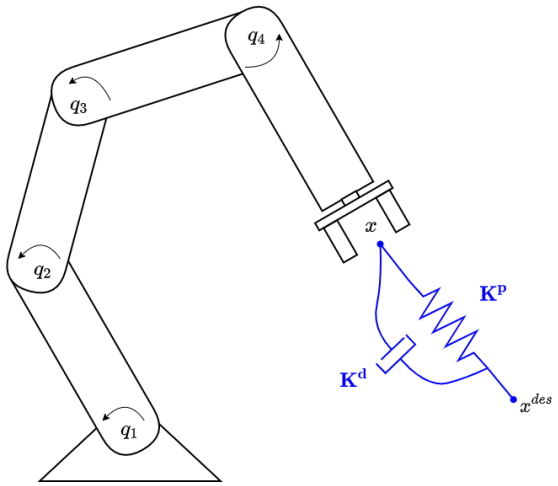


Fig. 2.1 A 3-link manipulator with joint positions denoted by q_i and corresponding end-effector pose x. Impedance control emulates the behaviour of a mass-spring-damper system at the end-effector of the robot as it attempts to reach the target x^{des}.

$$\boldsymbol{\Lambda}(\boldsymbol{q})\ddot{\boldsymbol{x}}+\boldsymbol{\Gamma}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{x}}+\boldsymbol{\eta}(\boldsymbol{q})=\boldsymbol{u}-\boldsymbol{W}^{ee} \tag{2.1}$$

where $\boldsymbol{\Lambda}(\boldsymbol{q}) = (\boldsymbol{J}\boldsymbol{M}(\boldsymbol{q})^{-1}\boldsymbol{J}^T)^{-1}$ is the 6×6 operational space inertia matrix, $\boldsymbol{\Gamma}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \boldsymbol{J}^{-T}\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\boldsymbol{J}^{-1} - \boldsymbol{\Lambda}(\boldsymbol{q})\dot{\boldsymbol{J}}\boldsymbol{J}^{-1}$ is the rotational effect wrenches including centrifugal and Coriolis effect, $\boldsymbol{\eta}(\boldsymbol{q}) = \boldsymbol{J}^{-T}\boldsymbol{g}(\boldsymbol{q})$ is the gravitational wrench, and $\boldsymbol{J}$ is the Jacobian mapping from joint to end-effector velocities ($\dot{\boldsymbol{x}} = \boldsymbol{J}\dot{\boldsymbol{q}}$). $\boldsymbol{M}(\boldsymbol{q}),\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})$ and $\boldsymbol{g}(\boldsymbol{q})$ are respectively the equivalent values defined in the joint space of the robot. The input joint torques $\boldsymbol{\tau}$ is mapped to its equivalent end-effector wrench as $\boldsymbol{u} = \boldsymbol{J}^{-T}\boldsymbol{\tau}$; and $\boldsymbol{W}^{ee}$ is the external wrench applied to the environment by the end-effector.

Consider the control law

$$\boldsymbol{u} = \boldsymbol{\Lambda}(\boldsymbol{q})\boldsymbol{\alpha}+\boldsymbol{\Gamma}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{x}}+\boldsymbol{\eta}(\boldsymbol{q})+\boldsymbol{W}^{ee} \tag{2.2}$$

where $\boldsymbol{\alpha}$ is a properly designed control input. This when cast into the robot dynamics (Eq. (2.1)) gives $\boldsymbol{\alpha} = \ddot{\boldsymbol{x}}$, proving that the control input has a meaning of acceleration in the base frame of the robot.

Task specification for a robot can be represented as a *kinematic trajectory* (a time-indexed sequence of end-effector poses or joint positions). Therefore, the approach of directly following kinematic trajectories using direct position control fails for tasks which involve interactions with other objects or application of specific forces with the environment as it does not account for the system dynamics. Such control scenarios result in stochastic disturbances

and hence is a challenging control problem. A solution for achieving goal tracking in these situations is to use controllers in the impedance paradigm.

The objective of task-space control in general is to achieve a desired behaviour for the end-effector for motion or force application. In case of impedance control, the objective is to emulate the dynamic behaviour of a second-order mechanical system with six degrees of freedom, characterised by a given mass, stiffness, and damping, known as mechanical *impedance* [66]. A well-designed task-space impedance controller would therefore cause the end-effector of the robot to behave as if it was a mass-spring-damper system whose properties are controlled/controllable (Fig. 2.1).

Assume the choice

$$\boldsymbol{\alpha} = \ddot{\boldsymbol{x}}^{des} + (\mathbf{K^m})^{-1}(\mathbf{K^d}\Delta\dot{\boldsymbol{x}} + \mathbf{K^p}\Delta\boldsymbol{x} - \boldsymbol{W}^{ee}) \tag{2.3}$$

where $\ddot{\boldsymbol{x}}^{des}$ is the desired acceleration of the end-effector, $\Delta\boldsymbol{x}$ and $\Delta\dot{\boldsymbol{x}}$ are respectively the displacement and velocity of the end-effector with respect to their references; $\mathbf{K^p}$, $\mathbf{K^d}$ and $\mathbf{K^m}$ are 6×6 symmetric positive-definite matrices defining the desired impedance behaviour, i.e., stiffness, damping, and mass (inertia) respectively.

This produces a closed-loop dynamics of the form:

$$\mathbf{K^m}\Delta\ddot{\boldsymbol{x}} + \mathbf{K^d}\Delta\dot{\boldsymbol{x}} + \mathbf{K^p}\Delta\boldsymbol{x} = \boldsymbol{W}^{ee} \tag{2.4}$$

Therefore any force $\boldsymbol{W}^{ee}$ acting at the end-effector will produce a dynamic response of the end-effector as if it was a mass-spring-damper that was defined. The final joint-space torque command can then be computed using the equation $\boldsymbol{\tau}^{\boldsymbol{u}} = \boldsymbol{J}^T\boldsymbol{u}$ of inverse dynamics and then be used for direct torque control of the robot. The control law can be shown to be asymptotically stable using classic Lyapunov method [199]. A diagram of the impedance control schema is shown in Fig. 2.2.

However, the above control law (Equations 2.2 and 2.3) requires accurate, noise-free force-torque measurements $\boldsymbol{W}^{ee}$ from the robot's end-effector for reliable performance, which in typical scenarios is hard to guarantee. Moreover, choosing an appropriate task-space inertia matrix and performing real-time dynamic compensation of the robot dynamics by transforming it to the joint-space for control (inertia shaping) can be notoriously challenging [119].

2.1.1 Impedance control without force-torque sensor

In the absence of a reliable force-torque sensor, external wrench $\boldsymbol{W}^{ee}$ cannot be measured in the control law (Eq. (2.2)) and the configuration-independent impedance behaviour (Eq. (2.4))

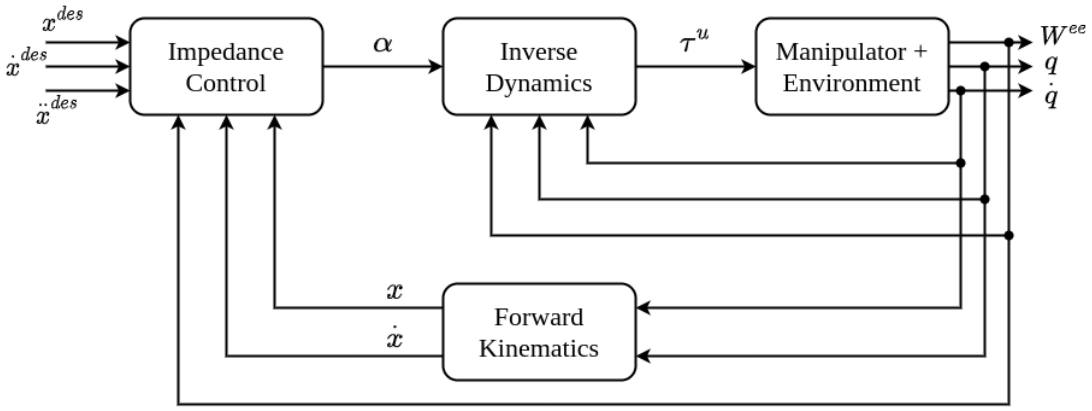


Fig. 2.2 A schematic of impedance control for a robot manipulator. Diagram adapted from [199].

cannot be obtained. However, a desired impedance behaviour can still be obtained using the control law

$$\boldsymbol{u} = \boldsymbol{\Lambda}(\boldsymbol{q})\ddot{\boldsymbol{x}}^{des} + \boldsymbol{\Gamma}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{x}}^{des} + \boldsymbol{\eta}(\boldsymbol{q}) + \mathbf{K}^{\mathbf{p}}\Delta\boldsymbol{x} + \mathbf{K}^{\mathbf{d}}\Delta\dot{\boldsymbol{x}} \tag{2.5}$$

Observe that the control command $\boldsymbol{u}$ is a wrench in the base frame that has to be applied at the end-effector of the robot to produce a desired motion of the end-effector. Substituting this control law to the dynamics (Eq. (2.1)) produces the the closed-loop equation

$$\boldsymbol{\Lambda}(\boldsymbol{q})\Delta\ddot{\boldsymbol{x}} + (\boldsymbol{\Gamma}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \mathbf{K}^{\mathbf{d}})\Delta\dot{\boldsymbol{x}} + \mathbf{K}^{\mathbf{p}}\Delta\boldsymbol{x} = \boldsymbol{W}^{ee} \tag{2.6}$$

Note that by formulating the control law without making use of a force-torque sensor, this impedance behaviour now preserves the actual configuration-dependent task-space inertia matrix of the robot $\boldsymbol{\Lambda}(\boldsymbol{q})$, which avoids the difficulties and complexities of performing inertia shaping [119]. This control law can also be shown to be asymptotically stable using classic Lyapunov stability analysis [199].

2.1.2 Compensating external disturbances using feed-forward control

In the absence of interactions with any objects, and therefore external wrench $\boldsymbol{W}^{ee}$, the control law described above provides asymptotic stability with the equilibrium state $(\dot{\boldsymbol{x}} = 0, \Delta\boldsymbol{x} = 0)$ for the closed-loop system. In the presence of non-zero $\boldsymbol{W}^{ee}$, however, a non-null $\Delta\boldsymbol{x}$ will be present at equilibrium.

For both fixed and non-stationary targets $\boldsymbol{x}^{des}$, if the external force $\boldsymbol{W}^{ee}$ is known to be due to a known/modelled resistance, then forces against the direction of motion can be cancelled out with an appropriate feed-forward term $\boldsymbol{u}^{\mathbf{ff}}$ in the control law (Eq. (2.5)), such that the

opposing interaction forces are selectively compensated in the direction of motion. For e.g., in a wiping task, the frictional force between end-effector and surface being polished can be computed using a good friction model. Then, appropriate feed-forward terms can be used to cancel out its effects so that the robot's motion is not hindered by it. This is similar to the gravity compensation term ($\boldsymbol{\eta}(\boldsymbol{q})$) which cancels out the constant force in the task-space acting at the end-effector of the robot (the effect is not a constant in the joint-space due to the non-linear mapping between the spaces). Therefore the control law can be modified to include such a feed-forward term to be used if and when such a predictive compensation model of external forces is available:

$$\boldsymbol{u} = \boldsymbol{\Lambda}(\boldsymbol{q})\ddot{\boldsymbol{x}}^{des} + \boldsymbol{\Gamma}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{x}}^{des} + \boldsymbol{\eta}(\boldsymbol{q}) + \mathbf{K}^{\mathbf{p}}\Delta\boldsymbol{x} + \mathbf{K}^{\mathbf{d}}\Delta\dot{\boldsymbol{x}} + \boldsymbol{u}^{\mathbf{ff}} \tag{2.7}$$

Feed-forward terms are also used to compensate for controller delays or other modelled imbalances in the system (see Section 2.2). In our controller, we use an incrementally updating forward model that predicts the wrenches acting at the end-effector, to provide the feed-forward term in our controller to compensate for the predicted external disturbances (explained in more detail later in Section 4.1).

2.1.3 Incorporating force control

A typical manipulation task is characterised by complex contact scenarios where some directions are subject to end-effector pose constraints while others are subject to interaction force constraints. For e.g., in an industrial grinding or polishing task, the robot has to apply a force on the surface of the object while moving along a perpendicular plane. Such tasks require both the generation of controlled force as well controlled motion. This can be achieved using a *hybrid force-motion controller*.

The hybrid force-motion controller is constrained by the force-velocity duality: *a direction of good velocity manipulability is a direction of poor force manipulability, and vice versa* [171]. In other words, it is only possible to control fully either velocity or force along any given direction. For instance, consider a robot firmly grasping a door [119]. The task space is $n = 6$ dimensional in SE(3), and the robot has 6 DoFs when free. However, when it grasps the door, it has only $6 - k = 1$ motion freedom of its end-effector, i.e., rotation about the door's hinges, and therefore $k = 5$ force freedoms. This means that the robot can apply any wrench that has zero moment about the axis of the hinges without moving the door.

The hybrid force-motion controller works by inherently defining *natural* and *artificial* constraints to the robot's end-effector [207]. Natural constraints are imposed on the robot *by the environment and/or task geometry*, either as a velocity constraint or a force constraint.

The robot can control only variables which are not subject to natural constraints (the hinge *naturally constrains* motion along 5 of the 6 motion dimensions in the previous example). The *reference values* for the variables in the remaining freedom dimensions are called artificial constraints as they are imposed by the task planner for executing the task. The two sets of constraints are complementary and cover the entire available degree of freedom for the robot in the space, allowing the complete specification of the task. Therefore, it possible to control motion along k direction and force along the other $6-k$ directions.

With this, we can include force control along directions complementary to that of motion in our task-space control equation (Eq. (2.7)):

$$\boldsymbol{u} = \mathbf{S^m}(\mathbf{K^p}\Delta\boldsymbol{x} + \mathbf{K^d}\Delta\dot{\boldsymbol{x}} + \boldsymbol{u^{ff}}) + \mathbf{S^f}(\boldsymbol{u^{fc}}) + \boldsymbol{H} \tag{2.8}$$

where $\boldsymbol{u^{fc}}$ is the force control term, The term $\boldsymbol{H}$ is a shorthand denoting the other dynamics compensation terms defined previously. $\mathbf{S^f}$ and $\mathbf{S^m}$ are binary vectors (comprising of 1s and 0s) complementary to each other so that the force and motion directions are orthogonal in the operational space. The notations for these complementary vectors will be ignored in text henceforth for easier readability, and the final control equation is simplified to

$$\boldsymbol{u} = \mathbf{K^p}\Delta\boldsymbol{x} + \mathbf{K^d}\Delta\dot{\boldsymbol{x}} + \boldsymbol{u^{ff}} + \boldsymbol{u^{fc}} + \boldsymbol{H} \tag{2.9}$$

where $\boldsymbol{u^{fc}}$ is to be understood to be force control commands in direction(s) orthogonal to its motion counterparts.

Force control equation for $\boldsymbol{u^{fc}}$

When the task requires applying specific forces and torques on the environment, the robot needs to make use of a force controller. A straightforward way of achieving force control is to make use of a force-torque measurements at the end-effector (either using an external force torque sensor or by estimating end-effector wrenches from joint efforts). Pure force control is something of an abstraction, since robots are usually able to move freely in at least some direction. However, this generalisation ties comfortably to the hybrid force-motion control described previously.

A typical force control law includes similar compensation terms for robot dynamics (at least gravity) as described previously for impedance control (see for e.g. [119] for derivation). A standard force control law used in many applications is a PI law of the form

$$\boldsymbol{u^{fc}} = \boldsymbol{W}^{des} + \mathbf{K^{fp}}\Delta\boldsymbol{W} + \mathbf{K^{fi}}\int \Delta\boldsymbol{W}(t)dt \tag{2.10}$$

where $\Delta \boldsymbol{W} = \boldsymbol{W}^{des} - \boldsymbol{W}^{ee}$ is the difference between the target wrench $\boldsymbol{W}^{des}$ and the measured end-effector wrench $\boldsymbol{W}^{ee}$, and $\mathbf{K^{fp}}$ and $\mathbf{K^{fi}}$ are the positive definite proportional and integral gain matrices respectively. The proportional term ensures that the force tracking error is reduced, while the integral term reduces any systematic bias. The first term ($\boldsymbol{W}^{des}$) ensures that the robot will measure the desired force at the end-effector when the force error $\Delta \boldsymbol{W}$ is zero. When a good gravity model is available and the robot is in stable contact, this control law can be directly plugged in to Eq. (2.9) to achieve and maintain the contact wrench $\boldsymbol{W}^{des}$.

Although this control law is appealing, it can be potentially dangerous if there is nothing for the robot to push against. When used in free space, such a control law will cause the robot to accelerate to make contact with an object in that direction. Since typical force control tasks require only small motion, one way to limit the acceleration is by adding a *velocity* damping term

$$\boldsymbol{u}^{\mathbf{fc}} = \boldsymbol{W}^{des} + \mathbf{K^{fp}} \Delta \boldsymbol{W} + \mathbf{K^{fi}} \int \Delta \boldsymbol{W}(t) dt - \mathbf{K^{fd}} \dot{\boldsymbol{x}} \tag{2.11}$$

The last term in the above equation ensures that the robot experiences a viscous resistance against high accelerations. Typically the value of $\mathbf{K^{fp}}$ is set to identity, while the others are chosen depending on the task and the scale of the target force.

2.1.4 Closing notes on task-space controller implementation

Regarding $\mathbf{S^f}$ and $\mathbf{S^m}$

The (6×6) matrix formats of $\mathbf{S^f}$ and $\mathbf{S^m}$ can be used instead of their vector forms (1×6) to achieve force or motion control in *any* direction in the base frame of the robot when designed correctly. Note that both the vector and matrix formulations allow for pure motion/impedance control (without force control) as well as pure force control (without motion control) if the appropriate active-direction vector is set to ones and its complementary to zeros.

Ignoring inertia term in the control law

The controller performance is extremely sensitive to the inertia term of the overall control law (Eq. (2.7)). Therefore, reliable control requires precise dynamics model of the robot and accurate acceleration sensors at the joints and/or end-effector. In practice, a perfect model of the dynamics of the robot may not be available and sensors are noisy. It is also observed that the inertia and control computations may be inefficient to calculate at the high controller frequency [119]. In such cases, it is common (especially if the desired accelerations are

small) to approximate Eq. (2.7) by removing the term $\boldsymbol{\Lambda}(\boldsymbol{q})\ddot{\boldsymbol{x}}^{des}$, which is followed in our implementations.

Null-space control

A side-effect of performing control in the task-space of the robot is the availability of null-space due to redundancy in the joint-space when there are more than 6 controllable joints for the robot. In these situations, the forces controlled in operational space have fewer dimensions than the robot has degrees of freedom. The redundancy makes it possible to accomplish a *secondary goal* along with the *primary goal* of tracking the target trajectory.

The null space of this primary controller is the region of the joint space where there is a redundancy of solutions. In these regions, the robot can move in certain ways and still not affect the pose of the end-effector. An example of this is the many configurations the elbow can be in while a person holds their hand at a fixed position in space. In such scenarios, a secondary controller can be designed to operate in the null space of the primary controller. The full control signal sent to the robot then becomes the sum of the primary control signal and a filtered version of the secondary control signal (for full derivation, refer to any standard textbook on robot manipulator control such as [171]). So the final transformed joint-space control command send to the robot is of the form:

$$\boldsymbol{\tau}^{\boldsymbol{u}} = \boldsymbol{J}^T \boldsymbol{u}_1 + (\mathbf{I} - \boldsymbol{J}^T \boldsymbol{J}^{+T}) \boldsymbol{u}_{null} \tag{2.12}$$

where $\boldsymbol{u_1}$ is the primary control goal (for e.g. Eq. (2.9)), and $\boldsymbol{u}_{null}$ is the secondary goal that the robot will try to achieve as long as it does not affect the primary objective (this is the function of the filtering term $\mathbf{I} - \boldsymbol{J}^T \boldsymbol{J}^{+T}$). In our implementations for the experiments in this thesis, we use a filtered secondary objective that tries to keep the robot away from joint limits (by staying close the joint median positions).

2.2 Forward models in robotics

An agent in an environment will have to decide on its actions based on observations as well as prediction about its own influence on the overall system. Therefore, the agent has to consider two important problems: (i) deduce or predict behaviour of the system (agent + environment), and (ii) use the information to decide on the action to be taken for manipulating the system.

In the domain of robotics, the first problem is seen as a pure modelling problem, where the system has to either determine missing data or predict future states from observations or

available data. The second problem is related to learning and developing control architectures which may make use of such models.

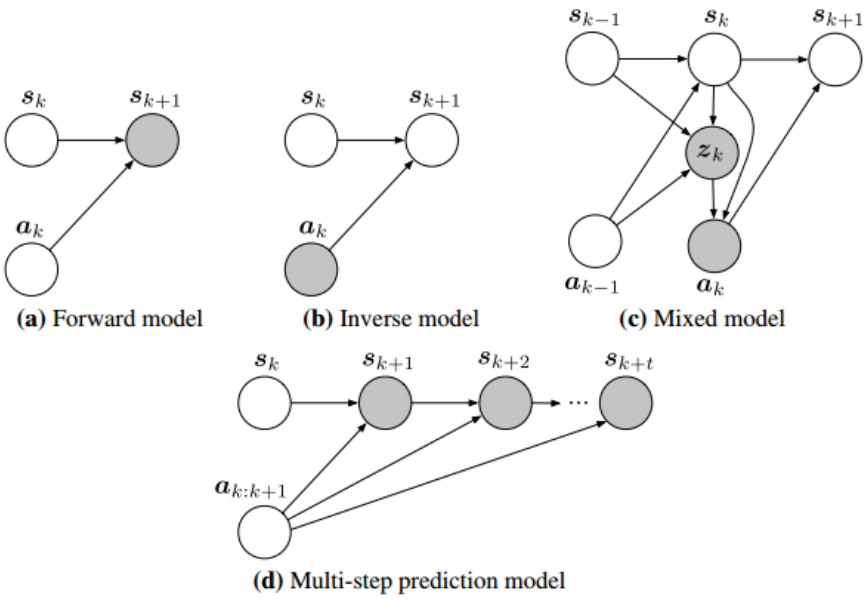


Fig. 2.3 Graphical illustrations of different types of models. The white nodes denote observed quantities, while the grey nodes represent inferred values. Image borrowed from [138].

Model learning provides a reference for the agent to deduce the evolution of the system when an action is taken, which allows the agent to estimate missing information. Depending on the type of observations and missing data, models can be distinguished as *forward models, inverse models, mixed models,* and *multi-step prediction models* [138].

Given an observed current state $\boldsymbol{s}_k$, and current action $\boldsymbol{a}_k$, a *forward model* attempts to predict the next state $\boldsymbol{s}_{k+1}$. *Inverse models*, on the other hand, are used to infer the current action from the current and desired/expected future state, i.e., $(\boldsymbol{s}_k, \boldsymbol{s}_{k+1}) \rightarrow \boldsymbol{a}_k$. Some approaches combine forward and inverse models to create *mixed models*. For applications where it is desirable to predict a series of states for the next t steps, we would use *multi-step/long-term prediction models*. Long-term prediction models can also be considered to be a form of forward model where prediction is done for a larger horizon instead of a one-step look-ahead. We do not discuss further about inverse or mixed models as they do not fall in the scope of this thesis.

As mentioned previously, forward models predict the next state of a dynamic system given the current state and action. Forward models also have a direct correspondence to state transition (transfer) functions f_s which typically describe the state evolution of dynamical systems: $\boldsymbol{s}_{k+1} = f_s(\boldsymbol{s}_k, \boldsymbol{a}_k) + \varepsilon_f$, where ε_f is the noise component. In practice, while the transfer function is used to express the physical properties and evolution of the system,

forward models try to capture the causal relationship between actions and states, so as to estimate/emulate the (unknown) transfer function.

Several studies in human motor control provides evidence of existence of forward-models in human motor control [129, 203, 61, 49, 128], often referred to as a body schema, i.e., a sensorimotor representation of the body used for action [65]. The predictions from these models are used for different purposes such as feed-forward control, motor system coordination, action planning and monitoring.

Forward models have been used in classical control since the Smith predictor, where a forward model was used to compensate for the delays in the feedback loop [178]. Forward models have since been widely used for developing model reference adaptive control (MRAC) methods [189, 208], where actions are adjusted based on the difference between the desired and current state of the system. Forward models (and long-term predictive models) have also been used in model predictive control (MPC) strategies, where optimal actions are computed by minimising some cost function over a certain prediction horizon in the future [158, 41]. (MRAC and MPC are discussed further in Section 3.2 along with some of their applications in literature.) With the evolution of machine learning, however, forward models have been modelled using different reinforcement and supervised learning settings. Several applications of forward models in robotics and manipulation are discussed in Section 3.1.

In this thesis, we develop an incrementally updating forward model to predict the end-effector forces and torques when the robot performs a manipulation task, which is used to obtain the feed-forward term as well as to guide the selection of the feedback gains of our controller online (see Section 4.1).

In problems such as open-loop control, it is more useful to have information of the system for the next t-steps. This is the multi-step prediction problem where the objective is to predict a sequence of states without the availability of measurements in the considered time frame. Developing multi-step prediction models is difficult due to the lack of measurements in the prediction horizon. A straightforward idea is to use single-step prediction models t times in sequence to obtain the multi-step prediction. However, this approach is susceptible to error accumulation due to prediction errors in previous values [138]. Alternative approaches that overcome such issues include using autoregressive models (commonly used in time-series prediction) where past predictions are used to predict future outcomes [9]. Since parametric structures are too limited for multi-step predictions of complex robotic systems, non-parametric models have also been explored in [84, 88]. However, due to the lack of system measurements in the form of feedback from the system, long-term prediction models are not suitable for real-time predictions of dynamical systems such as robot manipulation, especially when the interaction dynamics can be different and discontinuous due to making

and breaking of contacts. We explore this limitation of long-term predictive models in detail experimentally by comparing it with our online incremental hybrid model in Section 5.3.1.

2.3 Hybrid Dynamical Systems

Manipulation tasks involve changing contact states where the robot (or the object being manipulated by the robot) has to make or break contacts with other objects in the workspace. This brings rise to high discontinuities in the overall system dynamics (robot motion, acceleration, torques) as well as in the interaction dynamics (forces and torques at the end-effector and contact points). It is reasonable to consider these dynamics to be smooth at the other regions of the task where contact-changes do not occur. The evolution of the system state in such scenarios depends on the *contact mode* that the robot is in, i.e. the same action taken by the robot in different contact modes will result in different final states. This property of having continuous system states that are dependent on the value of a discretely varying mode makes robotic manipulation a prime example of a *hybrid system*. Hence, it is important to understand the characteristics and properties of hybrid dynamical systems before choosing a learning or control strategy to solve the challenges in a changing-contact manipulation task. This section briefly describes the formulation of hybrid systems and explains the main characteristics of such systems, which were utilised for developing the hybrid learning and control framework for manipulation tasks presented in this thesis (Chapter 5). The formulation closely follows the descriptions in [1], which is a comprehensive guide to the theory and control of hybrid systems.

While forward models of classical physics (discussed in Section 2.2) are unique mappings, there are several cases where one forward model alone do not provide sufficient information to uniquely determine the next system's state. In systems such as manipulation (as mentioned above), the system evolution can depend on other conditionals that describe the system. Hybrid systems fall under this category where the evolution of the continuous state s of the system depends on another discrete state (or mode) m.

Hybrid dynamical systems are characterised by the interaction of continuous and discrete dynamics [63]. As a simple example of a hybrid system, consider a temperature regulator in a room. In a simplistic form, the heating system can be assumed to either work at full power or to be turned off completely, hence becoming a system that can operate in one of two modes: "on" or "off." In each of these modes of operation, the evolution of the temperature can be described by a different continuous form. This system is then said to have a hybrid state consisting of a discrete member (taking the discrete values "on" and "off") and a continuous state taking values in the real numbers (temperature). In this example, the switching between

the two modes of operations is controlled by a thermostat, and depends directly on the current temperature value (continuous state). The thermostat switches the mode from "on" to "off" whenever the temperature reaches a pre-set threshold temperature. Vice versa, when the reaches a minimum value, the heating mode is switched "on."

This example can be used to explain the main properties and features of a hybrid system:

- The system is hybrid because it has a discrete state m (modes "on" and "off") and a continuous state s (temperature). The hybrid system can then be described using its state (s,m).
- The evolution of the continuous state s depends on the discrete state m, i.e., $s \in \mathcal{S}_m$ is an element in the continuous subspace $\mathcal{S}_m \subseteq \mathbb{R}^d$ (where the dimensionality $d = 1$ in the example) associated with the discrete mode m. (Depending on whether the heating mode is "on" or "off", the temperature change is reversed, and thereby the mode governs the evolution of the temperature.)
- The mode change of m depends directly or indirectly on the continuous state s and different conditions on the evolution of s can trigger the mode change of m. In the above example, the mode change between "on" and "off" is triggered by the temperature reaching the prescribed high or low limits.

In the simple example of the thermostat, the continuous state s does not 'jump' to a new value when the mode m changes, instead its velocity changes and it starts moving in a different direction. This is not generally the case in typical hybrid systems. A general hybrid system can have its continuous state s discretely move to a *new* disjoint value in a new continuous state $\mathcal{S}_{m_j}$ when it switches to a new mode m_j.

An example behaviour of a typical hybrid system with state (m,s) is shown in Fig. 2.4. It can be seen that the evolution consists of smooth phases in which the discrete state (mode) remains constant while the continuous state is smoothly varying. The discrete state m changes at the transition times (t_1,t_2,t_3). Simultaneously, the continuous state s can 'jump' to a discontinuous value far from its previous value. For instance, at time t_1 the state s changes abruptly from $s(t_1^-)$ to $s(t_1^+)$, which are values of s just before and just after the state jump, respectively.

At this point, it has to be made clear that the transitions are not necessarily prescribed by some clock (*'time-driven switching'*), but could also depend on other events such as some condition on the evolution of the discrete and/or the continuous state (*'event-driven switching'*. For instance, in the thermostat example these transition times were determined by the temperature reaching some pre-defined minimum or maximum values (state events).

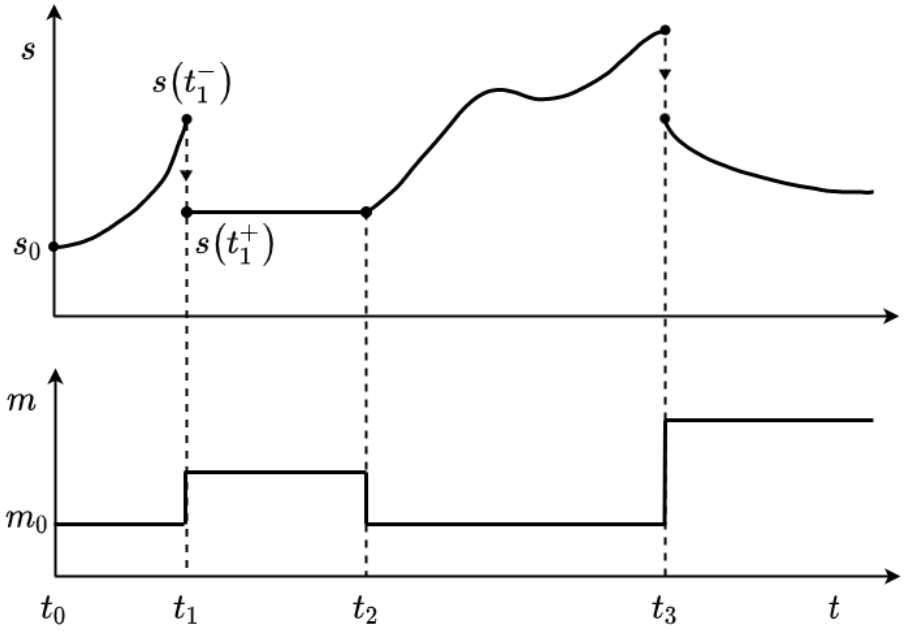


Fig. 2.4 Sample behaviour of a hybrid system with 1-dimensional continuous state s and discrete mode states m.

Similarly, in the task shown in Fig. 1.1, the robot experiences different interaction forces as it slides from one surface to the other (motion '2' in figure). Here the forces experienced by the robot can be considered to be smoothly changing when it slides on one surface, and it discretely switches to a new 'contact mode' when it slides across the boundary to the next surface. Similarly the contact modes also change when the robot makes or breaks contacts with different surfaces (e.g. at the end of motion '1' and '4' in figure), as it experiences different interaction forces for the same motion in these different modes.

In general, the trajectories of the hybrid system are partitioned into time intervals within which the continuous state evolves smoothly while the discrete state remains constant. At the boundaries of the intervals, the discrete state changes and the continuous state can 'jump' to a new state and follow a new pattern for continuous evolution.

2.3.1 Mathematical formulation of hybrid systems

Like any dynamical system that has some form of actuation, state evolution is usually also a function of some action $\boldsymbol{a}$ that the an agent takes. Without loss of generality, assume a d-dimensional system whose state $\boldsymbol{s}$ evolves according to the differential equation $\dot{\boldsymbol{s}} = \boldsymbol{f}(\boldsymbol{s}(t), \boldsymbol{a}(t))$. In a non-actuated system, $\boldsymbol{a} = 0$ at all times.

For a hybrid system, however, the transition vector field $\boldsymbol{f}$ depends on the currently active mode (discrete state) m, and hence the state evolution of a hybrid system can be represented as:

$$\dot{\boldsymbol{s}} = \boldsymbol{f}(\boldsymbol{s}(t), \boldsymbol{a}(t), m(t)) \tag{2.13}$$

As mentioned previously, the switching of the mode m could be time-driven or event-driven. In case of time-driven switching, the current mode is a function of time, and therefore the hybrid system state evolution (Eq. (2.13)) simplifies to $\dot{\boldsymbol{s}} = \boldsymbol{f}(\boldsymbol{s}(t), \boldsymbol{a}(t), t)$.

The switching can also be triggered by the continuous state $\boldsymbol{s}$ reaching a value in the **guard region**, $\mathcal{G}$, i.e., a mode switch from m_i to m_j can happen when $\boldsymbol{s}(t) \in \mathcal{G}(m_i, m_j)$. In the thermostat example, the guard region for $\mathcal{G}$("on","off") was a single-element set containing the maximum temperature for switching off the heating; when the temperature reaches this threshold, the operation mode changes to "off" and the temperature drops.

Another important phenomenon associated with hybrid systems is the *autonomous state jumps* that was described previously. At some time $\bar{t}$ when the mode transition occurs (such as at t_1 in Fig. 2.4), the state may jump from $\boldsymbol{s}(\bar{t}^-)$ to $\boldsymbol{s}(\bar{t}^+)$ abruptly. Another intuitive example is that of a bouncing ball whose velocity reverses (instantaneously) when the ball hits the ground. Typically, another relation called **reset map** $\mathcal{R}$ decides the value of $\boldsymbol{s}(\bar{t}^+)$ after a mode transition.

With this intuition, a hybrid dynamical system can be fully defined using the following:

- $\mathcal{S} \in \mathbb{R}^d$ is the continuous state space where the d-dimensional continuous state vector $\boldsymbol{s}$ operates.
- $\mathcal{M}$ is the discrete state space for all possible modes of operation m for the system, for e.g., $\mathcal{M} = \{0, 1, 2, ...\}$.
- $\boldsymbol{f}$ is a set of vector fields that describes the continuous dynamics of all separate modes $m \in \mathcal{M}$.
- *Init* is a set of initial values (m_0, s_0) of the hybrid system state.
- $\mathcal{G}$ is a set of guards describing when a discrete state transition occurs. For e.g., the guard $\mathcal{G}(m_0, m_1)$ poses a condition on the state $\boldsymbol{s}$ which has to be satisfied for triggering the mode transition $m_0 \rightarrow m_1$.
- δ is the discrete state transition function which describes the change of mode m to a new mode at $\mathcal{G}$.

- $\mathcal{R}$ is a set of reset maps that defines the state jumps during transitions. For e.g., if a hybrid system enters the guard region $\mathcal{G}(m_0, m_1)$ at $\bar{t}$, then the reset map $\mathcal{R}(m_0, m_1)$ defines the jump $\boldsymbol{s}(\bar{t}^-) \rightarrow \boldsymbol{s}(\bar{t}^+)$ during the transition. Another common way of representing the reset map as a function is: $\boldsymbol{s}(\bar{t}^+) = \mathcal{R}_{m_0,m_1}(\boldsymbol{s}(\bar{t}^-))$.

In some formulations, another set called the **invariants** (*Inv*) is described along with guards for technically differentiating the sufficient and necessary conditions for triggering a mode change [63]. In these descriptions, invariants and guards play complementary roles: while guards serve as an 'enabling' condition which denotes when a particular transition *may* occur, invariants describe when a transition *must* take place.

With the above formulation, any hybrid system can be considered to start with an initial hybrid state (m_0, s_0) which lies in the set defined by *Init*. From here, the continuous state $\boldsymbol{s}$ evolves according to its vector field $\dot{\boldsymbol{s}} = \boldsymbol{f}(\boldsymbol{s}(t), \boldsymbol{a}(t), m_0)$ with $\boldsymbol{s}(0) = \boldsymbol{s}_0$, and the discrete state m remains constant. If at some point $\boldsymbol{s}$ enters a guard region within the set $\mathcal{G}$, say $G(m_0, m_1)$, the transition $m_0 \rightarrow m_1$ is enabled. The discrete state may then change to m_1, and the continuous state will jump from its current $\boldsymbol{s}^-$ to $\boldsymbol{s}^+$ according to the reset map $\mathcal{R}_{m_0,m_1}$. The continuous state $\boldsymbol{s}$ will then follow the evolution according to $\dot{\boldsymbol{s}} = \boldsymbol{f}(\boldsymbol{s}(t), \boldsymbol{a}(t), m_1)$. This process is repeated till the system procedure completes. Therefore, in a hybrid system, the operating mode may switch between many discrete states where each mode is governed by its own continuous vector field. Mode transitions are triggered by some variable(s) crossing specific thresholds (state events) and/or by duration (time events) depending on the guard regions or invariants. With a change of mode, discontinuities or jumps in the continuous variable may occur defined by the corresponding reset map.

2.3.2 Relation to hybrid automata

The formulation of piece-wise continuous system described above directly leads to another common formulation in hybrid model theory called the *hybrid automaton*, which is often seen in robotics research [84, 108] for making use of the piecewise continuous nature of robotic tasks. Several definitions of hybrid automata exist in literature, but the difference is only in detail. A hybrid automaton is a more constrained formulation of the hybrid model described above, and the main difference is that while hybrid models work in a state space that is infinite, hybrid automata is defined in a finite space $\mathcal{H}$. Additionally, a hybrid automaton H is a tuple defined as:

$$H = (\mathcal{M}, \mathcal{S}, \boldsymbol{f}, Init, Inv, \delta, \mathcal{G}, \mathcal{R})$$

where the terms are as previously defined. The main difference is that $\mathcal{M}$ has to be a *finite* set of labelled discrete states called *control locations*, and $\mathcal{S}$ is a finite set of continuous variables. The hybrid state of the system H is given by $(m, \boldsymbol{s}) \in \mathcal{H}$ which exist in the finite state space $\mathcal{H} = \mathcal{M} \times \mathbb{R}^d$. For more details about hybrid automata, readers are referred to [96].

2.3.3 Closing notes on hybrid systems

Note that this section described hybrid systems using deterministic notations. However, it is important to understand that realistic systems are stochastic and practical applications use stochastic representations of hybrid systems for formulating problems and their control solutions. Stochastic formulation of hybrid system accounts for randomness and process noise in the state evolution, typically by formulating all the sets described above as probabilistic distributions with Gaussian white noise of appropriate dimensions. The stochastic formulation of hybrid systems follows directly from the discrete setup described above; interested readers are referred to [26].

Also, hybrid systems can depend more on system inputs $\boldsymbol{a}$ [63]. In such systems, the time at which the discrete mode changes, the value of the discrete mode after transition, the evolution of the discrete state, etc. can all be affected by the values of the input. The discrete mode changes in the hybrid systems considered in this report are assumed to not be affected by any inputs directly, or if they are, we consider them as an independent system evolving autonomously, over which the robot has no direct control.

The next chapter reviews some works from literature that are relevant to the presented framework as well to the different topics discussed in this thesis.

Chapter 3

Related Work

This chapter looks at different attempts in the research community to tackle the main challenges addressed in this thesis. Section 3.1 discusses some ways in which forward models have been used for robot manipulation. Section 3.2 then goes through different control strategies explored in literature for the purpose of tracking a prescribed target. In Section 3.3, we then look at some of the methods attempted to learn and apply variable impedance control for different robotic tasks. Some attempts at controlling hybrid dynamical systems are then considered in Section 3.4. We then discuss common methods that are seen in literature for handling collisions and contact changes (Section 3.5). Finally, we discuss the different state spaces used for learning and control in robot manipulation in Section 3.6. The chapter is concluded by briefly mentioning and justifying our choices for the learning and control spaces used in the development of the framework presented in this thesis.

3.1 Forward models for changing-contact manipulation

A typical manipulator robot will have to interact with the environment through contacts that can be either continuous (tasks like polishing, deburring etc) or discontinuous due to making or breaking of contacts (tasks like assembly). Such interactions will require some knowledge about the environment to understand how the interaction would affect the state of the robot, and to select or modify appropriate actions for achieving the objective. One way to tackle this problem is by enabling the robot to learn from its own experience, i.e., the robot observes (measures) the changes to the environment through its interactions and creates a model of the action-effects by using suitable machine learning/optimisation techniques. Access to such a *forward model* which can predict the effect of applying an action in a particular state (see Section 2.1.3) would help the robot anticipate/simulate the effects of its interaction in advance. In many practical cases, the response of the environment can be different from the

predictions of the learned model due to dynamic changes in the environment or due to unseen regions of the environment. In such cases, the robot would have to be able to incorporate new information as it observes them so as to modify its model of the environment, i.e., the learning framework should have the ability to adapt its model when new measurements cannot be explained by the current model of its environment. The ability to learn such adaptive forward models is important for a robot framework for it to be more generalisable, adaptable and autonomous [36]. The importance of learning models from observation to meet the demands of novel situations which cannot be met by traditional solutions is explored in [17].

Forward models have been applied to different fields of robotics; some of the applications include online sensor calibration [126], learning occupancy grid maps [192], and prediction of intent in human-robot collaborative works [37]. In [184], the authors used forward models for predicting the effects of a robot's action, and showed how such forward models lead to more efficient, robust and effective behaviour for achieving the objective in different task scenarios.

Several works in literature have used forward models to improve performance in robot manipulation tasks. In [193], a forward model created from high bandwidth tactile sensors is used to augment sensor measurements for better manipulation. A Bayesian framework was used to create forward models from the demonstrated samples in [170] to generate skills for a robot in real-time. Forward models have also been used for aiding push manipulation tasks [183, 13]. For an extensive survey on learning models for control in robots, readers are referred to [138].

In general, forward models are used to predict the behaviour of the robot [100, 69] and/or the objects [44] being effected while performing the task. The main challenge in the development of such a forward model is the selection of relevant state features of the task. Effective learning and performance of a forward model relies very much on the choice of the feature vector. forward models broadly fall into three categories (i) analytic models - which are typically modelled mathematically using Newtonian physics; (ii) learned models - built from data using techniques from machine learning; and (iii) hybrid models - a combination of learned and analytic models.

Analytic models are informed by the knowledge of mechanics to make predictions about robot and object motions [44, 116]. They assume that Newtonian mechanics govern the dynamics of the object and estimate parameters like the mass, inertia, Coriolis component and gravity effects using regression techniques [122, 123, 118]. However, most methods make unrealistic assumptions such as quasi-static mechanics, zero slippage and point contacts to make the derivations tractable [29, 45]. Hence analytic methods often suffer from the problem of inaccurate prediction. Many of these approaches also require an explicit representation

of its intrinsic parameters, such as friction, mass, mass distribution, and coefficients of restitution, which are not trivial to estimate [93].

The second approach is to use techniques from machine learning to build forward models of the system. These techniques learn an action-effect correlation either using data obtained from expert demonstrations [100, 69] or from self experience using trials [112, 105]. Many methods have been developed to address the learning and control problems in robot manipulation [99], especially methods based on reinforcement learning (RL) [185] and those combining deep networks and RL for learning flexible behaviors from complex data [12, 62, 117].

Learning a prediction model in a contact-rich environment poses several unique challenges to model learning approaches. Under rigid-body assumptions, establishing contacts is often mathematically modelled as discontinuous dynamics with an impulsive update of velocity, acceleration, and force [18]. This is usually overlooked by existing learning methods which leads to their poor performance. Furthermore, applying traditional model learning algorithms (such as from model-based reinforcement learning) requires large number of training trials, often collected through multiple repetitions of the task by the robot. These requirements are difficult to satisfy in practical domains, especially on a physical robot due to wear and tear of hardware. Therefore, such methods are data-inefficient and, in case many real-world changing-contact tasks, impractical. Also, the training process optimises several parameters, and the internal representations and decision-making mechanisms are opaque, making it computationally expensive to learn action policies and difficult to transfer them to new tasks. Although these methods do not require explicit mathematical representations of the task, robot, or the objects involved, the choices regarding the model representation, learning algorithm and state representation are challenges that need to be tackled. As mentioned, these methods are also not well-suited for modelling hybrid dynamics because they implicitly or explicitly consider a single model for the entire manipulation task [99]. RL and optimal control methods for robot manipulation often assume the task dynamics are smooth, which will result in poor performance in changing-contact tasks.

Recently, several attempts have been made to use deep neural networks (DNNs) for end-to-end learning of manipulation tasks which bypass the step of learning a separate policy for modelling dynamics and a control policy [12, 133, 6]. DNNs by design can easily overcome some of the issues of analytical methods such as knowledge of robot-environment models, physical parameters of models, and expertise is mechanics and optimisation. Their flexibility also circumvents making restrictive assumptions such as object rigidity, simplistic friction models, and inelastic impacts. However, they also introduce challenges that are unique to DNNs due to their high-dimensional optimisation setting. The most obvious and difficult

challenge is its need for a large amount of training data collected in almost all possible scenarios so as to cover the maximum region of the high-dimensional state space. This often requires a lot of training hours and high-end computing resources. Another unique challenge is its tendency to entangle complex physical attributes in the optimisation manifold. For instance, stiffness in the learned dynamics brings both stiffness and local minima into the optimisation landscape [144]. Analytical system identification methods attempt to solve this by exploiting second-order [90] or global [64] optimisation strategies. But it is not straightforward to apply such strategies to the high-dimensional parameter spaces of DNNs. Another important issue arises due to the feature (advantage and limitation) of DNNs which is their ability to approximate any dynamical system by selecting the smoothest interpolation of the training data, which conflicts with the inherently discontinuous nature of contact changes and impact dynamics. This behaviour is prevalent in common training methods such as stochastic gradient descent, and is the primary goal of including regularisers such as weight-decay in the optimisation algorithms. This smoothing nature is harmful and counterproductive for modelling discontinuous systems such as changing-contact manipulation. Some applications and limitations of DNNs used for end-to-end learning of manipulation tasks are discussed later in Section 3.6.

The third approach is to use a combination of both analytical and learning approaches discussed above. These forward models tend to learn the difference between analytical models and the true dynamics of the interactions, capturing the improvements to the analytical model needed to match the observed environment (Gandhi et al. [53]). The main advantage of these forward models is low requirement of data since only the difference in dynamics are to be learned. However, these strategies still rely on several restrictive assumptions regarding the type of contacts, friction models, object dynamics, etc., require significant knowledge about the mathematical models to be used, and also require at least a few trials and examples of the real-world possible scenarios to be modelled. Although sim-to-real strategies have been developed to further reduce the training on real robots, aspects such as the dynamics of rigid bodies with friction are too complicated to be modelled in a real-time dynamics simulator [79]. Often, even with reduced training on real robots, several hundreds of trials in the real world is then required for fine-tuning the models for transferring to a physical robot [6]. The main disadvantage of using physics simulators as proxies for physical robots lies in the difficulty in modelling contacts accurately and imitating real-world interactions between physical objects (discussed further in Section 3.5).

There have also been works which makes use of the piecewise continuous nature of changing-contact tasks by modelling such tasks as hybrid systems or hybrid automata (see Section 2.3). The models learnt in such fashion typically have a hierarchical structure, with

a high-level model that selects or identifies the current mode, and a gating strategy to use a corresponding low-level model which models the dynamics of that mode [108, 23, 95]. Such methods are also commonly used for bipedal locomotion [134]. Planning methods for manipulation also often take the discontinuous nature of manipulation into account [195, 75], but they assume prior knowledge of the models of the system, and of the actions and modes of interest. The disadvantages of such approaches are that they often require the knowledge of the number of modes and their sequence beforehand. In fact, several methods require generating synthetic training data to account for the sparseness of data at the guard regions [108].

In contrast to the above works, our framework (see Chapters 4 and 5) can learn dynamics of robot-environment interaction as a hierarchical model without assuming any prior knowledge about the system. The model is learned online, makes no assumptions on the structure of the dynamics, and can incrementally learn to model new scenarios as it sees them.

3.2 Control strategies for path-following

The research on developing controllers for establishing stability in different dynamical systems has been popular for almost two centuries. Several methods have been developed for controlling systems in a stable and robust manner. The most basic plant system are linear time-invariant (LTI) systems, for which methods from classical control theory can produce the desired performance. Such methods have been reviewed in many works in literature (eg: [58]).

In most real-world systems including robot manipulators, however, the model of the plant (robot and/or interacting objects) is non-linear and/or unknown. In case of manipulators, the model sometimes can be unmodelled or difficult to model due to interactions with other objects, unknown tools/objects at the end-effector, environment dynamics, etc. Research in classical control has developed many methods to achieve precise movement during interactions with the environment, e.g., hybrid force control [164], parallel force control [31], and impedance control [66]. These approaches require accurate knowledge of the system's dynamics and/or precise feedback schemes. For a manipulator robot, it is possible to achieve accurate motion tracking and better rejection of perturbances by using a constant high stiffness controller, but being very stiff expends more energy and the robot will not be *compliant* to external forces, hence making interactions unsafe. Such classical robot control methods characterised by high gain negative error feedback control are not suitable for tasks that involve interaction with the environment (possibly humans) because of possible high impact and interaction forces [3]. To overcome this at the low level, this is usually handled

by bringing a state-dependent or time-dependent variability in the controller parameters or the dynamics model of the system. These non-linearities in the dynamics raise the need for control schemes that are capable of handling system uncertainties, or are adaptive and can change according to the system's response and/or observations (measurements).

Model Predictive Controllers (MPCs) have shown remarkable results in systems whose models are known but the parameters are unknown *constants* [137]. MPC aims to predict the future behaviour of the process during execution to obtain the optimal closed-loop control to minimise the objective function in a limited span of time horizons within certain constraints [41]. MPCs have been widely used in controlling mobile and aerial robots [10, 11, 158]. They have also been used as control schemes for manipulation systems. For instance, in [72], an MPC formulation is used for controlling a 24-DoF pneumatically actuated soft robotic hand. MPCs have also been used for providing position and force tracking for robot-environment interactions [56]. The disadvantages of model predictive control are that it requires knowledge of the structure of the system model, and relies on it being fixed, which is not a valid assumption in dynamic environments where the interaction dynamics can change or if the robot has to make or break contacts with other objects in the environment. Moreover, the MPC algorithm can become complex and would need longer computation time than other controllers in many situations [41].

In scenarios where the system parameters can vary, but one knows the extent or bound for the parameter change, there are methods for developing a fixed controller which is able to handle such parameter changes and can guarantee stability as long as they are within in the pre-defined bounds. This type of control is referred to as *robust control* (see [39] for review). However, robust control strategies are generally known to be less efficient and has been shown to be limited in the extent of uncertainty it can handle even for simple first-order linear systems [209].

Adaptive control [14] is a popular class of controllers for dealing with systems whose dynamics are unpredictable, unmodelled, or variable. Adaptive control adjusts itself to manipulated systems whose parameters can change constantly or are unknown beforehand using feedback from the system. Non-adaptive controllers require the control system to be modelled on the basis of the prior knowledge regarding the system, i.e., the system is known and the controller is intended for that system by assuming that the system will not change. In contrast, adaptive controllers do not have to rely on the prior data from the system and is able to handle system changes by adjusting their parameters automatically. It is used when the controlled plant faces uncertain interference, or it is anticipated to experience changes in its parameters in a fashion that is unknown at the start [208]. There are typically two popular adaptive control strategies seen in control literature: (i) model reference adaptive controllers

(MRAC), and (ii) self-tuning regulators/controllers. Recently, however, a third strategy has become popular in many fields such as robotics: (iii) gain scheduling.

The basic idea of the *model reference adaptive controller (MRAC)* is to compute control signals which will force the controlled system to behave as a specified reference model [191, 208, 151]. It is an example of indirect control whereby the control law is adapted so as to mimic a reference model which has the same order as the original system [206]. The advantage of MRAC is that one does not have to exactly know the plant parameters, rather, approximations can be employed and the adaptive control system can use prior input/output data to improve these approximations. However, there are significant challenges for successfully deploying this type of control. The control system's stability performance is crucial because it is hard to develop a stable adaptation rule. Furthermore, the MRAC system depends on cancelling the non-linearities of the system dynamics via the reference model [189]. In practice, complete cancellation cannot be ensured and control engineers try to bring the non-linear segments to small points in the dynamics space where they can be neglected. Although the exact knowledge of the system dynamics is not needed for developing MRAC, it still requires a good understanding of the system and its model for defining a sensible reference model. Providing a wrong or incompatible reference model can produce instability in the system and have devastating effects (see Section 4.4.1). A comprehensive review of works from literature which employ MRAC to robotic manipulators is presented in [208].

Self-tuning regulators differ from MRAC in that they typically represent the controlled plant as a linear time-varying system, and tries to either (a) vary the controller parameters online based on system feedback, or (b) try to estimate the parameters of the controlled system during task execution for substituting in the control law [151, 89]. It has been demonstrated that under certain conditions, a self-tuning controller tracking a target point, designed using estimated system model parameters, can converge to the optimal controller obtained for the model with the true parameter values [55]. However, for manipulation tasks, the objective is transient instead of fixed, since the goal is not to reach a single point but to follow a trajectory. Therefore, the convergence of the parameter estimates and controller gains of traditional self-tuning controllers may not be achieved during the finite time over which the motion takes place, or it is possible that the values in one point of the trajectory is not sensible at other points due to different system dynamics [89] (see Section 4.4.2 for experimental validation). In repetitive tasks, however, the estimates of the parameters from the previous run can be used as the initial estimates, which can then be improved in subsequent trials. Thus, the controller is 'trainable' in repetitive tasks, resulting in a more accurate performance for systems whose dynamics vary in a fixed pattern across all trials. Under such situation, a controller can be designed or trained for every single anticipated

operating point. This type of control adaptation falls under the next category of adaptive control, called gain-scheduling.

Gain-scheduling is not commonly seen in classical control literature, but has become more popular with the adoption of imitation learning and reinforcement learning in robotics. Standard strategies for gain scheduling involve learning a time-indexed sequence of control parameters that are either learned by repeating the task and iteratively updating the gains by minimising some cost (e.g. [21, 80, 204, 97]), or by learning time- or state-dependent gain profiles from pre-collected training data (e.g. [100, 113, 107, 162]). Many of the variable impedance strategies mentioned in Section 3.3 using reinforcement learning or learning-from-demonstrations can be considered to be in this category.

The adaptive variable impedance control (AVIC) framework presented in this thesis (see Chapter 4) can be considered to be closest to the self-tuning adaptive control paradigm mentioned previously. However, the presented control framework uses the accuracy of a forward model (which is learned incrementally during task execution) for guiding the online adaptation of the controller parameters (Section 4.1.3). And unlike other online learning methods [204], our control framework does not require a periodically repeating trajectory, nor does it learn a time-series of controller parameters to be used in a repeatable dynamic environment. In Section 4.4, the performance of AVIC is experimentally compared with the three classes of adaptive control discussed in this section.

3.3 Variable impedance control for robot manipulation

Impedance control provides a feasible solution for overcoming position uncertainties and in avoiding large forces due to interaction since the full impedance control law for robot manipulators are designed to modulate their motion and/or compliance based on real-time system feedback. As explained in Section 2.1, an impedance controller resembles a virtual mass-spring-damper system between the environment and the robot. Impedance control has been used in the Cartesian space to control the interaction of the end-effector with the environment [172, 24, 115], as well as in the joint-space for improving safety [114]. However, in many real-world tasks robots may need to vary their impedance during the execution of the task. Consider a scenario where a domestic robot has to navigate an unstructured environment (eg. homes, industrial floors, etc.) where they have to interact with different objects. Such tasks demand the application of different control forces and torques depending on the different mass, friction, etc.

Variable impedance control in general provide a time-varying impedance profile for the robot controller during task execution; by definition, such control strategies belong to the

class of gain-scheduling adaptive controllers described in the previous section. In variable impedance control methods, the impedance parameters could change as a function of time, robot state, or other observations/measurements of the interacting system.

Many existing methods for robot manipulation use machine learning algorithms to compute suitable values of the impedance parameters for the task at hand [21, 80, 68, 104, 38]. These methods either represent the desired stiffness profiles as a time series or as a task-specific policy, and need large labelled training datasets. Several methods learn impedance profiles or impedance models from demonstrations that are provided in the form of kinesthetic teaching (e.g. [100, 2]), which is then used to obtain the desired impedance behaviour during task execution. If the provided demonstrations from the teacher do not overlap perfectly, it is possible to derive a policy that imitates the average motion of the demonstrated dataset, and the robot can then vary the stiffness according to the coherence of the trials [25, 100] or based on the force sensed during replay [2]. Such use of variable impedance as an action representation has been shown to be successful for adaptive grasping [113], manipulation of deformable objects [107], and human-robot collaboration [163]. In [2], a variable impedance controller is learned from demonstrations and sensed forces using tensor-based Gaussian Mixture Models (GMM), which is then used to provide a time-varying stiffness profile that allows the robot to react satisfactorily to new conditions. GMM was also employed by Kronander and Billard [100] to directly encode full stiffness matrices using Cholesky decomposition. However, the approach requires the teacher to wiggle the robot during training to make it more compliant, or increase the pressure with which the robot is held to make it stiffer. Such approaches may not be feasible in all cases and is not suitable for tasks that involve external physical contacts. Saveriano and Lee [165] uses a similar approach while exploiting the constraints derived by Khansari-Zadeh and Billard [86] to guarantee the convergence of the trajectory retrieved via Gaussian Mixture Regression (GMR). In [113], GMM was used to encode the pose of the end-effector, after which the impedance parameters and reference trajectory were estimated using optimisation. Rozo et al. [162] proposed a framework for learning stiffness profile in a cooperative assembly task which used visual and haptic sensor data, which was used to build Task-Parameterised GMM (TP-GMM) where each Gaussian component corresponds to an independent stiffness matrix. This was later extended to reformulate the stiffness estimation as a convex optimisation problem for ensuring optimality of the stiffness matrices in [163]. Peternel and Ajoudani [150] proposed a method based on Dynamic Movement Primitives (DMP) where a ‘novice’ robot can learn variable impedance behaviour from an ‘expert’ robot through collaborative task during online execution.

Some works propose iterative learning approaches where the agent is assumed to be able to repeat a task till some measure of 'good performance' is achieved [30, 52, 204, 97]. Conventional methods try to reduce trajectory tracking error as well as improve rejection of *periodic* disturbances. This typically involves learning a corrective term for the control law that linearly depends on tracking error, measured disturbance, and/or time. Such strategies can be included in the class of RL-based gain-scheduling adaptive control mentioned in Section 3.2.

Many learning methods often depend too much on the control strategy that it becomes difficult to define a clear separation between the controller design and task planning. In addition to modifying the controller parameters during execution, many such methods tend to also adapt the reference trajectory, while typically relying on constant inertia matrices [3]. An active learning control strategy for estimating variable stiffness is proposed by Calinon et al. [25]. However, their approach also requires demonstrations to have variability in position which is used to capture stiffness information. This does not arise in scenarios where the robot is constrained to follow a predefined plan. Khansari-Zadeh et al. [87] proposed the Integrated MOtion Generator and Impedance Controller (i-MOGIC) which derives the robot trajectory and variable impedance profile from a GMM which is then used to compute the control input. Khader et al. [85] proposed an RL-based variable impedance controller built over i-MOGIC for a robot peg-insertion task, where stability is guaranteed by bounding rollouts to the state-space and ensuring that they tend towards the goal of the task. Buchli et al. [21] used Dynamical Movement Primitives (DMP) to encode both reference and joint impedance trajectories in different simulated and physical robot experiments. A mixture of proportional-derivative systems is used for representing and learning a policy for flipping pancakes in [94].

Some approaches vary stiffness from the perspective of the object being manipulated [113, 166, 202]. These methods have mostly been designed for grasping and require accurate analytic models of the object; it is challenging to provide such models in dynamic domains. A variable impedance controller which varies stiffness based on the friction coefficient (estimated via prior exploration) is proposed in [7].

The stability of a variable impedance control scheme depends on how the impedance gains vary. Although stability issues of impedance control have been studied from the beginning [66], stability in VIC is not a trivial problem and has only been recently considered in literature [3]. Stability of variable impedance controllers have been shown using biomimetic controllers which are capable of automatically adjusting controller stiffness [54, 204]. However, these methods requires a fixed repeatable reference trajectory. Analysis of interaction stability of variable impedance controllers using Lyapunov-based tools becomes intractable

when the environment dynamics are unknown or too complex [3]. This is clearly the case when a robot physically interacts with a dynamic environment or with a human operator. Therefore, researchers quickly converged on using passivity-based proofs for investigating stability. In simplified terms, the passivity theory provides a mathematical method to describe and verify that a dynamical system is not producing more energy than it receives. For more information regarding passivity, the readers are referred to [197]. One such method where passivity-based approach is used to ensure the stability of time-varying impedance controller is [46], where passivity is ensured by incorporating a virtual energy tank and checking that the energy dissipated to the task is greater than the energy pumped into the system. Several other works have also used the strategy of constraining variable impedance matrices using passivity theory to guarantee the stability on the desired impedance dynamics (e.g. [101, 187, 87, 85]).

A robot can use a variable impedance controller to change the impedance parameters to match the desired motion profile. Most variable impedance control methods reviewed in this section are time-dependent or include joint space parameters as a part of the state description. This dependence can make the task model out of sync with task execution in the presence of unforeseen perturbances, limiting the ability to adapt impedance intelligently to dynamic and/or unseen environments.

Human motor control literature provides evidence that humans exhibit task-dependent varying impedance behaviour [141, 57, 50, 190, 200]. The impedance modulation in humans is achieved by co-contraction of muscles [160]. Studies have also shown that the motivation for this modulation of stiffness is to deal with unstable dynamics [22] and sensorimotor noise [167]. Yang et al. [204] proposed a controller based on this idea, which was later extended by Ganesh et al. [54]. The controller provided a constant feed-forward force for predictable perturbations and increasing stiffness in directions of unpredictable perturbations. Studies performed by Ajoudani et al. [8] report that human-assisted stiffness modulation via tele-operation outperforms constant or low stiffness behaviour for a given task. It has also been suggested that forward model-based control is often easier to learn [128].

We used these insights to develop our AVIC framework (Chapter 4) which uses an incrementally updating feed-forward model that compensates for continuously changing environments, and varies feedback gains based on the accuracy of the learned model.

3.4 Control for hybrid systems

Several control methods have been proposed in the control theory literature for specific classes of hybrid systems. For instance, a wide range of literature can be found for switched systems where stabilising controllers are developed using Lyapunov arguments [77, 34, 130].

Optimal control theory has been used for developing control approaches for hybrid systems in the context of manufacturing [34, 149]. Recently, model predictive control (MPC) has also been extended to some classes of hybrid systems [127, 34].

Benefits of incorporating modes or phases in the design of controllers for manipulation is evident in grasping tasks (e.g. [159]) where the controller has distinct objectives and behaviour requirements for phases such as 'approach', 'grasp' and 'release'. Different strategies for sequencing motion primitives have also been used to solve manipulation tasks, but they assume the existence of a library of modes or motion primitives, or alternatively segment a sequence of primitives from human demonstrations [140]. These methods do not consider the interaction dynamics or try to reduce effects of impact. Therefore, this type of modelling makes the learned policy dependent on the environment, movements and the sequence of modes.

Motivated from the effectiveness and flaws from the works discussed in the previous sections, the framework proposed in this thesis uses independent task-space adaptive variable impedance controllers for each identified mode in the task. The framework also learns transition-phase controllers to handle contact changes so as to minimise impact forces and jerk on collision, while trying to minimise the errors in tracking accuracy. This makes the framework more generalisable and adaptable to new environments.

3.5 Modelling and handling impacts & contact changes

Collisions and impact dynamics introduce critical challenges to planning, modelling and control of robots in applications such as locomotion [201] and manipulation [83]. Even a single collision is a complex interaction where object interpenetration is prevented by material deformation, and which often occurs on a scale far below the resolution of practical sensors [60]. Capturing these processes accurately requires an impractically precise set of knowledge regarding the materials, geometries, and initial conditions, on top of the complications in predicting dynamics using these information [28]. To avoid these challenges, most robotics approaches make several coarse approximations of contact dynamics such as the rigid-body assumption to make the problem more tractable (see [20] for background). When impacts occur, rigid-body models approximate the event as an instantaneous change in velocity due to an impulsive force. Even so, seemingly minor changes in the mathematical models can result in significantly different predictions from identical initial conditions, and in many cases, they are unable to capture real-world behaviours with any available model [45, 182].

There have been several attempts to analytically model the relation between relative motion between two colliding objects and their impact dynamics. The problem has mainly

been treated as the linear complementary problem (LCP) for the velocities and impulses [152] or the accelerations and forces [205] at contact points. Although such formulations can guarantee a physical consistency between motion (velocities and/or accelerations) and impulses, their computing cost is large because the LCP has to be solved at every time step of motion, and they assume the contact forces are uniquely determined for a motion. A different LCP formulation for evaluating velocities of colliding bodies is employed by Routh et al. [161], where they use a discrete model based on the law of conservation of momentum, Newton's law, Poisson's Hypothesis etc. Several formulations of the impact transition problem use continuous-time models that are based on equations of motion, with assumptions and simplifications such as approximating impulse to finite forces and finite impact time [71, 132]. A method for applying sums-of-squares verification to rigid bodies undergoing discontinuous, inelastic impact events is presented in [156]. These methods however require defining the compliances at contact points for preventing bodies from penetrating each other, which are difficult to determine. In [90], dynamics in robot manipulation is posed as a joint optimisation of both trajectory and model parameters, and is solved using Newton's second-order optimisation method. In all these studies, the static model of Coulomb friction is used, which can be unrealistic in many cases due to approximations made for overcoming its discontinuities. In [135], the authors use the LuGre model [35] which is dynamical friction model, for modelling contact transition in robot manipulation, which was shown to produce more realistic simulations although it requires several more parameters of the system to be known.

Unrealistic contact models is a primary reason for the gap between simulated and real-world performance in robotics problems [144]. When objects collide in the physical world, materials deform on an imperceptibly small spatial and temporal scale, preventing inter-penetration. The underlying material property driving this is mechanical stiffness, which significantly influence the equations of motion of these systems [144]. Even small errors in the initial conditions or model parameters can produce significantly different predictions. Furthermore, velocity measurements are extremely sensitive to the measured time, as they change almost instantaneously during impacts. These issues become even more significant when learning a model of a real system from noisy sensor measurements. Many simulators allow users to specify several physics parameters and physical properties of objects such as mechanical stiffness (e.g. MuJoCo [194], Bullet [33]), which can produce relatively realistic simulations. However, even with a perfect simulator, this requires accurate knowledge about object parameters such as material stiffness.

Impacts affect the interaction dynamics due to the almost instantaneous changes to the values of velocity, acceleration and forces. The smoothing effect of deep neural networks

mentioned in Section 3.1 is therefore particularly harmful for modelling impacts and collisions. This is especially significant due to the sparsity of reliable data points that can be collected during and around impact which is pronounced by the lower reliability of an average sensor in these regions. Parmar et al. [144] provides an excellent discussion on the main challenges of using deep learning systems (as well as analytical methods) for modelling contact dynamics, an important one being the degradation of model performance with increasing stiffness.

Therefore, deep learning models are unsuitable for contact modelling and prediction for these reasons while analytical approaches generally require information about the physical properties such as mass, restitution and frictional properties of the objects as well as, in many cases, detailed 3D object models. The information regarding the physical properties of all the involved objects may not always be readily available, and they are cumbersome to collect for different tasks/objects. Even with these information, analytical methods may not be reliable in real-time applications for providing accurate estimates due to the complex computations that are often required. However, instead of modelling contact dynamics directly, the robot can decouple the problem by first approximating the *positions of contact points*, and then using a 'safer' controller in the predicted regions. Static contact properties such as *contact positions* and *direction of impact* can be estimated more reliably (if the task plan is known beforehand) using either tactile/force-torque sensors or coarse depth images/point cloud models of the objects. Acquiring training samples for such measurements are also usually easier than performing an analytical analysis of the interaction. In [196], a robot is shown to acquire the training samples' labels autonomously by interacting with objects to learn high-level rules for the objects that can be used for planning. Such interactive perception techniques have been used in several scenarios such as to estimate constraints or physical properties of objects [81, 16]. The benefits of interactive perception are that they do not necessarily need pre-training and they help disambiguate between scenarios as well as in observing otherwise latent properties [99]. For instance, a robot can figure out if an object is fixed or movable by pushing it. However, such methods usually require the robot to perform the task multiple times to build models or optimise model parameters, especially when trying to model dynamics properties of the interaction. In our contact-anticipation model (Chapter 6), we focus on interactively improving the knowledge of the robot about the locations of the contacts involved in the task using a Kalman-filter based update algorithm. These estimates of the locations of contacts are then used to define 'transition' regions in the workspace of the robot, where it can expect contact-changes to occur, and therefore use a safer control strategy in such regions.

Using separate controllers is a common strategy for handling contact changes. Methods that use a transition-phase controller for changing-contact manipulation tasks focus on minimising the discontinuities in the dynamics such that the controller is stable and tracks accurately after transition [131, 120, 175]. However, these methods generally switch to a different controller only after a contact is detected, which can result in significant disruptions in the dynamics when the switch is made. These transitions results in the intake of large amount of energy into the system large spikes in forces and acceleration, and could potentially damage the robot or the domain objects. In contrast to these methods, our contact-handling module aims to predict the contacts and use it to transition to a safer 'transition-phase' controller smoothly on-the-fly by interpolating between the control outputs *before* collision occurs. The 'transition-phase' in our approach uses a low-stiffness controller and follows a slower velocity profile which would produce lower discontinuities at contact. Hyde et al. [73] uses a transition-phase controller with constant low velocity to reduce impact effects in the guard regions of manipulation tasks modelled as hybrid systems. However, unlike their approach, our framework also uses a low-stiffness controller to reduce impact on collision, and can also automatically update the choice of approach velocity based on a desired impact force (Section 6.4).

For the least delays and deviation from the original plan, the new approach velocity of the transition-phase controller should come into effect only when the robot is about to make a contact. Modifying the velocity requires modifying the timeline of the provided kinematic plan. To ensure that the trajectory plan is smooth, it is necessary for both the velocity profile and the geometric path to be continuous. When executing a planned trajectory, two of the most important factors are smoothness and accuracy of tracking. Since the velocity of motion has to be modified for safer collision, the tracking accuracy (in terms of time to task completion, but not in deviation from the path) has to be sacrificed. However, motion smoothness (motion derivatives at least up to jerk) can be guaranteed by making the transition trajectory continuous. To make the motion smooth in acceleration and jerk, a motion profile that is at least C^4 smooth is required. For kinematic time-optimal motion, different variants of trapezoidal velocity profiles are commonly seen [19]. For avoiding overshooting and undershooting of trajectories that happens with these methods due to lack of consideration the length of the original displacement trajectory, the motion constraints and total displacement can be recalculated iteratively [59, 92]. Several methods that create C^4 smooth trajectories using multiple trajectory segments have also been developed [5, 136]. However, the complexity of such methods can become high for obtaining a feasible trajectory plan at every iteration of the task and often has multiple task-dependent hyperparameters to tune. Several minimum-jerk motion profiles are also found in literature [153, 106, 51, 70].

A trapezoidal C^4 smooth motion profile for point-to-point motion has been described in [59] using a seventh-order C^3 polynomial function as the velocity profile. In contrast, our transition-phase controller uses a novel velocity profile which is simpler in formulation with no additional hyperparameters and has continuous derivatives of all orders at every point of the function, making it C^∞ smooth (Section 6.5.2).

3.6 On state spaces for learning and control

Performance in a manipulation task is significantly affected by the choice of state space used for learning and control. Several earlier works have focused on applying learning methods in the joint-space of the robot for solving manipulation problems. The two most popular choices for learning in the joint-space are learning joint torques directly [40, 111, 155], and learning reference positions for a joint-space PD controller [12, 148]. Using joint-space as the action space is favourable because they are easy to implement, and inject very little bias into the learning problem [198]. On the other hand, learning in the joint-space require learning to compensate for dynamic effects (e.g., inertial, gravitational, and centrifugal forces), which could otherwise have been modelled separately or, in most cases, are directly available. This raises the need for larger training examples and training time to accurately learn the dynamics model for reasonably good performance. Even then, it often becomes necessary to implement gravity compensation separately to learn successful manipulation strategies when using torque control [80].

Learning and optimisation methods work best in the space where the task objectives are defined directly. In a standard contact-rich manipulation task, the objectives are directly and easily defined in the Cartesian task-space of the robot in terms of end-effector trajectory, obstacle locations, end-effector target wrenches, etc. This intuitively supports the choice of learning policies in the task-space of the robots. Furthermore, task-space definitions (e.g. trajectories) are inherently independent of the robot kinematics and provides better transferability to other robots since the correspondence problem is solved by the robots' native forward and inverse kinematics models.

Surprisingly there are relatively fewer works that formulate the action space of a learning task in terms of the low-level Cartesian space of a robot. Some tasks that have been demonstrated and transferred to robot in the task space include pancake-flipping [94], block stacking [76], pick and place [176], peg-in-hole [102], and pouring water [145]. Martín-Martín et al. [121] showed that a control policy learned in the end-effector space transfer to different robots better than policies in any other space since the representation and policy are independent of the robot embodiment. They also demonstrated that the policies learned in the

task-space have higher task success rate and sample efficiency for contact-rich manipulation tasks compared to policies learned in the joint-space. Learning policies in the task-space tend to require fewer training samples and uses smaller state space if the task objectives can be represented in the task-space.

Although policies in the task-space are easier to learn and more transferable in most cases, the advantages of learning in the task-space can vanish if the reward-functions are complex, non-linear and non-separable [43]. Furthermore, generation of smooth trajectories are theoretically easier in the generalised coordinates of a system (joint-space in case of manipulators), as ultimately control commands are sent in the lower configuration space of the joint motors. However, research in task-space trajectory planning and generation have advanced enough that the difference is almost negligible [91, 59]. Similarly, planning and execution of trajectories are easier and more meaningful in the joint space if there are obstacles in the workspace that are to be avoided by parts of the robot other than the end-effector. Therefore, the choice of the space for control and learning in manipulation depends mostly on the objective of the task and trade-off that the user is willing to make.

Recently deep learning has been used to tackle the problem of changing-contact manipulation by using end-to-end strategies where all the intermediate problems such as contact dynamics, controller formulation, interaction dynamics, etc. are bypassed by training a unified 'black-box' model which learns a mapping from observations to control commands. Generally, this requires a large amount of training time with a simulated and/or physical robot. In-hand reorientation of a cube using a 24-DoF robotic hand was demonstrated in [12] where the system was entirely trained in simulation and transferred to a real robot. However, in addition to using a complicated vision and tracking system, the training data comprised of 3 years worth of experience which they compressed to 1.5 hours of training time using high-end computing power and resources. In [133], a similar 24-DoF hand was used to rotate two free-rolling spheres in the palm using 2 - 4 hours of purely real-world training data. In [6], a physics engine is augmented using an object-based neural network for performing a multi-object pushing task. Although using the physics engine helped reduce the training time, the task still required collecting over 1000 push trials from the physical robot to fine-tune the model. Similarly, a (relatively) shallow network architecture which fuses the inputs from three sensors (RGB camera, force-torque sensor, robot end-effector states) is presented in [110], where the model learns controllers for a peg-insertion task. The network is trained end-to-end using self-supervision with physical robot trials that took about 5 hours of wall-clock time. The authors demonstrated the effectiveness of the trained system in generalising to different instances of peg-insertion (different hole shapes) with reasonable accuracy.

In this thesis, we chose to define our learning and control problem in the task-space of the robot for the following reasons: (i) learning in the task-space is more sample-efficient [43] and is therefore more appropriate for online learning; (ii) learning interaction dynamics is more meaningful in the task-space as the robot is interacting with the environment using the end-effector and feels the interactions directly at its end-effector (iii) learning interaction dynamics in the task-space allows for using the learned model as a feed-forward model in the task-space controller (see Chapter 4); (iv) task-space impedance control is more meaningful for manipulation tasks as the Cartesian representation allows the robot to be selectively compliant along certain directions; and (v) contact anticipation and handling is directly suitable in the task-space of the robot as contact locations are independent of the robot configuration.

In the next chapters, we describe our framework and experimentally evaluate its performance in the context of a robot (simulated and/or real) performing a manipulation task in varying dynamical environments.

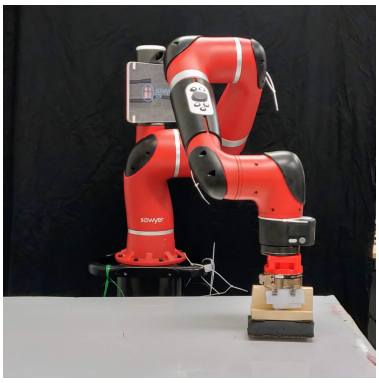
Chapter 4

Adaptive Variable Impedance Control for Continuous Interactions

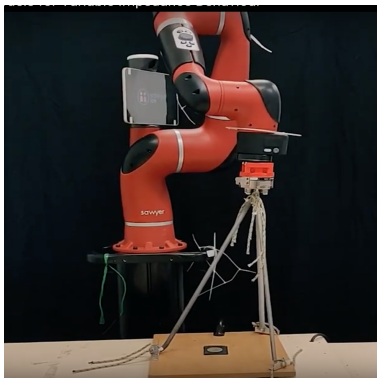
Manipulation tasks such as polishing, deburring and grinding involve complex contact interactions with the environment. These tasks require following desired trajectories while maintaining target contact forces. Resistive forces on the robot influence the controller performance due to surface properties such as friction. Therefore, control in such tasks requires the knowledge of the dynamics of the robot and environment under interaction.

Industrial robots working in closed cages typically use arbitrary high stiffness since they work in a structured and perfectly repeatable environment (i.e. specially designed environments) where all changes are predictable or modelled. However such pre-programming become cumbersome as robots are applied in unstructured environments and/or have to collaborate with humans. In principle, a motion planner can design and execute a plan for an entire interaction task accurately, if the complete interaction model of the robot and environment is available. However, it is impractical to assume that such models of the environment will be available for dynamic tasks. Errors in modelling leads to planning error and subsequently causes deviation from the desired trajectory, mainly due to incorrect estimation of contact forces. Such inconsistencies and environment unpredictabilities make the generated plans unusable without a proper adaptive controller.

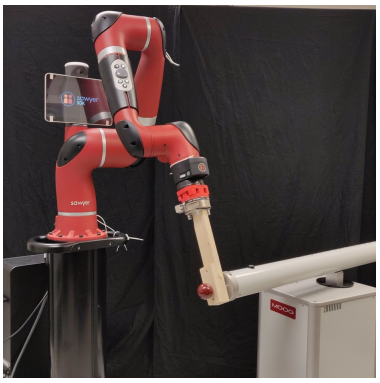
It is safe to assume that the interaction dynamics when a robot is in a continuous contact mode is smooth and continuous. For instance, the forces experienced by a robot polishing an even surface will be relatively smooth as long as the robot's motion is smooth. For a robot pulling springs, the forces it feels are a continuously changing effect depending on its position in the workspace (Fig. 4.1b). Similarly, if a robot is moving through a viscous liquid, the forces it feels will be a continuous function of its velocity. The dynamics will also be smooth even if the viscosity of the liquid was to change smoothly; this phenomenon



(a) Surface-polishing task where the robot experiences continuous resistive forces at the end-effector due to surface friction as it slides along the table.



(b) The end-effector is attached to three springs of different stiffness ('multi-spring' environment) resulting in continuously changing forces as it moves.



(c) The robot is attached to a Moog HapticMaster which emulates an environment whose viscosity increases as it moves ('porridge' environment).

Fig. 4.1 Example tasks with continuously changing interaction dynamics.

can be observed when heating certain non-Newtonian fluids, such as porridge which gets thicker as it gets cooked (Fig. 4.1c). Performing contact manipulations therefore demands reasoning about the forces and torques that are measured during task execution. Measuring contact wrenches is a straightforward way to describe environment interactions in the state space of a Cartesian controller. Usually, force measurement is achieved using a force-torque sensor mounted between the wrist and the end-effector. Prediction of environment dynamics (interaction forces) in such scenarios would clearly result in better control. It is argued that humans can handle such unforeseen and varying dynamics by adapting stiffness [22] and by building forward models [129] for the task.

This chapter presents a control framework inspired by these insights. The proposed Adaptive Variable Impedance Control (AVIC) framework incrementally learns a forward model of the interaction dynamics in the task space and uses it to adapt the parameters of a variable impedance controller on-the-fly such that the robot is stiff only if it has not modelled the interaction forces accurately. It forms an adaptive control strategy that can follow a provided task-space task plan accurately while navigating through continuously changing environment dynamics using lower controller stiffness (expending lower energy).

The chapter begins by describing the formulation of the AVIC framework in Section 4.1. A primitive formulation of this framework was published in the International Conference on Humanoid Robots (Humanoids, October 2019) as joint first-author with Dr. Michael Mathew [124]. The key differences from the original framework are briefed in Section 4.2. The performance of the proposed framework in different continuously varying dynamic environments and the need for incremental models in such environments is evaluated in

Section 4.3. In Section 4.4, the proposed controller is experimentally compared to a few baseline adaptive control strategies from literature. The chapter is concluded in Section 4.5 by discussing the advantages and limitations of the proposed control framework.

4.1 AVIC framework formulation

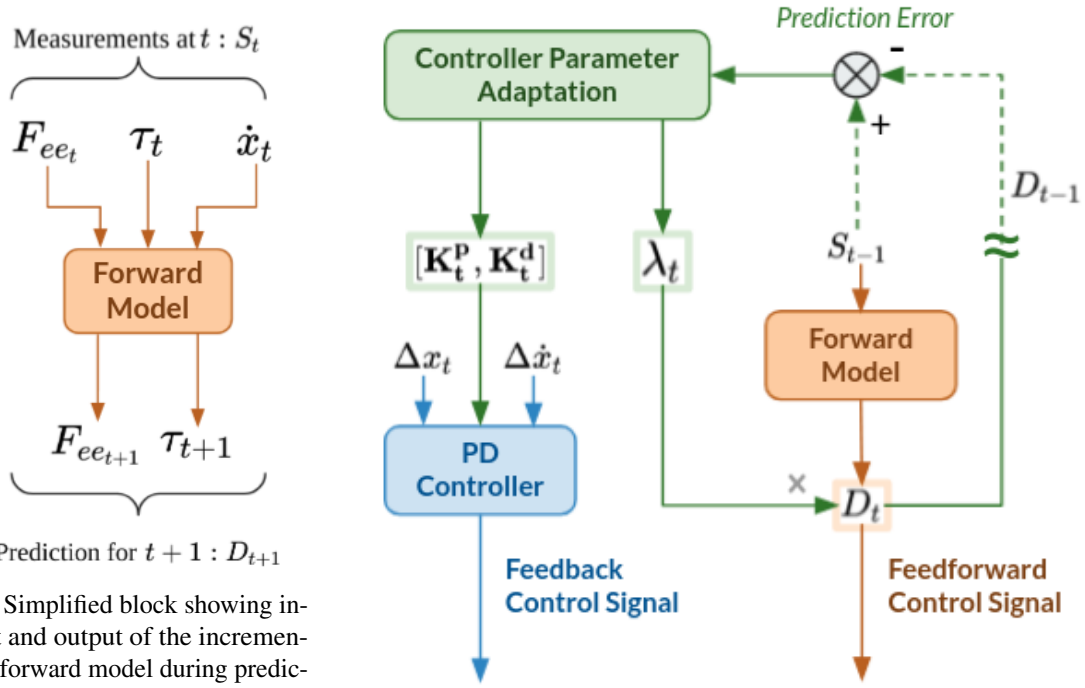
Existing work on robot manipulation use control strategies which learn or compute stiffness values based on large labelled training datasets or require comprehensive knowledge of the domain dynamics, impose unrealistic assumptions or hardware requirements, or use state representations that make it computationally expensive to estimate the stiffness parameters online. On the other hand, research in human motor control indicates that when performing a new manipulation task, humans initially use higher arm stiffness to accurately follow the desired trajectory in the presence of unforeseen external disturbances. This behaviour ensures that they can complete the task without being hindered by unknown effects. With sufficient experience, humans can then perform the same task accurately with much lower stiffness. Such a behaviour is achieved by building internal models of the task dynamics to predict the configurations of the object and the hand, and the forces, during task execution [48, 82, 78, 168]. Our Adaptive Variable Impedance Control (AVIC) framework draws inspiration from these insights to make a significant departure from existing adaptive control methods.

4.1.1 Basic structure

With AVIC, the human designer either provides a desired operational-space motion pattern (i.e., profile) for the task or teaches a trajectory via kinesthetic demonstration. The robot learns a forward model of the task online from the demonstration or during an execution of the task trying to follow the motion profile. The learned model predicts the forces and torques that can be expected in the next state and determines a feed-forward term in the control command. The forward model is continuously revised and the prediction error guides the selection of the gain (i.e., impedance) parameters of the controller that provides the feedback term in the control command. The hybrid force-motion controller separately controls force along the direction(s) in which compliance is desired. A simplified block diagram of the AVIC framework is shown in Fig. 4.2.

The AVIC framework builds on the task-space hybrid force-motion impedance controller form derived in Section 2.1:

$$\boldsymbol{u}_t = \mathbf{K}_t^{\mathbf{p}} \Delta \boldsymbol{x}_t + \mathbf{K}_t^{\mathbf{d}} \Delta \dot{\boldsymbol{x}}_t + \boldsymbol{u}_t^{\mathbf{ff}} + \boldsymbol{u}_t^{\mathbf{fc}} + \boldsymbol{H} \tag{4.1}$$



(a) Simplified block showing input and output of the incremental forward model during prediction (Section 4.1.2)

(b) Block diagram of framework

Fig. 4.2 Simplified block diagram of AVIC framework. Note that the internal robot dynamics compensation and force control components of the controller have been omitted from the figure for clarity.

Note that the control law is the same as Eq. (2.9), but with the subscript t denoting instantaneous values of the parameters at time t. The force control term $\boldsymbol{u}^{\mathbf{fc}}$ is used when a target force has to be applied along some direction.

The next sections will explain how the feed-forward model is learned and updated incrementally, and how it is used for determining the feed-forward term $\boldsymbol{u}_t^{\mathbf{ff}}$ and feedback impedance parameters $\mathbf{K}_t^{\mathbf{p}}$ and $\mathbf{K}_t^{\mathbf{d}}$ on-the-fly.

4.1.2 Learning forward model of interactions in task-space

The forward model of a contact mode in the AVIC framework is learned as the robot attempts to follow the desired trajectory in the task space for the given task. To avoid explicit dependence of the forward model on time, a Gaussian Mixture Model (GMM) is fit over points of the form $[\boldsymbol{S}_{t-1}, \boldsymbol{D}_t]$, where $\boldsymbol{S}_t$ can be any combination of features that uniquely represent the robot's state for the task, and $\boldsymbol{D}_t$ is interaction effects felt by the robot at its end-effector at time t. $\boldsymbol{S}_t$ can contain information about end-effector pose ($\boldsymbol{x}_t$), velocity ($\dot{\boldsymbol{x}}_t$), forces ($\boldsymbol{F}_t^{ee}$), etc. while $\boldsymbol{D}_t$ would be measurable interaction effects such as end-effector forces ($\boldsymbol{F}_t^{ee}$) and torques ($\boldsymbol{\tau}_t^{ee}$).

In our framework, we aim to predict the end-effector forces and torques from previous measurements of end-effector velocities and wrenches. However, instead of their 3D vector representations we use the *magnitudes* of force, torque, and end-effector velocity (linear and angular separately) for learning and prediction. Since the magnitudes of frictional forces and torques are independent of the direction of motion (for objects having consistent friction properties) and depend only on the relative speed of motion of the objects, such simplified representation is sufficient to learn and predict the end-effector forces and torques along the direction of motion. This reduced representation of forces and torques makes the learning process simpler, more computationally efficient, and also *independent of the direction of motion*. The learned model always predicts the forces and torques along (or against) the direction of motion. Since the end-effector's direction of motion is always known, the components of force and torques along the axes of motion can be recovered when needed. The final state space where the forward model is learned can therefore be represented as $X_t = [\boldsymbol{S}_{t-1}, \boldsymbol{D}_t]$ where

$$\boldsymbol{S}_{t-1} = \left[\|\dot{\boldsymbol{x}}^{lin}_{t-1}\|, \|\dot{\boldsymbol{x}}^{rot}_{t-1}\|, \|\boldsymbol{F}^{ee}_{t-1}\|, \|\boldsymbol{\tau}^{ee}_{t-1}\| \right] \tag{4.2}$$

$$\boldsymbol{D}_t = [\|\boldsymbol{F}^{ee}_{t}\|, \|\boldsymbol{\tau}^{ee}_{t}\|] \tag{4.3}$$

To incrementally update the model using new measurements during task execution, we used an online variant of GMM called the Incremental GMM (IGMM) [180, 42, 4]. IGMM can update model parameters and incorporate additional components to the mixture model using a measure of closeness and frequency which are defined by the user using hyperparameters. For more information regarding the incremental algorithm used, readers are referred to [42]. IGMM internally uses a variant of the Expectation-Maximisation (EM) algorithm to fit the model and maximise the following likelihood function:

$$L(\theta) = p(\mathbf{X}|\theta) = \prod_{n=1}^{T} p(X_n|\theta) = \prod_{n=1}^{T} \left[\sum_{j=1}^{M} p(X_n|j)p(j) \right] \tag{4.4}$$

where $\theta = (\mu_j, \sigma_j, p_j)$ for $j = 1...M$ are the parameters of the M components of the GMM. $\mathbf{X} = (X_1, ..., X_T)$ represents the points to be fit, with $X_t = [\boldsymbol{S}_{t-1}, \boldsymbol{D}_t]$. Each point contains information about the *previous* end-effector state, along with the *current* wrench.

Once trained, the forward model provides a function:

$$f_{\text{fm}} : \boldsymbol{S}_t \mapsto \boldsymbol{D}_{t+1} \tag{4.5}$$

that predicts D_{t+1} at the next time step as a function of the current (measured) value of $\boldsymbol{S}_t$, using Gaussian Mixture Regression (GMR) [188]. Recall that in our work, the primary sensor is the force-torque sensor at the end effector, and that the motion pattern, forward model, and control law are defined in the Cartesian/task space instead of the joint-space of the manipulator.

Note that although we described our forward model using Eq. (4.5), this formulation still follows the structure of standard forward models described in Section 2.2: the forward model f_{fm} estimates the next state $\boldsymbol{s}_{t+1}$ given the current state $\boldsymbol{s}_t$ and the current action $\boldsymbol{a}_t$. In our case, this would translate as $\boldsymbol{s}_t = [\|\boldsymbol{F}_t^{ee}\|, \|\boldsymbol{\tau}_t^{ee}\|]$ and $\boldsymbol{a}_t = \left[\|\dot{\boldsymbol{x}}_{t-1}^{lin}\|, \|\dot{\boldsymbol{x}}_{t-1}^{rot}\|\right]$. The forward model can then equivalently be represented as:

$$\boldsymbol{s}_{t+1} = f_{\text{fm}}(\boldsymbol{s}_t, \boldsymbol{a}_t) \tag{4.6}$$

There are a few options to set the initial values of the parameters of a dynamics model. In our implementation, the robot sets the initial values based on sensor measurements collected in a short time interval at the beginning of the task. This is then modified continuously if the measurements do not fit the model as the task progresses.

4.1.3 Using dynamics model for variable impedance control

Many manipulation tasks can be accomplished using a very high stiffness ($\mathbf{K}^{\mathbf{p}}_{max}$), but this expends energy. On the other hand, if the robot has to perform a task in free-space, accurate trajectory following can be achieved with a much lower stiffness ($\mathbf{K}^{\mathbf{p}}_{free}$). If the robot has a good forward model that can predict and compensate for the interactions in a task, the predictions should be able to provide a feed-forward term to cancel out the external forces, effectively reducing the motion to that in free-space. Similar to human behaviour with a familiar manipulation task, the feedback gains can then be closer to $\mathbf{K}^{\mathbf{p}}_{free}$. We use a similar strategy to modify the parameters of the task-space controller (Eq. (4.1)) in our framework online.

Since the final control output $\boldsymbol{u}$ is a wrench, the feed-forward term $\boldsymbol{u}^{\mathbf{ff}}$ must also be a wrench. The interaction dynamics model described in Section 4.1.2 provides exactly this by predicting the end-effector forces and torques that can be expected in the next instant. Therefore, we use a weighted form of the predicted wrenches to provide the appropriate feed-forward term, while using the prediction accuracy to guide the choice of the feedback

gains of the variable impedance controller:

$$\boldsymbol{u}_t^{\mathbf{ff}} = \lambda_{t-1} \boldsymbol{W}_t^{pred} \tag{4.7}$$

$$\mathbf{K}_t^{\mathbf{p}} = \mathbf{K}_{free}^{\mathbf{p}} + (1 - \lambda_{t-1})(\mathbf{K}_{max}^{\mathbf{p}} - \mathbf{K}_{free}^{\mathbf{p}}) \tag{4.8}$$

where $\boldsymbol{W}_t^{pred}$ is the predicted wrench provided by the forward model; and $\lambda \in [0,1]$ is a measure of accuracy of the learned model. The measure is also used for guiding the choice of the instantaneous stiffness matrix in the feedback control law (Eq. (4.8)). λ can be any function that maps the accuracy of the model prediction to a value between 0 and 1 such as the logistic function:

$$\lambda_t = 1 - \frac{1}{1 + e^{-r(\varepsilon_t - \varepsilon_0)}} \tag{4.9}$$

Here, the weighting factor is based on the error in the prediction (ε_t), while the growth rate r and sigmoid midpoint ε_0 are hyperparameters tuned experimentally. In our implementation, ε_0 is a simple Euclidean error between the predicted and measured values of the force-torque values in the last step, although other sensible measures such as model likelihood can be used.

The damping term in the control law is updated using the known constraint of the damping factor for a critically-damped system [100]:

$$\mathbf{K}_t^{\mathbf{d}} = \sqrt{\frac{\mathbf{K}_t^{\mathbf{p}}}{4}} \tag{4.10}$$

This formulation of the control law (defined in Eqs. (4.1), (4.7), (4.8) and (4.10)) provides online variation of feedback and feed-forward terms of the controller based on the instantaneous accuracy of the dynamics model. When the dynamics model predicts accurately, λ is closer to 1 and the feed-forward term can cancel out the external disturbances in the task. This allows the robot to complete the motion task while using lower stiffness (closer to $\mathbf{K}_{free}^{\mathbf{p}}$) as if it was moving in free space. Similarly when the model accuracy is low, the weightage of the feed-forward term is reduced and the controller will rely on the feedback component of the controller as it follows the trajectory more precisely using higher feedback gains. This high-stiffness state also allows the robot to improve its dynamics model online due to more reliable measurements from the sensors, which in turn allows the robot to use lower impedance. This strategy ensures that the tracking accuracy of the robot is not compromised even when the model is not accurate, while allowing the robot to use lower impedance gains when the model is reliable.

4.2 Key modifications made to previous framework

Forward model learning and state space

In our published work [124], we had explored two representations for $\boldsymbol{S}_t$ for predicting end-effector forces. The first is of the form $\boldsymbol{S}_t = [\dot{\boldsymbol{x}}_t^{lin}, \boldsymbol{F}_t^{ee}]$, where $\dot{\boldsymbol{x}}_t^{lin}$ corresponds to the translational (linear) component of the end-effector velocity. The second, motivated by studies of animal motor control [168], is of the form $\boldsymbol{S}_t = [\dot{\boldsymbol{x}}_t^{lin}, \boldsymbol{F}_t^{ee}, \boldsymbol{u}_t]$ where, $\boldsymbol{u}_t$ is the task space control command. This is similar to the 'efferent copy' mechanism in animal motor control, where a copy of control signals is fed back to the central nervous system and are used by internal forward models to predict the effects of actions. In that work, we showed that the efferent signals produced no significant improvement in performance in comparison with the forward model that does not use the efferent copy. The reason for this could be that the forward model is able to obtain enough information for force prediction from the current end-effector velocity and forces, making the information encoded in the efferent copy redundant. This observed performance, and the fact that adding dimensions to the state-space makes the learning more computationally demanding, led us to *not* use the efferent copy in the subsequent experiments. The final model learned was a mixture model learned in a 9-dimensional space (6 for $\boldsymbol{S}_{t-1}$ and 3 for $\boldsymbol{D}_t$) without the efferent copy[1].

In contrast to the feature space used in the above work, the modified framework also included rotational velocities ($\dot{\boldsymbol{x}}^{rot}$) as well as torques ($\boldsymbol{\tau}^{ee}$) in the state space. This helps in compensating for torsional effects due to contacts in manipulation tasks, such as when the robot is pivoting an object about its edge or sliding an object while maintaining an edge contact with another surface (see Section 5.3.3). For instance, the torsional strain felt at the wrist of the robot is different when sliding a block on its face as opposed to when sliding it by its leading edge (see Fig. 5.2). To capture these effects and to compensate for such torsional strains, the forward model was improved to include the predictions about the end-effector torques as well (this concept is explained with figures later in Section 5.2.1). This also required incorporating rotational velocities in the input space of the GMM, since changes in rotational velocities (acceleration) contribute to end-effector torques when the robot is in continuous contact.

Furthermore, the modified framework used *magnitudes* of the velocities and wrenches as dimensions for the state space as explained in Section 4.1.2. This improved the learning efficiency by reducing the size of the state space even when learning to predict more infor-

[1]These are the maximum number of dimensions learned for any task. In practice, only the dimensions along which motion control is applied are used for learning forward model since the force along the other directions is due to the force controller itself and would ideally be the commanded target force.

mation by incorporating end-effector rotational velocities and torques. The final state space is 6-dimensional with 4 for $\boldsymbol{S}_{t-1}$ and 2 for $\dot{\boldsymbol{D}}_t$. This reduces the computational complexity since the Expectation Maximisation (EM) algorithm includes inversion of covariance matrix which has a complexity that is cubic in the number of dimensions $O(d^3)$.

Controller formulation

The controller formulation in the published work defined the overall controller as a PD controller with a feed-forward term, ignoring the robot's dynamics. The dynamics formulation and the control law was modified to provide a complete impedance control law which also considers the robot dynamics (Eq. (4.1)).

The previous formulation also did not include the weighting term λ for the feed-forward term (in Eq. (4.7)). This meant that the feed-forward term was wrongly used even when the model's predictions were not reliable.

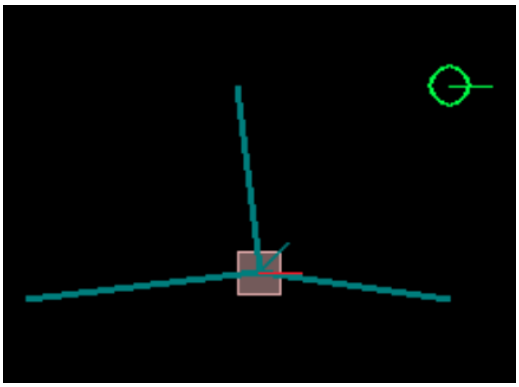
Furthermore, the feed-forward term in the new version was improved to incorporate torques to provide the balanced control equation in the task-space. The original formulation ignored the feed-forward torques, thereby having several limitations such as not being able to differentiate between the type of contacts and not compensating for torsional effects at the end-effector. The advantage of incorporating torques in the state space for differentiating different types of contacts is explained in Section 5.2.1.

4.3 Experimental evaluation

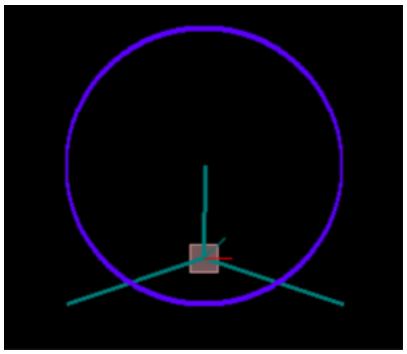
We established the advantages of the variable impedance control formulation in [124] using a 7 DoF robot in different continuous contact tasks (Fig. 4.1). When used in conjunction with the continuously-revised forward models, the gain parameters are adapted automatically to account for changes in dynamics within the mode, minimising the corresponding prediction errors. In addition, the hybrid force-motion controller provides compliance in the direction of force control while following the desired motion pattern. As a result, the manipulator is able to adapt its gain parameters automatically if, for instance, the surface is tilted or raised during task execution. We evaluated the dynamics learner and variable impedance controller in three different continuous-contact settings: (i) a board polishing task which requires the robot to move along a trajectory against friction while applying a normal force (Fig. 4.1a), (ii) a spring pulling task where the robot has to move in an environment against the unmodelled forces of three different springs (Fig. 4.1b), and (iii) a porridge stirring task where the robot has to move along a trajectory while the viscosity of the environment changes continuously (Fig. 4.1c). In all cases, it was shown that the framework was able to provide

tracking accuracy similar to the baseline high-stiffness strategy while using significantly lower impedance parameters.

This section includes further experiments for (a) testing the convergence of the variable impedance controller experimentally in the presence of continuously changing interaction dynamics; (b) analysing the advantages of having an incrementally updating model for learning and predicting continuously changing dynamics; and (c) comparing the performance of the control framework with respect to three other adaptive control strategies from literature, in terms of their tracking accuracy in and ability to handle continuously changing dynamics.



(a) Robot tracking a fixed target. The green circle represents the target pose for the robot.



(b) Robot attempting to follow a trajectory. The blue circle indicates the target trajectory for the robot to follow.

Fig. 4.3 Screenshots of the custom-built simulated 'Multi-spring environment'. The red block denotes the end-effector of the robot which can move in the workspace (black region) using task-space control. The 3 green lines indicate that 3 springs are attached to the end-effector of the robot which exerts dynamically changing forces due to the combination of their individual extensions on the end-effector as it moves.

These experiments are conducted in a simulated custom-built 2D environment using Box2D physics engine [27, 143]. The robot's end-effector is represented using a block mass on which different dynamically changing forces act as it follows a predefined trajectory (see example Fig. 4.3). The main measures of performance that were used are accuracy in trajectory tracking, controller delay, and prediction accuracy of forward model wherever applicable. It has to be noted that the units used (SI) in the figures and results are purely for convenience of the reader, and are not to be taken as an equivalent of a real physical system[2].

The incremental forward model is built during task execution in most cases (unless pre-training is specifically mentioned). In the simulated 2D setup, the state space of the forward

[2]In the simulator, applying 10 units of force on a free body (point) of mass 5 units, for instance, will produce an acceleration of 2 units (Newton's second law: $F = ma$). This can be interpreted as $[F = 10N, m = 5kg, a = 2m/s^2]$, or $[F = 10mN(\text{milliNewton}), m = 5g, a = 2mm/s^2]$, or any other balanced combination of appropriate units.

model does not include the rotational components (i.e., rotational velocity and end-effector torques) since the end-effector is assumed to maintain a fixed orientation for simplicity. In experiments with the real physical robot in subsequent experiments in the next chapters, the original state space described in Section 4.1.2 is used.

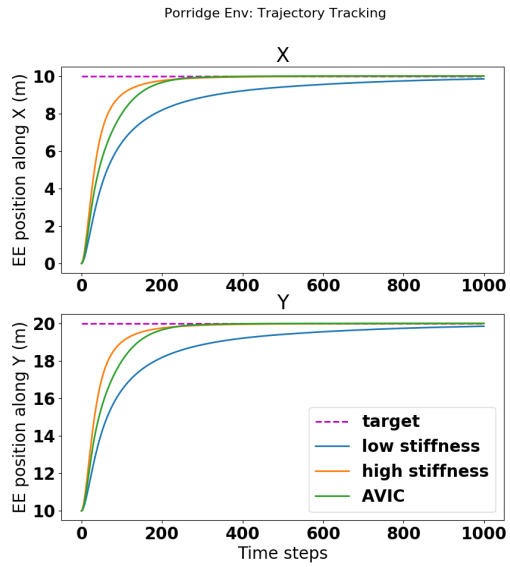
4.3.1 Evaluating controller convergence

This experiment was designed to test the hypothesis that the presented adaptive variable impedance control framework ensures that the end-effector of the robot can converge to a fixed (non-transient) goal smoothly in the presence of any continuously changing dynamics. The experiments are designed with the notion that any continuous interaction can be modelled as a spring-damper system.

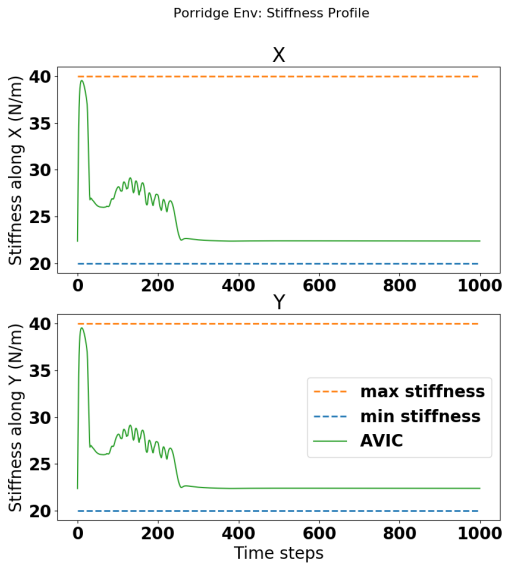
The system was subjected to two different environments in which the robot had to follow a pre-defined trajectory while experiencing continuously changing forces at the end-effector. The first environment emulates the dynamics of a non-Newtonian fluid such as a porridge, where the viscosity of the environment increases continuously till it reaches a limit (similar to the physical robot experiment shown in Fig. 4.1c). The second environment has 3 spring forces acting on the end-effector of the robot along with a constant damping (similar to Fig. 4.1b). In both cases, the robot was unaware of the environment beforehand and was only provided with a starting pose and a single fixed target pose. As baselines, we compared the position tracking when using a fixed high-stiffness controller and a fixed low-stiffness controller.

In the first experiment, the robot had to move from a starting point and move to the target while the viscosity of the environment continuously increases from 0.1 Ns/m^2 to 100 Ns/m^2 as time progresses. Fig. 4.4a shows the trajectory tracking along the X and Y axis when the robot uses a constant high stiffness (40 N/m), constant low stiffness (20 N/m), and when using the proposed control framework. It can be seen that the system is able to converge quickly to the target when using the proposed adaptive variable impedance controller (AVIC) comparable to the high-stiffness strategy, whereas the low-stiffness approach fails to converge to the goal in the allowed time. The AVIC has the added advantage that it requires much lower stiffness to achieve the same goal within a similar time as the high stiffness strategy, especially towards the end of the task (see Fig. 4.4b). The stiffness values can also be used to understand the accuracy the forward model as they are related proportionally (Eq. (4.8)).

In the second task, the robot end-effector was attached to 3 springs of different stiffnesses (10 N/m, 15 N/m, 12 N/m) and having bases fixed at different positions in the world (see Fig. 4.3a). The robot experiences completely different forces as it pulls them along different paths. The target position for the controller was the same as in the previous experiment.



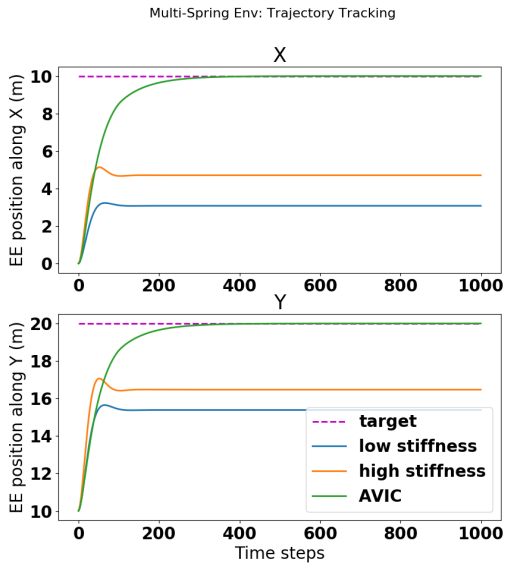
(a) Position tracking along X (top) and Y (bottom) directions when using a high stiffness controller (orange), low stiffness controller (blue), and the proposed AVIC framework (green).



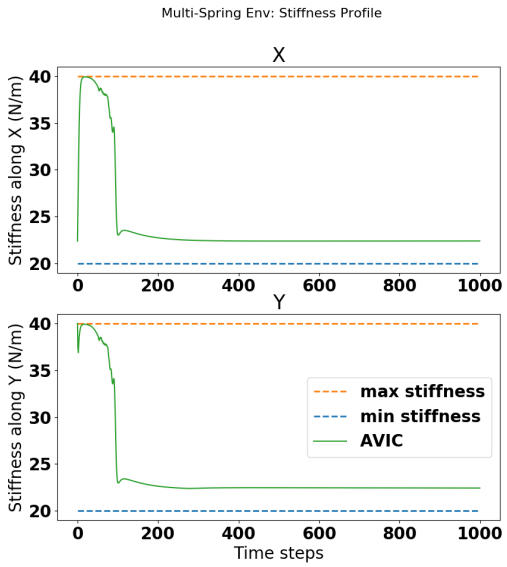
(b) Controller stiffness along X (top) and Y (bottom) directions when using a high stiffness controller (orange), low stiffness controller (blue), and the proposed AVIC framework (green).

Fig. 4.4 Trajectory tracking and stiffness profile adaptation for point-to-point motion in 'Porridge Environment'. The trajectory tracking performance when using the proposed control framework AVIC (green) is comparable to the high stiffness control strategy (orange) but requires much lower stiffness to achieve the objective.

The effectiveness of having an adaptive feed-forward model is more evident in this task (see Fig. 4.5a) because the combined stiffnesses of the 3 springs produce large resistive force as the robot pulls on them, such that even the high-stiffness controller is unable to bring the robot to the target position (orange curve). This is because the the pre-defined maximum value of allowed stiffness is not high enough to counter the external spring forces. It has to be understood that by increasing the value of the constant high stiffness, a tuned PD controller would have been able to counter these external forces. The purpose of this experiment was to demonstrate that AVIC uses much lower stiffness for tracking the trajectory than what would be needed otherwise in such extreme environments. Due the incremental learning nature of the dynamics model in AVIC, the system quickly learns to model the forces acting on the robot and uses its predictions to compensate for the additional external disturbances as the robot pulls the springs. This allows the robot to reach the target and hold that position using much lower stiffness than the high-stiffness strategy (Fig. 4.5b).



(a) Position tracking along X (top) and Y (bottom) directions when using a high stiffness controller (orange), low stiffness controller (blue), and the proposed AVIC framework (green).



(b) Controller stiffness along X (top) and Y (bottom) directions when using a high stiffness controller (orange), low stiffness controller (blue), and the proposed AVIC framework (green).

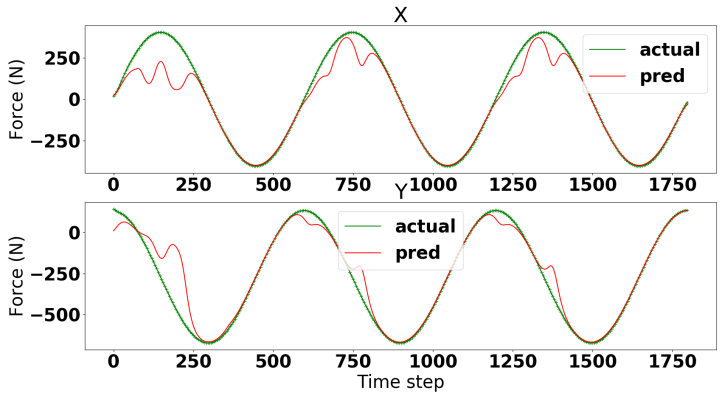
Fig. 4.5 Trajectory tracking and stiffness profile adaptation for point-to-point motion in 'Multi-Spring Environment'. The trajectory tracking performance when using the proposed control framework AVIC (green) is better than other baseline constant stiffness strategies due to feed-forward model.

4.3.2 Importance of incremental models in continuously changing dynamics

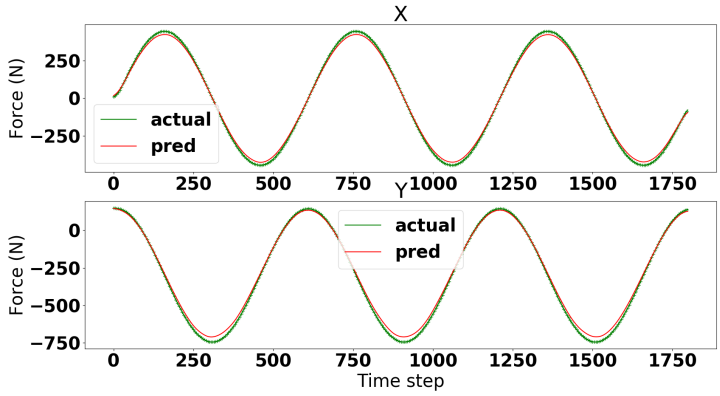
An important feature of the framework that allows the robot to use lower stiffness is its ability to adapt the forward model online and incrementally. This provides the robot with the capability to update its knowledge as the environment smoothly changes and hence provide more reliable predictions and thereby use lower impedance to proceed with the task.

To demonstrate the importance of having an incrementally adapting model when the environment is different to the previously experienced dynamics, we compare our incremental GMM forward model with a fixed GMM model that is created using the same training data. Note that the experiment was conducted in the same simulated setup with perfect noiseless measurements for objective comparison of the strategies.

For the first experiment, the forward model was trained in the 'multi-spring' environment mentioned previously (with individual spring constants 10 N/m, 15 N/m, and 12 N/m). For collecting the training data for the forward models, the robot was made to move around (with a fixed constant high stiffness controller) in a circular trajectory in this environment (Fig. 4.3b). The measurements from this experiment was used to build a fixed GMM model



(a) Predictions from a pre-trained fixed GMM forward model. *Top:* Predictions and measurements along X; *Bottom:* Predictions and measurements along Y.

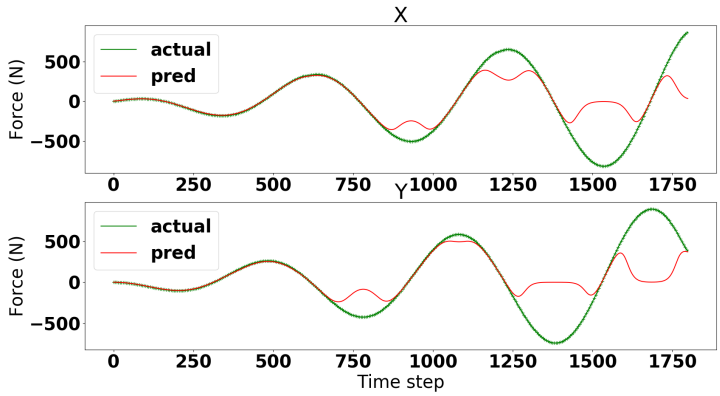


(b) Predictions from the incremental GMM forward model. *Top:* Predictions and measurements along X; *Bottom:* Predictions and measurements along Y.

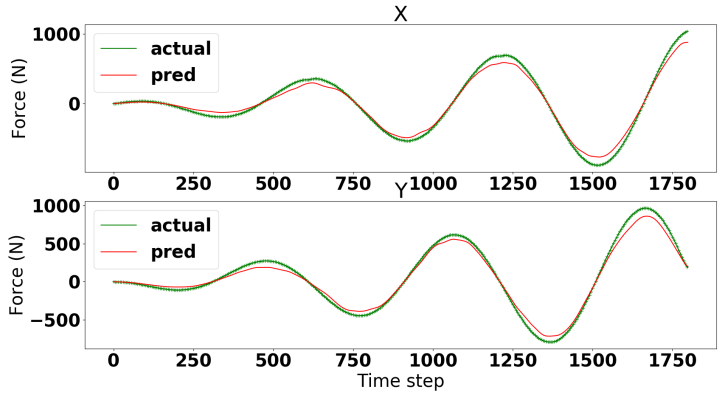
Fig. 4.6 Comparison of predictions from fixed and incremental GMM models in 'multi-spring' environment. *Red:* Predicted values; *Green:* Actual measurements.

for one-step prediction of end-effector forces from previous end-effector velocity and force. The fixed GMM consisted of 15 Gaussian components (chosen from the gradient plot of Bayesian Information Criteria scores (see Appendix A.1)) and the EM algorithm converged in ≈ 300 iterations. This model was then used as forward model for the same trajectory but when the springs were made stiffer by increasing their spring constants to 12 N/m, 20 N/m, and 18 N/m respectively. The robot moved in this new environment with the same constant high-stiffness controller and the predictions from the forward model was compared with the measured values. The results of the prediction provided by this forward model is shown in Fig. 4.6a. It is evident that the model predictions are only reliable when the experienced forces are within some bounds of the training data. This is expected and is because the GMM is able to provide predictions by conditioning only if the measurements are within its

components in the state-space. It is unable to extrapolate its predictions due to the properties of Gaussian conditioning whereby the predictions tend towards the mean of the mixture as the query point moves beyond the model in the state-space. On the other hand, when using the incremental variant of the GMM on the new environment, the model is able to adapt quickly to incorporate the new batches of measurements and provide reliable predictions in continuously changing environments (Fig. 4.6b).



(a) Predictions from a pre-trained fixed GMM forward model. *Top:* Predictions and measurements along X; *Bottom:* Predictions and measurements along Y.

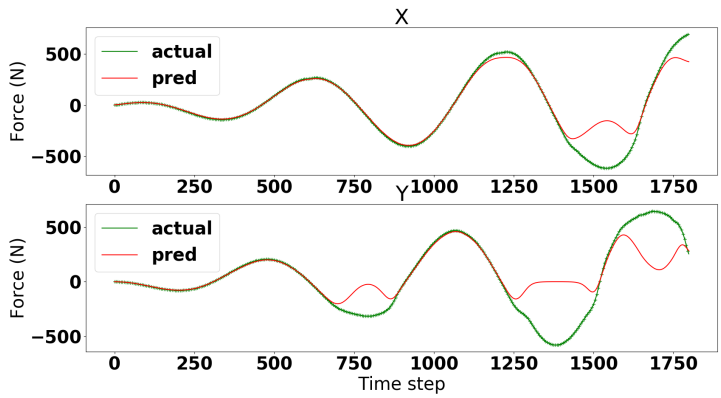


(b) Predictions from the incremental GMM forward model. *Top:* Predictions and measurements along X; *Bottom:* Predictions and measurements along Y.

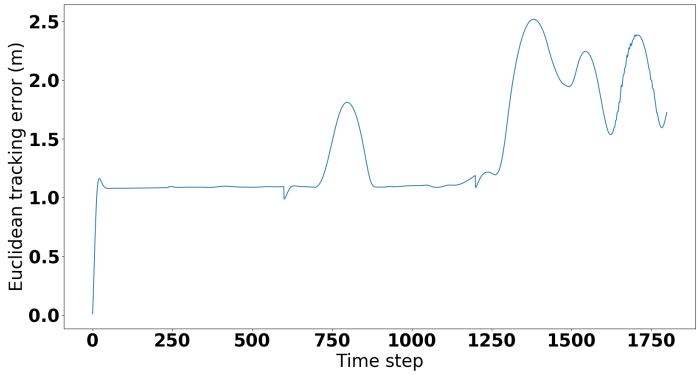
Fig. 4.7 Comparison of predictions from fixed and incremental GMM models in 'porridge' environment. *Red:* Predicted values; *Green:* Actual measurements.

The need for an adaptive model is easier to observe in the case of the porridge environment where the model has to be continuously adapted since the viscosity of the environment keeps increasing and is therefore difficult to model using pre-training unless the exact pattern of change is known beforehand. For this experiment, the robot was made to move along the same circular motion but in an environment whose viscosity increases continuously from

$0\ Ns/m^2$ to $80\ Ns/m^2$ in steps of $0.1\ Ns/m^2$ every simulation step (the total circular trajectory had 600 steps). The GMM had 21 components and the EM algorithm converged in ≈ 370 iterations. The trained GMM was then used in an environment whose viscosity increased from $0\ Ns/m^2$ to $200\ Ns/m^2$ in steps of $0.1\ Ns/m^2$. The predictions from this model in the new environment is shown in Fig. 4.7a. As expected, the model is able to predict the forces accurately when the input forces have been modelled, but as in the case of the previous experiment, the model fails to predict the feed-forward forces when the measured forces are beyond the training data. It also has to be mentioned that the accuracy of this model in the other regions is because of the constant high-stiffness controller that is being used. When this model is used to provide the variable impedance behaviour (using the relation in Eq. (4.8)), this causes unreliable trajectory tracking as well as worse predictions (Fig. 4.8).



(a) Predictions from the pre-trained fixed GMM forward model. *Top:* Predictions and measurements along X; *Bottom:* Predictions and measurements along Y.



(b) Trajectory tracking error (Euclidean) in metres. The tracking becomes unstable as the model prediction error increases and the controller gains vary.

Fig. 4.8 Controller based on a fixed forward model is not reliable when it used for providing variable impedance control. The predictions from the forward model are less accurate, and the trajectory tracking accuracy drop.

These results clearly indicate the need for using an incremental model as a feed-forward model for predicting interaction dynamics. Additionally, it has to be mentioned that pre-training a fixed model requires training data that covers a considerable region of the state space. This requires training time as well as memory depending on the complexity of the model. Although the fixed GMM models took less than 30 seconds for training in each of the above case, realistic training sets would be much larger and would have higher-dimensional data points. Incremental learning becomes all the more important when the robot experiences a previously unseen environment and it has to quickly build a new model from scratch (see Chapter 5).

4.4 Comparing AVIC with other adaptive control methods from literature

This section aims to compare the performance of the proposed control framework with three other adaptive control strategies from literature in a continuous dynamics setting. The objective is to demonstrate that the online nature of the forward model along with the adaptive variable impedance strategy presented in this chapter provides better trajectory tracking performance without any pre-requisite knowledge about the task or the environment.

Adaptive controllers are able to adjust their parameters according to the manipulated systems whose parameters change constantly, or for systems whose parameters are not known beforehand. As discussed in Section 3.2, adaptive control strategies fall broadly into three categories:

1. Model reference adaptive control (MRAC): An indirect control strategy where the control law is adapted online to mimic a user-defined reference model which has the same order as the original system.

2. Self-tuning controllers: Controllers which adapt the parameters of the control law (or the system model parameters used in the control law) online based on the real-time system feedback.

3. Gain-scheduling: Controllers which have a time-indexed sequence of controller parameters to be used during task execution. The sequence of parameters are either learned from pre-collected data from the same system, and/or iteratively improved by repeating the task multiple times.

This section compares the performance of the proposed AVIC framework with one strategy from each of these categories. The implementations of these methods are chosen

based on the ease of implementation, reproducibility (of presented results), and the tasks presented in the respective published work. In some cases, the implementations had to be modified to fit the experimental setup and for providing an objective comparison (such as changing joint-space representation to task-space form for the MRAC implementation).

4.4.1 Comparison with MRAC

As described previously, model reference adaptive control (MRAC) works by trying to adapt its control law based on a pre-defined desired reference model. The implementation of MRAC we used for comparison is based on the work presented by Tarokh [191]. The original formulation was described for an n-DoF manipulator in the joint-space and was only formulated theoretically. The formulation was adapted to the task-space (2D without orientation consideration) for our implementation. This can be seen as equivalent to a 2-DoF manipulator defined in the joint-space (since we don't consider orientation), where the 2D positional coordinates x and y can be considered equivalent to the joint positions q_1 and q_2.

In brief, the MRAC implementation uses as reference model a second-order spring-damper system for each dimension i which are parameterised using user-defined values for desired natural frequency ω_i and damping ratio ζ_i. The control law consists of a 'feedback' term, a 'cascade' term, and an 'auxiliary' feed-forward term, which loosely imitates a PID control behaviour. The 'feedback' component is dependent on the current position and its derivative, the 'cascade' term varies based on goal position and velocity, and the 'auxiliary' term depends on a modified tracking error. Each of these three terms in the control law have gain matrices which are adapted online using separate adaptation laws based on separate error measures consisting of tracking error, goal positions, etc. For the full description of the MRAC formulation, the readers are invited to refer [191].

The main complication with the implementation of the MRAC was the number of hyperparameters that are to be tuned for it to work well for any task. For an n-dimensional system, the user has to first define an $n \times 1$ vector each for ω and ζ for defining the reference model, along with an $n \times n$ positive semidefinite matrix which is critical for ensuring system stability. Furthermore, the update rule for the gains of each of the three control terms ('feedback', 'auxiliary', and 'cascade') have six $n \times n$ matrices (proportion, derivative, and integral update gains) each, which have to be defined by the user. Finally, the initial values for the gains (three $n \times n$ matrices) have to be provided as well. Therefore, even for a 2-dimensional system, this requires empirical tuning of about 40 hyperparameters (even assuming all matrices to be diagonal)[3].

[3]Many of these parameters are often set to zeros or ones, as suggested in the original formulation [191].

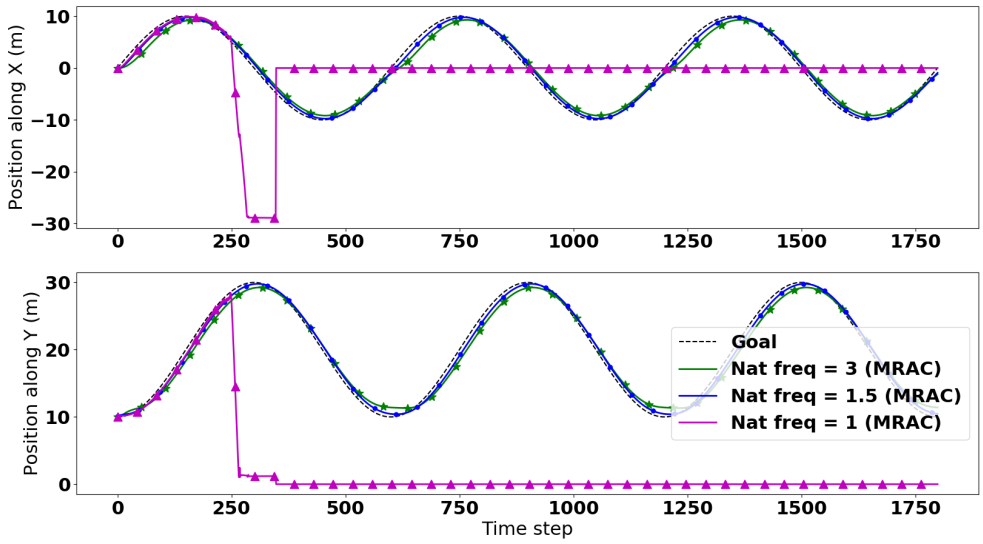


Fig. 4.9 Performance of MRAC is highly dependent on the design of the reference model. Here small variation in the value of natural frequency ω resulted in instability and failure in completing the task.

The performance of MRAC is best observed in the 'multi-spring' environment where the robot had to move along a circular trajectory three times. The tuning of the hyperparameters was found to be quite cumbersome, especially for a continuous changing dynamic setup such as the 'multi-spring' environment. This is clear in Fig. 4.9, where the trajectory tracking is significantly affected by the choice of the natural frequency parameter ω of the reference model (with all other hyperparameters fixed). It can be seen that by changing the value of ω from 1.5 to 1 has adversely affects the performance and leads to controller instability. Once the hyperparameters are correctly chosen, however, the tracking performance is comparable to that of our AVIC framework (Fig. 4.10).

Discussion

As expected, MRAC can perform well if the reference model and other hyperparameters are defined correctly by the designer. However, this can be quite challenging and wrong values can quickly lead to poor performance. On the other hand, AVIC has minimal hyperparameters to tune (mainly the parameters for IGMM) and can be tuned quite intuitively. With properly tuned hyperparameters, MRAC showed similar performance when the spring constants in the 'multi-spring' environment were changed. However, hyperparameter selection for the 'porridge' environment proved difficult and a comparable performance could not be achieved

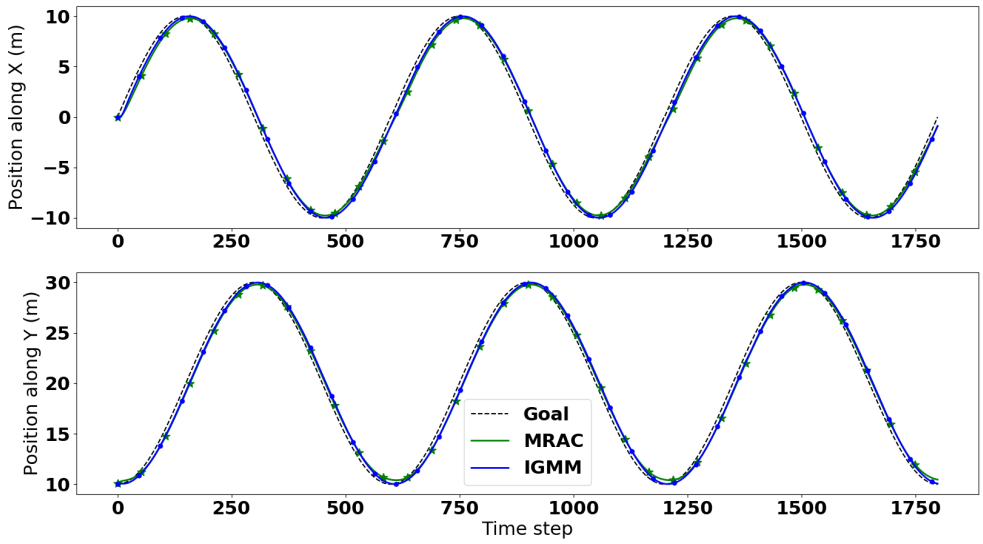


Fig. 4.10 Trajectory tracking comparison of MRAC and AVIC. With correctly tuned hyperparameters, performance of MRAC can be matched with that of our AVIC framework in the simulated 'multi-spring' environment. The performance of the correctly tuned MRAC (green) matches AVIC (blue) almost exactly.

with the MRAC. This is because it proved to be difficult to tune the hyperparameters of the reference model to match the varying viscosity of the 'porridge' environment.

4.4.2 Comparison with a self-tuning controller

As mentioned previously, due to the online adaptation nature of the controller parameters, our AVIC framework fits most closely in the category of self-tuning controllers. However, in contrast to typical self-tuning methods which do not make use of external force-torque measurements, AVIC has an incremental forward model that models end-effector wrenches online so as to counter the external disturbances more directly.

As an example of self-tuning controller for comparison, we chose a formulation called Active Inference Control (AIC) as presented by Pezzato et al. [151]. AIC is formulated using a Bayesian representation for the system state evolution such that the 'free energy' of the system is minimised. It iteratively updates its belief about the state of the system using the measurements it gets from system feedback at each instant. It is an adaptive control strategy which continuously updates for unmodelled environment dynamics. The control command is then selected which fulfils a prior expectation about a desired goal, by refining the internal belief about the system (implicit state estimation) using measurements. The

control parameters are therefore iteratively updated to account for unaccounted energy in the system so as to reach a single fixed target.

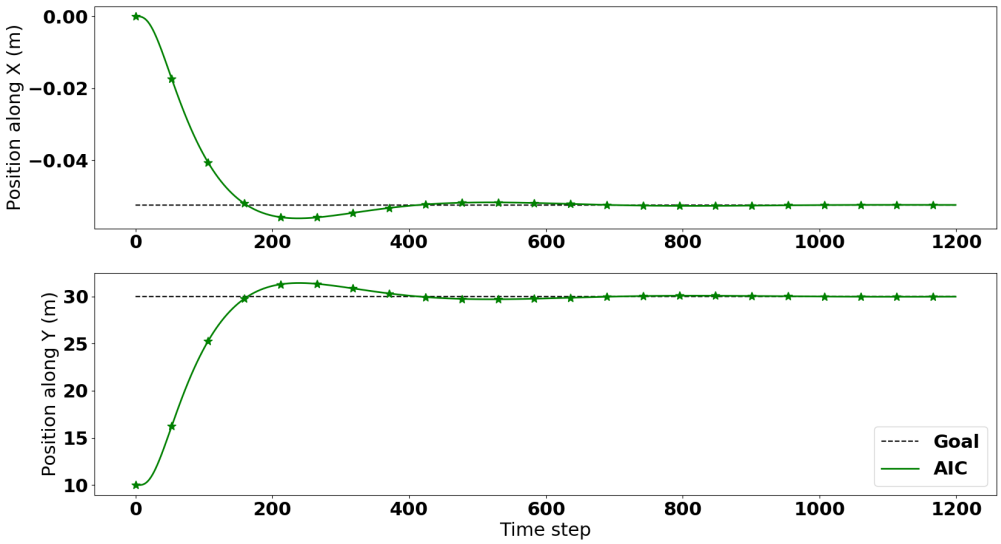
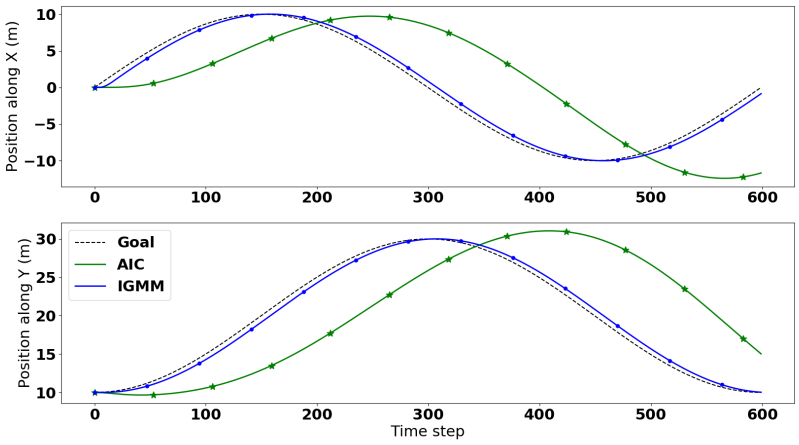
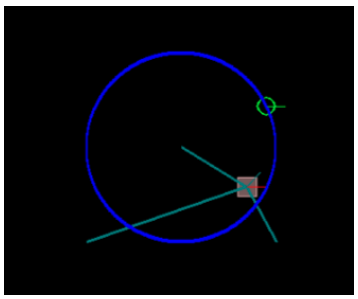


Fig. 4.11 Given time, AIC converges smoothly and stably to a fixed target.



(a) Trajectory tracking comparison between AIC and AVIC.



(b) A screenshot of the robot end-effector (red square) lagging behind the target (green circle) when using AIC.

Fig. 4.12 For trajectory tracking, AIC parameters fail to converge in time before target updates which results in delays.

Although it is difficult to prove convergence in a dynamically changing system, AIC achieves convergence in practice. Given time, AIC converges to the target smoothly without any appearance of instability for any type of dynamically changing environment (see

Fig. 4.11). However, this is not useful when the robot has to follow a trajectory in such an environment as the parameters of the controller in general will not converge in time before the target changes to the next point in the trajectory. Since the optimal control parameters are dependent on the target, this means the same parameters may not necessarily apply for the new target as the trajectory progresses. Therefore, the robot always lags behind the target when following a transient or changing goal trajectory (see Fig. 4.12).

Discussion

Self-tuning regulators are good adaptive control methods that work well when the set-point being tracking is fixed till the controller converges. However, for trajectory tracking, this behaviour produces delays in following the plan. This effect is pronounced in AIC as the control law does not have an intuitive form (such as a PID) with parameters that can directly influence delays and trajectory tracking performance. Instead the control law produces commands based on its belief of where the system is and where it should be by assuming that the target is fixed. The AVIC framework on the other hand, compensates for external disturbances by directly predicting and cancelling out the forces, and hence minimises lag in trajectory tracking.

4.4.3 Comparison with gain-scheduling adaptive control

Gain-scheduling can either be done using data collected previously to fit a profile model (supervised), or by iterative updating time-indexed control parameters online (RL-based). For comparison, we implemented an RL-based iterative update strategy called Biomimetic Adaptive Control (BAC) [204] where control parameters for each instant in a repeating trajectory are iteratively updated in each trial. This obviously means that it requires multiple trials of the task, or alternatively a periodically repeating trajectory. The algorithm then iteratively updates controller parameters at time t using their values from previous iteration at time t based on the tracking error at t.

The authors of BAC tested their controller in a 2D simulated environment (similar to the 2D block-world presented in this chapter) in three different environments: (i) presence of a constant force in one direction, (ii) the previous environment with an additional position-based divergent field (spring environment), and (iii) further adding a velocity-dependent divergent field (viscous environment with constant viscosity/damping) to the previous environment. The controller learns T-dimensional vectors for each of the controller parameters (stiffness, damping, and feed-forward terms) where T is the total time-steps in the trajectory. The trajectory is repeated multiple times till the values converge or when it reaches a maximum

number of trials. At timestep $t \in T$, the controller parameters at time t is updated from their previous values to minimise a cost function based on tracking error.

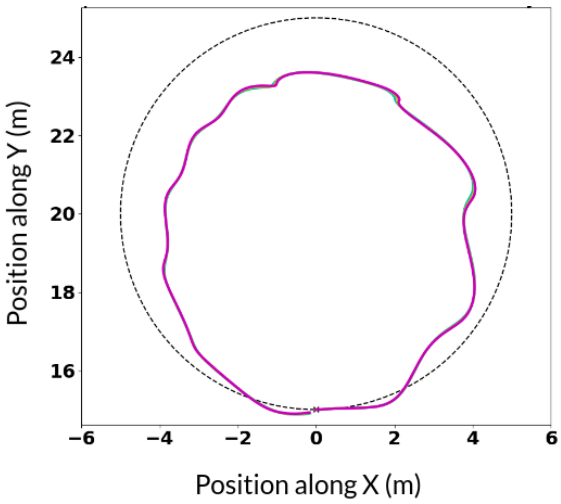


Fig. 4.13 Trajectory followed using BAC by robot in the 30th trial when bad initial parameters were provided. Dotted line indicates the target trajectory; the solid line is the actual path followed by the robot. The controller instability does not allow for further improvement of parameters.

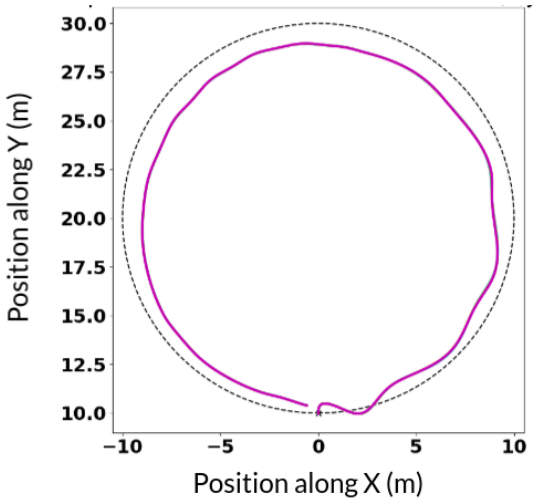


Fig. 4.14 Trajectory followed using BAC by robot in the 30th trial of the 'multi-spring' environment. Dotted line indicates the target trajectory; the solid line is the actual path followed by the robot. BAC parameters fail to converge in 30 trials for complex environment dynamics.

It was observed that good initialisation of the parameters is necessary to ensure that the learning converges. With bad parameter initialisation, the parameters never converge due to instabilities and tracking irregularities that accumulate as the trajectory progresses. Such accumulation of errors and temporal relation to previous control commands are not accounted for in the algorithm. Therefore, BAC is unable to recover from such failures as it updates parameters at each time instant based only on their previous values in the last iteration of the task and the current computed cost. One such effect of wrongly initialising the stiffness parameters is shown in Fig. 4.13. Here, the robot is subjected to a constant downward force, while a single spring is attached to it from the centre of the circular trajectory. Due to bad initialisation of parameters, the trajectory tracking is not improved even in the 30th iteration (trial) of the task because the controller has become unstable, which hinders further improvement.

As with most reinforcement-learning-based algorithms, BAC requires multiple trials of the task and depends on the environment to be the same as in the previous trials. With reasonably good parameter initialisation and enough trials (≈ 30), BAC is able to converge to and provide good tracking in the environments presented by the authors (see [204] for results). However, with more complex environments such as the 'multi-spring' environment,

the parameters of BAC has a more difficult time to converge to good set of parameter values within the specified number of maximum trials (30) – see Fig. 4.14.

Discussion

The main disadvantage of RL-based gain-scheduling strategies is that they usually require significant training data or repetitions on the real system, and still relies on the system being exactly the same during task execution. Furthermore, it is difficult to ensure smooth trajectory tracking when learning a time-indexed parameter profile since such strategies do not typically account for the temporal relation of the parameters within each trial. The value of a parameter at time t affects the choice of its value at time $t+1$ since the system dynamics is a function of the control command. Although, gain-scheduling methods that fits models on pre-collected data implicitly account for this by training on sequential data, RL strategies such as BAC which updates parameters based on their values from the previous trial of the task can be adversely affected by the lack of explicit temporal considerations. Furthermore, such methods rely on the duration of the task (total number of time steps in each trial) to be fixed across trials.

4.5 Summary and discussion

This chapter introduced the Adaptive Variable Impedance Control (AVIC) framework for performing manipulation tasks in continuously changing dynamical environments. The framework uses an incremental GMM to continuously learn to predict end-effector wrench during a task, which is then used to adapt the parameters of a task-space variable impedance controller online. The learning process and the control parameter adaptation were described. A previous base version of the framework had been published previously in collaboration with another researcher, and hence the results from the original publications were omitted. The framework presented in this chapter is based on this work, but has significant improvements and modifications which were also explicitly mentioned.

The framework was experimentally evaluated in a custom-built 2D simulated environment for performing objective noiseless comparisons. First, the performance of the framework in several challenging continuous dynamics was evaluated in terms of its convergence to a target. The experiments also proved that the framework allows the robot to use much lower stiffness to reach the goal smoothly compared to the baseline high-stiffness controller. It was also showed that having the feed-forward model allowed the system to counter extreme environments which could not be tracked even with a constant high-stiffness controller. Next, the necessity of having incrementally updating forward models was experimentally

demonstrated by comparing the online learning model with a fixed pre-trained Gaussian Mixture Model for the same data. It was made clear that incremental models are important for handling dynamically varying environments and for handling new environments.

The AVIC framework was then experimentally compared with methods from 3 classes of adaptive control which were implemented based on state-of-the-art formulations in literature. The advantages and limitations of these methods in the context of continuously changing environments were discussed.

The proposed AVIC framework performed better than the baseline methods in all the dynamically changing environments that were considered. It is able to quickly learn to predict end-effector wrenches and thereby rely on lower impedance parameters for providing accurate trajectory tracking. The method also has very few hyper-parameters that can be tuned easily depending on the task. Maintaining a consistent learning thread and a separate controller loop that makes use of these predictions are the only implementation challenges that had to be solved. The main issue was that it required a separate learning thread which independently and continuously learns to predict the end-effector wrenches. However, since the controller parameters are updated online based on the error in prediction, trajectory tracking is not sacrificed even when the learner is not reliable (due to lack of data points, noisy sensor measurements, etc.); it simply uses higher impedance parameters when the model is not predicting accurately, which results in precise tracking as well as helps in improving the forward model.

The experiments in this chapter were designed to test the effectiveness of the framework in a contained, objective, and controlled setting which is not affected by external factors such as sensor noise or delays in communication; hence a simple 2D simulated environment was selected. Results from real-robot experiments were not included in this chapter also because similar experiments were already presented in the collaborative publication mentioned previously. Furthermore, subsequent chapters in this thesis include other experiments done on a physical robot making use of the AVIC framework presented in this chapter.

The main limitation of the proposed AVIC framework is that it relies on the interaction dynamics to be continuous. It performs poorly when there are discrete changes in the environment. The next chapter describes this limitation further and proposes a hybrid learning framework for handling such discontinuities in dynamics.

Chapter 5

Hybrid Model Learning for Discretely Changing Interaction Dynamics

A manipulation task can be modelled as a hybrid system (see Section 2.3), wherein the dynamics of the system is continuous within each of a number of discrete dynamical modes [99]. The dynamics then become piece-wise continuous, where the system 'jumps' between these distinct modes depending on the current state. In manipulation tasks, these mode switches often correspond to making or breaking of contacts or discrete environment changes (sudden changes in environment forces, types of contacts etc.). The mode switches provide a modular structure to manipulation tasks where the mode transition can be considered to be sub-goals or triggers for identifying a need for changing the current model of the environment.

Unfortunately, having distinct modes also makes the interaction dynamics of manipulation tasks inherently discontinuous, which makes it difficult to guarantee smoothness when learning a policy; small changes in the state can have significant effect on the task if the system is near a transition. It is therefore important that the robot monitors its actions for unexpected and undesired mode transitions, and is able to update its knowledge about the environment modes online. Therefore, an ideal manipulation framework should be able to detect mode transitions and be able to deal with it without causing dynamical or performance instability.

Using this knowledge about the inherent discontinuity in the system dynamics in a manipulation task, it is more meaningful to have a framework that is at least aware that the dynamics of the system can be discrete. The most straightforward way to capture this while learning is to have separate models for the discrete modes. Each of these low-level models may correspond to one distinct manipulation mode (e.g. type of contact, environment forces, etc.), which can provide the required behaviour to navigate that mode successfully.

Each individual model is continuous and learns the environment in that mode, but a higher-level discrete mode-identification and switching model should be present that captures the high-level task mode information. Such model representation is known as hybrid model representation, where a continuous model is learned for a discrete sub-task [139] while a transition model learns to choose the appropriate low-level model to use for the observed state. This approach allows the robot to exploit the modularity of the distinct sub-tasks for the discrete high-level transition model [108]. Planning approaches for manipulation domains often explicitly take this multi-modal structure into account for planning. Many works on learning continuous models for individual skills or sub-tasks can be seen as learning part of a larger hybrid model. The model learned by the AVIC framework (Chapter 4) can be seen as one such low-level model that works for *one* manipulation mode (all tasks considered in Chapter 4 had continuous contact without discrete mode switches). The framework, due to its reliance on a single continuous model, will not be sufficient for handling discrete changes in dynamics (see Section 5.3.1).

Hybrid models have been used for performing various manipulation tasks [32, 98, 177]. Most works make use of the discrete 'jumps' between modes that arise due to making or breaking of contacts, which capture the changing dynamics and constraints caused by the changing contacts. The dynamics within a mode can be learned using standard continuous model approaches. The 'guard regions' (transition regions) can be modelled as explicit sets of states or using classifiers [99]. More flexible models, such as non-parametric and neural network models, may also be able to implicitly capture hybrid dynamics of manipulation tasks [47]. However, exploiting the modularity of the hybrid system becomes more difficult when using an implicit continuous model of the hybrid structure.

This chapter introduces our hybrid framework for learning and control of a robot manipulator performing tasks with piecewise-continuous dynamics. It is an expansion of our publication [175] to the 2020 journal, *Advances in Cognitive Systems*, with additional experiments and baseline comparisons. The chapter begins by formulating manipulation as a hybrid system in Section 5.1. The proposed hybrid framework and the feature representations used are then described in Section 5.2. The framework is experimentally evaluated in Section 5.3 with a 7-DoF robot performing different changing-contact tasks. The importance of having incremental learning for hybrid systems is demonstrated in Section 5.4 by using a fully offline long-term prediction model as the baseline. The chapter is concluded in Section 5.5 by discussing the advantages and limitations of the presented hybrid framework.

5.1 Manipulation as a hybrid dynamical system

Consider a robot manipulator (Fig. 5.1) that has to slide an object over a surface along a given motion pattern. The interaction dynamics of the robot performing these tasks are discontinuous when a contact is made or broken and continuous elsewhere. The dynamics also varies based on the type of contact (e.g., surface or edge contact), surface friction, applied force, and other factors.

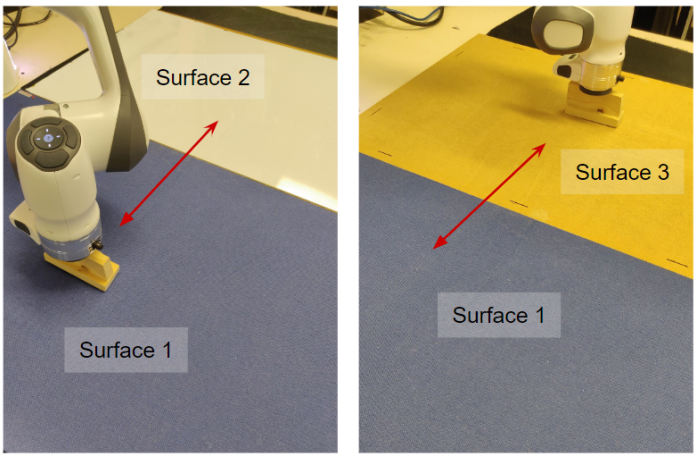


Fig. 5.1 Manipulator sliding an object in a pattern along three surfaces with different friction. The interaction dynamics switches discretely when the robot moves across the boundary between the surfaces.

As explained in Section 2.3, the state of a piecewise-continuous hybrid system can be described using the tuple $(m, \boldsymbol{s})$. In the context of manipulation, $m \in \mathcal{M}$ is a *contact mode* from a discrete set of modes $\mathcal{M}$, and $\boldsymbol{s} \in \mathcal{S}_m$ is a $d-$dimensional element describing the interaction dynamics in the continuous subspace $\mathcal{S}_m \subseteq \mathbb{R}^d$ associated with m. For our formulation, we assume that continuous subspaces do not intersect or overlap, i.e., $\mathcal{S}_m \cap \mathcal{S}_n = \emptyset \quad \forall \quad m \neq n$. The evolution of $\boldsymbol{s}$ within a mode is determined by some discrete-time continuous function $\boldsymbol{f}$, but the state transition is discrete and discontinuous at the boundaries between modes. In the guard regions $\mathcal{G}_{m,m'} \subseteq S_m$ between modes m and m', $\boldsymbol{s}$ is transported to $\boldsymbol{s}_{t+1} \in S_{m'}$ through a *reset function* $\mathcal{R}_{m,m'}(.)$. In a stochastic setup, the evolution of interaction dynamics in a changing-contact manipulation tasks is thus governed by the following state propagation law:

$$\boldsymbol{s}_{t+1} = \begin{cases} \mathcal{R}_{m_t,m_{t+1}}(\boldsymbol{s}_t) + w_t & \text{if } \boldsymbol{s}_t \in \mathcal{G}_{m_t,m_{t+1}} \\ \boldsymbol{f}(\boldsymbol{s}_t, \boldsymbol{a}_t, m_t) + w_t & \text{if } \boldsymbol{s}_t \in \mathcal{S}_{m_t} \end{cases} \tag{5.1}$$

where w_t is additive (Gaussian) process noise in sensor measurements.

In the context of the manipulation tasks considered in this thesis, the forces, torques, and velocities measured by the robot at its end-effector constitute the observable state $\boldsymbol{s}$ of the system that can describe the interaction dynamics and varies continuously within each contact mode. This formulation makes the reasonable assumption that properties such as friction are continuous across the surface of each object. Several factors such as the physical properties of the interacting objects, the control strategy used, etc. can determine the state transition $\boldsymbol{f}(\boldsymbol{s}, \boldsymbol{a}, m)$ governing the evolution of $\boldsymbol{s}$ in the mode m. When mode changes occur in the guard region $\mathcal{G}_{m,m'}$ between modes m and m', the dynamics corresponds to a new state in a mode m' where the state evolution is then guided by function $\boldsymbol{f}(\boldsymbol{s}, \boldsymbol{a}, m')$.

For changing-contact tasks, measurements within the guard region are typically pronounced and significantly different when compared with the readings within a dynamic mode. For instance, when a robot makes a new contact with an object in its environment, there will be significantly disruptive discontinuities in its end-effector force-torque measurements. The mode switches thus impose a hybrid structure on manipulation tasks; as we describe in Section 5.2.1, the transitions can be considered as triggers for changing the current dynamics model of the domain. The types of mode switches observed in manipulation tasks and strategies for detecting them are discussed later in Chapter 6 when describing our contact anticipation and handling module.

5.2 Hybrid model learning framework

In our framework, a mode-detection model learns to identify the interaction mode that the robot is in, where each mode comprises a: (i) forward (dynamics) model that predicts part of the observable state (end-effector forces and torques) as described in Section 4.1.2; (ii) an adaptive variable impedance control law as explained in Section 4.1; and (iii) relevance condition that (in)validates a mode based on the magnitude of changes in sensor measurements.

The performance of the framework is experimentally evaluated on a robot performing changing-contact tasks where multiple discrete modes are involved. The robot has to identify the mode and use appropriate dynamics models and controllers to navigate the identified mode. A simplistic mode-detection model could have just one mode or use an ad hoc strategy to assign particular models to corresponding parts of the motion pattern. Our formulation, however, seeks to account for changes in relevant factors in order to improve: (i) performance within each dynamic mode through online adaptation of the parameters of the forward model and the control law; and (ii) overall performance by automating the recognition of mode changes and the learning of models for previously unseen modes.

5.2.1 Mode identification and learning

Our approach for recognising known modes and identifying new modes in changing-contact tasks is based on the observation that any change in mode is accompanied by a sudden significant change in the sensor readings. In our framework, the robot responds to pronounced changes in force-torque measurements by briefly using a high-stiffness control strategy while quickly obtaining a batch of sensor data to confirm and respond to the transition. The robot learns a new dynamics model if a new mode is detected, and transitions to (and revises) an existing dynamics model of a known mode if transitioning to a known mode.

The management of modes is based on an online incremental clustering algorithm called Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) [210, 211]. This algorithm incrementally and dynamically clusters incoming data for given memory and time constraints, without having to examine all existing data points or clusters. We used the implementation of BIRCH in the Scikit-learn library [147]. Each cluster is considered to represent a mode in a feature space (more details below), with the clusters being updated using batches of the feature data. The fraction of the input feature vectors assigned to any existing cluster determines the confidence in the corresponding mode being the current mode. If the highest such confidence value is above a threshold, the dynamics model of that mode is used and revised until a mode change occurs. If the feature vectors are not sufficiently similar to an existing cluster, a new cluster (i.e., mode) and the corresponding dynamics model are constructed and revised (as described in Section 4.1.2) until a mode transition occurs.

The dynamics model of each mode is a separate incremental model that uniquely tries to capture the interaction dynamics of the current mode. This formulation builds and stores one forward model in memory for each identified mode, i.e. $f_{\text{fm}|m_i} \quad \forall \quad m_i \in \mathcal{M}$, where each $f_{\text{fm}|m_i}$ is a separate IGMM model providing the forward model for that mode according to Eq. (4.6). Once the current mode is identified by the mode-detection module, the corresponding forward model is used in AVIC for navigating the mode in the task.

Reduced state-space representation

The key factor influencing the reliability and generalisability of the mode-detection module is the choice of feature representation for distinguishing between the modes in the clustering algorithm. This representation is task-dependent but the objective is to identify one or more properties that vary substantially when change occurs while concisely and uniquely representing the modes. For instance, the torques experienced while performing a screwing task can be used to identify the tightness of the coupling and the appropriate control strategy for the task.

For the task of sliding an object over surfaces with different values of friction, the property that strongly influences the end-effector forces (F^{ee}) is the friction coefficient between the object and the surface. When two objects slide over each other at constant velocity, F^{ee} is proportional to the applied normal force (R) and the friction coefficient (μ) (assuming the relative orientation of their surface normals do not change); μ can then be estimated as:

$$\mu \propto \frac{\|F^{ee}\|}{R} \tag{5.2}$$

A concise feature representation for this task is thus $\frac{\|F_t^{ee}\|}{R_t}$, which has the effect of making mode classification independent of the magnitude of the applied force.

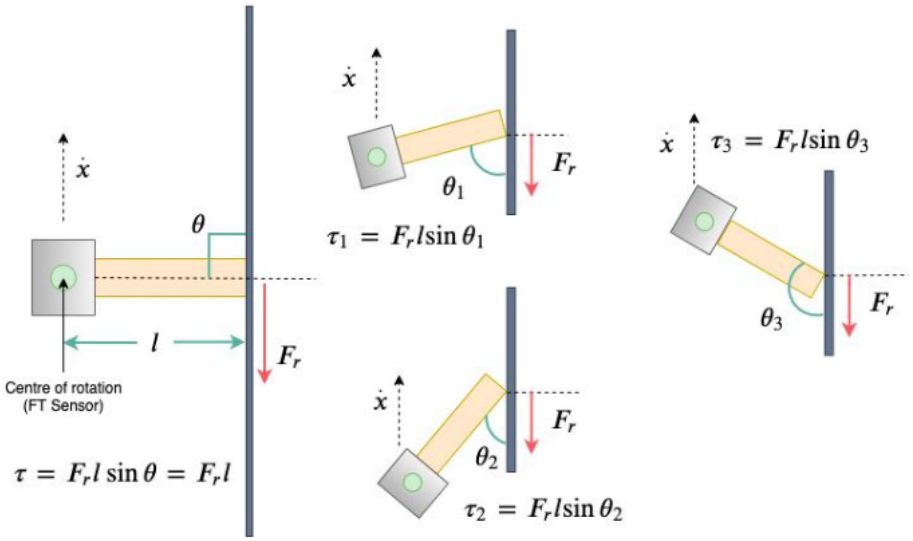
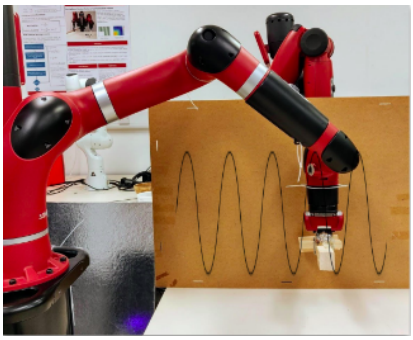


Fig. 5.2 The torque measured at the pivot (τ) varies for different relative orientation of the object (θ), unlike the force at the tip (F_r). The object is moving along $\dot{x}$ resulting in a frictional resistance F_r at the point of contact in the opposite direction.

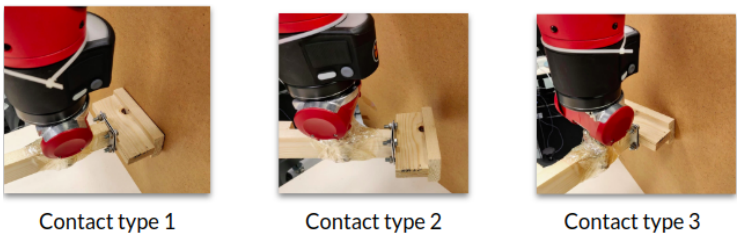
In a similar manner, for changes in the type of contact, end-effector orientation is a useful feature, but small changes in orientation may require different modes. A more reasonable feature is the magnitude of the end-effector torques that can be measured using the force-torque sensor in the wrist:

$$\tau = F_r l \sin\theta \tag{5.3}$$

where F_r is the force at the tip, l is the length of the pivot arm, and θ is the orientation between the surface normals. Fig. 5.2 indicates that for any object, τ is different for the different types of contacts.



(a) The black curve indicates the trajectory the robot has to follow while applying a predefined force on the surface.



(b) Different contact modes used for comparing force-torque values during sliding.

Fig. 5.3 Task setup used for comparing torques when the robot uses different types of contacts for sliding on a surface.

To verify that torque patterns are indeed more distinguishing of different contact modes than force patterns, a simple experiment was devised using the 7-DoF Rethink Robotics Sawyer robot. The robot was made to slide a rigid object against a fixed surface along a predefined trajectory (Fig. 5.3a). The experiment is repeated using different contact modes (Fig. 5.3b), and the end-effector force and torque patterns were compared. It can be seen from the results (Fig. 5.4) that the torque patterns (especially about Z axis) show more variation as compared to the forces, and are more distinguishable between contact types.

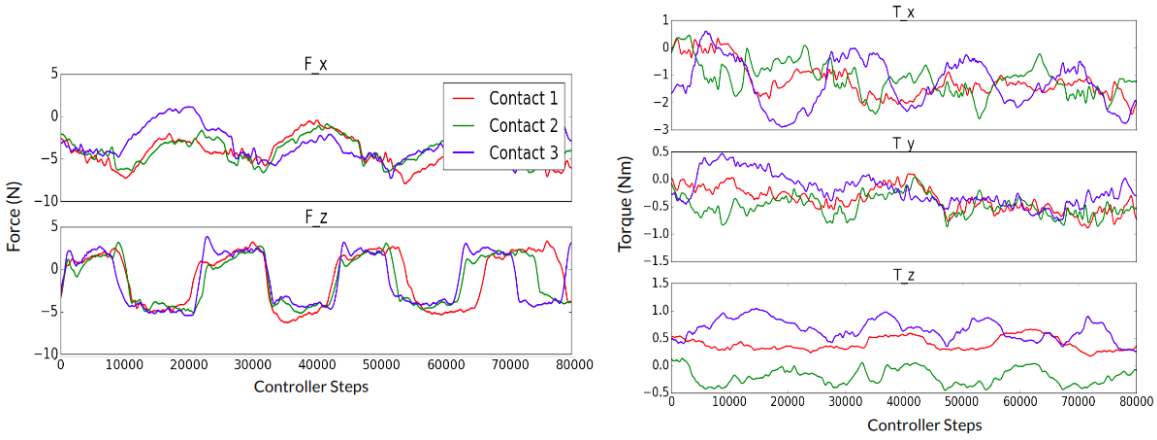


Fig. 5.4 Left: Force readings along X and Z axes (Y is force-controlled). **Right:** Torque measurement readings about the 3 axes. Torque patterns are more varied across the different contact modes compared to the forces.

An important note to be made here is that torque measurements allow distinguishing between contact types only because *the geometry of the object is assumed to be fixed across the trials.* As depicted in Fig. 5.2, the torque felt is a function not only of the orientation

between the contact surfaces (θ), but is also dependent on the length of the moment arm (l) which varies if the geometry of the object changes. In this work, we assume the object being manipulated by the robot is rigidly fixed and non-deformable, as are other objects that are in the workspace and involved in the task.

With the magnitude of the torques ($\|\tau\|$) as the feature representation, modes can be classified independent of the motion direction and object orientation. This representation would not work when the magnitude of the applied force differs. If we instead assume that the force measured at the wrist (F^{ee}) approximates the force at the tip of the object (F_r), Equations 5.2 and 5.3 imply that $\frac{\|\tau_t\|}{R^t}$ is invariant to the magnitude of the applied force for a fixed relative orientation between the objects in contact since:

$$\tau = \mu R l \sin(\theta)$$

Ideally, $\frac{\|\tau\|}{R}$ is constant for each mode (based on θ) provided object geometry (l) and friction (μ) do not change. Experimental analysis revealed that this parameter by itself is insufficient to distinguish between contacts when the applied normal force changes because the assumption about kinematic friction ($F_r = \mu R$) often does not hold in many real-world situations [15]. We thus use $[\frac{\|\tau\|}{R}, \frac{\|F^{ee}\|}{R}]$ as the feature representation for this task to provide more information regarding the current mode; it supports better generalisation over different normal forces while reliably distinguishing different changing contacts.

5.2.2 Framework algorithm

Algorithm 5.1 is an overview of the framework's control loop for a changing-contact manipulation task, e.g., sliding an object on a surface. It proceeds until a desired motion pattern is completed. Our control, adaptation, and learning methods are not used when the manipulator is moving in free space (lines 11-13); they are only used after there is contact with a surface (lines 2-10). The robot detects mode changes when there are substantial changes in the sensor measurements (line 3). The robot responds by setting a high stiffness (line 4), collecting sensor measurements, determining the transition to a new or existing mode (line 5), and creating new models if necessary (lines 6-8). In the absence of a mode transition (e.g., the detected change in sensor measurement was an anomaly), the robot continues with the current mode and dynamics model (line 10). A video demonstrating the operation of our framework and some results discussed in this chapter can be found online[1].

[1] https://youtu.be/m210rxIDZ7Q

Table 5.1: Control loop of framework

Input : Desired motion pattern as sequence of task space way-points, Control parameters: $\mathbf{K}^p_{free}, \mathbf{K}^p_{max}$; Dynamics models corresponding to modes $M = \{f_i : i \in [1,N]\}$; Current mode: $m = 0$.

while *Motion pattern not complete* **do**
  if *mode transition detected* **then**
    `// Set high stiffness`
    $\mathbf{K}^p_t \leftarrow \mathbf{K}^p_{max}$
    `// Detect (new/existing) mode`
    m = detect_classify_mode()
    `// Populate new model for new mode`
    if *new mode found* **then**
      $\mathcal{M} = \mathcal{M} \cup f_m$
    end
  end
  Update $f_{\text{fm}|m}$ online and use it for control in AVIC (Eqs. (4.1), (4.7), (4.8) and (4.10))
end

5.3 Experimental Evaluation

We used a 7-DoF Franka Emika Panda manipulator robot for our experiments in this section. The robot had to slide an object on a surface along an assigned motion pattern provided by an external planning module or encoded based on a single demonstration of the task by the human designer, e.g., human moves the manipulator along a desired path. We experimentally evaluated the following hypothesis:

H5.1: *Building separate dynamics models for the different modes results in better performance than using a single model that is revised continuously.*

H5.2: *Our hybrid framework provides reliable and efficient performance for changing-contact manipulation tasks.*

H5.3: *Our hybrid framework's performance is robust to changes in motion direction and changes in applied force.*

H5.1 explored the need for learning different dynamics models for different modes; **H5.2** and **H5.3** examined whether the framework can reliably and efficiently transition to the appropriate mode (and dynamics model) in the presence of changes in direction of motion and applied forces. We mainly used the *root mean squared error* (RMSE) in related

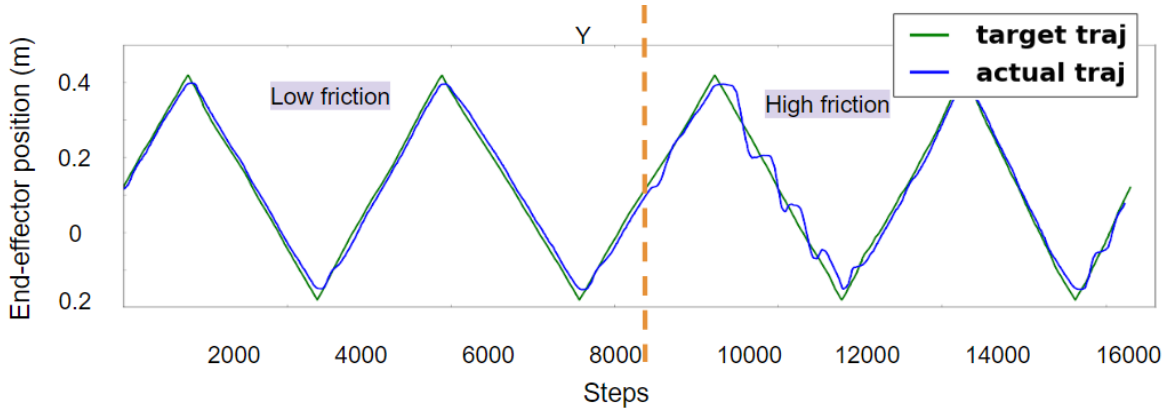
measurements (e.g., end-effector position, forces, and stiffness) as the key performance measure. Unless stated otherwise, each data point in the results below is the result of 10 repeated trials on the robot.

5.3.1 Need for hybrid models in manipulation tasks

The Adaptive Variable Impedance Control (AVIC) framework introduced in Chapter 4 is designed for handling smoothly changing interaction dynamics during manipulation, and cannot handle discontinuities. This is mainly because the feed-forward model that is incrementally built using IGMM is a continuous model and cannot model discontinuities reliably. Furthermore, the direct application of the weighted prediction from this model as a feed-forward term (Eqs. (4.1) and (4.7)) to the controller also means that the predicting wrong values during a discontinuity can have adverse effects on the robot and/or the objects involved.

Discontinuities in robot manipulation are typically caused due to the robot making or breaking contacts with an object in its workspace, or due to a physical or geometric discontinuity on the surface of the object that the robot is interacting with. For instance, if a robot is polishing a surface whose material friction changes discretely, the AVIC framework may not be able to handle the discontinuity in the forces safely. This was also observed when a physical had to slide from a surface of high friction to that of considerably low friction using the AVIC framework – see Fig. 5.1 (left). Here, the high friction of the first surface causes the framework to predict high forces even at the boundary between the surfaces, which results in the robot overshooting when it enters the low friction surface. This is also experimentally tested below.

We ran two experiments to evaluate hypothesis **H5.1**, i.e., the need for separate dynamics models for different modes of changing-contact tasks. The robot's task was to slide an object (rigidly fixed to the end-effector) over a flat surface, and the surface friction was changed to obtain two distinct surfaces. In the first experiment, the robot had a dynamics model for the first (rougher) surface but not for the second (smoother) surface. We expected the robot to overshoot the trajectory and experience a sudden increase in the joint torques when it transitioned from the first surface to the second. Experimental results matched these expectations. In 90% of the trials, the robot was unable to complete the task; it stopped before the trajectory was completed. The feed-forward values predicted by the model for the rougher surface were much higher than the actual values for the smoother surface. This discrepancy made the robot overshoot when it transitioned to the smoother surface; the joint torques reached safety limits and the robot stopped moving.



Purple: Tracked trajectory; **Green:** Desired trajectory

Fig. 5.5 Trajectory tracking for the surface polishing task when adapting a single model to a new surface. The vertical line indicates where the surface friction changes during the task.

Fig. 5.5 shows the position tracking for one of the few attempts where the robot was able to finish tracking when sliding from a surface of low friction to that of high friction. It can be seen that the tracking accuracy falls from the point of change. This is due to the forward model predicting the frictional resistance of the previous surface instead of the new surface and providing disruptive feed-forward terms to the control command. It was also observed that the model tries to explain the new observations by increasing the number of Gaussian kernels in its GMM model, which can slow down prediction and learning (see Appendix A.1).

In contrast, having separate models and switching to the second model when the predictions are inaccurate proved to give better results (mean prediction error: 1.97 N ($\pm$ 1.1) as opposed to 8.52 N ($\pm$ 6.2) in the previous case). The position tracking and stiffness adaptation for this strategy is shown in Fig. 5.7. These results indicated that a single incrementally-revised dynamics model was unable to handle pronounced, discrete mode changes.

In the second experiment, the robot repeated the same task starting with a single dynamics model but used a high stiffness controller to build a new model from scratch when a change to the second surface is detected; the robot does not initially have a forward model for the second surface. We considered both transitions (i.e., rougher to smoother surface and vice versa), and the robot performed better than using a single dynamics model that is incrementally revised; the task was completed successfully in all the trials. Fig. 5.6 shows the position tracking performance in one such trial. Operating with a high value of stiffness until a reliable forward model is constructed results in more energy being expended than when the dynamics models for the two modes are available. When the dynamics models

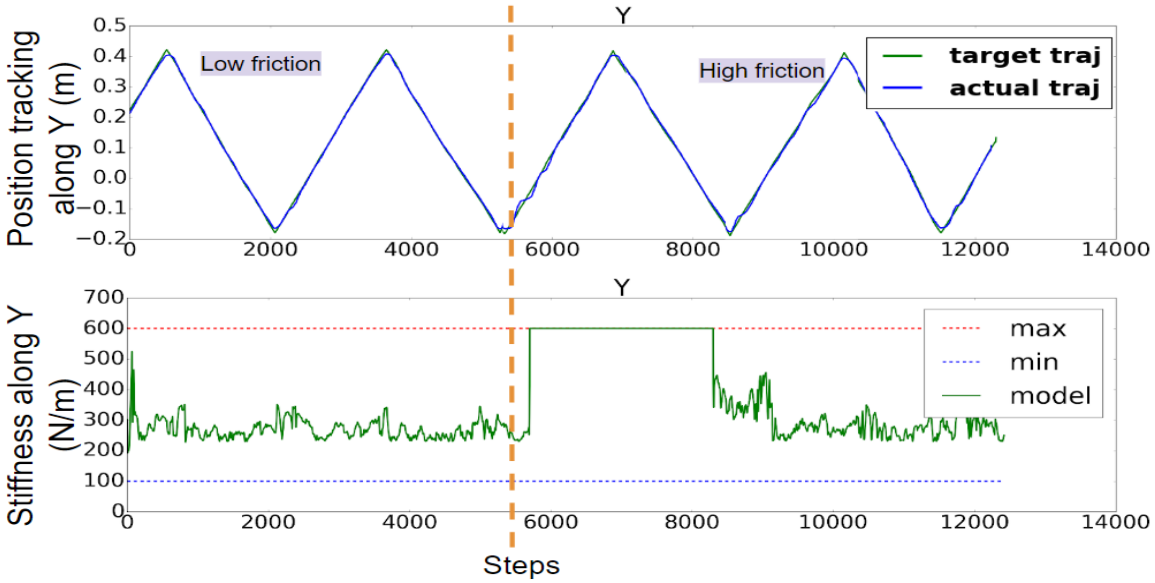


Fig. 5.6 Performance when a dynamics model is constructed from scratch for a new mode. **Top:** position tracking; **Bottom:** variation in controller stiffness. Robot spends considerable time under high stiffness when transition to new mode occurs (dashed vertical red line).

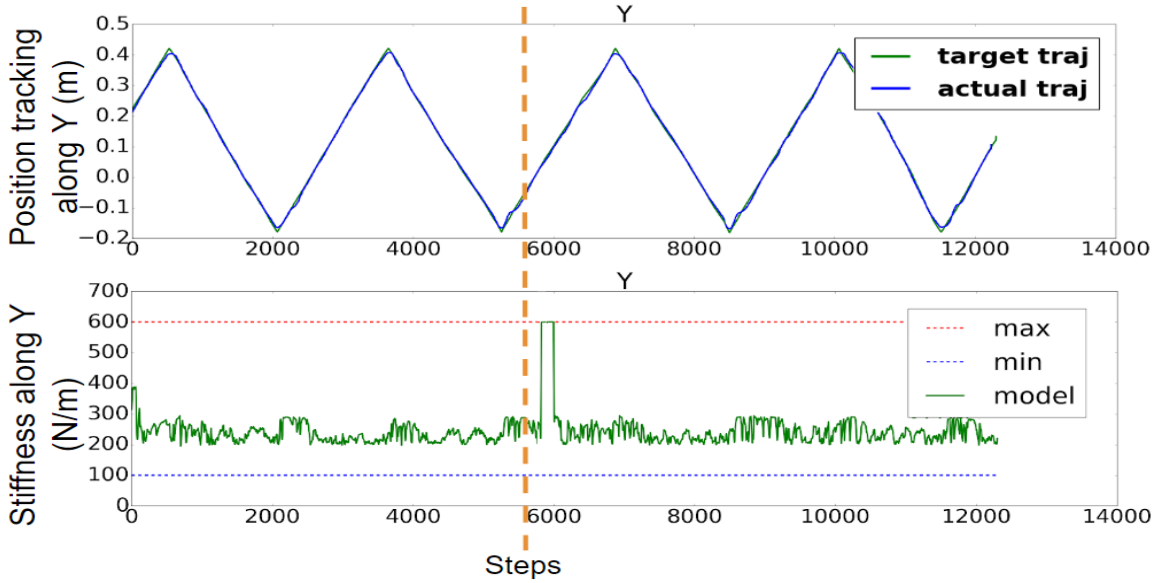


Fig. 5.7 Performance when separate dynamics models are available for two distinct modes. **Top:** position tracking; **Bottom:** variation in controller stiffness. The robot spends very little time under high stiffness when the transition between modes occurs (dashed vertical red line).

for the two surfaces are available, the robot is able to switch between them when needed, spending much less time under high-stiffness, as observed by comparing the stiffness plots in Fig. 5.6 and 5.7. The difference in performance is statistically significant, e.g., the robot had to spend ≈ 3.5 seconds in high stiffness to build a new model from scratch while it only

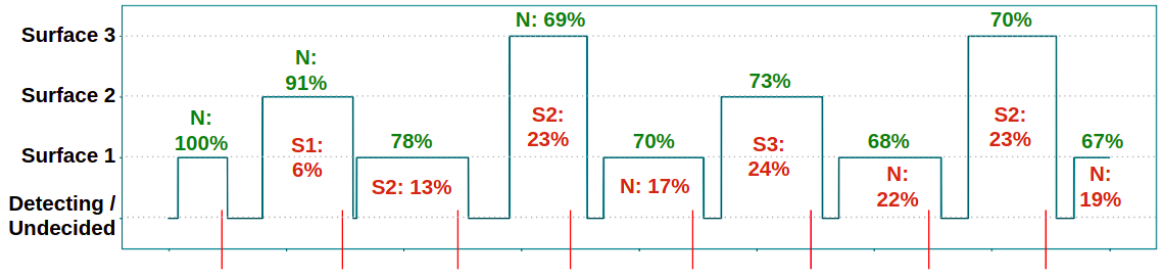


Fig. 5.8 Modes detected and their confidence values; red vertical lines on the x-axis indicate actual mode transitions. The number on top of a peak (in green) indicates the confidence with which the transition was identified; the number below a peak (in red) corresponds to the mode with the next highest confidence. 'N' indicates a transition to a new mode.

required ≈ 0.2 seconds to detect and switch to a previously learned model; RMSE for the position tracking plots in Fig. 5.6 and Fig. 5.7 are 0.017 and 0.015 respectively. We repeated these experiments for other combinations of surfaces (with different friction) and for motion patterns executed over more than two different surfaces with different surface friction. In each case, the robot was able to detect the new mode and incrementally revise the parameter values of the new mode, and to transition to using the existing models when appropriate. These results support hypothesis **H5.1**.

5.3.2 Detecting discretely changing surfaces

To evaluate hypotheses **H5.2** and **H5.3**, we first considered the changing surface task. The robot was asked to slide an object along a desired trajectory, and it experienced three previously unseen surfaces with different values of friction (Fig. 5.1). We expected the robot to identify a transition to each new mode (i.e., each surface) and incrementally build a dynamics model for the mode while operating under high stiffness. Once the dynamics models for a mode had been built, we expected the robot to respond to subsequent transitions to this mode by using the corresponding dynamics model.

Fig. 5.8 shows the robot's ability to detect mode changes in one trial of this experiment. The robot was able to identify transitions to existing or new modes with high confidence. In each instance, the second best choice of mode was associated with a much lower value of confidence. The results also show that our hybrid framework and feature representation make performance robust to changes in the direction of motion, i.e., a new mode is not identified when the manipulator moves over a previously seen surface in a new direction. There was some similarity in the confidence values for surfaces 2 and 3 (S2 and S3 in the plot) because of the similarity in their friction values. This is clear in Fig. 5.10 where it can be observed that the frictional force due to surfaces 2 and 3 are similar.

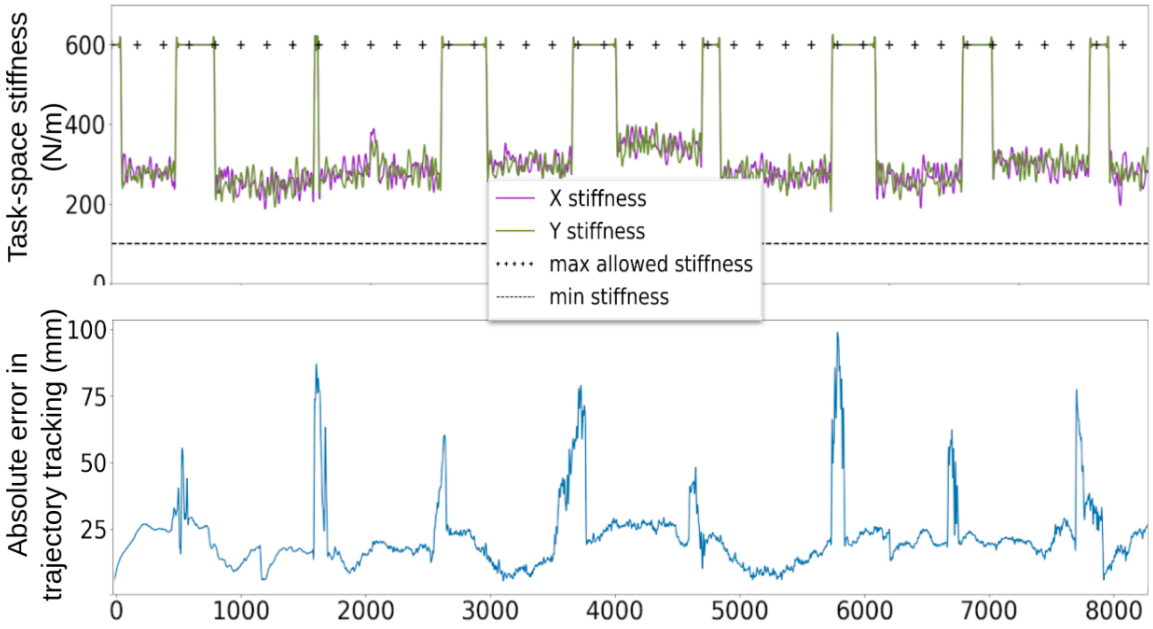


Fig. 5.9 Performance for changing-surface task. **Top:** controller stiffness. **Bottom:** absolute error in trajectory tracking. The spikes during trajectory tracking correspond to a temporary, incorrect feed-forward prediction by the previous model after the guard regions.

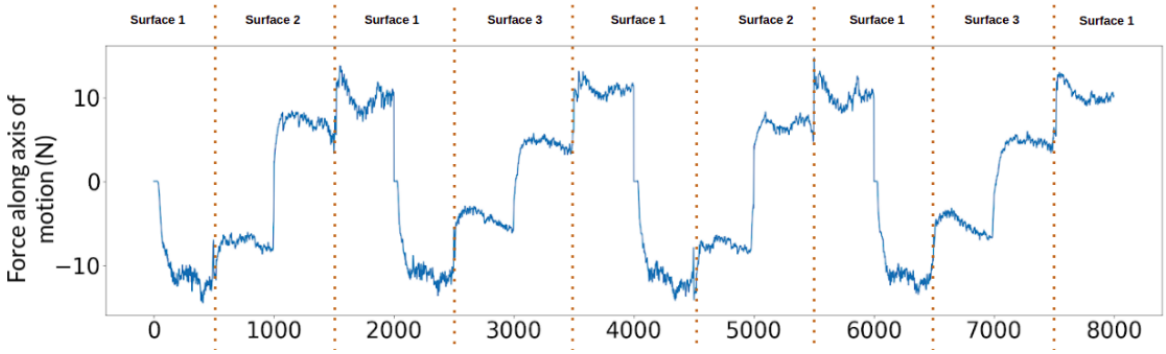


Fig. 5.10 End-effector force measured along the axis of motion. The change in sign of forces indicate change in direction of motion. The dotted vertical lines indicate the points where the surfaces change.

Fig. 5.9 shows the trajectory tracking error and the values of the stiffness parameters of the controller during the trial. The peaks in the trajectory error plot correspond to a sudden change of surface. During each such instance, the predictions made by the dynamics model of the previous mode caused a momentary error in the trajectory tracking ability, until the robot switched to the high-stiffness mode and identified the current mode; the robot then used suitable low(er) stiffness to complete the task. As discussed earlier, switching to a previously seen mode requires a much shorter period of high stiffness compared with building a new dynamics model. These results support hypothesis **H5.2** and to some extent, **H5.3**.

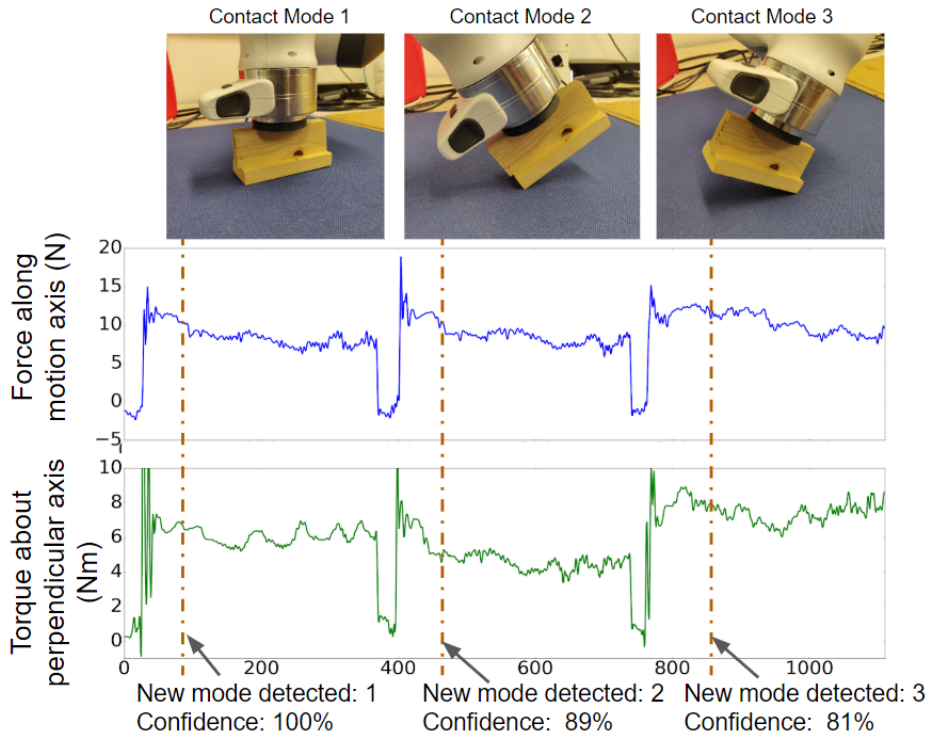


Fig. 5.11 Illustrative trial of changing contact type task. **Top:** Contact types when the end-effector's motion is towards the right; **Middle:** Forces measured along the direction of motion; **Bottom:** Torques measured about axis parallel to surface and perpendicular to direction of motion. Spikes in the measurements correspond to contact transitions; dashed (brown) vertical lines indicate when the framework managed to build a reliable dynamics model of the corresponding mode.

5.3.3 Testing framework robustness and generalisability

To further explore hypotheses **H5.2** and **H5.3**, we ran trials with the changing contact-type task. The robot's task was to slide an object along a desired trajectory on a surface while experiencing each of the three different types of contacts shown in the top part of Fig. 5.11. As stated earlier, the motion pattern was extracted from a single demonstration of the task; the robot did not have prior knowledge of the different dynamic modes. During each trial, the robot approached the table to execute a particular type of contact while maintaining a normal force of 10 N. Contact with the surface triggered a transition, and the robot proceeded to slide the object along the surface with a normal force of 10 N. As before, we expected the robot to confirm any transition to a previously seen mode using data from a brief period of high stiffness motion, and to use the corresponding dynamics model. Also, we expected any transition to a previously unseen mode to require data collection over a longer period of high stiffness to build a dynamics model of the mode. We also expected the robot to be robust to changes in sequence of contacts, motion direction, and normal force.

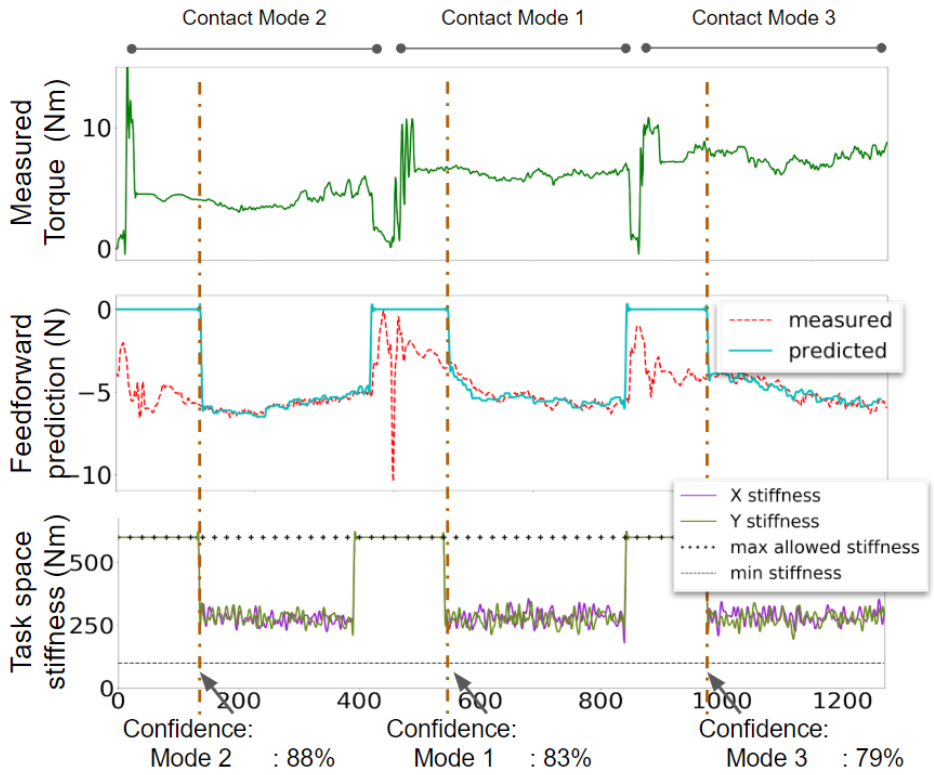


Fig. 5.12 Testing previously constructed dynamics models for the changing contact type task, but with contacts appearing in a different sequence. **Top:** Torques measured about the axis parallel to surface and perpendicular to motion direction; spikes in measurements correspond to contact; **Middle:** End-effector forces predicted by forward model for the current mode; **Bottom:** Variations in the controller stiffness based on errors in the predicted forces.

Fig. 5.11 summarises the learning of dynamics models for a particular trial consisting of the three different modes. It shows the incremental construction of dynamics models for each mode, along with the variation of end-effector force and the torque measured along the axis that is most affected by the motion. We observe that separate models were learned for the three contact types (i.e., three different modes) with high confidence.

Next, Fig. 5.12 shows the results of testing the existing dynamics models with the same overall trajectory but with a different sequence of contacts. We observed that the robot was still able to recognise the modes quickly and accurately, regardless of the order in which they occurred. The second plot in Fig. 5.12 shows the end-effector forces predicted by the dynamics model for the current contact mode. The feed-forward term was used and revised online when the model is reliable, but the term's value was zero when the robot had not identified the mode. Similarly, the stiffness parameters of the impedance controller were varied according to the prediction error of the dynamics model; recall that high stiffness

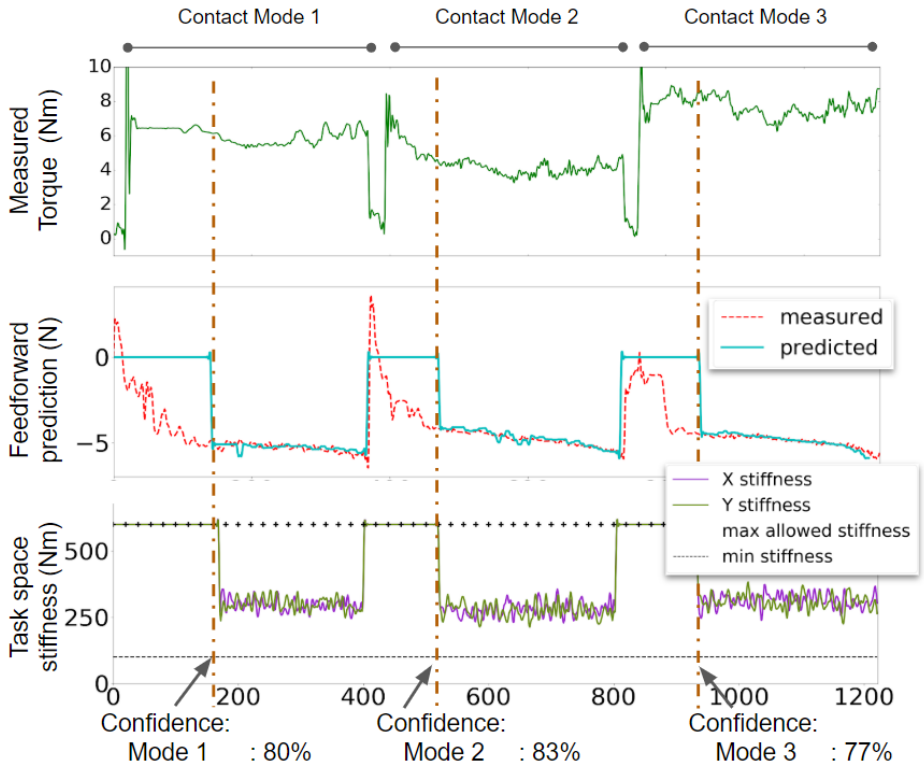


Fig. 5.13 Testing previously constructed dynamics models for the changing contact type task, but with motion in a different direction. **Top:** Torques measured about the axis parallel to surface and perpendicular to direction of motion; **Middle:** End-effector forces predicted by forward model for the current mode; **Bottom:** Variations in the controller stiffness based on errors in predicted forces.

Normal Force = 10 N	Ground Truth		
Detected Mode	Contact 1	Contact 2	Contact 3
Contact 1	83	9	16
Contact 2	2	88	1
Contact 3	14	2	79
New Mode	1	1	4
Normal Force = 20 N	Ground Truth		
Detected Mode	Contact 1	Contact 2	Contact 3
Contact 1	81	10	17
Contact 2	3	86	1
Contact 3	15	2	77
New Mode	1	2	5

Table 5.2 Probability of match computed by algorithm (in %) over 10 trials of mode recognition based on previously constructed dynamics models for the three types of contacts (top part of Fig. 5.11). **Top:** Normal force of 10 N; **Bottom:** Normal force of 20 N.

values are used temporarily when the robot is collecting data to build the dynamics model for the mode.

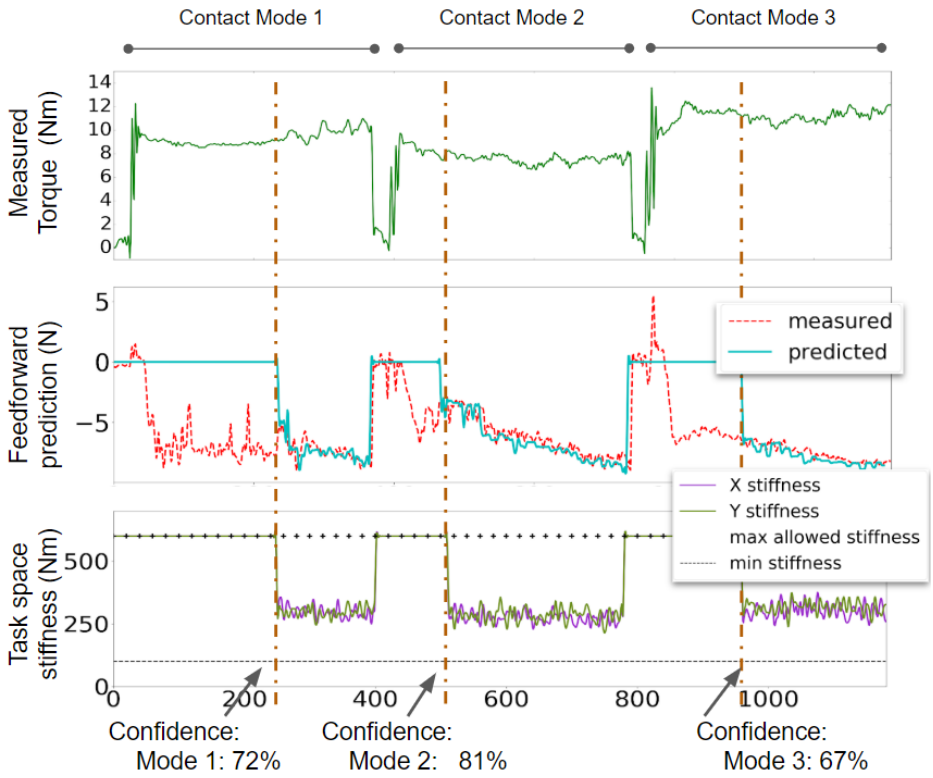


Fig. 5.14 Testing previously constructed dynamics models for the changing contact type task, but with a different normal force (20 *N* instead of 10 *N*). **Top:** Torques measured about the axis parallel to surface and perpendicular to direction of motion; **Middle:** End-effector forces predicted by current mode's forward model; **Bottom:** Variations in the controller's stiffness values due to errors in predicted forces.

Next, Fig. 5.13 demonstrates the robustness of the framework to motion along a direction different from that used during training. The feed-forward model predictions and the corresponding variable impedance behaviour for one of the trials is shown, along with the dynamics model chosen with the highest confidence (bottom of the figure). The modes were always identified correctly.

The framework was then tested for the same task and set of contacts but with the manipulator applying a different constant normal force on the surface. The robot's performance in one experimental trial of identifying the modes and adapting the the existing dynamics models is summarised in Fig. 5.14. These results match those in Table 5.2; although the confidence associated with the modes was slightly lower and the robot took a little longer to recognise the modes when the normal force was changed, the hybrid framework was able to recognise the modes correctly and complete the task successfully using variable impedance control.

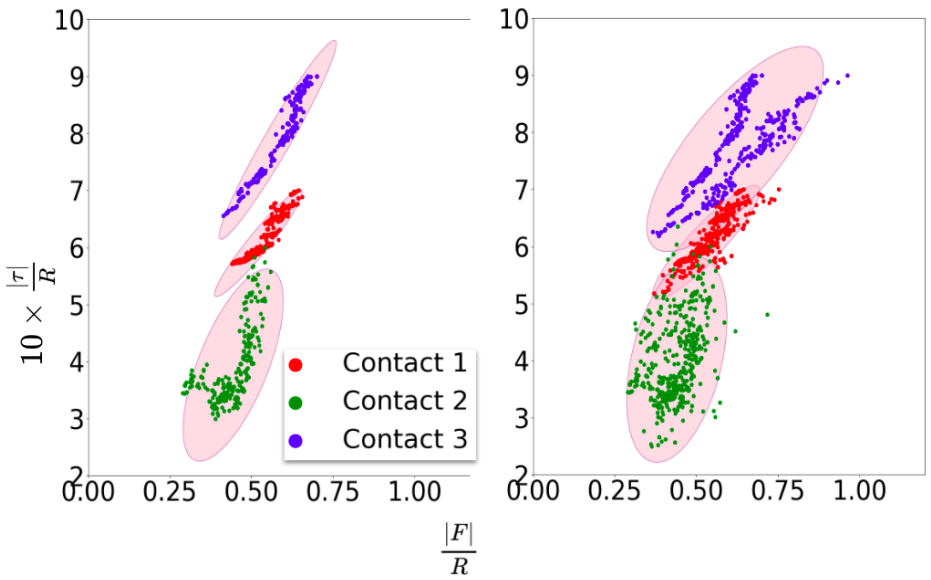


Fig. 5.15 Visualization of the feature representation used for learning the transition models for recognizing different types of contacts. **Left:** 2D features from one trial of the three different contacts with 10 N normal force; **Right:** 2D features from when combining the data for 10 N and 20 N normal force; feature vectors are less separable.

The reason for the worse performance when different normal forces come into play can be attributed to the kinetic friction assumption ($\mu = F/R$) being unrealistic in many real world tasks. Fig. 5.15 (left) shows a sparsely populated feature space for a single trial of the changing contact type task when applying a constant normal force of 10 N. The ellipses denote the mean and covariance of the features for the three contacts. The ellipses are distinct and separable using clustering in this space. However, when data from another trial of the task where the applied normal force is 20 N is added to the dataset, the clusters are less separable (Fig. 5.15, right). This indicates that the feature representation is less capable of distinguishing between these types of contact modes when the applied normal forces are varied. The results in Figs. 5.11 to 5.13 also indicate that the time taken to recognise modes is longer if the modes under consideration are similar, e.g., modes 1 and 3 in these experiments. These results support hypotheses **H5.2** and **H5.3**, but indicate the need for further research on the choice of the abstract feature representation for the dynamic modes of changing-contact manipulation tasks.

5.4 Evaluating the need for online learning in hybrid systems

This section will try to motivate the need for having an incrementally adaptive and reactive framework for predicting and handling discontinuous dynamics. The intention is to show that the overall performance of the our hybrid framework presented in this chapter can be attributed to: (i) learning in the task-space of the robot as opposed to the joint-space, (ii) having a feedback mechanism (albeit noisy) for obtaining the last measured values of forces and torques, (iii) a reactive mode-change detection component that identifies mode changes based on actual real-time force feedback instead of relying solely on historical training data, and (iv) having an incremental learning style that not only relies on training experiences but can also use new information to improve knowledge about the interaction dynamics.

As the baseline for comparison, we choose a fully offline long-term dynamics prediction framework (hereafter referred to as 'baseline framework') developed by Khader et al. [84] which learns a set of probabilistic Gaussian Process (GP) models corresponding to the different modes identified from a dataset. The baseline framework automatically identifies different dynamic modes in the task from a provided training dataset, builds separate dynamics models for each identified mode, and uses a probabilistic algorithm for multi-step prediction of joint-space state variables for a contact-rich manipulation task. The framework is originally proposed as a data-efficient way of predicting long-term state evolution of joint-space dynamics for tasks involving discontinuous dynamics such as changing-contact manipulation, and is not proposed as a solution for real-time control of a robot in such tasks. However, by making incremental changes to the framework and algorithm set up, it was possible to show the benefit of having a short-term predictive model, real-time feedback, and an incremental hybrid model learning framework in the task-space for practical manipulation tasks. Another important factor for selecting this framework as baseline is the similarity of the tasks that were selected by the authors for demonstrating their framework. The next section provides a brief overview of the original baseline framework, before proceeding to the experimental evaluation.

Overview of the baseline framework

The original baseline framework learns a hybrid model from a training set of demonstrations of a task done with a few variations across trials to capture the variability. The training data consists of several sequences (corresponding to trials of tasks) of joint positions, joint velocities, as well as 3D positions and velocities of three points on the end-effector of the robot. The main components of the framework are:

1. *Mode discovery through clustering*: Modes are clustered from the training data using the feature vector defined by concatenating the positions and velocities of three non-collinear points at the end-effector of the robot expressed in Cartesian coordinates. This representation acts as a proxy for capturing the rotational information of the end-effector in addition to its position. The clustering method used is the Dirichlet Process Gaussian Mixture Model (DPGMM) automatic clustering method which automatically infers the number of modes in the clustering data given an upper bound. This method often results in over-clustering (clustered modes are more than the actual interaction modes in the task) due to the feature representation, but the authors claim that "such over-identification is practically advantageous since it reduces the GP training time" for the individual clusters.

2. *Mode dynamics*: Individual mode dynamics (for each cluster of modes) are learned from the training data using Gaussian Process Regression (GPR). For each mode, a corresponding multi-output GP learns the function $p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{u}_t)$, where $\boldsymbol{s}_t = [\boldsymbol{q}_t \quad \dot{\boldsymbol{q}}_t]$ is the state vector comprising of joint positions and velocities, and $\boldsymbol{u}_t$ is the control command at time t. The dynamics model learns to predict the state of the system in the next timestep given the current state and command.

3. *Guard functions*: The guard function is a deterministic classifier that is trained to predict the next mode m_{t+1} as a function of $(\boldsymbol{s}_t, \boldsymbol{u}_t)$. A multiclass support vector machine (SVM) is used for building the guard function using the labels obtained previously through clustering (see point 1 above).

4. *Reset maps*: Reset maps are a critical component in the baseline framework for handling the discontinuities in the dynamics. The reset map learns to predict the evolution of states across the boundaries of the modes (guard regions) as it transitions across modes $m_i \rightarrow m_j$. For every (i, j) in the set of observed mode sequences, a multi-output GP is learned to approximate the reset map state transition $p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t\boldsymbol{u}_t)$ across the guard regions.

In addition to the above, the framework also has a method for propagating uncertainty through the learned hybrid model using a particle-based method which makes use of unscented transform (UT), as well as a strategy for probabilistic switching between modes for long-term prediction, but these will not be described further as they are not directly relevant to our discussions, although they are a critical contribution of the paper. For more details regarding the overall framework and its components, the readers are referred to the original paper [84].

5.4.1 Experimental setup

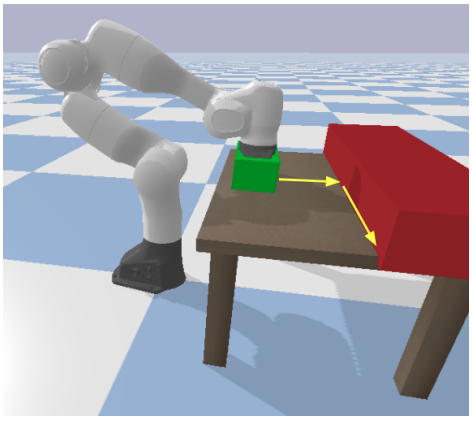


Fig. 5.16 The simulated task setup used for comparison with baseline framework. After approaching the table from top, the robot slides its end-effector (green) along the table (brown) till it collides with a wall obstacle (red), after which it slides along the wall while maintaining contact with the table.

As explained in the previous section, the baseline framework learns to predict the evolution of joint states (joint positions and velocities) from training data. By training the model with multiple trials (≈ 40) with different controller stiffnesses, the authors were able to show that the framework can predict the long-term joint-space dynamics reasonably well using a new controller stiffness. The task considered was a changing-contact task where a robot (with a block fixed at the end-effector) approaches a table making contact with it, and slides along the table till it collides with a wall, after which it has to slide on the table along the wall. The contact changes occur when the robot makes contact with the table, and then when it collides with the wall. We replicated a similar task in simulation (using PyBullet physics – see Fig. 5.16), and were able to achieve similar results for the experiments stated in the original paper (these results are not included here).

It is to be noted that the clustering, individual GP training, and SVM training together requires significant training time depending on the density & length of data in the training set, and the implementation of GP used. We used a sparse multi-GP implementation provided by Matthews et al. [125]. Our re-implementation of the baseline framework required ≈ 55 minutes training time on an 8-core, 16GB RAM computer without using a GPU, for a dataset consisting of 40 trials of the task mentioned above; the long-term prediction process during testing takes ≈ 20 minutes.

For demonstrating the need for an incrementally updating model in the task-space for handling the dynamics of changing-contact tasks, the baseline framework was incrementally modified after each experiment to test the following hypotheses:

H5.4: *Long-term prediction of joint-space dynamics in terms of joint-positions and velocities is unreliable for discretely changing dynamics unless the robot is always using high-stiffness control.*

H5.5: *Long-term prediction of dynamics requires the system to be trained with the same environment that it will be tested on, since the prediction does not make use of real-time system feedback during each step of the prediction.*

H5.6: *Consecutive one-step prediction of task-space dynamics (in the form of end-effector wrenches) is more reliable than one-shot long-term prediction, but an incrementally updating model is required to model unseen environments.*

5.4.2 Long-term prediction of joint dynamics

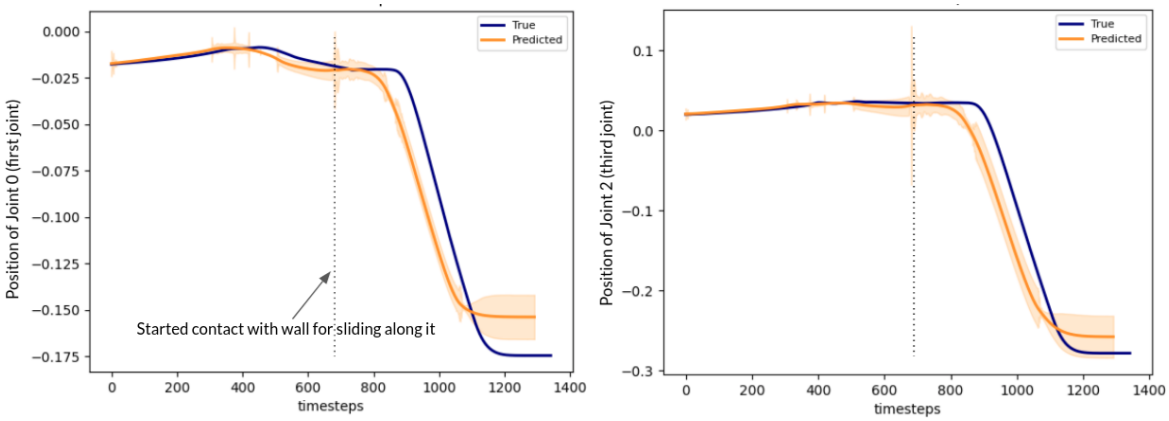


Fig. 5.17 Long term prediction of two joints' positions produced by the baseline framework. The system was trained with wall friction 0.6, and tested on a wall of friction 0.9. The dotted vertical line indicate when the robot made contact with the wall obstacle.

For testing hypotheses **H5.4** and **H5.5**, we modified the task so that the friction coefficient of the wall (second collision – red block in Fig. 5.16) can vary across trials. The training data now contained trials from when the robot performed the same task (using a fixed medium-stiffness controller) but for different friction values of the wall. The system was first trained with fixed controller stiffness values, but with a fixed wall friction (0.6). Testing was then done with an unseen higher friction value (0.9) and the same controller. Fig. 5.17 shows the prediction and actual values of the most relevant joint positions in the task (joints 1 and 3). Being unaware of the change in friction, the offline framework predicted the joints to move as freely as it did during training. The true joint positions were, however, affected by the higher friction resulting in the "true" values lagging behind the "predicted" values.

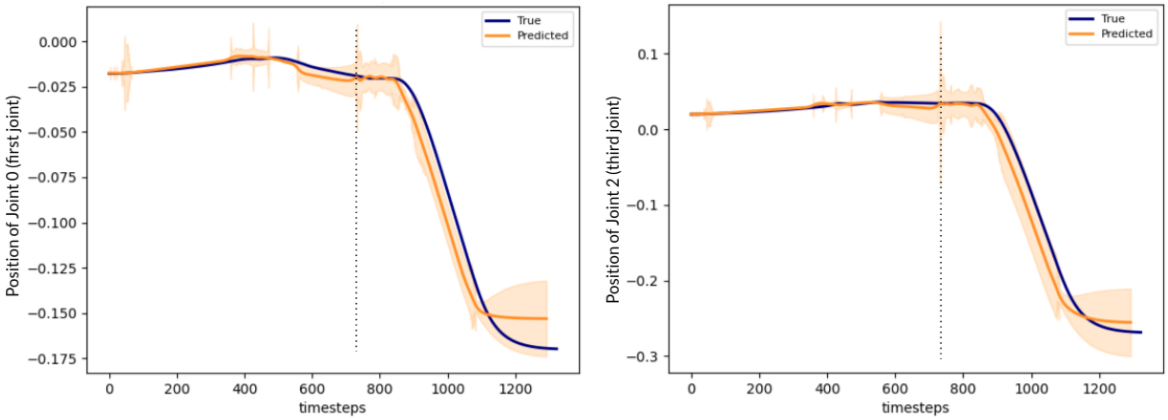


Fig. 5.18 Long term prediction of two joints' positions produced by the baseline framework. The system was trained with wall friction values 0.1 through 0.7, and tested on a wall of friction 0.9. The dotted vertical line indicate when the robot made contact with the wall obstacle.

For further exploration, the training set was modified to include wall friction values between $0.1 - 0.7$ at increments of 0.1; the system was then tested on a wall with friction 0.9. This results in a prediction that is similar to the previous set of experiments, but with a wider band of uncertainty around the prediction due to the larger variability in the training data (Fig. 5.18). The predicted band just covers the true value in its uncertainty band, but the mean is still off from the actual joint values. This showed that the framework learns to model the differences in measurements across trials as uncertainty in prediction.

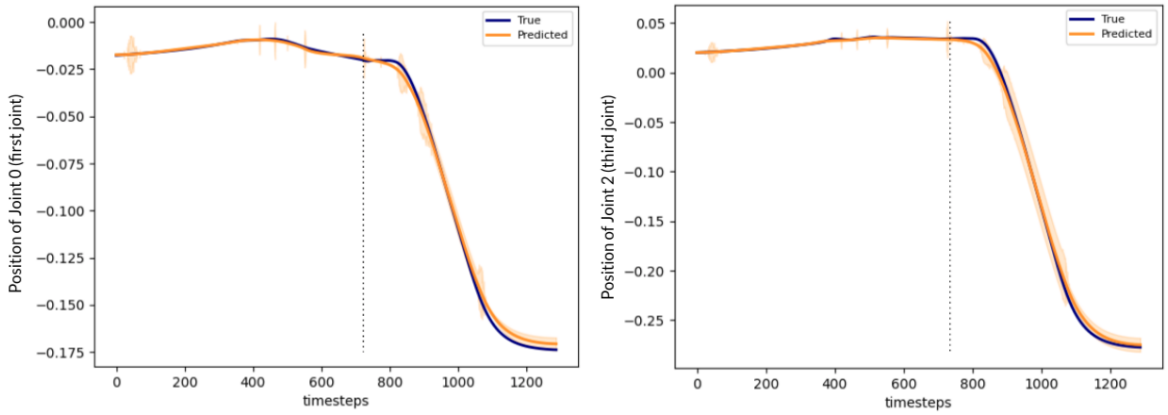


Fig. 5.19 Long term prediction of two joints' positions produced by the baseline framework. The system was trained and tested using a constant high-stiffness controller (friction values same as in Fig. 5.18). The dotted vertical line indicate when the robot made contact with the wall obstacle.

However, when the controller used for training and testing is changed to a *high stiffness* controller, the predictions are more accurate (Fig. 5.19), since the trajectory tracking is now more accurate and robust to external forces due to different friction. The robot therefore

follows the trajectory quite closely regardless of friction value. This results in lower variability across trials (marked by the narrower uncertainty band around the prediction), and also matches the actual values during testing.

Discussion

Although the example used for the above experiments is a case where the surface friction changes, similar prediction errors will be observed if any other dynamics properties of the environment are changed, such as the mass of an object being handled or the viscosity of a fluid the robot is interacting with.

The main reason for the poorer performance of the framework in the presence of unseen environments is the fact that the system is not aware of the dynamics properties of the environment (in this case, frictional coefficient of the wall) before it performs the prediction. This is not a direct flaw of the framework as it was not designed for such use cases. It can also be argued that by making the prediction depend on the friction coefficient (or some other proxy for differentiating the surfaces), the predictions can be improved. This may be possible, but would still require training the system with surfaces of different friction, as well as being aware beforehand what the value of the friction coefficient is.

The experiments in this section showed that for accurate long-term prediction of joint-space dynamics, the environment has to be fixed and/or the robot has to use a high-stiffness controller at all time, hence supporting **H5.4** and **H5.5**. For predicting long-term dynamics for a variable environment, the system will at least have to be trained with several examples of possible environments and will need a way of knowing the environment before performing prediction. Some of these issues can be partially addressed if the robot learns the dynamics in the task space and has access to a more direct measurement of the interaction dynamics such as end-effector force-torque measurements.

5.4.3 Long-term prediction in the task-space

The baseline framework was now modified to learn interaction dynamics in the same space as the forward model of our AVIC framework, so as to make the comparison easier and more intuitive. This required modifying the mode dynamics GPs and the reset map GPs of the framework to now learn the task space dynamics $p(F_{ee_{t+1}}|F_{ee_t}, \dot{x}_t])$ similar to our dynamics learner (see Section 4.1.2). The other parts of the original framework were left mostly unchanged.

The same experiment was then repeated with the new state-space. The system was again trained with examples of the state evolution when the surface friction is between 0.1 and

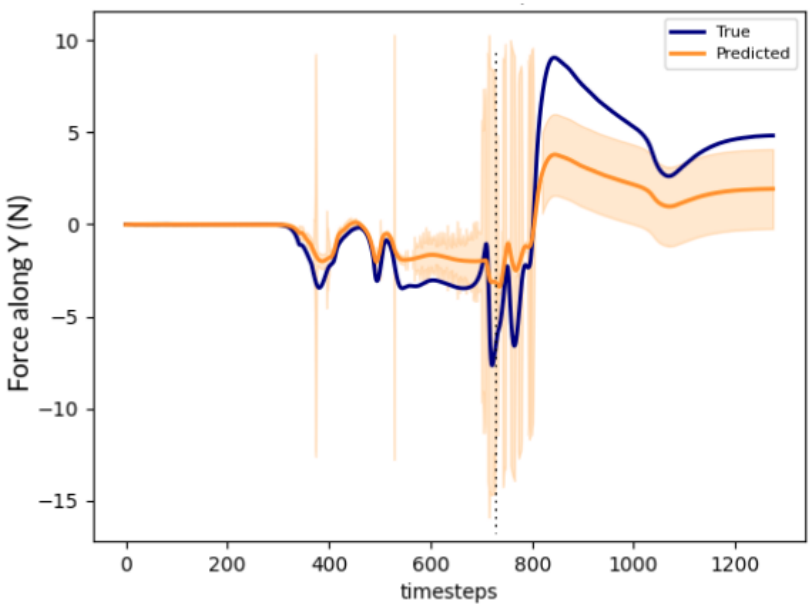


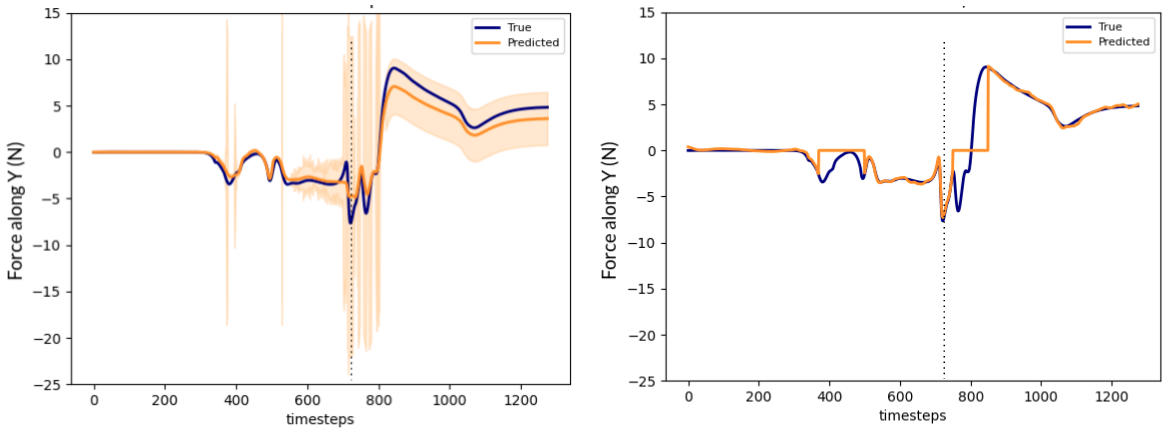
Fig. 5.20 Long term prediction of end-effector force along Y (along the wall). The system was trained with wall friction values 0.1 through 0.7, and tested on a wall of friction 0.9. The dotted vertical line indicate when the robot made contact with the wall obstacle.

0.7, and then was compared with the measurements taken when the surface friction is 0.9. Fig. 5.20 shows the measured and predicted force measurements along Y axis (the direction along which the friction of the wall acts). The spikes at ≈ 400 along the X-axis corresponded to when the robot made contact with the table, and the spikes between $700-800$ corresponded to when the robot collided with the wall. We observed that the actual values of friction force after contact with the wall did not fall within the predicted band of uncertainty. Also, the effect of different levels of friction was more observable in the task space, indicating that it is more meaningful to represent the interaction dynamics for changing-contact tasks in the task space.

As another intuitive example to show the effectiveness of such a task-space representation of dynamics in identifying modes and mode changes, consider a robot approaching a movable object from a side with a constant velocity v. If the object is movable (and light enough), the robot can continue to move at velocity v while pushing the object. With the original joint-space representation used in the baseline framework, the mode change due to the new contact will not be captured. On the other hand, the modified task-space representation which uses end-effector wrenches will recognise the contact change due to the spike in the measurements, and identify the subsequent pushing motion as a new mode due to the markedly different interaction dynamics (the new end-effector wrench pattern after contact will be a function of the weight of the object, friction, etc.).

The region in the figure where the collision occurs has large variability across trials. This is again because the system has no knowledge of the actual environment properties, i.e. friction of the surface, and blindly predicts a long-term profile from the initial state, based on the training data. This experiment also supports hypothesis **H5.5**, and suggests that interaction dynamics cannot be reliably predicted without using real-time system feedback.

5.4.4 One step prediction in task-space using real-time feedback



(a) Prediction from the modified baseline framework.

(b) Predictions from our hybrid framework.

Fig. 5.21 One-step prediction of end-effector force along Y-axis (along the wall). Each system was trained with wall friction values between $0.1 - 0.7$, and tested on a wall with friction 0.9. Dotted vertical line indicates when the robot made contact with the wall (obstacle).

For the next experiment, real measurements from the robot ($[F_{ee_t}, \dot{x}_t]$) was provided as feedback to the system online during testing. At each timestep, the system then only had to predict sensor values for the next timestep using the dynamics model for the identified mode. This is more similar to our predictive learner which does online prediction based on previous measurements from the system. Although this strategy produced better results (Fig. 5.20) by using the current newly seen values to predict corresponding outputs by extrapolating from the learned model, the predictions are still not accurate. This is because the model is not incrementally updated to account for the new environment dynamics[2]. Our hybrid framework (AVIC), on the other hand, supports incremental, real-time updates to the dynamics models. This resulted in more reliable predictions for the same experiment (see Fig. 5.21b) and thus more accurate trajectory tracking in the presence of discretely changing dynamics and previously unseen environments. These results support **H5.6**.

[2]It is also affected by the fact that Gaussian model representations such as GP and GMM tend to move towards the mean of the output dimensions of the models on conditioning, especially when the query points are far from the training data.

Inference and Conclusion

The above experiments showed that prediction of dynamics is more useful for changing-contact tasks when the learning is done in the task-space, and when a good feedback of the system state is available for predicting the next state. Feedback from the actual system during task execution provides relatively better predictions even in unseen environments by extrapolating from the learned model. This is also largely due to the feature representation and state-space used for learning. However, even with real-time feedback, having a fixed learned model cannot provide accurate predictions when the environment is too different from the training data. Furthermore, the disadvantages of having a fixed learned model in the context of continuously changing environments (such as the spring environment or porridge environment from Chapter 4) as well as in discretely changing environments (changing surface friction as in Section 5.3.2) has been previously demonstrated (Section 4.3.2). These experiments show that a hybrid control framework that supports such control and learning is better able to adapt to previously unseen variations in the dynamics and provide smooth control of changing-contact robot manipulation tasks.

5.5 Summary and discussion

This chapter described a computational framework for changing-contact manipulation by formulating changing-contact manipulation tasks as a piece-wise continuous hybrid system. Any such task is considered to be made up of discrete modes with continuous dynamics and distinct control strategies. Each mode comprises a forward (predictive) model, a hybrid force-motion (feedback) control law, and a relevance condition. The use of different representations for a mode's components enables the robot to automatically, reliably, and efficiently identify mode changes and incrementally adapt the dynamics models for the modes. Unlike data-driven methods that require many labelled training examples, our framework is able to build and revise the dynamics model for each observed mode from very few examples. Unlike existing control methods for manipulation tasks, our method is not limited to the sequence of modes seen during demonstrations, and it does not require prior information about the number of modes in the task.

The proposed framework was experimentally evaluated on a physical robot performing the representative changing-contact manipulation task of sliding an object held in its end-effector along a desired motion trajectory on a surface. Experimental results illustrate the ability to reliably follow the desired motion trajectory in the presence of changing surface friction, type of contacts, and applied force, invariant to changes in the motion direction and magnitude of applied forces. In addition, the framework formulates changing-contact

manipulation tasks such that it can be applied to other tasks in this category such as peg-insertion, block pushing, stacking, etc. Furthermore, our hybrid framework may be adapted to other dynamics and control problems such as a mobile robot navigating, exploring, and collecting samples from different terrains.

The necessity of having an incrementally adapting learning framework for handling the discontinuous dynamics of changing-contact tasks was explored by comparing with a baseline framework from literature. The baseline was a long-term prediction framework for hybrid systems which learned to cluster different modes of dynamics from pre-collected data and then used a probabilistic method to predict the evolution of dynamics. By making incremental changes to this framework, we were able to show that the prediction performance improves when real-time system feedback is used for consecutive single-step predictions of task-space dynamics, but for reliable prediction of dynamics in changing contact tasks in new environments, an incrementally adaptive model is required.

The presented hybrid framework relies on the incremental mode detection module and the capability to learn separate dynamics models for each mode. However, the current approach for detecting mode changes will be difficult to use in tasks with many more modes. Further research in the choice of feature representations and the state space used for learning could help tackle such issues. It would also be interesting to explore the automatic selection of the abstract feature representation suitable for the modes of each changing-contact manipulation task.

The final objective of this thesis is to enable reliable, efficient, and smooth control in the context of a robot manipulator performing complex assembly tasks with multiple objects in complex domains. The next chapter will try to address some of the limitations of the current framework and explore new directions. For instance, the current strategy of switching between modes (and dynamics models) is not smooth, with large spikes in sensor measurements in the guard (i.e., transition) regions due to collisions and mode changes that are not predicted or otherwise accounted for using an appropriate control strategy. The next chapter will investigate the possibility of accurate prediction of the time and location of contact, and try to use this information to achieve smooth overall motion dynamics in changing-contact tasks.

Chapter 6

Contact Anticipation and Handling Contact Changes

Many core industrial assembly tasks, e.g., peg insertion and stacking, and human manipulation tasks, are changing-contact tasks whose discontinuous dynamics can result in poor transition-phase behaviour or instability [146]. Handling these discontinuities smoothly with minimum energy intake to the system is very important to reduce damage to the robot and/or the domain objects.

As discussed in Section 3.5, several attempts to model contacts and use switching algorithms to change controllers have been attempted for many years [120, 131]. Smooth motion along a desired trajectory can be achieved in a changing-contact manipulation task using an accurate analytical model of the transitions or a learned model that predicts the transition dynamics. However, analytical models of the impact dynamics of a system of objects require comprehensive knowledge of the objects' physical and geometric attributes, and often impose assumptions not satisfied in practical domains [71, 135]. Methods that learn the attributes of the objects, build object classifiers based on these attributes, and/or learn sequences of parameters (e.g., joint angles) to achieve the desired trajectory, find it challenging to acquire sufficient examples of different objects, contacts, and attributes to learn generalisable models [16, 67].

Another approach is to use a *transition-phase* controller that has properties such as low velocity and stiffness to reduce impact forces, vibration, and jerk on impact. Existing transition control strategies switch to a different controller *after* a contact is detected. This switch can cause substantial discontinuities in the interaction dynamics, damaging the robot or the objects [131, 175]. Instead, an ideal framework should be able to predict contacts and adapt the velocity and stiffness during the transition phase to minimise discontinuities, with the robot switching between controllers smoothly to produce smooth motion overall.

With these insights as motivation, we extend our online learning and control framework to include a contact-change-handling module that decouples the contact prediction and control problem into independent components. The contact anticipation module tries to predict *when* a contact change will occur, while a separate 'transition-phase' controller is used while approaching these regions. The transition phase controller has some set of ideal properties that would help reduce the effects of impacts and contact changes while ensuring that the overall motion is smooth. To ensure that the modules fit in with the overall online framework, both components need to have intuitive task-specific parameters that can be easily tuned from task specifications and/or improved from very few repeats of the task.

This chapter presents a contact-change-handling module that respects these requirements. Instead of building a unified model for predicting contact dynamics and learning a sequence of control commands, we seek to improve the estimates of contact locations in a few trials and adapt the velocity and stiffness during the transition phase to minimise discontinuities. The formulation tries to answer the following questions: (**Q1**) How best to predict contacts accurately? (**Q2**) When to activate the transition-phase controller? (**Q3**) How best to adapt the transition-phase controller's parameters to the task? and (**Q4**) What representation and strategy to use for reliable and efficient control with limited examples for providing smooth, jerk-free motion?

This chapter is an expansion of the paper published at the 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) ([**?**]). The chapter starts by explaining the two types of mode changes that could occur during manipulation in a static environment (Section 6.1). We then describe the contact prediction model that is used to estimate the 'regions of anticipated mode transition' in a task (Section 6.3). Then in Section 6.4, we discuss the properties and structure of our 'transition-phase' controller which would be used when the robot enters a region of anticipated mode transition. Section 6.5 describes how the framework can smoothly transition between the default controller and the 'transition-phase' controller by ensuring smooth and jerk-free motion during the transition. Section 6.6 describes how the task-space controller stiffness matrix of the transition-phase controller is rotated such that the robot is compliant along the direction of impact. We then evaluate our framework on a physical robot and in simulation, using the motivating example of sliding tasks that involve making and breaking contacts with objects and surfaces of different attributes (Section 6.7). The chapter is concluded in Section 6.8 by summarising and critically analysing the advantages and limitations of the presented contact-change-handling module in the context of real-world changing-contact tasks.

6.1 Types of mode transitions

As explained previously, mode transitions bring about discontinuities in the interaction dynamics for a robot performing any manipulation task, making it a piecewise continuous hybrid system. These discontinuities can occur due to making or breaking contacts, or due to discrete changes in the physical properties of the environment/object that the robot is interacting with. In a static environment, therefore, the robot may experience discontinuities in dynamics (mode transitions) of two types: (i) impact transitions (collisions), and (ii) impactless transitions. The robot experiences these in different ways, and each has distinguishable effects on its dynamics. The control strategy required for the robot to handle these changes so as to produce a smooth motion is dependent on the effect each has on the robot and the overall dynamics. The following sections discusses these two types of mode transitions and their characteristics, which would help us design ideal control strategies to deal with each so as to incorporate a contact-change-handling module to our incremental framework for changing-contact manipulation.

6.1.1 Impact transitions (or collisions)

The most easily observable type of mode transition for a robot performing changing-contact tasks is collision. Collisions occur when a moving robot (end-effector) comes in contact with a fixed object in its workspace. Given that a task plan is known in a fixed static environment, there are several methods of estimating the contact locations for such mode changes. Unless occluded or visually indistinguishable, visual sensors are prime candidates for providing rough initial estimates of where collisions can occur in a planned motion.

Characteristics of impact transitions

- An important characteristic of collision-based mode change is that it will always remove at least one degree of freedom (DoF) of the robot in a functional coordinate space. Suppose $\boldsymbol{q} = (q_1, ..., q_n)$ be the set of joint positions of an n-DoF robot manipulator, and let $\boldsymbol{x} = P_{FK}(q)$ be the 6-dimensional mapping to a functional coordinate system such as the end-effector frame or contact frame, which describes position and orientation in the most intuitive way for any particular task, for e.g. $\boldsymbol{x} = (x, y, z, \theta, \phi, \psi)$. When motion along a direction is blocked, say, by an obstacle, a constraint is introduced. If we suppose that a contact restricts k DoFs in the functional space, then k coordinates from the set $\boldsymbol{x}$ reduce to zero, creating a separation $\boldsymbol{x} = (\boldsymbol{x}^f, \boldsymbol{x}^c)$, where $\boldsymbol{x}^f$ is the set of

$6-k$ free coordinates and $\boldsymbol{x}^c$ is the set of k constrained coordinates expressed in the functional coordinates as $\boldsymbol{x}^c = P^c_{FK}(q) = 0$.

From our original problem formulation where we assume that all objects in the workspace of the robot are fixed in its base frame, it is straightforward to see that colliding with an obstacle in the world will always bring a motion constraint on the robot, which will always be along the direction of the normal impact force. Using the constraint representations explained above, if making contact with a table constrains motion of a freely moving robot along the z axis in a functional coordinate frame, then $\boldsymbol{x}^c = z = 0$, which restricts $k = 1$ dimensions of the full functional coordinate space, leaving $\boldsymbol{x}^f = (x, y, \theta, \phi, \psi)$.

- Collisions are marked by large spikes in force-torque measurements as a function of the relative velocity of approach as well as material properties such as coefficient of restitution and hardness of the objects involved. Furthermore, collisions also bring about a discontinuity in the velocity of the end-effector by forcing at least one dimension in the functional coordinates to go to zero. This causes spikes in acceleration, jerk etc. All these effects can be attributed to the sudden loss in a degree of freedom in the functional coordinate space.

- In our framework, the robot will have to switch the direction of force control after a collision. Typically, some force target is now to be achieved along the direction of the new contact, while following a motion trajectory along the remaining free coordinates. In general, after a collision, the robot will have to break contact if it has to continue with the previously active control representation (i.e. to keep the direction of motion control and force control unchanged). In the example above where the robot makes contact with a table and loses a DoF along z, the robot will either use this collision to mark the end of the task, or will switch to force control along the z direction, so as to follow a motion pattern on the surface of the table ($x-y$ plane) to proceed with the task. If it has to continue with the previous controller (motion control without force control in any direction), the robot will have to break contact with the object.

Detecting collisions

With the use of an end-effector force-torque sensor, it is straightforward to identify large spikes as potential contact changes. However, to differentiate collisions from impact-less contact changes, we have to use our knowledge about the characteristics of collisions as discussed above.

Formally, our framework considers the robot to have collided if the following conditions are all met:

$$|\boldsymbol{F}_t^c - \boldsymbol{F}_t^{pred,c}| > \Lambda_1 \tag{6.1}$$

$$|\frac{d\boldsymbol{F}^c}{dt}| > \Lambda_2 \tag{6.2}$$

$$\dot{\boldsymbol{x}}_t^c \approx 0 \tag{6.3}$$

where $\boldsymbol{F}_t^c$ is the end-effector force in the frame of anticipated contact along the contact normal measured at time t; $\boldsymbol{F}_t^{pred,c}$ is the force predictions made by the dynamics model of that contact mode (the forward model of AVIC), and $\dot{\boldsymbol{x}}^c$ is the end-effector velocity in the contact frame. Condition 6.1 checks if the predicted force is different from the instantaneous measured force by a user-defined threshold Λ_1. Directly checking if the measured force is above a threshold is not meaningful when the robot is moving in an environment where it is continuously moving against a force (e.g. friction, viscous liquid, etc.). Using the predictions provided by the forward model ($\boldsymbol{F}_t^{pred,c}$) as basis removes this issue. Similarly, condition 6.2 checks for sudden spikes in the force sensor measurements, while condition 6.3 checks if the robot lost a degree of motion freedom along the direction of contact. If these three conditions are found to be satisfied, the framework registers it as a collision-based contact change, which would normally mark the end of that segment of the plan.

In practice, however, the contact frame is not known/used in our framework and the direction of impact is not known beforehand. Therefore, we use the magnitude of the force instead of its vector representation in conditions 6.1 and 6.2. Condition 6.3 is then checked along the direction of the force vector. The direction of the detected force is also used for guiding the direction in which the robot has to be compliant in subsequent trials (see Section 6.6).

6.1.2 Impact-less mode transition

The second type of contact change occurs when a the interaction discontinuities occur due to discrete changes in the environment dynamics that are not due to impact. A simple example is when robot slides across two surfaces whose friction are different (Fig. 5.1); here, the robot experiences sudden changes in frictional resistance as it crosses the boundaries between the surfaces, but the transition does not necessarily cause large impact forces or drop in velocity.

Characteristics of impact-less transitions

- The main feature that distinguishes impact-less transitions from collisions is that these mode transitions occur without a loss in degree of motion freedom; in fact, they can sometimes result in additional DoFs, such as when a robot breaks an existing contact. Consider the previous example of a robot approaching and making contact with a table to lose a DoF along the z axis. If the robot is now moving along the table, it has full control only along 5 dimensions ($\boldsymbol{x}^f = (x, y, \theta, \phi, \psi)$). Assuming that the surface of the table is smooth and the robot is moving along a jerk-free trajectory on the table, the interaction forces felt by the robot is smooth and continuous. However, breaking contact with the surface will provide the robot controller with the additional degree of freedom along z as well, and reduces the number of constraint DoFs. This can happen without impact and/or drop in velocity, and is hence falls into the category of impact-less mode changes.

- Impact-less mode transitions will also result in sudden jumps in force-torque profiles similar to collisions, however unlike collisions, the peaks in the curves are typically much less prominent.

- Unlike collisions, these contact changes may not be easily visible to an external vision sensor before task execution (e.g. different surfaces on the same plane may not be visually distinct in a camera image), and often the robot can learn about them only during the first execution of the task.

- Since there is no loss of DoF, the robot can continue to use the same controller or a similar control representation to continue with the task, and does not necessarily require a change in force-motion control directions.

Detecting impact-less transitions

Since impact-less transition does not cause a loss in motion DoF, an impact-less transition can be considered to have happened if conditions 6.1 and 6.2 are satisfied without meeting condition 6.3.

$$|\boldsymbol{F}_t^c - \boldsymbol{F}_t^{pred,c}| > \Lambda_1 \tag{6.4}$$

$$|\frac{d\boldsymbol{F}^c}{dt}| > \Lambda_2 \tag{6.5}$$

During implementation, the checks for mode changes are done by following the logic shown in Table 6.1. This is a simple yet effective way to detect and differentiate between mode changes in most scenarios. However, other than relying on a good force-torque sensor and an accurate forward model, this strategy also assumes that the robot velocity is not too low during impacts. It can be hard to detect spikes in force torque-measurements (especially when the robot is sliding on a surface which offers frictional resistance) due to collision if the robot is already moving too slow. In such cases, the thresholds Λ_1 and Λ_2 become very sensitive hyperparameters. This can often result in the robot identifying collisions a short time after the actual contact occurred (as the robot presses against the contact and crosses the thresholds). Similarly, detecting impact-less mode changes can sometimes be difficult if the robot uses high stiffness (see discussion at the end of Section 6.7.5). Depending on the task, the mode transitions involved, and the velocity of motion, the values for Λ_1 and Λ_2 may also have to be separately tuned for collisions and impact-less transitions.

Table 6.1: Mode change detection logic

if $|\boldsymbol{F}_t^c - \boldsymbol{F}_t^{pred,c}| > \Lambda_1 \quad and \quad |\frac{d\boldsymbol{F}^c}{dt}| > \Lambda_2$ **then**
  `// A mode change has occurred`
  if $\dot{\boldsymbol{x}}_t^c \approx 0$ **then**
    Collision detected
  else
    Impact-less mode transition detected
  end
end

6.2 Framework overview

Figure 6.1 presents an overview of our framework. The inputs are the desired motion trajectory, the force-torque sensor measurements, and the end-effector position. The default controller is the hybrid force-motion adaptive variable impedance controller (AVIC for hybrid systems) that was described in the previous chapters. As demonstrated previously, operating in task (i.e., Cartesian) space allows this controller to use suitable abstractions to learn accurate forward models from very few examples, provide compliance along specific directions, and accurately track the desired trajectory, thus partially addressing Q4 mentioned in the introduction. The framework in this chapter builds on this default controller's representation, enabling a task-space contact anticipation model that incrementally updates its contact prediction using a Kalman filter (Q1). These predictions are used to minimise the time spent

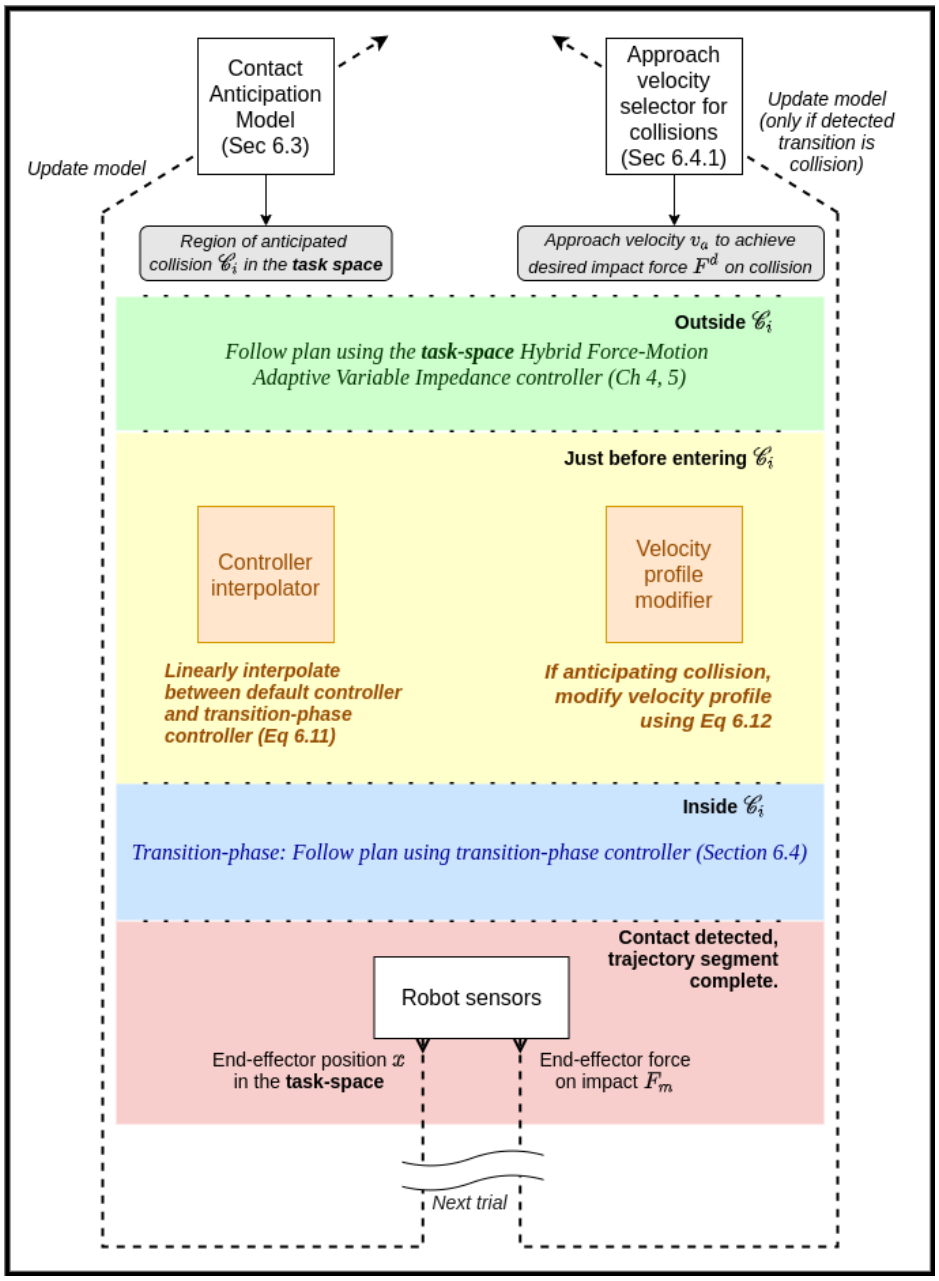


Fig. 6.1 Overview of the modified framework for smooth control of changing-contact manipulation tasks.

in the transition phase (Q2), and the controller properties to be used in the transition phase is set adaptively to achieve a smooth motion profile (and a desired impact force if collisions are expected) (Q3). Once the transition is completed using a suitable controller to minimise discontinuities, the robot moves to using the default controller and revises the parameter values suitably. We begin with a description of the contact prediction method.

6.3 Kalman filter-based contact anticipation

We model the robot's belief about the position of each expected mode change (collision or impact-less) while executing the assigned motion trajectory as a multivariate Gaussian in the workspace, with the covariance ellipsoid denoting the uncertainty along different motion control dimensions; we call this the 'region of anticipated mode transition' denoted by $\mathscr{C}$ (see Fig. 6.2). In the context of this work, a 'contact position' refer to a position in the task-space of the robot where a mode switch occurs when following the provided task plan. Therefore, in any trial of the task, the robot expects a specific contact position $\mathbf{c}$ to lie within the corresponding $\mathscr{C}$.

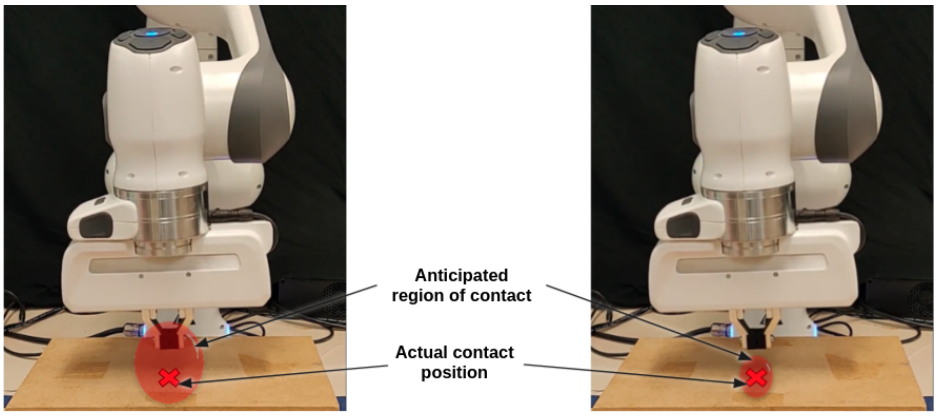


Fig. 6.2 The Kalman filter contact anticipation method models expected contact locations as Gaussians in the task space. The 'region of anticipated mode transition' $\mathscr{C}$ becomes tighter in a few trials as the robot becomes more confident about the contact location. **Left:** In the first trial, $\mathscr{C}$ is large due to uncertainty; **Right:** In trial 3 of the task, the volume of $\mathscr{C}$ is significantly smaller.

The representation of each contact position is compact and is updated over very few trials of the task using a Kalman filter; this also fits with the controller used by the robot which is also defined in the task-space. For the Kalman filter-based contact anticipation model, we assume that each contact position is guided by the state update equation: $\dot{\mathbf{c}} = \mathbf{A}\mathbf{c} + \mathbf{B}\mathbf{u}_k + w$, where $\mathbf{c}$ is the contact position, $\mathbf{A}$ is the object's self-activation (I for positively activated objects), $\mathbf{B}$ is the control matrix capturing the effect of action $\mathbf{u}$ on contact position, and w is Gaussian noise modelling the uncertainty in the contact location. The sensor model uses the end-effector pose (as given by forward kinematics from joint positions) as measurement when a mode change is detected. This sensor model (with noise depending on the joint encoder noise and forward kinematics) will provide a corrected estimate of the contact point once a contact is made, resulting in a reduced covariance ellipsoid for subsequent trials.

The filtering can then be repeated after each trial: the prediction step given by the state transition model and the update step done using the forward kinematics at contact. The initial

estimate of the contact position is assumed to be given as a rough guess by the planner or an external vision sensor, which is used to initialise the Kalman filter.

The Kalman filter's update equations are as follows:

$$\hat{\mu}_{k|k-1} = \mathbf{A}_{k-1}\hat{\mu}_{k-1|k-1} + \mathbf{B}_{k-1}\mathbf{u}_{k-1} \tag{6.6a}$$
$$\Sigma_{k|k-1} = \mathbf{A}_{k-1}\Sigma_{k-1|k-1}\mathbf{A}_{k-1}^{\mathbf{T}} + \mathbf{Q}_{k-1} \tag{6.6b}$$
$$\mathbf{v}_k = \mathbf{y}_k - \mathbf{H}_k\hat{\mu}_{k|k-1} \tag{6.6c}$$
$$\mathbf{S}_k = \mathbf{H}_k\Sigma_{k|k-1}\mathbf{H}_k^{\mathbf{T}} + \mathbf{R}_k \tag{6.6d}$$
$$\mathbf{K}_k = \Sigma_{k|k-1}\mathbf{H}_k^{\mathbf{T}}\mathbf{S}_k^{-1} \tag{6.6e}$$
$$\hat{\mu}_{k|k} = \hat{\mu}_{k|k-1} + \mathbf{K}_k\mathbf{v}_k \tag{6.6f}$$
$$\Sigma_{k|k} = \Sigma_{k|k-1} - \mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^{\mathbf{T}} \tag{6.6g}$$

where $\hat{\mu}_{i|i-1}$ and $\Sigma_{i|i-1}$ are the predicted mean and covariance of the contact location $\mathbf{c}$ at step i, $\hat{\mu}_{i|i}$ and $\Sigma_{i|i}$ are the corrected mean and covariance based on measurement $\mathbf{y}_i$ (of pose on contact) step i, $\mathbf{K}$ is the Kalman gain, and $\mathbf{Q}$ and $\mathbf{R}$ are noise matrices. Note that our representational choices enable us to develop a contact anticipation model that is simple, efficient and yet reliable. We are able to use a linear model to estimate contact locations in the workspace, arriving at an accurate estimate from very few (noisy) repetitions of the task. This representation supports contact with movable objects if their motion dynamics are modelled reasonably well using $\mathbf{A}$ and $\mathbf{B}$, but we assume (in this work) that the end effector only makes contact with stationary objects ($\mathbf{A} = \mathbf{I}$, $\mathbf{B} = \mathbf{I}$), which is reasonable for many manipulation tasks such as industrial tooling. Also, $\mathbf{H} = \mathbf{I}$ since state and measurements are in the same space.

6.4 Transition-phase controller formulation

The transition controller follows the same control structure as our default variable impedance controller (Eq. (4.1)) and varies only in the choice of the control parameters. Depending on the type of mode transition (as discussed in Section 6.1), the choice of the control parameter values can vary, but the basic structure remains:

$$\boldsymbol{u} = \mathbf{K}^{\mathbf{p}*}\Delta\boldsymbol{x} + \mathbf{K}^{\mathbf{d}*}\Delta\dot{\boldsymbol{x}} + \boldsymbol{u}^{\mathbf{ff}} + \boldsymbol{H} \tag{6.7}$$

where $\mathbf{K}^{\mathbf{p}*}$ and $\mathbf{K}^{\mathbf{d}*}$ are the stiffness and damping parameters of the transition-phase controller that can vary depending on the type of mode transition. The different desired properties of

the transition-phase controller for the two types of mode transitions are described in the next sections.

6.4.1 Transition-phase controller for impact transitions

As explained previously, collisions are marked by impacts and will result in the robot coming to rest (in the direction of impact). Since the permitted impact force may differ based on the task, e.g., large forces can damage delicate objects in certain tasks, it is reasonable to assume a safe limit on the maximum allowed impact force for a particular task. It is known that the velocity of approach is directly proportional to the force on impact[1], especially when the robot registers a contact while moving in free space. One advantage of the design of our controller (and the related representational choices) is that a simple approach (linear regression) can be used to fit the relationship between impact force and the approach velocity between a pair of objects. If available, this relationship can be used to compute the approach velocity for a desired impact force (see Section 6.7.2).

However, the robot may not have a model of these relationships when performing a task for the first time. So, for any given target impact force, the robot starts with a safe low velocity during the first trial, using the difference between the target impact force and the measured impact force to revise the approach velocity for the next iteration of the task:

$$\Delta v_a = \beta(\boldsymbol{F}^d - \boldsymbol{F}^c) \tag{6.8}$$

where Δv_a is the value used to revise the approach velocity, $\boldsymbol{F}^d$ is the desired impact force along motion direction, $\boldsymbol{F}^c$ is the measured impact force, and β is a learning rate that is ideally a value less than or equal to the slope of the plot relating impact force to the approach velocity. Over time, this approach enables the robot to learn a task-specific velocity of approach for a desired impact force. The learned linear model can also be reused for other target impact forces.

Also, experimental analysis indicated that reducing the controller stiffness helps reduce the jerk in motion after impact by providing compliance, but has no significant effect on impact forces because the robot has to make the contact for the error and stiffness term in the feedback control loop to come into effect. Thus, reducing stiffness during impact helps in absorbing vibrations after contact, and is hence a desirable feature for impact-handling transition-phase controller. The stiffness and damping matrices are adjusted such that the robot is most compliant along the direction of expected impact while it maintains higher

[1]This was also observed experimentally (see Section 6.7.2 and Appendix B.3)

stiffness along other directions so as to reduce tracking errors. The rotation of the stiffness matrix and the selection of the parameter values are explained in Section 6.6.

6.4.2 Transition-phase controller for impact-less transitions

If the robot expects an impact-less transition in $\mathscr{C}$, it does not have to reduce its velocity and can continue following the original plan. Reducing velocity would bring delays in the task which is undesirable. However, as mentioned in the previous chapter, having a high-stiffness controller helps in identifying new modes after transition. This is because high stiffness ensures that the robot motion is accurate and the sensor measurements are more accurate, which helps in mode identification. For this reason, the ideal transition-phase controller for impact-less transitions should switch to a high-stiffness controller. The parameter values for this transition phase controller (Section 6.4) in our framework is set to use the maximum allowed values of the active AVIC, i.e., $\mathbf{K}^{\mathbf{p}*} = \mathbf{K}^{\mathbf{p}}_{max}$.

6.5 Switching to Transition-Phase Controller

Recall that a lower stiffness in the transition phase can reduce vibrations on impact, and a lower velocity reduces the impact forces. Since any such strategy will cause the robot to deviate from the desired trajectory, the robot should ideally switch to this control phase just before the contact is made, and switch out of it immediately after stable contact is established. Since this is not possible in practice, it is safer to switch to this control mode when it enters a region in the task space where the contact is highly likely to occur, and switch out of it once stable contact is achieved.

As stated previously, we use the covariance of the multivariate Gaussian estimating the contact location to define the region of anticipated contact $\mathscr{C}$ in the task-space. Activating the transition-phase controller just before or after it enters $\mathscr{C}$ ensures that the transition-phase is only active when a contact/collision is anticipated. The part of the target motion trajectory $\mathcal{P}$ within $\mathscr{C}$ can be found by checking if the points in $P(t)$ satisfy the relation given by:

$$(\mathcal{P}(t) - \mu)^T \Sigma^{-1} (\mathcal{P}(t) - \mu) \leq \lambda \tag{6.9}$$

where μ is the mean of the Gaussian predicting contact position, Σ is the covariance, λ is the scaling factor governed by the confidence in the covariance estimate; it is modelled as the chi-squared percent point function of the desired confidence value. The first point in trajectory $\mathcal{P}$ to satisfy this condition is the boundary p_c of the anticipated collision region $\mathscr{C}$ along the direction of approach. When the robot does a task for the first time, the position

uncertainty and hence the volume of $\mathscr{C}$ are large, and the robot switches to the appropriate transition phase controller (as described in Section 6.4) earlier than actual transition. Over time, as the covariance ellipsoid shrinks, the robot switches to the transition controller when it is about to make contact.

6.5.1 Smooth Transition between Controllers

To avoid discontinuities in motion dynamics, the robot needs to smoothly transition from a normal (pre-contact) controller with output $\mathbf{u_1}$ to the appropriate transition-phase controller with output $\mathbf{u_2}$. Linear interpolation of $\mathbf{u_1}$ and $\mathbf{u_2}$ over a time window $[0,T]$ is possible since they are of the same task-space representation, and can produce smooth transition between these controllers:

$$\mathbf{u} = (1-\alpha)\mathbf{u_1} + \alpha\mathbf{u_2}; \qquad \alpha = t/T \qquad t \in [0,T] \tag{6.10}$$

where T is the desired duration of the transition between the controllers. As long as the outputs from the two controllers ($\mathbf{u_1}$ and $\mathbf{u_2}$) are individually smooth, the output of the combination will also be smooth. In this work, controllers use the task-space representation described earlier, with $\mathbf{u_2}$ being the output of the fixed transition-phase controller (low- or high-gain depending on the transition type) as the arm approaches the contact point. We use this strategy to smoothly transition between controllers of different stiffnesses such that the transition is completed by the time the robot reaches p_c. A similar approach is used to smoothly transition from the transition-phase controller to a normal controller after contact is made.

6.5.2 C^∞ smooth velocity profile for jerk-free motion

Transition-phase controllers for handling impact transitions typically use a lower velocity than the original kinematic sequence P to reduce the force at impact. Also, as the region $\mathscr{C}$ is revised by the Kalman filter, the robot will need to switch to a different desired approach velocity at different points in the target trajectory (in different trials). Therefore, the timeline of the trajectory has to be modified to account for the modified velocity profile.

To modify the given motion trajectory such that velocity changes smoothly, we enable the robot to create a new velocity profile and time-mapping. Our approach builds on the trapezoidal formulation used in literature for velocity profiles; it can be viewed as either the *lift-off* or *set-down* phase of a trapezoidal profile. Unlike other representations, our formulation results in motion that is smooth and continuous at all orders, i.e., it is C^∞ smooth.

Without loss of generality, assume that the original motion trajectory $\mathcal{P}$ is along one dimension with velocity v_1. Assuming that transition starts at time t_1 with v_1 and has to be completed at t_2 (i.e., over a period $T = t_2 - t_1$) with approach velocity v_2 (the velocity as the robot crosses boundary point p_c of $\mathscr{C}$), the velocity profile is defined as:

$$v(\tau) = \begin{cases} v_1 + \frac{(v_2 - v_1)e^{-1/\tau}}{e^{-1/\tau} + e^{-1/(1-\tau)}} & \text{if } 0 < \tau < 1, \\ v_1 & \text{if } \tau \leq 0 \\ v_2 & \text{if } \tau \geq 1 \end{cases} \tag{6.11}$$

where $\tau = t/T = t/(t_2 - t_1)$. For $\tau \in (0,1)$, $e^{-1/\tau}$ has continuous derivatives at all orders at every point τ on the real line. Since $v(\tau)$ has a strictly positive denominator for all points in its domain and velocity limits are enforced $\forall \tau \notin [0,1]$, this profile provides a smooth transition from v_1 to v_2 over $[t_1, t_2]$ and $v(\tau)$ is continuous despite its piece-wise definition. Acceleration and jerk are computed as first- and second-order derivatives of $v(\tau)$ with respect to τ, and position trajectory is obtained by integrating the profile; all motion derivatives are continuous—see Figure 6.3. Moreover, the simplistic formulation has the added advantage that it does not have additional hyperparameters to tune like many jerk-free velocity profiles in literature [5, 153, 59].

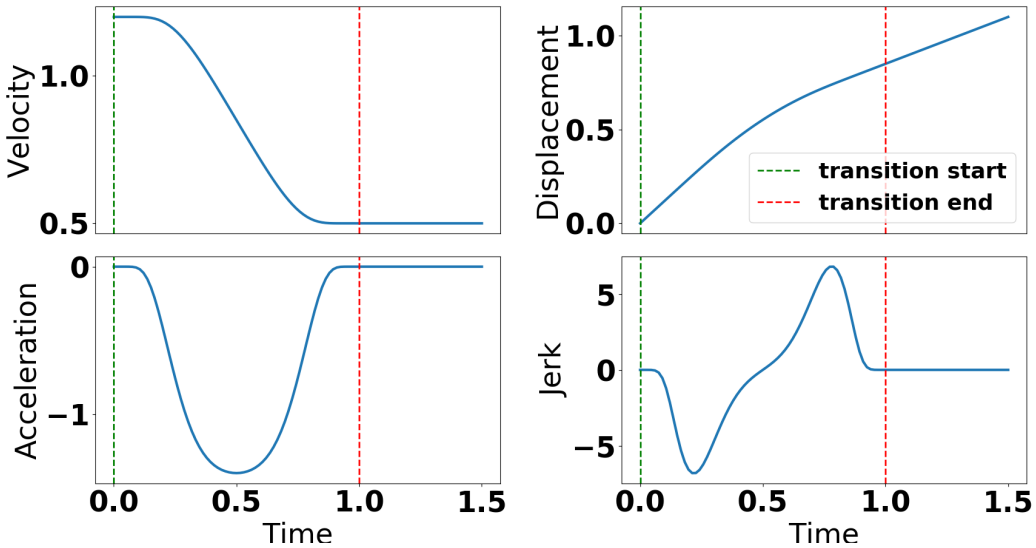


Fig. 6.3 Velocity plots (generated using Eq. (6.11)) with corresponding position, acceleration, and jerk plots. Velocity varies from 1.2 to 0.5 in unit time. It is clear that all curves are smooth and continuous.

In general, it is possible that the total displacement of the original plan may be less than the total distance obtained by integrating the desired velocity profile[2]. In this case, the robot

[2]This happens only when $v_2 > v_1$ and total motion distance is short

may overshoot the original planned goal pose. In such cases, it is possible to incorporate a scaling factor to the function $v(\tau)$, which can modify the value of v_2 or T so as to restrict the maximum distance travelled. A similar scaling factor is used by Grassmann et al. [59] for automatically adapting the motion constraints when the trajectory is too short. In our case however, this scenario is improbable since this cannot happen if $v_2 < v_1$ (which is the normal case when entering a region of expected collision). Furthermore, the requirement of having to complete the transition before it reaches a certain position p_c in the plan P automatically removes this scenario ever happening in the approach phase.

The timeline of the new velocity profile $v(\tau)$ for any given contact can be used to modify the target trajectory P such that the velocity transition is completed as the robot reaches p_c. As the velocity transition duration (Eq. (6.11)) and the controller transition duration (Eq. (6.10)) are the same, the robot will become compliant and slow down just as it enters $\mathscr{C}$.

6.6 Stiffness frame realignment to reduce vibrations

<MAYBE WORTH MOVING TO APPNDX>

In certain situations involving impact transitions when compliance is required in a direction that is not along one of the axes of the operational space coordinate frame, it would be easier to define the impedance parameters in a different frame. For instance, when a robot is approaching a collision, the robot would benefit from being compliant along the direction of contact (which may be different to the direction of motion) so as to minimise the amount of energy absorbed by the system. A simple solution is to use lower impedance gains along all axes. However, this would result in loss of trajectory tracking accuracy along all dimensions, which is especially unacceptable in tasks requiring precision.

A better solution is to use low stiffness only along the direction in which impact is expected, which would reduce the error in trajectory tracking along the other directions. Suppose a lower translational stiffness of $K^{\mathbf{p},low}$ is desired along the direction of impact, while a different stiffness $K^{\mathbf{p}}$ is the original value to be used in the other directions. Then the custom approach-phase stiffness matrix (for translation) can be defined as

$$\mathbf{K}^{\mathbf{p}'}_{[tr]} = \begin{bmatrix} K^{\mathbf{p},low}_{[tr]} & 0 & 0 \\ 0 & K^{\mathbf{p}}_{[y]} & 0 \\ 0 & 0 & K^{\mathbf{p}}_{[z]} \end{bmatrix} \tag{6.12}$$

The ideal rotation should align $K^{\mathbf{p},low}_{[tr]}$ with direction of impact. For this purpose, we define an impact frame in the cartesian space, with origin at the point of impact on the

end-effector of the robot. The X axis of the impact frame represents the direction of impact vector (direction of anticipated impact from the point of contact to the obstacle). If the orientation of the impact frame in the base frame is given by the rotation matrix $\boldsymbol{R_c}$, the required translational stiffness matrix of the controller in the base frame can be obtained using the relation

$$\mathbf{K}^{\mathbf{p}*}_{[tr]} = \boldsymbol{R_c}\mathbf{K}^{\mathbf{p}'}_{[tr]}\boldsymbol{R_c}^T \tag{6.13}$$

However, for rotational stiffness, compliance is required about axes *perpendicular* to the direction of impact. For the sake of example, assume the robot expects a collision along the X axis of a global frame where the operational space is defined. For translational stiffness, the robot needs low stiffness along the X axis so that any motion towards the obstacle is compliant. However, if the end-effector is rotating, it has to be compliant about Z and Y axes so that any rotation that moves the end-effector towards the obstacle is compliant, while rotation about X axis alone would not cause any collision. Therefore, the stiffness matrix for the rotational component has to be

$$\mathbf{K}^{\mathbf{p}'}_{[rot]} = \begin{bmatrix} K^{\mathbf{p}}_{[\theta]} & 0 & 0 \\ 0 & K^{\mathbf{p},low}_{[rot]} & 0 \\ 0 & 0 & K^{\mathbf{p},low}_{[rot]} \end{bmatrix} \tag{6.14}$$

and it can then be rotated using the same relation with the rotation matrix $\boldsymbol{R_c}$:

$$\mathbf{K}^{\mathbf{p}*}_{[rot]} = \boldsymbol{R_c}\mathbf{K}^{\mathbf{p}'}_{[rot]}\boldsymbol{R_c}^T \tag{6.15}$$

The final control stiffness to be used in the control equation (Eq. (6.7)) can be then obtained by diagonally block-stacking the two matrices as:

$$\mathbf{K}^{\mathbf{p}*} = \begin{bmatrix} \mathbf{K}^{\mathbf{p}*}_{[tr]} & \mathbf{0}_{[3,3]} \\ \mathbf{0}_{[3,3]} & \mathbf{K}^{\mathbf{p}*}_{[rot]} \end{bmatrix} \tag{6.16}$$

The corresponding damping matrix can also be obtained by either performing a similar rotation or using the critically-damped relation to compute the damping matrix corresponding to $\mathbf{K_p}^*$.

In the framework, the values of $K^{\mathbf{p},low}_{[tr]}$ and $K^{\mathbf{p},low}_{[rot]}$ are hyperparameters that are known to provide a desired compliant behaviour on contact, while the remaining parameter values (of $K^{\mathbf{p}}_{[x,y,z,\theta,\phi,\psi]}$) are are varied across trials such that they are inversely proportional to the certainty (or proportional to covariance) of the anticipated contact region. This way the robot is not highly stiff along other directions if it is uncertain about the predicted contact and is

compliant to reduce damage. As the prediction certainty increases, it is safe to assume that the robot can be stiff along directions in which it does not expect impact to act.

6.7 Experimental Analysis

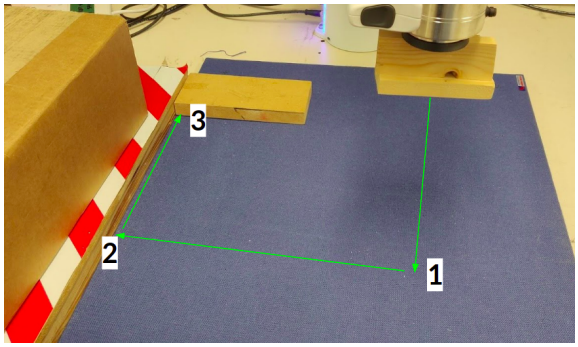


Fig. 6.4 A sliding task that involves making contacts with the table's surface at "1", with a wall at "2", and with another object at "3".

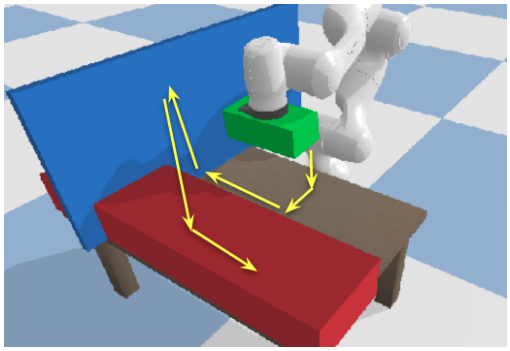


Fig. 6.5 A similar sliding task implemented in PyBullet for testing framework effectiveness in providing smooth motion.

In this section, the contact anticipation and handling module presented in this chapter is evaluated experimentally. We start by first testing the following hypotheses:

H6.1: *Our contact prediction approach incrementally improves the estimate of each contact's position over time, reducing task-completion delay and trajectory tracking error;*

H6.2: *The learned linear relationship between approach velocity and impact force provides an accurate estimate of the approach velocity for a desired impact force; and*

H6.3: *The overall framework produces smooth motion dynamics (i.e., velocity, acceleration etc.) for manipulation tasks with multiple contact changes.*

For experiments, we used a seven degrees of freedom (DoF) Franka Emika Panda robot operating on a tabletop (Fig. 6.4) and its simulated version in PyBullet (Fig. 6.5). The performance measures include accuracy (e.g., position tracking, impact force), task completion time, and the time spent in the transition-phase.

6.7.1 Contact Anticipation

To evaluate the ability to incrementally improve the estimate of contact position (**H6.1**), we used a task-space trajectory that required the robot to approach a (static) table from above,

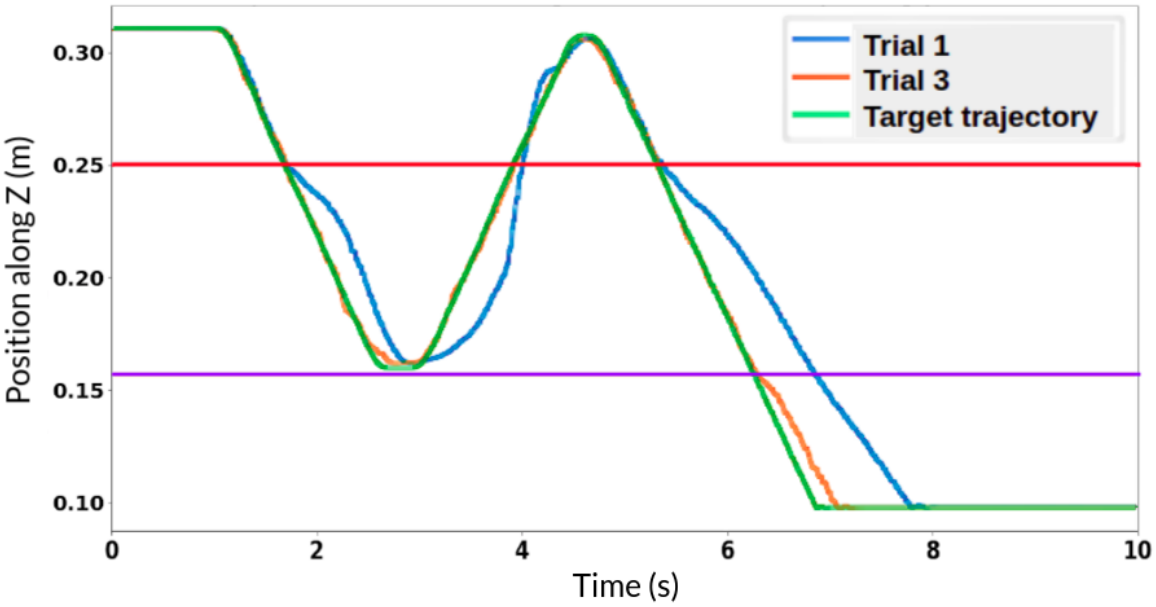


Fig. 6.6 Position of the end-effector (EE) during trial-1 and trial-3 of the contact prediction experiment. The red horizontal line is the edge of the covariance ellipsoid in trial 1; the violet line is the ellipsoid boundary in trial 3. Updated covariance in trial 3 enables the robot to avoid going to the transition-phase in the first dip of zig-zag trajectory and reduce the tracking error.

move back up without making contact with the table, and move down and make contact with the table, resulting in a zig-zag trajectory along the z-axis. As described earlier, the robot was expected to move with a lower velocity when approaching a contact point, but spend as little time as possible in this low-velocity, low-stiffness transition phase to reduce tracking error and delay in task completion.

An initial estimate of each contact position (based on target trajectory) was provided manually to simulate input from an external planner or vision system. Each initial estimate had a large covariance (0.175 along each dimension, with distance measured in meters) to simulate the uncertainty associated with a visual sensor or planner. Due to the large covariance, the region of anticipated contact ($\mathscr{C}$) overlapped with points in the first 'valley' of the target (zig-zag) trajectory although there was no actual contact with the table's surface. We expected the robot to obtain an improved estimate of $\mathscr{C}$ over time and not switch to the transition-phase controller in the first valley; the switch was only expected when the robot approached the table the second time. Given the focus on contact prediction, we empirically chose safe values for the transition-phase control parameters (i.e., approach velocity and stiffness).

We observed a significant reduction in covariance, e.g., from 0.175 to 0.07 in just three successive trials in an experiment, as summarized in Fig. 6.6, which enabled the robot

to avoid going to the transition-phase in the first dip in the trajectory. Also, the average Euclidean tracking error (per time step) in the position of the end-effector (EE) reduced from $1.3\,cm$ in the first trial to $0.16\,cm$ in the third trial, and the task completion time reduced from $7.9\,s$ in the first trial to $7.2\,s$ in the third trial; the expected (ground truth) motion duration is $7\,s$. Similar results were obtained with other target trajectories, indicating support for **H6.1**, i.e., that the uncertainty in the contact position is reduced quickly, which reduced delays in task completion as well as errors in trajectory tracking. These results also indicate that using the transition-phase controller only when it is required reduces the deviation from the desired motion trajectory.

6.7.2 Approach Velocity and Impact Force

To test the relation between approach velocity and impact force on contact, the robot was given a target motion trajectory that required it to move in free space and make contact with the table; this is also shown in the supplementary video. The task was repeated with different velocities ranging from $0.02\,m/s$ to $0.16\,m/s$ in steps of 0.02, each repeated four times, and we measured the corresponding force on contact. We observed that a line whose parameters were estimated by linear regression provided a reasonably good fit for the relationship between end-effector approach velocity and the end-effector force along the direction of motion, as shown in Fig. 6.7. The variance in the fit can be attributed largely to the noise in the force-torque sensor, which can be large during discontinuities such as collisions.

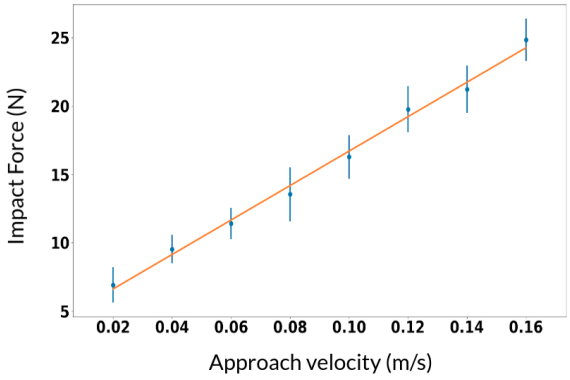


Fig. 6.7 Approach velocity vs force on impact. Orange line denotes the estimated linear relationship.

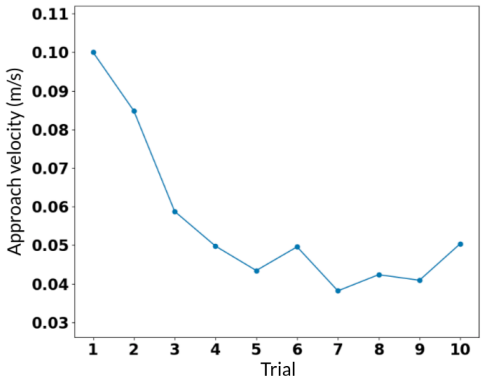


Fig. 6.8 Approach velocity over a sequence of trials to achieve a target impact force of 10 N using gradient descent.

Given such a learned relationship, the robot was asked to perform the same target trajectory (as above), but it had to now choose its approach velocity so as to achieve a desired impact force on contact. The measured contact force was compared with the desired impact

Target Force (N)	Estimated reqd velocity (m/s)	Measured force (N)	Error force (N)
10	0.047	7.4	2.6
12	0.063	15.1	3.1
15	0.086	15.3	0.3
18	0.11	16.7	1.3

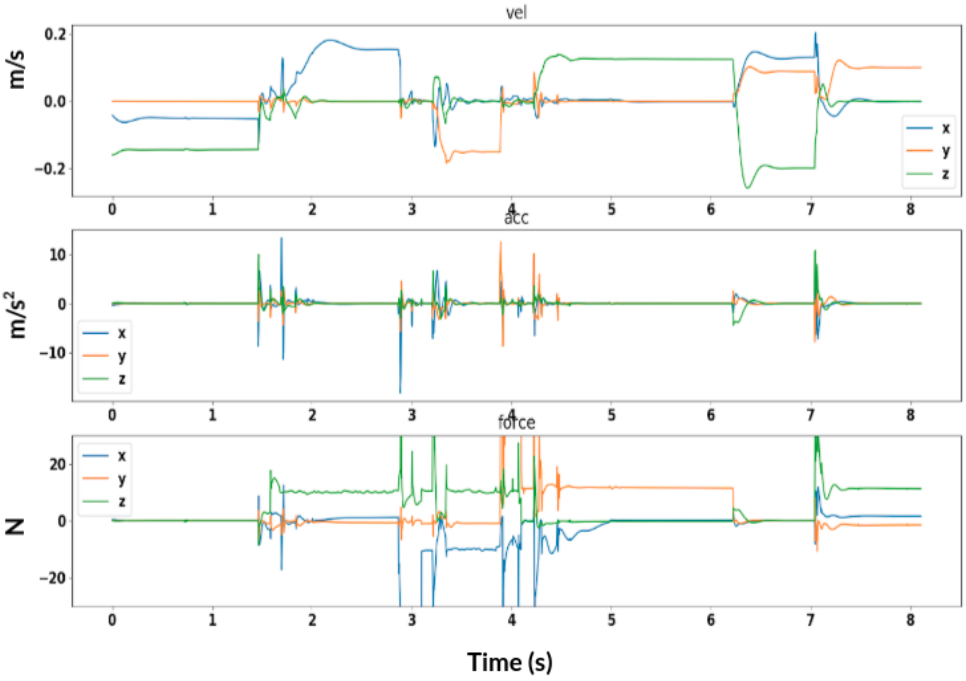
Table 6.2 Errors in contact force with the learned function specifying approach velocity as a linear function of the impact force; errors were higher at lower values of target force due to sensor noise.

force. Table 6.2 summarizes results for four trials for four of the 11 target force values we tested ($10-20$N at 1N increments). We observed that the robot was able to compute an approach velocity that resulted in an impact force similar to the desired value, with an error of ~ 3N. These errors were more pronounced at lower values of the target impact force, which can be attributed to sensor noise, i.e., the learned model was limited by the accuracy, sensitivity, and resolution of the force-torque sensor, joint encoders, and the robot's forward kinematics model.

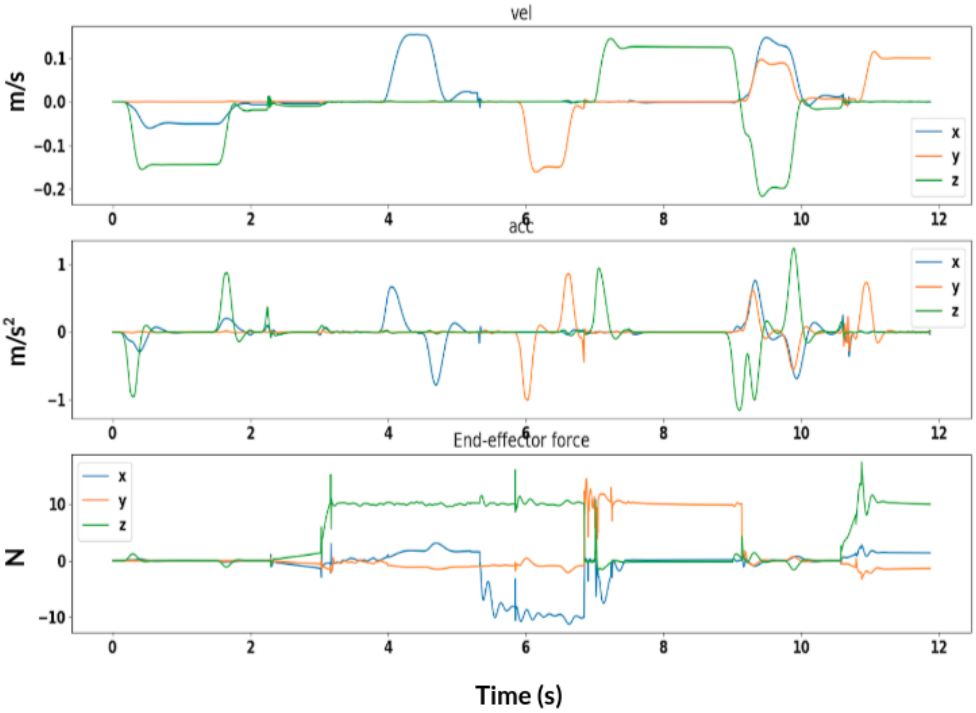
For the next experiment, Eq. (6.8) was used to incrementally update the approach velocity of the robot without providing the learned linear model. The initial value of the approach velocity was set to $0.1\,m/s$, the target impact force (F_d) was 10 N, and $\beta = 0.003$. Fig. 6.8 shows the evolution of approach velocity over 10 successive trials. We observed that the approach velocity was $\approx 0.045\,m/s$ in the fifth trial, after which the noise in the force torque sensor measurements at impact made it difficult to converge further to a single value of approach velocity. The error between the measured force and desired force reduced from $8.5\,N$ to $0.2\,N$ at the end of 10 trials. Similar results were obtained for other values of initial approach velocity and target impact force, indicating that in the absence of the learned linear model, the framework can still converge to a suitable approach velocity for a target impact force, although it requires more trials. These results thus support **H6.2**.

6.7.3 Smoothness of Motion

The motion profiles (e.g., velocity, acceleration profiles) of a changing-contact manipulation task are expected to have large spikes in the absence of our contact-change-handling module. This hypothesis (**H6.3**) was tested in a simulated environment. To test the effectiveness of the framework in providing an overall smooth dynamics and safer interaction, a baseline case was used for comparison in a simulator. In the baseline, the robot performs the task using the original task-space plan without predicting contacts or modifying velocities for lower impact effects.



(a) Without using contact-change-handling module



(b) When using the contact-change-handling module

Fig. 6.9 End-effector velocity, acceleration, and force in the simulated changing-contact task: (a) without using the contact-change-handling module; (b) when using the contact-change-handling module, the curves are smoother and the peaks are reduced.

The task (Fig. 6.5) involved the robot moving in free space first and then making contact with a table. The robot then proceeds to slide along the surface (force control along 1 direction) till it encounters a contact in the form of a rigid wall. It then slides along this wall (force control in 2 directions) to make contact with another wall. The robot then has to slide up this wall (force control along 1 direction) till it reaches the end of the wall. It then moves in free space towards another surface which it makes contact with and slides along. The task has multiple contacts and requires using force and motion controls along different directions in different segments of the plan. Following the original plan without contact anticipations or controller modifications (baseline) would cause the robot to experience high impact forces and discontinuities in the dynamics (see Fig. 6.9a). It can be seen that the motion produces large sharp peaks in acceleration and hence high jerks. The accelerations are as high as 18 m/s^2, while the peak impact forces are well over 50N which could be damaging for a real robot.

This was then compared with the robot performing the same task but by using the proposed framework for contact detection and smooth transition. It was provided with rough initial guesses of the contacts involved in the task mimicking values from a noisy vision sensor, with different noise uncertainty (covariance) for the different contacts involved. The robot was able to complete the task while producing significantly smoother motion in the first trial. The overall velocity and acceleration curves are smooth, even when the robot has multiple contacts. The peaks forces are significantly lower with the highest impact force being less than 8N and the peak acceleration on impact is less than $1m/s^2$, due to the lower velocity in the 'transition-phase'. The time to complete the task was however increased by about 3.7 seconds due to the modified velocity profiles of the plan. The framework therefore trades off task completion time for improving the overall smoothness of motion. By incrementally updating the knowledge about contact locations, however, the delay can be reduced. This is explored with a real robot in the next section.

6.7.4 Performance in task involving multiple collisions

To evaluate the overall framework and the resulting dynamics on a physical robot performing tasks involving multiple collisions, the robot (with a wooden block attached to end-effector) was asked to move vertically down to the table (contact 1), slide along y-axis (the table's surface) to a wall (contact 2), and slide along the wall (while in contact with the table's surface) to another obstacle (contact 3), as shown in Fig. 6.4. The robot was provided significantly incorrect initial guesses of the contact positions with substantial noise (see Table 6.3). The robot had to repeat the task while reducing the deviation from the given motion pattern by improving its estimate of the contact positions. The robot also had to

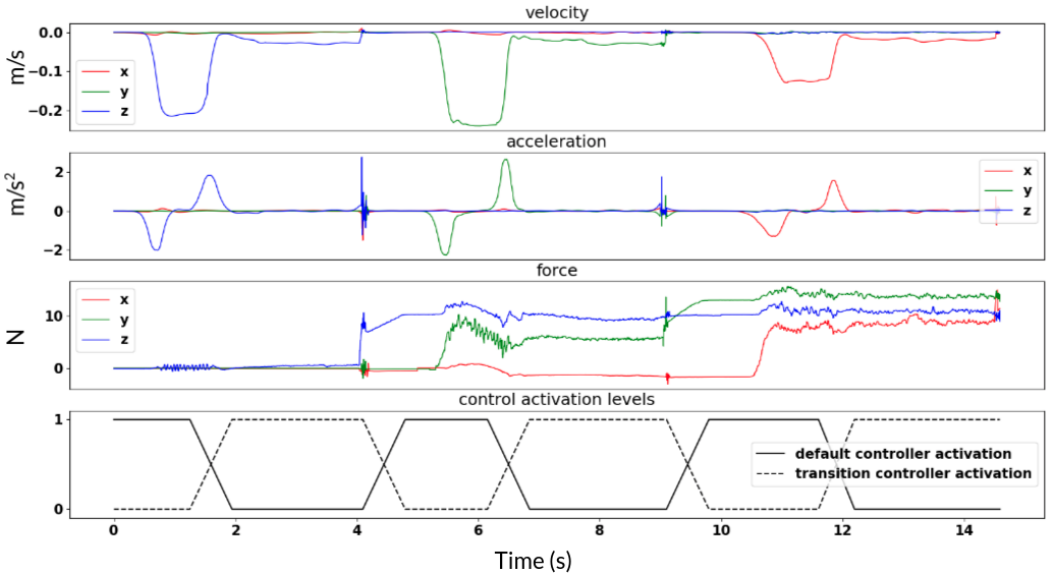
Prediction Error (m)	Initial	Final (trial 5)
Contact 1 (Z-axis)	0.12 ± 0.3	0.016 ± 0.039
Contact 2 (Y-axis)	0.09 ± 0.2	0.011 ± 0.04
Contact 3 (X-axis)	0.1 ± 0.2	0.018 ± 0.036

Table 6.3 Error in the estimated contact location along the most significant axis for the contact (in parenthesis) in the first and fifth trials of the task in Fig. 6.4. The value along the diagonal of the corresponding covariance matrix is shown as the standard deviation ($\pm$ term).

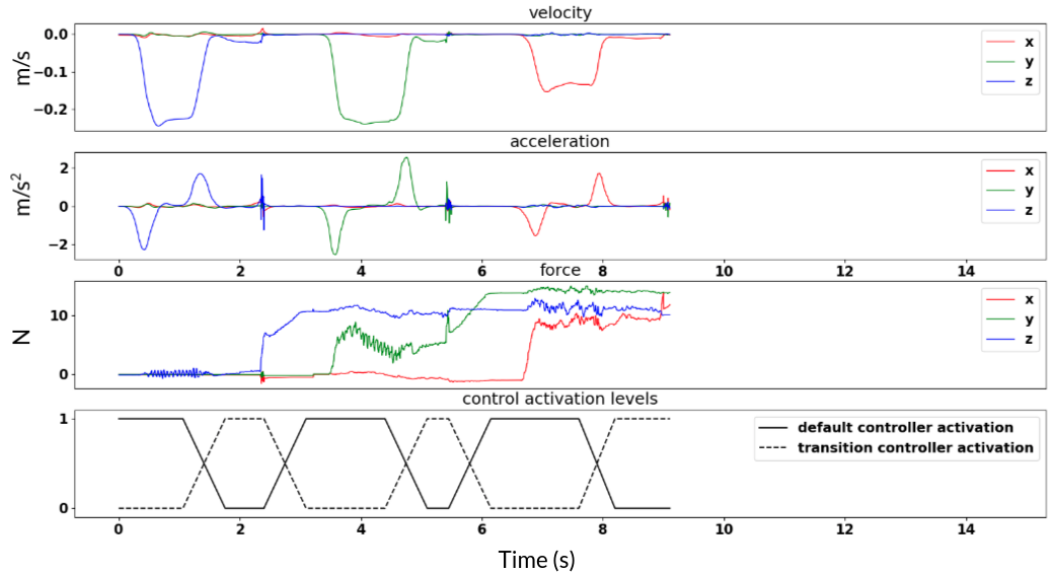
modify its approach velocity from the initial value of $0.05\,m/s$ to produce a desired impact force of 8 N. Since each contact in the task is in the presence of different environment dynamics (e.g., motion in free space, motion against surface friction), the velocity required to attain the desired impact force was expected to be different. The robot also had to incrementally update its approach velocity for each contact using gradient descent till the desired velocity for that environment was achieved. Furthermore, the robot had to perform all the trials with smooth overall motion dynamics with minimum spikes in the velocity or acceleration profiles.

Fig. 6.10a shows the velocity, acceleration, and EE force in the first trial, and Fig. 6.10b shows these values after five trials. The results in these figures and in Table 6.3 show that the uncertainty in the estimate of the contact positions is reduced, as indicated by a significant reduction in the size of the covariance ellipsoids, and the robot spends significantly less time using the transition-phase controller and the associated lower velocity. The last plot in Fig. 6.10a and Fig. 6.10b show the activation of the default controller and the transition-phase controller. The overall task was completed in 9.2 *s* in the fifth trial as opposed to 14.4 *s* in the first trial. The covariance ellipsoids converged in the first three trials of the task, but the task was repeated to evaluate the ability to compute and set the approach velocity for different transition-phase controllers.

With our framework, the robot converged to a suitable approach velocity for the first contact (from motion in free space) in five iterations. It was, however, difficult for the robot to adjust its approach velocities for contacts 2 and 3, which required the robot to use force control along one and two directions (respectively). Contact 3 was particularly challenging because it involved sliding along two different surfaces, resulting in very noisy readings from the force-torque sensor due to the different values of frictional resistance offered by the two surfaces. Since the impact force was along the same direction as friction, it was more difficult to isolate the impact force from the force due to surface friction, which made updating the approach velocity more challenging.



(a) Experiment trial 1.



(b) Experiment trial 5.

Fig. 6.10 Velocity, acceleration, force, and controller activation levels in: (a) experimental trial 1; (b) experimental trial 5. Use of our framework reduces uncertainty in estimates of contact positions, reduces the time spent using the transition-phase controller, and reduces discontinuities.

6.7.5 Framework effectiveness in tasks involving impact-less transitions

As explained previously discontinuities in interaction dynamics do not always occur due to collisions, and can also be due to discrete changes in the type of environment that the robot is in contact with. In this experiment, the objective was to test the effectiveness of the

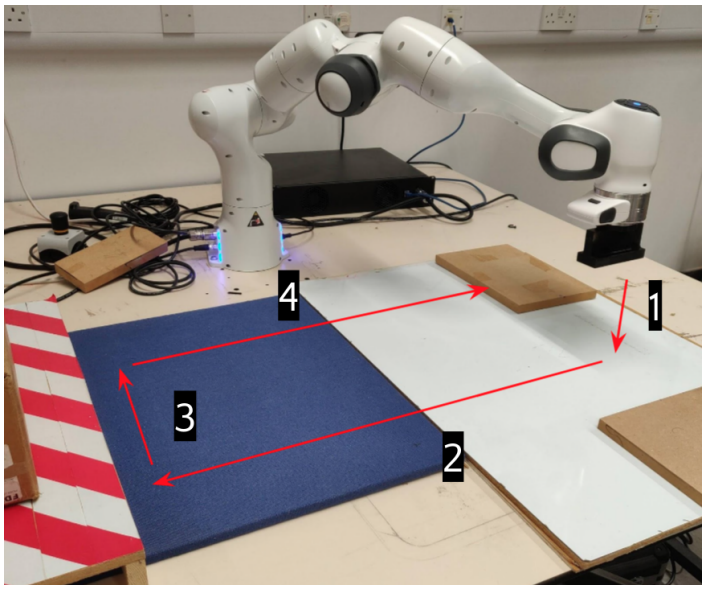


Fig. 6.11 A sliding task that involves both impacts and impact-less mode transitions. The robot experiences impact-less mode transitions as it performs motions '2' and '4' as the surfaces switch in between. These changes are not known to the robot beforehand, and has to be learned from experience.

framework in learning to predict contact changes that are not due to collisions (impact-less transitions).

The experiment involved the robot performing a similar changing-contact experiment as before, but this time, the surface along which the robot has to slide suddenly changes without the knowledge of the robot (see Fig. 6.11). The location of the surface switches are not known beforehand, and the robot has to learn to anticipate them in subsequent trials once it realises that there are previously unknown contact changes in the provided plan. The task involved the robot approaching a surface (surface A) from top, and sliding along the surface to make contact with a wall as before; however, the surface changes midway to one with higher friction (surface B) unknown to the robot. The robot then has to slide along the wall, and then move to another wall parallel to it, on the way to which the robot will slide surface B to A again. As in the previous experiment, the robot is provided with initial guesses of where *collisions* occur in the task, but it is not provided any knowledge about the changes in surfaces; this, the robot has to detect in its first trial and should then be able to predict and handle in subsequent trials.

The desired behaviour from the robot is that it learns to anticipate collisions as well as impact-less contact changes, and smoothly switches to an appropriate controller, that reduces the discontinuities in the motion dynamics during task execution.

In this experiment, we focus on the performance of the framework in the regions where the surfaces change, i.e., where there are impact-less mode transitions. This occurs at two

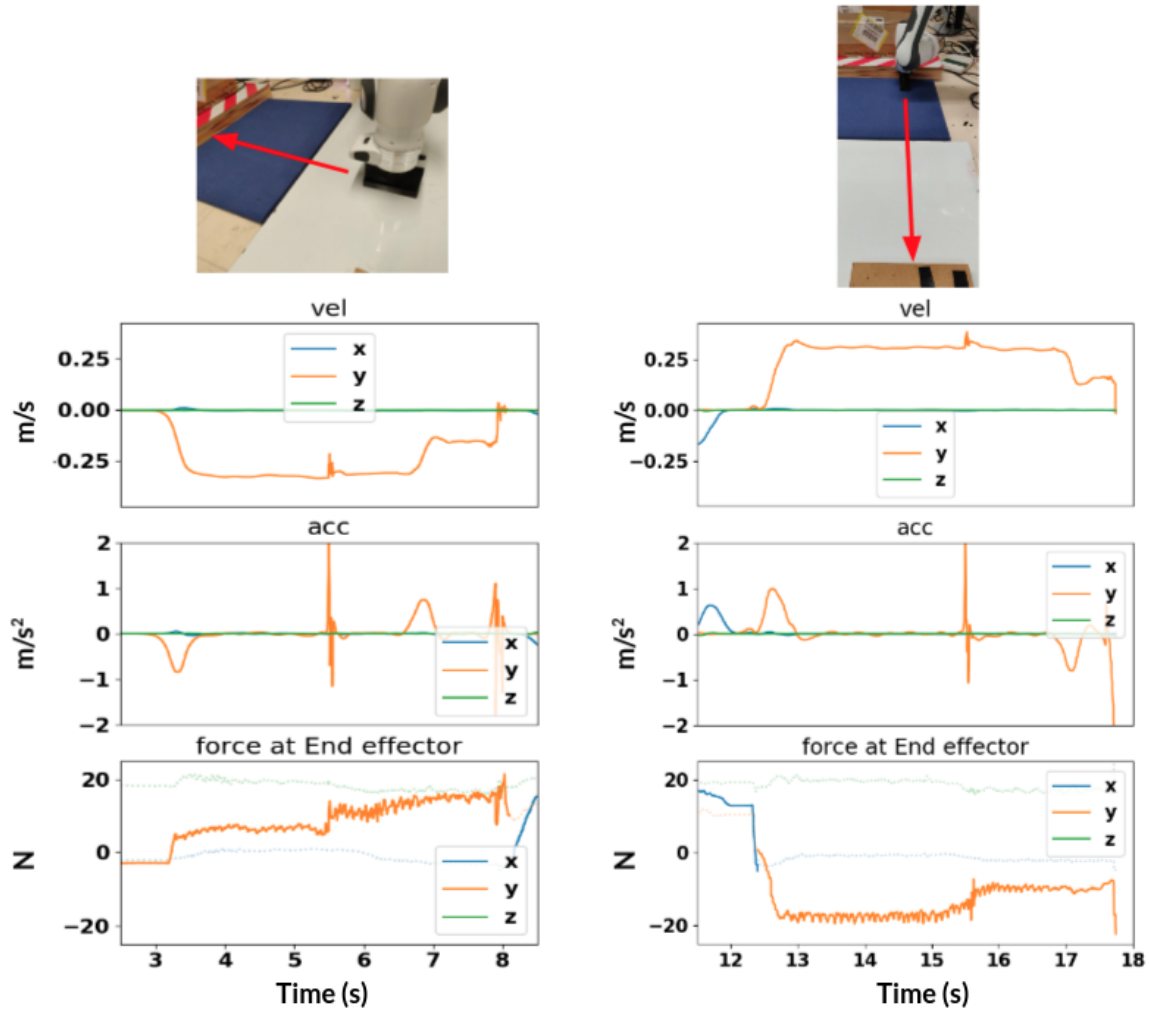
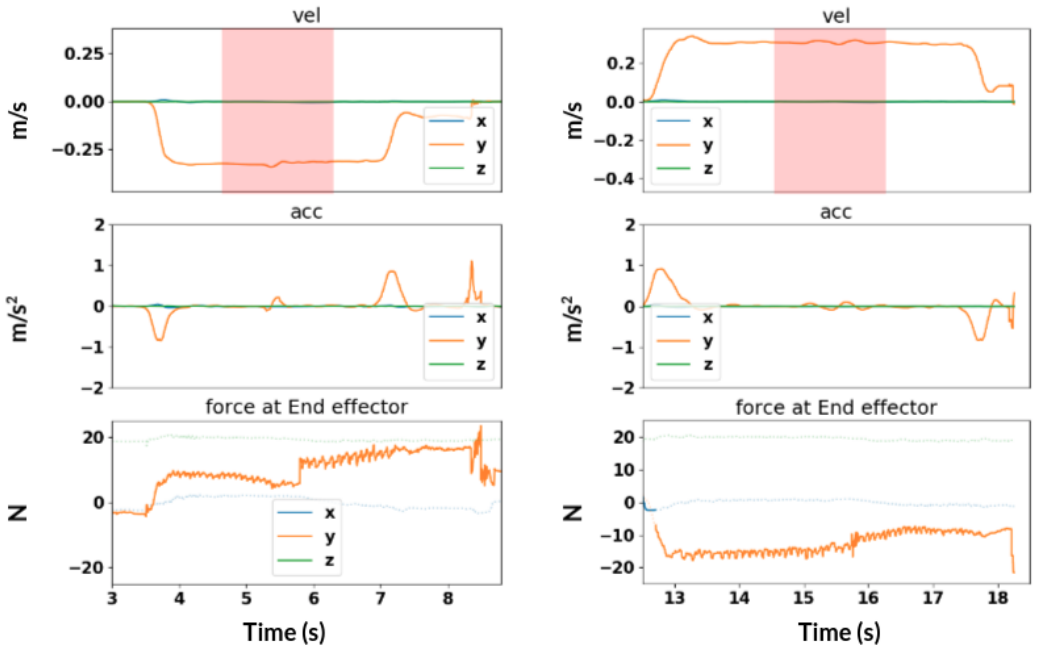
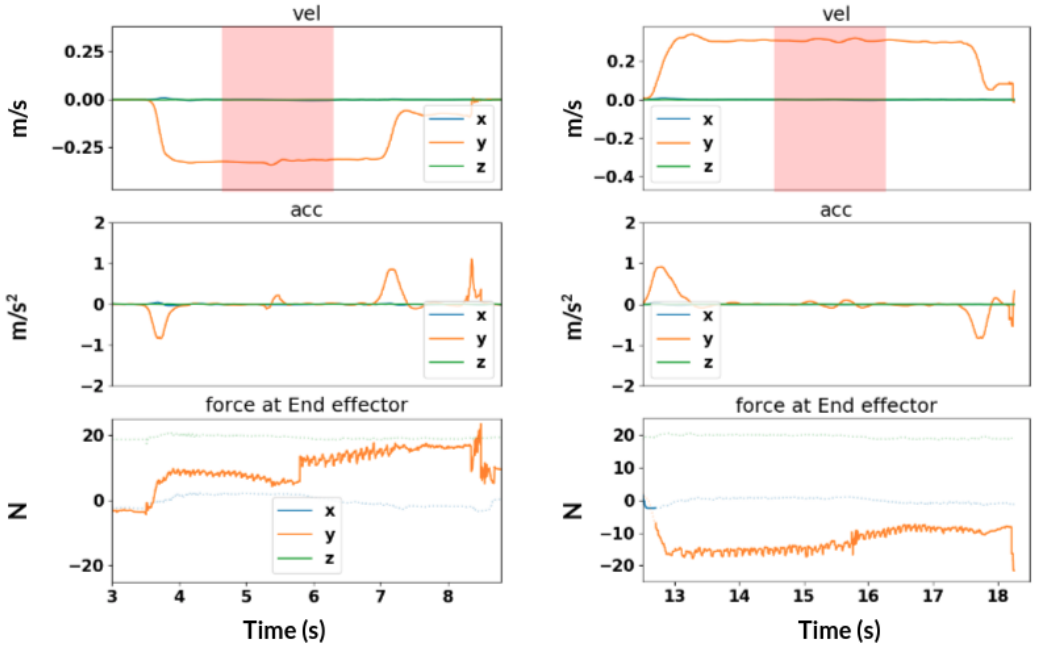


Fig. 6.12 Selected sections of the velocity (top), acceleration (middle), and force (bottom) curves in the first trial of the task; the robot is unaware of the impact-less transitions in the task beforehand. **Left:** Curves for the first impact-less transition (motion '2' in Fig. 6.11); **Right:** Curves for the second impact-less transition (motion '4' in Fig. 6.11)

sections in a trial of the task: Fig. 6.12 shows the velocity, acceleration, and end-effector force profiles in these regions measured during the first trial of the task. As expected, the robot experiences sudden spikes in velocity and acceleration (jerk) when the surface changes unexpectedly. Recall from previous chapter that this is due to the wrong predictions provided by the dynamics model which provides a feed-forward term for the controller that either overestimates or underestimates the environment forces in the new mode. When the robot detects such a discontinuity, it identifies this as a contact change and quickly switches to a high-stiffness controller to identify the new mode it is in. Seeing that this is a new surface which it hasn't seen before, the robot learns a new dynamics model for the mode which



(a) Experiment trial 2.



(b) Experiment trial 3.

Fig. 6.13 Selected sections of the velocity (top), acceleration (middle), and force (bottom) curves in trials 2 and 3. **Left:** Curves for the first impact-less transition (motion '2' in Fig. 6.11); **Right:** Curves for the second impact-less transition (motion '4' in Fig. 6.11)

it uses as a forward model for the AVIC in that mode. The surface of the table changes

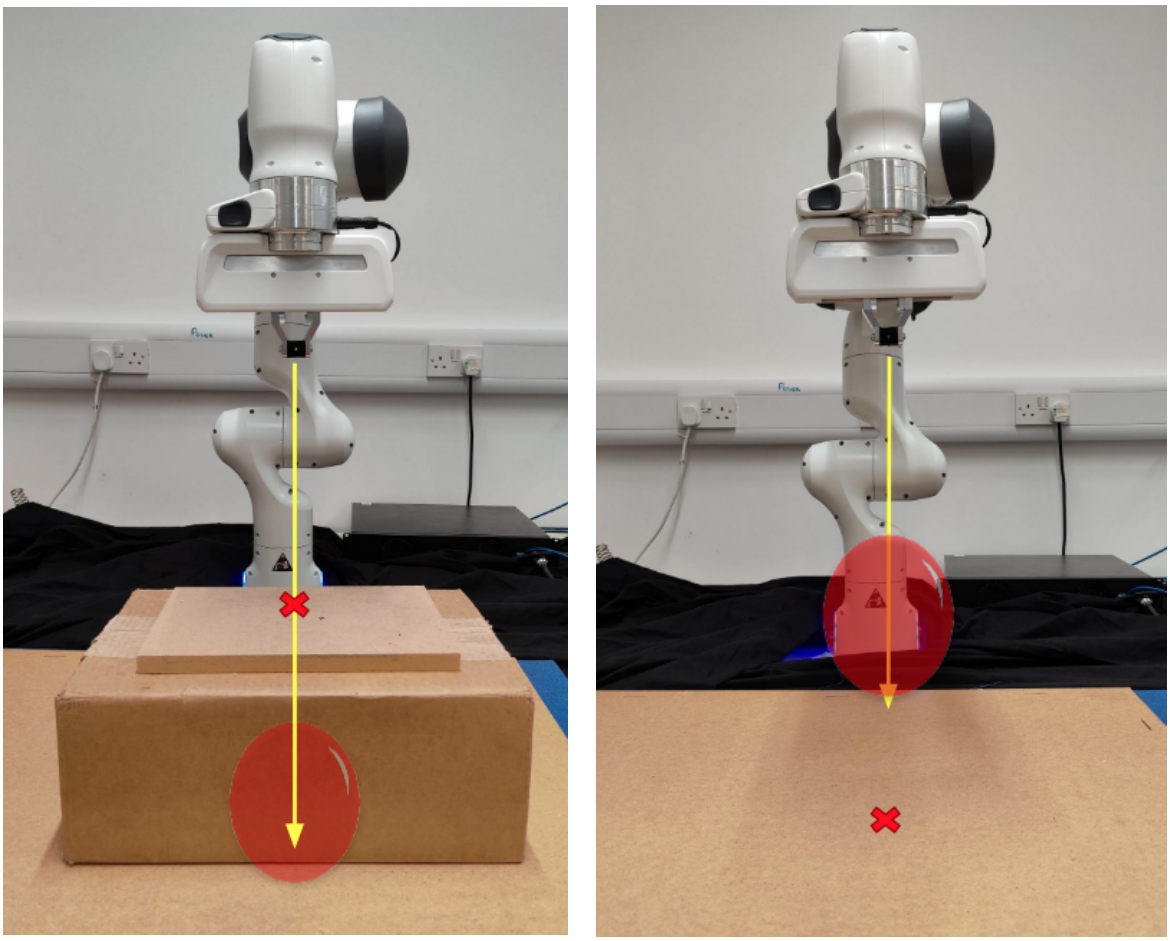
twice during the task; however, for the second surface change, the robot recognises the new dynamics mode as a surface that it has seen before (surface A).

In the next trial (Fig. 6.13a), the robot expects these mode switches (the anticipated region is marked in the figure), and because it anticipates an impact-less contact change, the robot switches to a high stiffness controller which would help the robot to identify the new mode quickly while also reducing sudden velocity-acceleration spikes during motion. As desired, the size of the anticipated region reduces in subsequent trials (Fig. 6.13b). However, the uncertainties (volume of the anticipated region $\mathscr{C}$) regarding the locations of these impact-less contact changes do not reduce as much as the ones for the anticipated collisions due to the higher noise parameter that is set (by the user) for impact-less transitions.

An interesting phenomenon that was observed during the experiment was that the robot using this *high-stiffness controller* is able to easily detect that a contact change has occurred when it slides from a surface of lower friction (A) to that of higher friction (B); however, it does not always detect that a mode change has occurred when sliding from B to A. This was understood to be because of the fact that when sliding a block from a surface of low friction to that of high friction, the sudden increase in friction at the region of the block in contact with the new surface offers the highest resistance, and contributes to the frictional resistance of the block. This change is sudden and pronounced (as seen in figure). On the other hand, when sliding from a high-friction surface to a low-friction surface, the trailing part of the block is in contact with the high friction surface which still offers resistance. This causes the robot to feel that the frictional resistance is being reduced gradually and not suddenly as in the other case. This lack of discontinuity in the sensed forces, and the absence of spikes in velocity due to the high-stiffness controller motivated the need to add another component to the contact-change-detection module of the framework: a threshold for the error between the measured end-effector force and the prediction from a (non-updating) forward model for that mode.

6.7.6 Handling collisions that are outside anticipated regions

This section tests the ability of the framework to adapt to situations where collisions occur outside the anticipated regions of contact. Such scenarios can occur, for instance, when the initial estimates of potential collisions are wrong. In case of initial estimates provided by visual sensors, this could be due to noise in sensors or due to occlusions in the line of sight of the camera. The Kalman filter-based update method allows for dealing with such cases as demonstrated in the below experiments.



(a) Collision occurs before anticipated region.

(b) Collision occurs after the anticipated region.

Fig. 6.14 Task setup used for testing the capability of the framework to handle collisions when they occur in regions outside the anticipated region $\mathscr{C}$. The red ellipse indicate $\mathscr{C}$ where the robot expects collision to occur according to the initial guess; the red cross denotes the location where the contact would actually occur; the yellow line is the simple path the robot was made to follow in the experiments.

Collision occurs before anticipated region

For the first experiment in this section, the robot was provided with an initial contact estimate that is wrong and lies beyond the actual contact (Fig. 6.14a). This scenario is equivalent to the robot having to deal with an unseen/unknown collision. The robot has to learn to make contact with the surface by ensuring smooth motion dynamics and low force on impact. As expected, in the first trial, the robot hits the surface with high impact force as it expects the contact to happen much later (Fig. 6.15). Since the robot experienced the collision outside the region of anticipated collision, it resets and re-initialises $\mathscr{C}$ with the new measurements of the end-effector position during impact with pre-defined default value for the covariance.

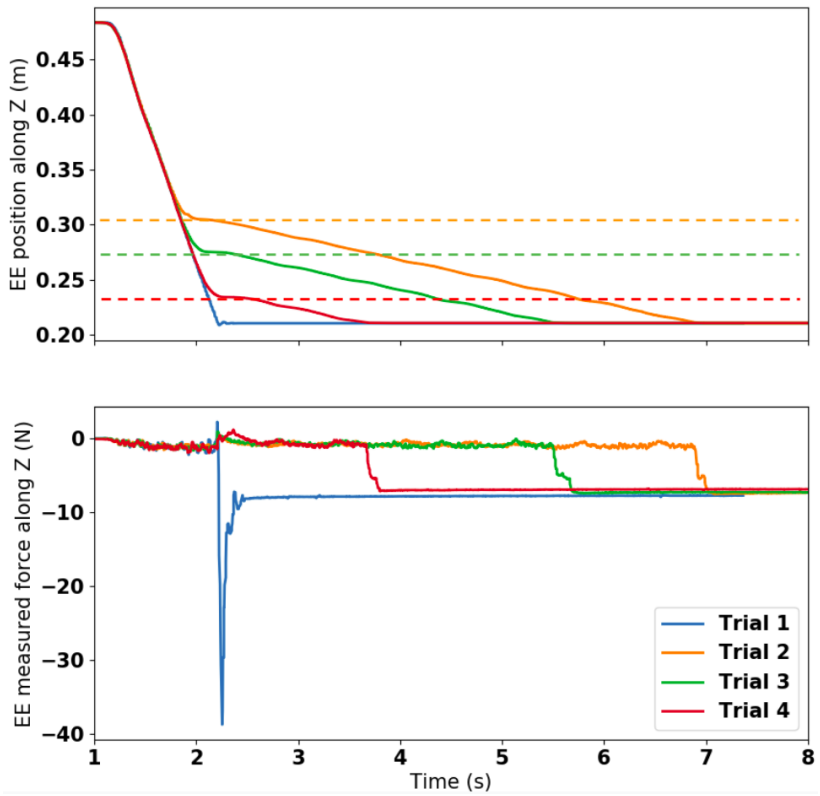


Fig. 6.15 End-effector position and force plots along the direction of motion during different trials of the task when the robot is not expecting a contact. Each colour represents a different trial. **Top:** End-effector position tracking in each trial; the dotted horizontal lines indicate the boundary of $\mathscr{C}$ in the corresponding trial. In the first trial (blue), the robot does not expect a contact in the region and hence the boundary is not seen in the figure. **Bottom:** End-effector force along impact direction. The first trial has highest impact force as it collides with an unexpected contact. In subsequent trials, the impact is reduced using the transition-phase controller. The delay in task completion is reduced in subsequent trials as the robot becomes more certain about the contact location and the size of $\mathscr{C}$ reduces.

The new estimate, however, enables to the robot to perform the task in the next trial with much lower impact force, although the task takes longer to complete due to the large $\mathscr{C}$. As in previous experiments, the robot can then reduce the uncertainty of $\mathscr{C}$ in subsequent trials, thereby reducing the delay in task while producing smooth motion and low impact force on collision (see Fig. 6.15). These results show that the robot can learn to handle unseen collision quickly in a few repeats of the task.

Collision occurs after anticipated region

In the next experiment, the robot is provided an initial estimate of $\mathscr{C}$ that lies before the actual collision (Fig. 6.14b). In such scenarios where the robot is expecting a contact and the contact

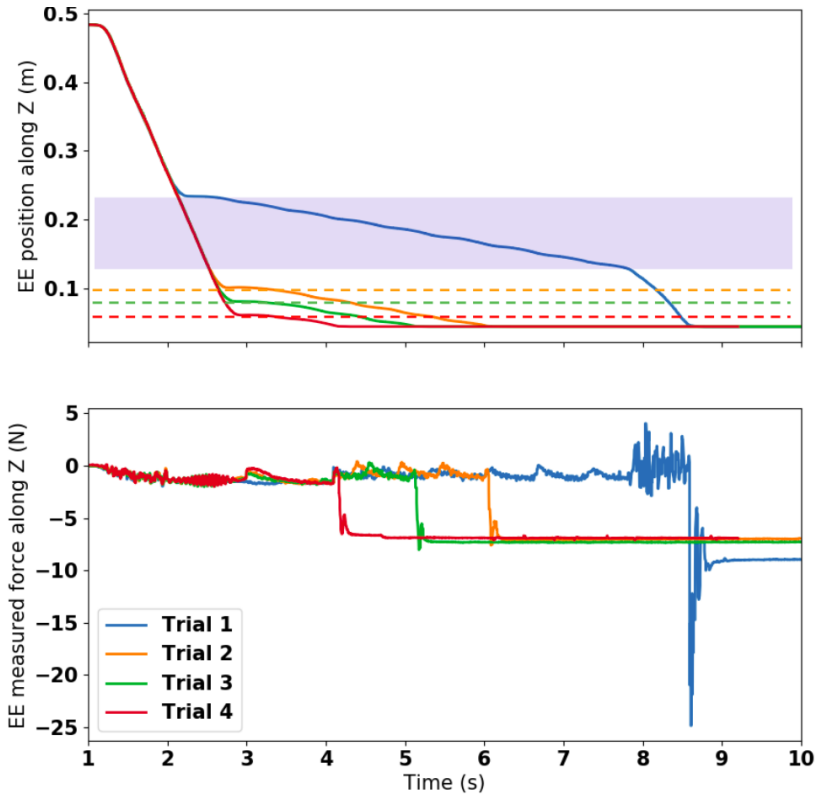


Fig. 6.16 End-effector position and force plots along the direction of motion during different trials of the task when $\mathscr{C}$ is before the actual contact. Each colour represents a different trial. **Top:** End-effector position tracking in each trial; the dotted horizontal lines indicate the boundary of $\mathscr{C}$ in the corresponding trial. The shaded (blue) region denotes the initial $\mathscr{C}$ where the robot expected the contact to occur. **Bottom:** End-effector force along impact direction. The first trial has highest impact force as it switches to a velocity-damped force controller to make contact when it does not find a contact in the anticipated region. In subsequent trials, the impact is reduced using the transition-phase controller in the newly estimated $\mathscr{C}$. The delay in task completion is also reduced in subsequent trials as the robot becomes more certain about the contact location and the size of $\mathscr{C}$ reduces.

never occurs, the controller in the framework is designed to switch to a velocity-damped force controller (Eq. (2.11)) till it makes contact with an object. The velocity damping allows the robot to approach a contact without uncontrolled acceleration as mentioned in Section 2.1.3. If the robot does not make contact after moving a predefined distance threshold, however, the robot stops the task and considers it failed, in which case the planner is asked to provide a new plan for the task.

This experiment, however, was designed such that the robot does make a contact, although it happens after the region where it anticipates a collision. Fig. 6.16 shows how the region of anticipated collision changes after each subsequent trials once the robot resets $\mathscr{C}$ at the end of its first trial. The impact force in the first trial is high as the robot is using force control

to make a contact after not having made a contact in the anticipated region. Even with the velocity damping, this is clearly not a safe impact. However, in the next trials, the robot uses the transition-phase controller within its new $\mathscr{C}$, thereby reducing the impact force and task completion time in subsequent trials.

This is conceptually similar to the approach taken by Steinmetz et al. [181] where their robot switches to a constant low-velocity controller till a contact is established, if an anticipated collision did not occur. However, unlike their approach, we try to reduce impact effects in subsequent trials by allowing the robot to reset and improve its estimate of contact location. We also use a velocity-damped force controller for a pre-defined distance after which the robot stops moving if does not experience a contact.

6.8 Summary and discussion

This chapter described a framework for addressing the discontinuities in changing-contact manipulation tasks that arise due to contact changes. The framework introduces a transition-phase controller in a hybrid force-motion variable impedance controller for continuous-contact tasks. Our representational choices enable us to simplify and address the associated challenges reliably and efficiently. Specifically, a Kalman filter formulation is adapted to incrementally improve the estimates of the contact positions. These estimates are used to minimise the time spent in the transition phase with appropriate transition-phase controllers. For controlling impact forces on collision, the velocity profile of corresponding transition-phase controllers is modified automatically to achieve smooth motion and a desired impact force.

The chapter began by describing the types of discontinuities that arise in a fixed-world changing-contact manipulation task, and by discussing their characteristics. We then proposed a novel Kalman filter-based incremental contact anticipation strategy that can improve the estimate of contact locations in a task in a few trials. Next, we introduced the concept of transition-phase controllers in the framework and the properties required for handling the two types of mode transitions. We also described our approach for automatically identifying the required approach velocity for reducing impact forces on collision. In the next sections, we described our strategy for ensuring smooth transition between the default variable impedance controller and the appropriate transition-phase controller such that the overall motion is smooth. Finally, we experimentally evaluated the proposed contact-change-handling module on a physical robot performing different changing-contact tasks involving collisions as well as impact-less mode transitions. We were able to demonstrate the effectiveness of the framework in reducing impact forces and vibrations on collision, and show the capability of the

framework to adapt to wrong contact estimates as well as its ability to reduce the uncertainty of contact locations in a few repeats of the task. Finally we showed the framework's ability to handle unexpected and missing contacts in a planned task.

This work opens up several directions for further research. First, we only focused on collisions due to translational motion, and did not address collisions due to rotations of the end-effector. This could be addressed by defining a region of anticipated collision in $\mathcal{SO}(3)$. Second, we observed that updating approach velocity for collisions when the robot is already in contact with another surface is more complicated. This is because of the difficulty in differentiating the sensor readings obtained due to reactive forces from the existing contact and the sensor readings obtained due to the impact force generated by the collision with another object. One possible way to address this issue could be to learn a better forward model for the contact mode such that it can accurately predict the forces due to the first contact. However, the problem is more likely to be more fundamental in terms of sensor accuracy and resolution at times of impact. Third, we only modified the velocity profile to achieve the desired smooth motion, and future work can further explore the relationship between stiffness values and the impact forces and vibrations. Initial experiments indicate that this is a challenging problem, as summarised in Appendix B.3. Moreover, reducing the stiffness during approach (to a contact position) makes the motion more sensitive to inertia, e.g., the velocity drops almost to zero before settling on the target approach velocity at time 1.5s in Fig. 6.10a. This behaviour is due to the lag in tracking the target trajectory and the uncompensated end-effector mass, which are due to the lower value of the stiffness used as the robot approaches a contact. We also acknowledge that the linear state transition formulation used for the Kalman filter could be oversimplifying the location of contacts, especially if the objects in the environment are non-static. This is also another potential direction for research to explore the possibility of extending this formulation for movable objects. Furthermore, we note that we have assumed all contact changes, specifically collisions, to be point contacts or simultaneous contacts that can be distinctly identified using a force-torque sensor at the wrist of the robot. However, this assumption may not hold in some real-world scenarios where collisions occur simultaneously or in quick succession, and often over a large area [60]. Despite these limitations, the proposed contact-change-handling module is a promising step towards developing a strategy for incrementally improving knowledge about contact changes and producing smooth motion overall with minimum delays and tracking errors in a few trials of the task.

Chapter 7

Conclusion

This research was motivated by the need for having a data-efficient framework for performing changing-contact manipulation tasks which can ensure the tracking accuracy as well as robot/object safely by handling continuous and discontinuous interaction dynamics, and reducing damaging effects of contact changes and collisions. This thesis primarily investigated the possibility of an incremental framework for handling continuous and discontinuous dynamics that occur in manipulation, which can quickly adapt to new and unseen environment dynamics, and deal with the large dynamics discontinuities that occur when contact changes occur, with minimum pre-training.

It was clear from reviewing literature and from our experiments that an adaptive variable impedance approach is required for efficiently navigating new and/or continuously changing environments. In Chapter 4, we presented Adaptive Variable Impedance Control (AVIC), an online framework for tracking a trajectory efficiently in continuously changing environments. We developed an incrementally updating feed-forward model for our variable impedance controller which could easily adapt to continuous changing environment dynamics and provide appropriate feed-forward term to compensate for the anticipated external disturbances. The feedback gains of the controller are also guided by the accuracy of the forward model, which ensures that the tracking accuracy is not compromised even if the forward model is unreliable. The representational choices made for our learning component enables the robot to incrementally learn to predict the end-effector forces and torques and adapt the model online and in real-time.

We experimentally demonstrated the convergence and trajectory-following capabilities of AVIC in different simulated continuous environments. We had previously published the results of conducting similar experiments on a physical robot in [124]. In the chapter, we also demonstrated the need for having an incrementally updating forward model for handling continuously changing environments by comparing our incremental forward model

with a fixed learned model. The AVIC framework was also experimentally compared with three other adaptive control strategies from literature in the context of trajectory-following in dynamically changing environments. The Model Reference Adaptive Control (MRAC) adaptive control strategy relies heavily on having a good reference model to navigate through any environment. This can become difficult for non-trivial and dynamically varying environments, as demonstrated in our experiments. Self-tuning adaptive controllers typically require fixed goal for the controller so as to converge to a good set of parameters and often does not converge in time for transient or time-varying goals. This makes such controllers unsuitable for the purposes of trajectory following. Gain-scheduling controllers learn a time-indexed sequence of controller parameters for following a trajectory in any given environment. However, such methods usually require large training data and relies on the environment to be the same during testing. On the other hand, our AVIC framework incrementally updates its forward model to capture the variations in the environment dynamics directly by modelling end-effector wrenches and is therefore able to navigate any continuously changing environment.

In Chapter 5, we extended AVIC for changing contact manipulation tasks by posing manipulation as a piecewise continuous hybrid system and learning the dynamics using a hybrid model. In this formulation we use a hierarchical model with a high-level mode-detection model which can incrementally learn to detect new contact modes, and low-level dynamics models which are the incremental forward models for AVIC corresponding to each identified contact mode. Using appropriate feature representations, we were able to demonstrate that the hybrid framework is able to successfully detect and model the interaction dynamics for different contact modes, as well as model previously unseen modes from scratch. We first showed that having separate models for distinct modes are better than having a single model or having to re-learn a model for the new mode from scratch. We then evaluated the framework in the context of a robot performing changing contact tasks where mode changes occur due to distinct environments (surfaces of different friction) or due to different types of contact the robot makes with an object (edge contact, surface contact, etc.). We then evaluated the need for having an incrementally updating hybrid framework and real-time feedback for model correction for learning piecewise-continuous systems by comparing it with a baseline fully offline long-term prediction model. We incrementally made changes to the baseline framework to motivate the choices made in the development of our hybrid framework.

In Chapter 6, we investigated a module for handling and reducing discontinuities in the interaction dynamics that happen when a contact change occurs. For instance, there will be large spikes in acceleration and forces at the end-effector when the robot collides with

an object in the environment, which could be damaging to the robot and/or the object. We proposed an incrementally improving Kalman filter-based contact anticipation module that iteratively updates the robot's knowledge about the contact locations in a changing-contact task by representing the regions of anticipated contact changes as ellipsoids in the task-space. This representation allows the robot to switch to appropriate 'safe' transition-phase controllers when it enters these regions so that the discontinuities during mode transitions are reduced. We described the two types of contact changes that could occur in a manipulation task in a static environment – impact transitions (collisions) and impact-less transitions – and discussed the ideal properties of a transition-phase controller for dealing with each. The chapter also proposed strategies to smoothly transition between controllers as the robot enters and leaves these regions of anticipated transitions such that the overall motion is smooth and jerk-free. The proposed transition-phase controllers for handling collisions also have the ability to modify their approach velocity automatically so as to achieve a desired force on impact. Together, the framework was shown to be able to handle collisions and impact-less mode transitions by sacrificing some *tracking accuracy in the form of delay in task completion*. The delays, however, reduce within a few trials of the task as the robot becomes more certain about the contact locations and hence spend less time in the transition-phase. The framework was tested for different changing-contact tasks that involved multiple collisions as well as impact-less transitions. We demonstrated that the robot is able to improve its knowledge regarding contact locations in a few trials and produce smooth overall motion by introducing minimum delay in task completion. Finally, we also experimentally evaluated the ability of the proposed contact-change-handling module to deal with contact-changes that occur outside the anticipated regions.

Most experiments in the thesis were conducted using a physical robot performing a manipulation task where the robot approached a surface and slid along it to make contact with other objects in the work space. This class of tasks was chosen as it is possible to quickly design new tasks which involves multiple collisions, impact-less transitions, multiple force-control directions, and have a piecewise continuous interaction dynamics by assuming that the friction of each surface is uniform across its face. These tasks can also be seen as representatives of different sub-tasks/segments of changing-contact tasks in industrial settings such as insertion and stacking, or tooling tasks such as grinding and polishing.

7.1 Challenges

The objective of the research was to capture the model of the hybrid dynamical system that is changing-contact manipulation. This, coupled with the need to have an adaptive and

incremental framework that requires little or no pre-training, introduced a lot of interesting and difficult challenges.

The AVIC framework presented in Chapter 4 relies heavily on the adaptive forward model used to update the parameters of the control law online. The incremental GMM algorithm (IGMM) used for building our forward models can become memory-intensive if the hyperparameters are not chosen correctly. The main hyperparameter that affects the performance of IGMM relates directly to the information stored in memory at a time. By choosing to keep more information in memory, the learning and prediction can slow down due to the need for inverting larger matrices (see Appendix A.1). On the other hand, not storing enough information in memory would result in the model 'forgetting' the data captured earlier in time. This means that the model will have to relearn the system dynamics if the robot encounters that part of the state space again. This could happen in some continuously varying environments such as the 'porridge environment' introduced in Chapter 4, where having shorter memory would mean the model would forget the model of the older dynamics as the viscosity of the environment increases. Choosing appropriate hyperparameters for the IGMM depending on the type of environment was critical to ensure that the incremental model could be used in real-time. This required running the dynamics learner on a separate machine while the controller and experiments were run from another computer in the same network. It also led to the development of the reduced one-dimensional magnitude representations that were used for learning instead of their vector formats.

As mentioned in our initial assumptions (Chapter 1), we assume that good dynamics models of the robot is available. This is a critical part of our controller formulation (Eq. (2.9)) to compensate for the robot dynamics during motion. However, for most of our experiments on the physical robot, the robot had to be attached with a custom end-effector representing a tool. To compensate for the additional load at the end-effector, the dynamics models of the robot had to be modified. Franka Emika provides a user interface for inserting the inertial information of the additional load into the model. However, this was not always reliable, especially when the attached load was heavy. So we had to use light-weight objects as end-effectors such as a wooden block (e.g. Fig. 5.1) or 3D printed objects (e.g. Fig. 6.11).

Another important issue was that the mode identification accuracy using our high-level mode-detection module presented in Chapter 5 depends very heavily on the choice of feature representation to distinguish between the different possible modes. This often required a more in-depth knowledge of the type of contact modes that the robot can encounter in a task. Since most tasks in this thesis involves different modes due to different surface friction, this did not require making too much change to the choice of feature representation. However, in the experiment where the robot had to differentiate between the different types

of contacts, a new feature representation that considers end-effector torques in addition to the original form of just forces, was required. Even with this formulation, it can become difficult to differentiate between contact modes if the robot applies different normal forces on the surface (see Fig. 5.15). Furthermore, it often becomes difficult to distinguish between contact modes if there are many possible modes in a task. In such scenarios, the distinction between modes are less evident and mode identification can take longer and sometimes even produce low-confidence results.

A critical portion of the challenges in the final part of this research can be attributed to the highly non-linear nature of the interaction dynamics when contact-changes occur. At these points, the force-torque sensors tend to be unreliable, especially if there are other forces already acting at the end-effector. This was noticed during tasks where the robot slides on multiple surfaces before colliding with another object; the force-torque measurements tend to be extremely noisy in such scenarios and the framework often mistook a measurement as a mode transition even when the actual transition had not happened. Therefore, the force-torque sensor readings had to be filtered and smoothed using a low-pass-filter to remove the noise. The Franka robot comes with in-built low-pass filtering of the end-effector force-torque estimates, which was used for most experiments. However, this often meant the measured impact forces are not the actual impact force that the robot felt (see Appendix B.3).

7.2 Framework limitations and future research

The presented framework is a step forward to developing an incrementally learning robot framework for performing changing-contact manipulation tasks efficiently. However, there are several parts in the framework that has to be improved before this can be deployed on a real-world system reliably.

As mentinoed previously, the AVIC framework heavily relies on having a good robot dynamics model and a reliable FT sensor at the robot end-effector to model the end-effector forces. It also becomes important the the FT sensor is located at a good point in the kinematic chain of the robot such that it measures forces and torques in the space where the task-space control command is computed, or a good definition of the transformation to this space is required. It is also important to have a properly designed control loop with an independent learning thread for modelling the interaction dynamics in real-time. This could be challenging requirements for very low-spec workstations.

Although the IGMM hyperparameter tuning is intuitive, it often has to be tested a few times before converging to a good set of parameters for forward models in a particular task.

The gradual 'unlearning' of the forward model mentioned in the previous section can also cause difficulties in selecting a reasonable set of hyperparameters.

Similarly, the control law in AVIC (Eq. (2.9)) relies on a forward model and feedback from the FT sensor. This means that in the absence of a good forward model, the controller reduces to a constant high-stiffness PD controller with dynamics compensation (impedance controller). This does not necessarily result in task failure, but does not achieve the objective of completing the task with low stiffness and energy.

As discussed above, the mode detection model presented in Chapter 5 requires good understanding of the contact modes that could appear in a task so as to design a good feature representation for the mode-detection module. The choice of the state space is crucial for being able to distinguish between modes, and would require good understanding of the difference in the observable values for the robot when it is in different contact modes. Another interesting challenge is to decide the resolution at which the distinction between modes is to be made. For instance, it may not be worth distinguishing between two surfaces whose friction are very similar; in such cases, it may be enough for the robot to use the same dynamics model from a similar surface and proceed with improving and using the model in AVIC to navigate the other surface. The level of resolution between modes is also a hyperparameter that the designer has to choose, which is dictated by the minimum 'distance' needed between the features to classify them as different modes.

Another important assumption made during the development of the hybrid framework is that manipulation tasks are piecewise-continuous. Although this is a fair assumption in a broad sense, it may not always be easy to separate out the contact modes involved in a task. For instance, even in a polishing task where the robot slides over the same surface, there could be several regions where the friction is not uniform due to surface irregularities; these could potentially trigger the mode-detection module in our framework. Similarly many manipulation task would involve the end-effector making contact with multiple points in the environment at the same time or in quick succession, which can often result in complicating the notion of different contact modes. Another critical assumption that is implicit for formulating the interaction dynamics as piecewise continuous is that the object at the end-effector is rigidly grasped by the robot. The interaction dynamics can be affected significantly if the held object can move even slightly. The measurements by an FT sensor at the wrist of the robot would be affected by the motion of a loose end-effector.

The contact anticipation module presented in Chapter 6 uses a simplified linear model to represent the state transition of contact locations for formulating the Kalman filter. This may be an oversimplification which may not generalise to movable objects, whose motion dynamics may not be linear. This should generalise to simple sliding objects, but it could

get much more complicated for objects that can roll or those that have irregular shapes. This assumption of linearity, however, is the main reason that the approach can converge successfully to the actual contact location in a few trials. Also, by setting a constant noise in the sensor model, we assume that the noise in the kinematics model of the robot is a constant. However, this would in reality be much more complex and would depend on the noise in the joint encoder as well. Any noise in the joint sensor would be mapped to the task-space non-linearly which itself could break the linearity assumption made by the Kalman filter. Similarly, another important limitation of this formulation is that the ellipsoid representation used to denote the region of anticipated mode transitions is applicable only because the trajectory followed by the robot is repeatable and is performed in a static (rigidly fixed) world.

The transition-phase controllers used for approaching a collision does not update its stiffness automatically, although we have some evidence that having lower stiffness helps reduce jerk on contact. It relies on the designer providing a 'safe' approach stiffness that would produce low vibrations while being stiff enough to compensate for end-effector inertia. Further research could explore strategies to allow the agent to intelligently learn to adapt the stiffness to minimise vibrations without sacrificing tracking accuracy too much. The complete contact-change-handling module presented in Chapter 6 ignore all contact changes that occur due to end-effector rotations. The C^∞ smooth velocity described in Section 6.5.2 is also only applicable for translational motion. The rotational velocity was linearly interpolated using SLERP (spherical linear interpolation) [169] in our experiments, which generally will not align with the time-remapping of the translational segment of the trajectory. It would be interesting to explore the velocity interpolation using a similar C^∞ form.

As mentioned previously, the overall framework relies on having good dynamics model of the robot. The control law used also implicitly assumes that the end-effector of the robot is included in the dynamics model. This would typically require incorporating the end-effector's dynamics parameters in the overall dynamics model of the manipulator system, which could be cumbersome especially if the end-effector is variable across trials of the task. Although our framework can account for this implicitly by automatically learning to increase the controller stiffness when the forward model accuracy is low (using higher stiffness can partially compensate for dynamics imbalance in the system), the effect of uncompensated end-effector dynamics can affect the performance when the robot is forced to use low stiffness (see discussion at the end of Chapter 6). This part of the framework could be improved by incorporating some system identification routine in the beginning of a trial to estimate the model of the end-effector dynamics.

Having the same end-effector across trials is also critical for the Kalman filter-based contact anticipation model; the model relies on being able improve the estimate of the contact location in the form of end-effector position during impact, and this makes the critical assumption that the end-effector is the same across trials and rigidly fixed to the robot.

An interesting (and challenging) problem that was not explored in this thesis is the importance of having a regular-shaped object as the end-effector for the effectiveness of the presented framework. The mode detection module relies heavily on the interaction dynamics to be affected only by the changes in environment or due to change in contact with the environment, and does not account for having irregularly shaped objects at the end-effector. It is possible that the overall framework would require a more generalisable formulation to account for irregular objects at the end-effector such as objects that do not have flat surfaces.

The overall framework also assumes that all collisions occur only at the end-effector of the robot and not at any other points on the robot's body. This could be a limiting assumption especially because the robot expects a task-space trajectory from the planner to follow. Joint-space plans could account for obstacles in the joint-space of the robot by adding them as constraints for the planner, which is not possible when generating a task-space plan. A possible solution to account for this problem is by using the joint-space plan as the secondary goal in the control law (see note on 'null-space control' in Section 2.1.4) while following the task-space plan as the primary goal; this way the robot would try to follow the joint-space plan (which avoids the obstacles in the joint space) as long as it does not impede the objective of achieving the primary goal (following the task-space trajectory).

Include notes on "being stiff when uncertain is not always good"

References

[1] (2009). *Handbook of Hybrid Systems Control: Theory, Tools, Applications*. Cambridge University Press.

[2] Abu-Dakka, F. J., Rozo, L., and Caldwell, D. G. (2018). Force-based variable impedance learning for robotic manipulation. *Robotics and Autonomous Systems*, 109:156–167.

[3] Abu-Dakka, F. J. and Saveriano, M. (2020). Variable impedance control and learning—a review. *Frontiers in Robotics and AI*, 7:177.

[4] Ahmad, W. (2006). Incremental learning of gaussian mixture models.

[5] Ahn, K., Chung, W. K., and Yourn, Y. (2004). Arbitrary states polynomial-like trajectory (aspot) generation. In *30th Annual Conference of IEEE Industrial Electronics Society, 2004. IECON 2004*, volume 1, pages 123–128. IEEE.

[6] Ajay, A., Bauza, M., Wu, J., Fazeli, N., Tenenbaum, J. B., Rodriguez, A., and Kaelbling, L. P. (2019). Combining physical simulators and object-based networks for control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3217–3223. IEEE.

[7] Ajoudani, A., Hocaoglu, E., Altobelli, A., Rossi, M., Battaglia, E., Tsagarakis, N., and Bicchi, A. (2016). Reflex control of the pisa/iit softhand during object slippage. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1972–1979. IEEE.

[8] Ajoudani, A., Tsagarakis, N., and Bicchi, A. (2012). Tele-impedance: Teleoperation with impedance regulation using a body–machine interface. *The International Journal of Robotics Research*, 31(13):1642–1656.

[9] Akaike, H. (1998). Autoregressive model fitting for control. In *Selected Papers of Hirotugu Akaike*, pages 153–170. Springer.

[10] Alexis, K., Nikolakopoulos, G., and Tzes, A. (2012). Model predictive quadrotor control: attitude, altitude and position experimental studies. *IET Control Theory & Applications*, 6(12):1812–1827.

[11] Alexis, K., Papachristos, C., Siegwart, R., and Tzes, A. (2016). Robust model predictive flight control of unmanned rotorcrafts. *Journal of Intelligent & Robotic Systems*, 81(3-4):443–469.

[12] Andrychowicz, M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al. (2018). Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*.

[13] Arruda, E., Mathew, M. J., Kopicki, M., Mistry, M., Azad, M., and Wyatt, J. L. (2017). Uncertainty averse pushing with model predictive path integral control. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 497–502.

[14] Aström, K. (2014). *History of Adaptive Control*, pages 1–9. Springer London, London.

[15] Baraff, D. (1991). Coping with friction for non-penetrating rigid body simulation. *ACM SIGGRAPH computer graphics*, 25(4):31–41.

[16] Barragän, P. R., Kaelbling, L. P., and Lozano-Pérez, T. (2014). Interactive bayesian identification of kinematic mechanisms. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2013–2020. IEEE.

[17] Beetz, M., Stulp, F., Esden-Tempski, P., Fedrizzi, A., Klank, U., Kresse, I., Maldonado, A., and Ruiz, F. (2010). Generality and legibility in mobile manipulation. *Autonomous Robots*, 28(1):21.

[18] Bender, J. and Schmitt, A. (2006). Constraint-based collision and contact handling using impulses. In *Proceedings of the 19th international conference on computer animation and social agents*, pages 3–11.

[19] Biagiotti, L. and Melchiorri, C. (2008). *Trajectory planning for automatic machines and robots*. Springer Science & Business Media.

[20] Brogliato, B. (2019). *Nonsmooth Mechanics. Models, Dynamics and Control: Erratum/Addendum*. PhD thesis, INRIA Grenoble-Rhone-Alpes.

[21] Buchli, J., Stulp, F., Theodorou, E., and Schaal, S. (2011). Learning variable impedance control. *IJRR-2011*, 30(7):820–833.

[22] Burdet, E., Osu, R., Franklin, D. W., Milner, T. E., and Kawato, M. (2001). The Central Nervous System Stabilizes Unstable Dynamics by Learning Optimal Impedance. *Nature*, 414(6862):446.

[23] Buşoniu, L., de Bruin, T., Tolić, D., Kober, J., and Palunko, I. (2018). Reinforcement learning for control: Performance, stability, and deep approximators. *Annual Reviews in Control*, 46:8–28.

[24] Caccavale, F., Chiacchio, P., Marino, A., and Villani, L. (2008). Six-dof impedance control of dual-arm cooperative manipulators. *IEEE/ASME Transactions On Mechatronics*, 13(5):576–586.

[25] Calinon, S., Sardellitti, I., and Caldwell, D. G. (2010). Learning-based control strategy for safe human-robot interaction exploiting task and robot redundancies. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 249–254. IEEE.

[26] Cassandras, C. G. and Lygeros, J. (2018). *Stochastic hybrid systems*. CRC Press.

[27] Catto, E. (2017). Box2d. http://box2d.org.

[28] Chatterjee, A. (1997). *Rigid body collisions: some general considerations, new collision laws, and some experimental data*. Cornell University.

[29] Chatterjee, A. (1999). On the realism of complementarity conditions in rigid body collisions. *Nonlinear Dynamics*, 20(2):159–168.

[30] Cheah, C.-C. and Wang, D. (1998). Learning impedance control for robotic manipulators. *IEEE Transactions on robotics and automation*, 14(3):452–465.

[31] Chiaverini, S. and Sciavicco, L. (1993). The parallel approach to force/position control of robotic manipulators. *IEEE Transactions on Robotics and Automation*, 9(4):361–373.

[32] Choudhury, S., Hou, Y., Lee, G., and Srinivasa, S. S. (2017). Hybrid ddp in clutter (chddp): trajectory optimization for hybrid dynamical system in cluttered environments. *arXiv preprint arXiv:1710.05231*.

[33] Coumans, E. and Bai, Y. (2016–2021). Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org.

[34] De Schutter, B., Heemels, W., Lunze, J., Prieur, C., et al. (2009). Survey of modeling, analysis, and control of hybrid systems. *Handbook of Hybrid Systems Control–Theory, Tools, Applications*, pages 31–55.

[35] De Wit, C. C., Olsson, H., Astrom, K. J., and Lischinsky, P. (1995). A new model for control of systems with friction. *IEEE Transactions on automatic control*, 40(3):419–425.

[36] Dearden, A. and Demiris, Y. (2005). Learning forward models for robots. In *IJCAI*, volume 5, page 1440.

[37] Demiris, Y. (2007). Prediction of intent in robotics and multi-agent systems. *Cognitive processing*, 8(3):151–158.

[38] Deniša, M., Gams, A., Ude, A., and Petrič, T. (2015). Learning compliant movement primitives through demonstration and statistical generalization. *IEEE/ASME Transactions on Mechatronics*, 21(5):2581–2594.

[39] Dorato, P. (1987). A historical review of robust control. *IEEE Control Systems Magazine*, 7(2):44–47.

[40] Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016). Benchmarking deep reinforcement learning for continuous control. In *International conference on machine learning*, pages 1329–1338. PMLR.

[41] Ekaputri, C. and Syaichu-Rohman, A. (2013). Model predictive control (mpc) design and implementation using algorithm-3 on board spartan 6 fpga sp605 evaluation kit. In *2013 3rd International Conference on Instrumentation Control and Automation (ICA)*, pages 115–120. IEEE.

[42] Engel, P. and Heinen, M. (2010). Incremental learning of multivariate gaussian mixture models. In *BSAI*. Springer.

[43] Fabisch, A. (2019). A comparison of policy search in joint space and cartesian space for refinement of skills. In *International Conference on Robotics in Alpe-Adria Danube Region*, pages 301–309. Springer.

[44] Fan, Y., Gao, W., Chen, W., and Tomizuka, M. (2017). Real-time finger gaits planning for dexterous manipulation. *IFAC-2017*, 50(1):12765–12772.

[45] Fazeli, N., Zapolsky, S., Drumwright, E., and Rodriguez, A. (2020). Fundamental limitations in performance and interpretability of common planar rigid-body contact models. In *Robotics Research*, pages 555–571. Springer.

[46] Ferraguti, F., Secchi, C., and Fantuzzi, C. (2013). A tank-based approach to impedance control with variable stiffness. In *2013 IEEE international conference on robotics and automation*, pages 4948–4953. IEEE.

[47] Finn, C. and Levine, S. (2017). Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793. IEEE.

[48] Flanagan, J. R., Bowman, M. C., and Johansson, R. S. (2006). Control Strategies in Object Manipulation Tasks. *Current Opinion in Neurobiology*, 16(6):650–659.

[49] Flanagan, J. R., Vetter, P., Johansson, R. S., and Wolpert, D. M. (2003). Prediction precedes control in motor learning. *Current Biology*, 13(2):146–150.

[50] Franklin, D. W., Burdet, E., Tee, K. P., Osu, R., Chew, C.-M., Milner, T. E., and Kawato, M. (2008). CNS Learns Stable, Accurate, and Efficient Movements using A Simple Algorithm. *Journal of Neuroscience*, 28(44):11165–11173.

[51] Freeman, P. (2012). Minimum jerk trajectory planning for trajectory constrained redundant robots.

[52] Gams, A., Nemec, B., Ijspeert, A. J., and Ude, A. (2014). Coupling movement primitives: Interaction with the environment and bimanual tasks. *IEEE Transactions on Robotics*, 30(4):816–830.

[53] Gandhi, M., Pan, Y., and Theodorou, E. (2017). Pseudospectral model predictive control under partially learned dynamics. *arXiv preprint arXiv:1702.04800*.

[54] Ganesh, G., Jarrassé, N., Haddadin, S., Albu-Schaeffer, A., and Burdet, E. (2012). A versatile biomimetic controller for contact tooling and haptic exploration. In *2012 IEEE International Conference on Robotics and Automation*, pages 3329–3334. IEEE.

[55] Gawthrop, P. (1980). On the stability and convergence of a self-tuning controller. *International Journal of Control*, 31(5):973–998.

[56] Gold, T., Völz, A., and Graichen, K. (2020). Model predictive position and force trajectory tracking control for robot-environment interaction. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7397–7402. IEEE.

[57] Gomi, H. and Osu, R. (1998). Task-dependent viscoelasticity of human multijoint arm and its spatial characteristics for interaction with environments. *Journal of neuroscience*, 18(21):8965–8978.

[58] Gopal, M. (1984). *Modern Control System Theory*. Halsted Press, USA.

[59] Grassmann, R., Johannsmeier, L., and Haddadin, S. (2018). Smooth point-to-point trajectory planning in *se*(3) with self-collision and joint constraints avoidance. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9. IEEE.

[60] Halm, M. and Posa, M. (2021). Set-valued rigid body dynamics for simultaneous frictional impact. *arXiv preprint arXiv:2103.15714*.

[61] Haruno, M., Wolpert, D. M., and Kawato, M. (2001). Mosaic model for sensorimotor learning and control. *Neural computation*, 13(10):2201–2220.

[62] Hausman, K., Springenberg, J. T., Wang, Z., Heess, N., and Riedmiller, M. (2018). Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*.

[63] Heemels, W., Lehmann, D., Lunze, J., and De Schutter, B. (2009). Introduction to hybrid systems. In *Handbook of Hybrid Systems Control–Theory, Tools, Applications*, pages 4–30. Cambridge, UK: Cambridge University Press.

[64] Heiden, E., Millard, D., Coumans, E., Sheng, Y., and Sukhatme, G. S. (2020). Neuralsim: Augmenting differentiable simulators with neural networks. *arXiv preprint arXiv:2011.04217*.

[65] Hoffmann, M., Marques, H., Arieta, A., Sumioka, H., Lungarella, M., and Pfeifer, R. (2010). Body schema in robotics: a review. *IEEE Transactions on Autonomous Mental Development*, 2(4):304–324.

[66] Hogan, N. (1984). Impedance control: An approach to manipulation. In *IEE-ACC-1984*, pages 304–313. IEEE.

[67] Högman, V., Björkman, M., Maki, A., and Kragic, D. (2015). A sensorimotor learning framework for object categorization. *IEEE Transactions on Cognitive and Developmental Systems*, 8(1):15–25.

[68] Howard, M., Braun, D. J., and Vijayakumar, S. (2013). Transferring human impedance behavior to heterogeneous variable impedance actuators. *IEEE Transactions on Robotics*, 29(4):847–862.

[69] Huang, B., Li, M., De Souza, R. L., Bryson, J. J., and Billard, A. (2016). A modular approach to learning manipulation strategies from human demonstration. *Autonomous Robots*, 40(5):903–927.

[70] Huang, P., Xu, Y., and Liang, B. (2006). Global minimum-jerk trajectory planning of space manipulator. *International Journal of Control, Automation, and Systems*, 4(4):405–413.

[71] Hunt, K. H. and Crossley, F. R. E. (1975). Coefficient of restitution interpreted as damping in vibroimpact.

[72] Hyatt, P. and Killpack, M. D. (2020). Real-time nonlinear model predictive control of robots using a graphics processing unit. *IEEE Robotics and Automation Letters*, 5(2):1468–1475.

[73] Hyde, J. M., Tremblay, M. R., and Cutkosky, M. R. (1997). An object-oriented framework for event-driven dextrous manipulation. In *Experimental Robotics IV*, pages 51–61. Springer.

[74] Ijspeert, A. J., Nakanishi, J., and Schaal, S. (2002). Movement imitation with nonlinear dynamical systems in humanoid robots. In *International Conference on Robotics and Automation*, volume 2, pages 1398–1403. IEEE.

[75] Jain, A. and Niekum, S. (2018). Efficient hierarchical robot motion planning under uncertainty and hybrid dynamics. *arXiv preprint arXiv:1802.04205*.

[76] Johannink, T., Bahl, S., Nair, A., Luo, J., Kumar, A., Loskyll, M., Ojea, J. A., Solowjow, E., and Levine, S. (2019). Residual reinforcement learning for robot control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6023–6029. IEEE.

[77] Johansson, M. K.-J. (2003). *Piecewise linear control systems: a computational approach*, volume 284. Springer.

[78] Johansson, R. S. and Cole, K. J. (1992). Sensory-motor coordination during grasping and manipulative actions. *Current Opinion in Neurobiology*, 2(6):815–823.

[79] Johnson, A. M., Burden, S. A., and Koditschek, D. E. (2016). A hybrid systems model for simple manipulation and self-manipulation systems. *The International Journal of Robotics Research*, 35(11):1354–1392.

[80] Kalakrishnan, M., Righetti, L., Pastor, P., and Schaal, S. (2011). Learning force control policies for compliant manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4639–4644.

[81] Katz, D. and Brock, O. (2011). A factorization approach to manipulation in unstructured environments. In *Robotics Research*, pages 285–300. Springer.

[82] Kawato, M. (1999). Internal Models for Motor Control and Trajectory Planning. *Current Opinion in Neurobiology*, (6):718–727.

[83] Kemp, C. C., Edsinger, A., and Torres-Jara, E. (2007). Challenges for robot manipulation in human environments [grand challenges of robotics]. *IEEE Robotics & Automation Magazine*, 14(1):20–29.

[84] Khader, S. A., Yin, H., Falco, P., and Kragic, D. (2020a). Data-efficient model learning and prediction for contact-rich manipulation tasks. *IEEE Robotics and Automation Letters*, 5(3):4321–4328.

[85] Khader, S. A., Yin, H., Falco, P., and Kragic, D. (2020b). Stability-guaranteed reinforcement learning for contact-rich manipulation. *IEEE Robotics and Automation Letters*, 6(1):1–8.

[86] Khansari-Zadeh, S. M. and Billard, A. (2010). Bm: An iterative algorithm to learn stable non-linear dynamical systems with gaussian mixture models. In *International Conference on Robotics and Automation*, pages 2381–2388. IEEE.

[87] Khansari-Zadeh, S. M., Kronander, K., and Billard, A. (2014). Modeling robot discrete movements with state-varying stiffness and damping: A framework for integrated motion generation and impedance control. *Proceedings of robotics: Science and systems X (RSS 2014)*, 10:2014.

[88] Kocijan, J., Murray-Smith, R., Rasmussen, C. E., and Girard, A. (2004). Gaussian process model based predictive control. In *Proceedings of the 2004 American control conference*, volume 3, pages 2214–2219. IEEE.

[89] Koivo, A. and Guo, T.-H. (1983). Adaptive linear controller for robotic manipulators. *IEEE Transactions on Automatic Control*, 28(2):162–171.

[90] Kolev, S. and Todorov, E. (2015). Physically consistent state estimation and system identification for contacts. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 1036–1043. IEEE.

[91] Kolter, J. Z. and Ng, A. Y. (2009). Task-space trajectories via cubic spline optimization. In *2009 IEEE International Conference on Robotics and Automation*, pages 1675–1682. IEEE.

[92] Kong, M.-X., Ji, C., Chen, Z.-S., and Li, R.-f. (2013). Application of orientation interpolation of robot using unit quaternion. In *2013 IEEE International Conference on Information and Automation (ICIA)*, pages 384–389. IEEE.

[93] Kopicki, M., Zurek, S., Stolkin, R., Moerwald, T., and Wyatt, J. L. (2017). Learning modular and transferable forward models of the motions of push manipulated objects. *Autonomous Robots*, 41(5):1061–1082.

[94] Kormushev, P., Calinon, S., and Caldwell, D. G. (2010). Robot motor skill coordination with em-based reinforcement learning. In *2010 IEEE/RSJ international conference on intelligent robots and systems*, pages 3232–3237. IEEE.

[95] Koval, M. C., Pollard, N. S., and Srinivasa, S. S. (2016). Pre-and post-contact policy decomposition for planar contact manipulation under uncertainty. *The International Journal of Robotics Research*, 35(1-3):244–264.

[96] Kowalewski, S., Garavello, M., Guéguen, H., Herberich, G., Langerak, R., Piccoli, B., Polderman, J., and Weise, C. (2009). Hybrid automata. In *Handbook of Hybrid Systems Control–Theory, Tools, Applications*, pages 57–85. Cambridge, UK: Cambridge University Press.

[97] Kramberger, A., Shahriari, E., Gams, A., Nemec, B., Ude, A., and Haddadin, S. (2018). Passivity based iterative learning of admittance-coupled dynamic movement primitives for interaction with changing environments. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6023–6028. IEEE.

[98] Kroemer, O., Daniel, C., Neumann, G., Van Hoof, H., and Peters, J. (2015). Towards learning hierarchical skills for multi-phase manipulation tasks. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1503–1510. IEEE.

[99] Kroemer, O., Niekum, S., and Konidaris, G. (2019). A review of robot learning for manipulation: Challenges, representations, and algorithms. *arXiv preprint arXiv:1907.03146*.

[100] Kronander, K. and Billard, A. (2014). Learning compliant manipulation through kinesthetic and tactile human-robot interaction. *IEEE Transactions on Haptics*, 7(3):367–380.

[101] Kronander, K. and Billard, A. (2016). Stability considerations for variable impedance control. *IEEE Transactions on Robotics*, 32(5):1298–1305.

[102] Krüger, N., Ude, A., Petersen, H. G., Nemec, B., Ellekilde, L.-P., Savarimuthu, T. R., Rytz, J. A., Fischer, K., Buch, A. G., Kraft, D., et al. (2014). Technologies for the fast set-up of automated assembly processes. *KI-Künstliche Intelligenz*, 28(4):305–313.

[103] Kupcsik, A., Deisenroth, M. P., Peters, J., Loh, A. P., Vadakkepat, P., and Neumann, G. (2017a). Model-based contextual policy search for data-efficient generalization of robot skills. *Artificial Intelligence*, 247:415–439.

[104] Kupcsik, A., Deisenroth, M. P., Peters, J., Loh, A. P., Vadakkepat, P., and Neumann, G. (2017b). Model-based contextual policy search for data-efficient generalization of robot skills. *Artificial Intelligence*, 247:415–439.

[105] Kupcsik, A. G., Deisenroth, M. P., Peters, J., Neumann, G., et al. (2013). Data-efficient generalization of robot skills with contextual policy search. In *AAAI 2013*, pages 1401–1407.

[106] Kyriakopoulos, K. J. and Saridis, G. N. (1988). Minimum jerk path generation. In *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, pages 364–369. IEEE.

[107] Lee, A. X., Lu, H., Gupta, A., Levine, S., and Abbeel, P. (2015). Learning force-based manipulation of deformable objects from multiple demonstrations. In *International Conference on Robotics and Automation*, pages 177–184.

[108] Lee, G., Marinho, Z., Johnson, A. M., Gordon, G. J., Srinivasa, S. S., and Mason, M. T. (2017). Unsupervised learning for nonlinear piecewise smooth hybrid systems. *arXiv preprint arXiv:1710.00440*.

[109] Lee, J., Grey, M. X., Ha, S., Kunz, T., Jain, S., Ye, Y., Srinivasa, S. S., Stilman, M., and Liu, C. K. (2018). Dart: Dynamic animation and robotics toolkit. *Journal of Open Source Software*, 3(22):500.

[110] Lee, M. A., Zhu, Y., Srinivasan, K., Shah, P., Savarese, S., Fei-Fei, L., Garg, A., and Bohg, J. (2019). Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8943–8950. IEEE.

[111] Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373.

[112] Levine, S. and Koltun, V. (2013). Guided policy search. In *International Conference on Machine Learning*, pages 1–9.

[113] Li, M., Yin, H., Tahara, K., and Billard, A. (2014). Learning object-level impedance control for robust grasping and dexterous manipulation. In *International Conference on Robotics and Automation*, pages 6784–6791.

[114] Li, Y., Sam Ge, S., and Yang, C. (2012). Learning impedance control for physical robot–environment interaction. *International Journal of Control*, 85(2):182–193.

[115] Lippiello, V., Siciliano, B., and Villani, L. (2007). A position-based visual impedance control for robot manipulators. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 2068–2073. IEEE.

[116] Liu, C. K. (2009). Dextrous manipulation from a grasping pose. In *ACM Transactions on Graphics*, volume 28, page 59. ACM.

[117] Lowrey, K., Kolev, S., Dao, J., Rajeswaran, A., and Todorov, E. (2018). Reinforcement learning for non-prehensile manipulation: Transfer from simulation to physical system. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, pages 35–42.

[118] Lynch, K. M. (1992). The mechanics of fine manipulation by pushing. In *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pages 2269–2276. IEEE.

[119] Lynch, K. M. and Park, F. C. (2017). Robot control. In *Modern Robotics*, pages 403–460. Cambridge University Press.

[120] Marth, G., Tarn, T. J., and Bejczy, A. K. (1993). Stable phase transition control for robot arm motion. In *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 355–362. IEEE.

[121] Martín-Martín, R., Lee, M. A., Gardner, R., Savarese, S., Bohg, J., and Garg, A. (2019). Variable impedance control in end-effector space: An action space for reinforcement learning in contact-rich tasks. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1010–1017. IEEE.

[122] Mason, M. T. (1982). Manipulator grasping and pushing operations.

[123] Mason, M. T. (2001). *Mechanics of robotic manipulation*. MIT press.

[124] Mathew, M. J., Sidhik, S., Sridharan, M., Azad, M., Hayashi, A., and Wyatt, J. (2019). Online Learning of Feed-Forward Models for Task-Space Variable Impedance Control. In *International Conference on Humanoid Robots*.

[125] Matthews, A. G. d. G., van der Wilk, M., Nickson, T., Fujii, K., Boukouvalas, A., León-Villagrá, P., Ghahramani, Z., and Hensman, J. (2017). GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 18(40):1–6.

[126] Maye, J., Sommer, H., Agamennoni, G., Siegwart, R., and Furgale, P. (2016). Online self-calibration for robotic systems. *The International Journal of Robotics Research*, 35(4):357–380.

[127] Mayne, D. Q. and Raković, S. (2003). Model predictive control of constrained piecewise affine discrete-time systems. *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, 13(3-4):261–279.

[128] Mehta, B. and Schaal, S. (2002). Forward models in visuomotor control. *Journal of Neurophysiology*, 88(2):942–953.

[129] Miall, R. C. and Wolpert, D. M. (1996). Forward models for physiological motor control. *Neural networks*, 9(8):1265–1279.

[130] Mignone, D., Ferrari-Trecate, G., and Morari, M. (2000). Stability and stabilization of piecewise affine and hybrid systems: An lmi approach. In *Proceedings of the 39th IEEE Conference on Decision and Control (Cat. No. 00CH37187)*, volume 1, pages 504–509. IEEE.

[131] Mills, J. K. and Lokhorst, D. M. (1993). Control of robotic manipulators during general task execution: A discontinuous control approach. *The International Journal of Robotics Research*, 12(2):146–163.

[132] Moore, M. and Wilhelms, J. (1988). Collision detection and response for computer animation. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 289–298.

[133] Nagabandi, A., Konolige, K., Levine, S., and Kumar, V. (2020). Deep dynamics models for learning dexterous manipulation. In *Conference on Robot Learning*, pages 1101–1112. PMLR.

[134] Nakamura, Y., Mori, T., Sato, M.-a., and Ishii, S. (2007). Reinforcement learning for a biped robot based on a cpg-actor-critic method. *Neural networks*, 20(6):723–735.

[135] Nakashima, A., Ooka, Y., and Hayakawa, Y. (2015). Contact transition modelling on planar manipulation system with lugre friction model. In *2015 10th International Workshop on Robot Motion and Control (RoMoCo)*, pages 300–307. IEEE.

[136] Nam, S.-H. and Yang, M.-Y. (2004). A study on a generalized parametric interpolator with real-time jerk-limited acceleration. *Computer-Aided Design*, 36(1):27–36.

[137] Narendra, K. S. and Annaswamy, A. M. (1989). *Stable Adaptive Systems*. Prentice-Hall, Inc., USA.

[138] Nguyen-Tuong, D. and Peters, J. (2011). Model learning for robot control: a survey. *Cognitive processing*, 12(4):319–340.

[139] Nguyen-Tuong, D., Seeger, M., and Peters, J. (2009). Model learning with local gaussian process regression. *Advanced Robotics*, 23(15):2015–2034.

[140] Niekum, S., Chitta, S., Barto, A. G., Marthi, B., and Osentoski, S. (2013). Incremental semantically grounded learning from demonstration. In *Robotics: Science and Systems*, volume 9, pages 10–15607. Berlin, Germany.

[141] Osu, R. and Gomi, H. (1999). Multijoint muscle regulation mechanisms examined by measured human arm stiffness and emg signals. *Journal of neurophysiology*, 81(4):1458–1468.

[142] Pajarinen, J. and Kyrki, V. (2017). Robotic manipulation of multiple objects as a pomdp. *Artificial Intelligence*, 247:213–228.

[143] Parberry, I. (2013). *Introduction to Game Physics with Box2D*. CRC Press.

[144] Parmar, M., Halm, M., and Posa, M. (2021). Fundamental challenges in deep learning for stiff contact dynamics. *arXiv preprint arXiv:2103.15406*.

[145] Pastor, P., Hoffmann, H., Asfour, T., and Schaal, S. (2009). Learning and generalization of motor skills by learning from demonstration. In *2009 IEEE International Conference on Robotics and Automation*, pages 763–768. IEEE.

[146] Paul, R. P. (1987). Problems and research issues associated with the hybrid control of force and displacement.

[147] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[148] Peng, X. B., Abbeel, P., Levine, S., and van de Panne, M. (2018). Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)*, 37(4):1–14.

[149] Pepyne, D. L. and Cassandras, C. G. (1998). Modeling, analysis, and optimal control of a class of hybrid systems. *Discrete Event Dynamic Systems*, 8(2):175–201.

[150] Peternel, L. and Ajoudani, A. (2017). Robots learning from robots: A proof of concept study for co-manipulation tasks. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 484–490. IEEE.

[151] Pezzato, C., Ferrari, R., and Corbato, C. H. (2020). A novel adaptive controller for robot manipulators based on active inference. *IEEE Robotics and Automation Letters*, 5(2):2973–2980.

[152] Pfeiffer, F. and Glocker, C. (1996). *Multibody dynamics with unilateral contacts*. John Wiley & Sons.

[153] Piazzi, A. and Visioli, A. (2000). Global minimum-jerk trajectory planning of robot manipulators. *IEEE transactions on industrial electronics*, 47(1):140–149.

[154] Pinto, R. C. and Engel, P. M. (2015). A fast incremental gaussian mixture model. *PloS one*, 10(10):e0139931.

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

[156] Posa, M., Tobenkin, M., and Tedrake, R. (2015). Stability analysis and control of rigid-body systems with impacts and friction. *IEEE Transactions on Automatic Control*, 61(6):1423–1437.

[157] Rajeswaran, A., Kumar, V., Gupta, A., Vezzani, G., Schulman, J., Todorov, E., and Levine, S. (2017). Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*.

[158] Rezaee, A. (2017). Model predictive controller for mobile robot. *Transactions on Environment and Electrical Engineering*, 2(2):18–23.

[159] Romano, J. M., Hsiao, K., Niemeyer, G., Chitta, S., and Kuchenbecker, K. J. (2011). Human-inspired robotic grasp control with tactile sensing. *IEEE Transactions on Robotics*, 27(6):1067–1079.

[160] Rosenbaum, D. A. (2009). *Human motor control*. Academic press.

[161] Routh, E. J. et al. (1955). *Dynamics of a system of rigid bodies*. Dover New York.

[162] Rozo, L., Calinon, S., Caldwell, D., Jiménez, P., and Torras, C. (2013). Learning collaborative impedance-based robot behaviors. In *Proceedings of the AAAI conference on artificial intelligence*, volume 27.

[163] Rozo, L., Calinon, S., Caldwell, D. G., Jimenez, P., and Torras, C. (2016). Learning physical collaborative robot behaviors from human demonstrations. *IEEE Transactions on Robotics*, 32(3):513–527.

[164] Salisbury, J. K. (1980). Active stiffness control of a manipulator in cartesian coordinates. In *IEEE Conference on Decision and Control, including the Symposium on Adaptive Processes*, volume 19, pages 95–100.

[165] Saveriano, M. and Lee, D. (2014). Learning motion and impedance behaviors from human demonstrations. In *2014 11th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pages 368–373. IEEE.

[166] Schneider, S. A. and Cannon, R. H. (1992). Object impedance control for cooperative manipulation: Theory and experimental results. *IEEE Transactions on Robotics and Automation*, 8(3):383–394.

[167] Selen, L. P., Franklin, D. W., and Wolpert, D. M. (2009). Impedance control reduces instability that arises from motor noise. *Journal of neuroscience*, 29(40):12606–12616.

[168] Shadmehr, R. and Krakauer, J. (2008). A Computational Neuroanatomy for Motor Control. *Experimental Brain Research*, 185(3):359–381.

[169] Shoemake, K. (1985). Animating rotation with quaternion curves. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 245–254.

[170] Shon, A. P., Storz, J. J., and Rao, R. P. (2007). Towards a real-time bayesian imitation system for a humanoid robot. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 2847–2852. IEEE.

[171] Siciliano, B., Sciavicco, L., Villani, L., and Oriolo, G. (2009). Differential kinematics and statics. *Robotics: Modelling, Planning and Control*, pages 105–160.

[172] Siciliano, B. and Villani, L. (2001). An inverse kinematics algorithm for interaction control of a flexible arm with a compliant surface. *Control Engineering Practice*, 9(2):191–198.

[173] Sidhik, S. (2020a). Franka ros interface: A ros/python api for controlling and managing the franka emika panda robot (real and simulated).

[174] Sidhik, S. (2020b). panda_simulator: Gazebo simulator for Franka Emika Panda robot supporting sim-to-real code transfer.

[175] Sidhik, S., Sridharan, M., and Ruiken, D. (2020). Learning hybrid models for variable impedance control of changing-contact manipulation tasks. In *Eighth Annual Conference on Advances in Cognitive Systems (ACS)*.

[176] Silver, T., Allen, K., Tenenbaum, J., and Kaelbling, L. (2018). Residual policy learning. *arXiv preprint arXiv:1812.06298*.

[177] Siméon, T., Laumond, J.-P., Cortés, J., and Sahbani, A. (2004). Manipulation planning with probabilistic roadmaps. *The International Journal of Robotics Research*, 23(7-8):729–746.

[178] Smith, O. J. (1959). A controller to overcome dead time. *ISA J.*, 6:28–33.

[179] Smith, R. et al. (2005). Open dynamics engine.

[180] Song, M. and Wang, H. (2005). Highly efficient incremental estimation of Gaussian mixture models for online data stream clustering. In Priddy, K. L., editor, *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 5803 of *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, pages 174–183.

[181] Steinmetz, F., Montebelli, A., and Kyrki, V. (2015). Simultaneous kinesthetic teaching of positional and force requirements for sequential in-contact tasks. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 202–209. IEEE.

[182] Stoianovici, D. and Hurmuzlu, Y. (1996). A critical study of the applicability of rigid-body collision theory.

[183] Stüber, J., Zito, C., and Stolkin, R. (2019). Let's push things forward: A survey on robot pushing. *arXiv preprint arXiv:1905.05138*.

[184] Stulp, F. and Beetz, M. (2008). Refining the execution of abstract actions with learned action models. *Journal of Artificial Intelligence Research*, 32:487–523.

[185] Stulp, F., Theodorou, E. A., and Schaal, S. (2012). Reinforcement learning with sequences of motion primitives for robust manipulation. *IEEE Transactions on robotics*, 28(6):1360–1370.

[186] Suárez-Ruiz, F., Zhou, X., and Pham, Q.-C. (2018). Can robots assemble an ikea chair? *Science Robotics*, 3(17):eaat6385.

[187] Sun, T., Peng, L., Cheng, L., Hou, Z.-G., and Pan, Y. (2019). Stability-guaranteed variable impedance control of robots based on approximate dynamic inversion. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*.

[188] Sung, H. G. (2004). *Gaussian mixture regression and classification*. PhD thesis, Rice University.

[189] Sutherland, J. L. (1987). *Model reference adaptive control of a two link manipulator*. PhD thesis, Carleton University.

[190] Takahashi, C., Scheidt, R. A., and Reinkensmeyer, D. (2001). Impedance control and internal model formation when reaching in a randomly varying dynamical environment. *Journal of neurophysiology*, 86(2):1047–1051.

[191] Tarokh, M. (1991). Hyperstability approach to the synthesis of adaptive controllers for robot manipulators. In *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, pages 2154–2159 vol.3.

[192] Thrun, S. (2003). Learning occupancy grid maps with forward sensor models. *Autonomous robots*, 15(2):111–127.

[193] Tian, S., Ebert, F., Jayaraman, D., Mudigonda, M., Finn, C., Calandra, R., and Levine, S. (2019). Manipulation by feel: Touch-based control with deep predictive models. *arXiv preprint arXiv:1903.04128*.

[194] Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE.

[195] Toussaint, M., Allen, K., Smith, K. A., and Tenenbaum, J. B. (2018). Differentiable physics and stable modes for tool-use and manipulation planning. In *Robotics: Science and Systems*.

[196] Ugur, E. and Piater, J. (2015). Bottom-up learning of object categories, action effects and logical rules: From continuous manipulative exploration to symbolic planning. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2627–2633. IEEE.

[197] Van der Schaft, A. J. and Van Der Schaft, A. (2000). *L2-gain and passivity techniques in nonlinear control*, volume 2. Springer.

[198] Varin, P., Grossman, L., and Kuindersma, S. (2019). A comparison of action spaces for learning manipulation tasks. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6015–6021. IEEE.

[199] Villani, L. and De Schutter, J. (2016). Force control. In *Springer handbook of robotics*, pages 195–220. Springer.

[200] Wang, T., Dordevic, G. S., and Shadmehr, R. (2001). Learning the dynamics of reaching movements results in the modification of arm impedance and long-latency perturbation responses. *Biological cybernetics*, 85(6):437–448.

[201] Wieber, P.-B., Tedrake, R., and Kuindersma, S. (2016). Modeling and control of legged robots. In *Springer handbook of robotics*, pages 1203–1234. Springer.

[202] Wimböck, T., Ott, C., Albu-Schäffer, A., and Hirzinger, G. (2012). Comparison of object-level grasp controllers for dynamic dexterous manipulation. *International Journal of Robotics Research*, 31(1):3–23.

[203] Wolpert, D. M., Ghahramani, Z., and Jordan, M. I. (1995). An internal model for sensorimotor integration. *Science*, 269(5232):1880.

[204] Yang, C., Ganesh, G., Haddadin, S., Parusel, S., Albu-Schaeffer, A., and Burdet, E. (2011). Human-like adaptation of force and impedance in stable and unstable interactions. *IEEE transactions on robotics*, 27(5):918–930.

[205] Yashima, M. and Yamaguchi, H. (2003). Complementarity formulation for multi-fingered hand manipulation with rolling and sliding contacts. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, volume 2, pages 2255–2261. IEEE.

[206] Yechiel, O. and Guterman, H. (2017). A survey of adaptive control. In *ICRA 2017*.

[207] Yoshikawa, T., Sugie, T., and Tanaka, M. (1988). Dynamic hybrid position/force control of robot manipulators-controller design and experiment. *IEEE Journal on Robotics and Automation*, 4(6):699–705.

[208] Zhang, D. and Wei, B. (2017). A review on model reference adaptive control of robotic manipulators. *Annual Reviews in Control*, 43:188–198.

[209] Zhang, J.-F. et al. (2001). Fundamental limitations and differences of robust and adaptive control. In *Proceedings of the 2001 American Control Conference.(Cat. No. 01CH37148)*, volume 6, pages 4802–4807. IEEE.

[210] Zhang, T., Ramakrishnan, R., and Livny, M. (1996). BIRCH: An Efficient Data Clustering Method for Very Large Databases. *ACM SIGMOD Record*, 25:103–114.

[211] Zhang, T., Ramakrishnan, R., and Livny, M. (1997). Birch: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, 1(2):141–182.

Appendix A

Relevant algorithms used in framework

This section provides more information about some of the algorithms that were adopted from existing literature for the development of the framework in this thesis.

A.1 Gaussian Mixture Models

By formulation, a Gaussian Mixture Model (GMM) behaves as an unsupervised clustering system. While traditional clustering algorithms such as K-means cluster each data point in to a *single* cluster (hard clustering), GMMs define the *probability* of a point belonging to a particular cluster (soft clustering).

A.1.1 Standard derivation

A Gaussian Mixture is a function combining K Gaussians, where each Gaussian is identified by $k \in \{1,2,...K\}$. Each Gaussian k in the mixture is comprised of: a mean μ which defines its centre, a covariance Σ that defines its width (or equivalently the dimensions of an ellipsoid in multivariate scenarios), and a mixing probability π that determines the weightage of the Gaussian function in the overall mixture, such that

$$\sum_{k=i}^{K} \pi_k = 1 \tag{A.1}$$

It is also convenient to define a latent variable z in the context of GMMs, which represent the whether a point belongs to a particular cluster in the mixture or not, i.e., it can only take values 0 or 1. Therefore, $p(z_{nk} = 1|x_n)$ would be read as "Given a point x_n, what it the probability of it being generated from Gaussian k?". Hence, this is equivalent to the mixing coefficient π, i.e., $\pi_k = p(z_k = 1)$.

Now, if $\mathbf{z}$ is the set of all latent variables $\{z_1, ..., z_K\}$, then their mutual independence would lead to prove

$$p(\mathbf{z}) = p(z_1 = 1)^{z_1} p(z_2 = 1)^{z_2} ... p(z_K = 1)^{z_K} = \sum_{k=i}^{K} \pi_k^{z_k}$$

The probability of observing our data given that it came from Gaussian k turns out to be the Gaussian function itself:

$$p(x_n|\mathbf{z}) = \prod_{k=1}^{K} \mathcal{N}(x_n|\mu_k, \Sigma_k)_k^z \tag{A.2}$$

The value that we are interested in is actually the probability of $\mathbf{z}$ given our data $\mathbf{x}$. The product rule of probabilities provide us with:

$$p(x_n, \mathbf{z}) = p(x_n|\mathbf{z}) p(\mathbf{z}) \tag{A.3}$$

Marginalising the above equation for $\mathbf{z}$ and using Eq. (A.2) leads to:

$$p(x_n) = \sum_{k=1}^{K} p(x_n|z) p(z) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)_k^z \tag{A.4}$$

To determine the optimal parameters $\theta = (\pi, \mu, \Sigma)$, we need to determine the maximum likelihood of the model. This can be found as the joint probability of all data points x_n

$$p(\mathbf{x}) = \prod_{n=1}^{N} p(x_n) = \prod_{n=1}^{N} \sum_{k=1}^{K} \pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)_k^z \tag{A.5}$$

Applying log for easier operations:

$$\ln p(\mathbf{x}) = \sum_{n=1}^{N} \ln \sum_{k=1}^{K} \pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)_k^z \tag{A.6}$$

In order to find the optimal parameters for the Gaussian mixture, we have to differentiate this equation with respect to the parameters and equate to zero. However, the logarithm around the second summation of the above equation makes this very hard. Therefore, we need to use an iterative method to estimate the parameters. And we typically use the *Expectation-Maximisation (EM)* algorithm for this.

However, we first have to rearrange the equation so that we can find the probability of z given x. By using Bayes' rule, we can get:

$$p(z_k = 1|x_n) = \frac{p(x_n|z_k = 1)p(z_k = 1)}{\sum_{j=1}^{K} p(x_n|z_j = 1)p(z_j = 1)} \tag{A.7}$$

The above equation with appropriate substitutions become

$$p(z_k = 1|x_n) = \frac{\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_n|\mu_j, \Sigma_j)} = \gamma(z_{nk}) \tag{A.8}$$

The Expectation-Maximisation (EM) algorithm

The EM algorithm is an iterative method which is widely used for optimisation problems where the objective functions have complexities that makes it difficult to obtain closed-form solutions.

In our GMM formulation, we will use the EM algorithm for optimising parameters $\theta = (\pi, \mu, \Sigma)$. At the beginning of the algorithm, θ is initialised using some strategy. Typically, we use the results obtained from a previous K-means run.

i. Expectation step:

In the expectation (E) step, we evaluate the expectation of the form

$$Q(\theta^*, \theta) = \mathbb{E}\left[\ln p(\mathbf{x}, \mathbf{z}|\theta^*)\right] = \sum_{\mathbf{z}} \gamma(z_{nk}) \ln p(\mathbf{x}, \mathbf{z}|\theta^*) \tag{A.9}$$

where $\gamma(z_{nk})$ is defined in Eq. (A.8). The complete model likelihood $p(\mathbf{x}, \mathbf{z}|\theta^*)$ can be obtained as

$$p(\mathbf{x}, \mathbf{z}|\theta^*) = \prod_{n=1}^{N} \prod_{k=1}^{K} \pi^{z_{nk}} \mathcal{N}(x_n|\mu_j, \Sigma_j)^{z_{nk}} \tag{A.10}$$

Applying logarithm operator and substituting this in Eq. (A.9), we get

$$Q(\theta^*, \theta) = \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma(z_{nk}) \left[\ln \pi_k + \ln \mathcal{N}(x_n|\mu_k, \Sigma_k)\right] \tag{A.11}$$

ii. Maximisation step:

The maximisation (M) step tries to find the revised parameters θ^* using

$$\theta^* = \arg\max_{\theta} \mathcal{Q}(\theta^*, \theta) \tag{A.12}$$

To take into account the restriction in Eq. (A.1), we format the optimisation function using the suitable Lagrange multiplier as follows:

$$Q(\theta^*, \theta) = \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma(z_{nk}) \left[\ln \pi_k + \ln \mathcal{N}(x_n | \mu_k, \Sigma_k) \right] - \lambda \left(\sum_{k=1}^{K} \pi_k - 1 \right) \tag{A.13}$$

By taking the derivative of Eq. (A.13) with respect to π, μ, and Σ one after the other and equating to zero, we obtain the solution

$$\pi_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk})}{N} \tag{A.14a}$$

$$\mu_k^* = \frac{\sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x_n}}{\sum_{n=1}^{N} \gamma(z_{nk})} \tag{A.14b}$$

$$\Sigma_k^* = \frac{\sum_{n=1}^{N} \gamma(z_{nk}) (\mathbf{x_n} - \mu_k)(\mathbf{x_n} - \mu_k)^T}{\sum_{n=1}^{N} \gamma(z_{nk})} \tag{A.14c}$$

These values are then used to determine the new value for γ in the next E step. This cycle is repeated till convergence.

Gaussian Mixture Regression (GMR)

GMR offers a simple solution to generate data from a GMM. It relies on the standard properties of Gaussians such as linear transformation and conditioning, and provides a probabilistic synthesis mechanism in which the model can compute output distributions online.

GMR can handle different sources of missing data, as the system can consider during the retrieval any combination of input-output mapping, and the expectations on the remaining dimensions can be computed very efficiently.

Consider the block decomposition of the data point ξ_n in a GMM, with mean and covariance μ_k and Σ_k. At each iteration n, $p(\xi_n^{\mathcal{O}} | \xi_n^{\mathcal{I}})$ can then be computed as a conditional distribution

$$p(\xi_n^{\mathcal{O}} | \xi_n^{\mathcal{I}}) \sim \sum_{k=1}^{K} h_k \mathcal{N}(\hat{\mu}_k^{\mathcal{O}}, \hat{\Sigma}_k^{\mathcal{O}}) \tag{A.15}$$

where

$$\hat{\mu}_k^{\mathcal{O}} = \mu_k^{\mathcal{O}} + \Sigma_k^{\mathcal{O}\mathcal{I}} {\Sigma_k^{\mathcal{I}}}^{-1} (\xi_n^{\mathcal{I}} - \mu_i^{\mathcal{I}}) \tag{A.16a}$$

$$\hat{\Sigma}_k^{\mathcal{O}} = \Sigma_k^{\mathcal{O}} - \Sigma_k^{\mathcal{O}\mathcal{I}} {\Sigma_k^{\mathcal{I}}}^{-1} \Sigma_k^{\mathcal{I}\mathcal{O}} \tag{A.16b}$$

$$h_k = \frac{\pi_n \mathcal{N}(\xi_n^{\mathcal{I}} | \mu_n^{\mathcal{I}}, \Sigma_n^{\mathcal{I}})}{\sum_k \pi_n \mathcal{N}(\xi_k^{\mathcal{I}} | \mu_k^{\mathcal{I}}, \Sigma_k^{\mathcal{I}})} \tag{A.16c}$$

Therefore, for a GMM that is encoded with the data $p(\xi_n^{\mathcal{I}}, \xi_n^{\mathcal{O}})$, GMR can be used to retrieve $p(\xi_n^{\mathcal{O}} | \xi_n^{\mathcal{I}})$.

Computationally the most expensive operation in GMR is the inversion of Σ, with the combined approximate order complexity being almost $O(NKD^3)$, for N data points, K Gaussian components and D dimensions. However, there are research which have successfully been able to bring the complexity of GMM modelling and retrieving down to about $O(NKD^2)$ [180, 154].

A.1.2 Incremental GMM

There have been several works that use incoming batches of data to modify an existing GMM by updating the mean, covariance and weightages of the components, with functionality to add or remove components based on some relevance criteria [180, 154, 4, 42].

The incremental GMM algorithm used in this thesis IGMM was adopted from the work by Engel and Heinen [42]. The algorithm creates and continually adjusts a Gaussian Mixture Model consistently to sequentially presented data. The IGMM algorithm is not described in detail here, and interested readers are referred to the original publication [42].

The critical part of the algorithm is the recursive update equations that are approximate incremental counterparts of the update equations used by the EM algorithm. This allows for incrementally updating the GMM parameters online using incoming batch of data. In general, the algorithm depends on hyperparameters such as the *novelty threshold* which determines the *minimum likelihood* for a data point to be grouped into an existing component. If the point is rejected by all the components, a new component is added to the mixture and the parameters are appropriately adjusted. Another important hyperparameter is a user-specified configuration parameter, which is typically left at a large enough value to avoid singularities. The other parameters are basically the standard hyperparameters in a regular GMM such as initial values of the mean, covariance, and weightage of the initial components.

A.2 BIRCH Incremental Clustering

BIRCH (balanced iterative reducing and clustering using hierarchies) [211] is an unsupervised clustering algorithm which particularly effective over large datasets. The main advantage of BIRCH is its ability to incrementally and dynamically cluster incoming, multi-dimensional data points, trying to produce the best quality clustering given a set of memory and/or time constraints. In most cases, BIRCH only requires a single scan of the database.

Without going into the mathematical formulation and derivation, the BIRCH algorithm takes as input a set of N data points, and a desired maximum number of clusters K, or the maximum 'diameter' of the cluster. It usually has 4 steps, the second and fourth of which are optional.

In the first step, the algorithm creates a hierarchical structure called clustering feature tree (CF tree) using the data points. It is a height-balanced tree with each node comprising of several clustering features (CF). Each CF is a tuple of the form $CF = (N, LS, SS)$, where $LS = \sum_{i=1}^{N} X_i$ is the linear sum of the data points, $SS = \sum_{i=1}^{N} (X_i)^2$ is the square sum. CFs are organised in a height-balanced CF tree following several rules.

In the second (optional) step, the algorithm scans all the leaf entries in the initial CF tree to rebuild a smaller CF tree, while removing outliers and grouping crowded subclusters into larger ones. In the third step, the leaf entries are clustered using an existing clustering algorithm. Usually, an agglomerative hierarchical clustering algorithm is applied directly to the subclusters. At this point, a set of clusters is obtained that captures major distribution pattern in the data. Minor localised inaccuracies that could be present can then be handled by the optional fourth step, where the centroids of the clusters produced in step 3 are used as seeds to redistribute the data points to its closest cluster to obtain a new set of clusters. For the full algorithm description, readers are invited to read [211].

Appendix B

Additional experiments

In this appendix, we discuss the experiments of some of the less relevant experiments that were not included in the main thesis text. These experiments are considered less important because (i) their results are inconclusive, (ii) their results provide the same conclusion as other experiments in the main text, or (iii) their results are unimportant or do not directly relate to the overall framework presented in this thesis.

B.1 AVIC trajectory tracking in simulated environments

This section presents some results of the trajectory-tracking performance of the AVIC framework in the simulated 'multi-spring' and 'porridge' environments that were described in Section 4.3. The section in the main thesis text only presented the result demonstrating the convergence ability and results of comparing AVIC with other adaptive control frameworks. This section simply provides the full trajectory-tracking and stiffness variation results of the AVIC framework following a task-space plan in the two environments.

The tracking accuracy of AVIC in the 'multi-spring' environment is shown in Fig. B.3. The prediction error by the incremental forward model and the corresponding stiffness adaptation by the AVIC framework can be seen in Fig. B.2.

Similarly, tracking accuracy of AVIC in the 'porridge environment' environment is plotted in **??**, while the prediction error by the incremental forward model and the corresponding stiffness adaptation by the AVIC framework is in Fig. B.2. Although the high forces make the learning harder, the error in prediction does not affect the trajectory tracking due to the corresponding adaptation of the controller stiffness.

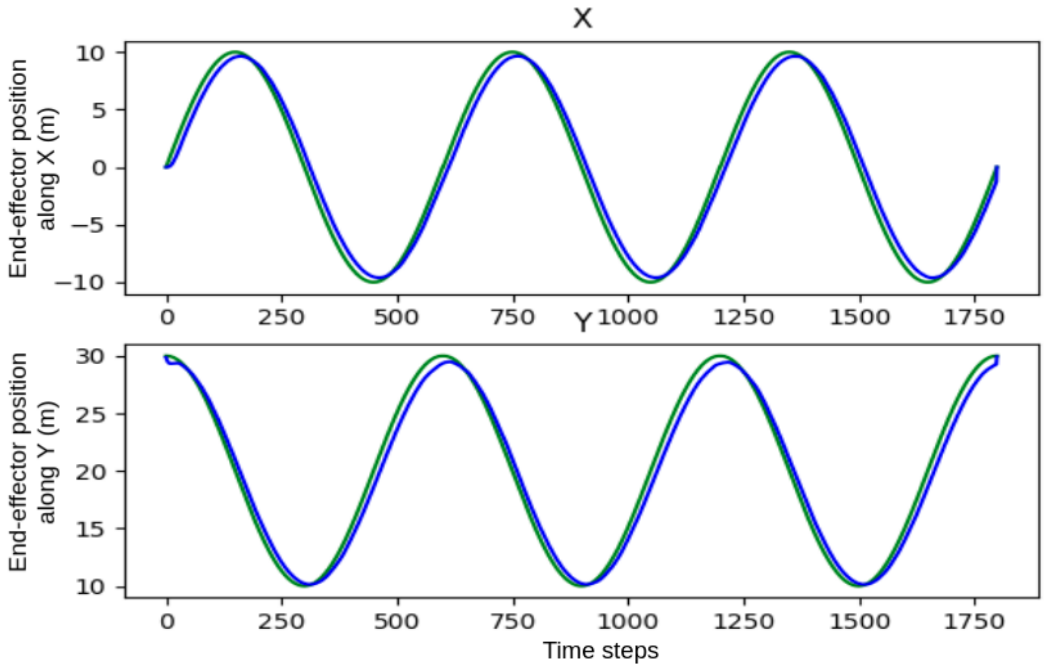
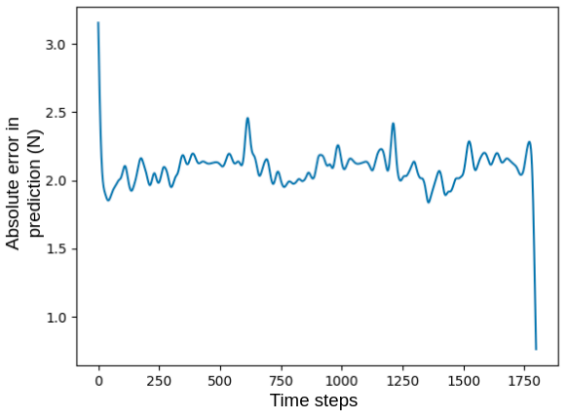
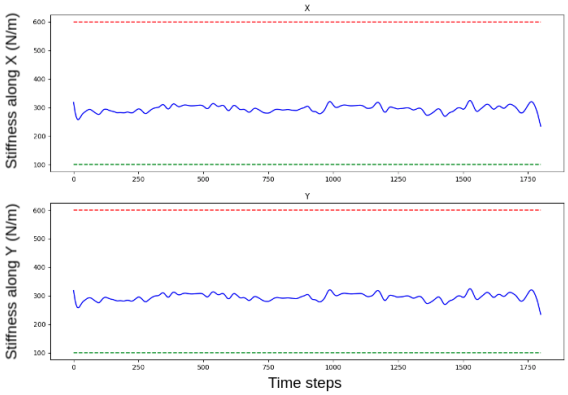


Fig. B.1 Trajectory tracking using AVIC in the 'multi-spring' environment along X (top) and Y (bottom). **Green:** Target trajectory; **Blue:** Actual end-effector position.



(a) Error in force prediction.



(b) Corresponding stiffness profile. **Green:** Minimum allowed stiffness; **Red:** Maximum allowed stiffness; **Blue:** Stiffness profile adaptation by AVIC.

Fig. B.2 Force prediction accuracy by the incremental forward model and corresponding stiffness variation of AVIC in the 'multi-spring' environment.

B.2 Friction in simulators

This section presents some of the experiments that were conducted in simulation to test the effect of different friction on the interaction dynamics in terms of end-effector forces and

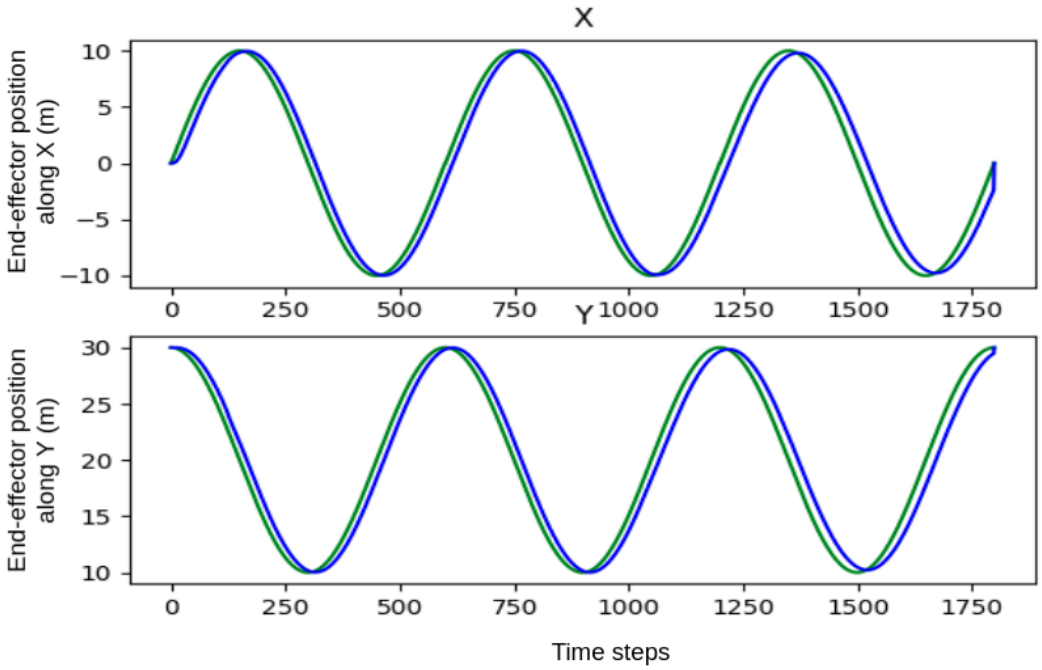
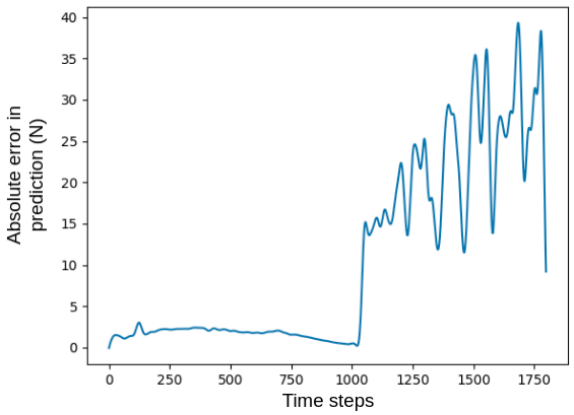
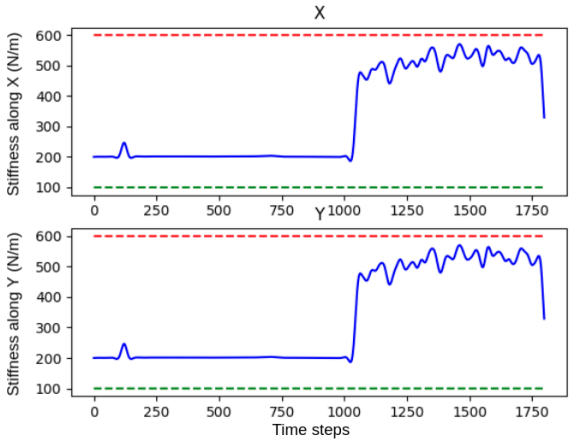


Fig. B.3 Trajectory tracking using AVIC in the 'porridge' environment along X (top) and Y (bottom). **Green:** Target trajectory; **Blue:** Actual end-effector position.



(a) Error in force prediction.



(b) Corresponding stiffness profile. **Green:** Minimum allowed stiffness; **Red:** Maximum allowed stiffness; **Blue:** Stiffness profile adaptation by AVIC.

Fig. B.4 Force prediction accuracy by the incremental forward model and corresponding stiffness variation of AVIC in the 'porridge' environment. When the model accuracy is low, the stiffness is increased to produce reliable tracking.

velocities. This part of the experiments were done for developing and testing the hybrid framework for discontinuous dynamics presented in Chapter 5. This had to be done in

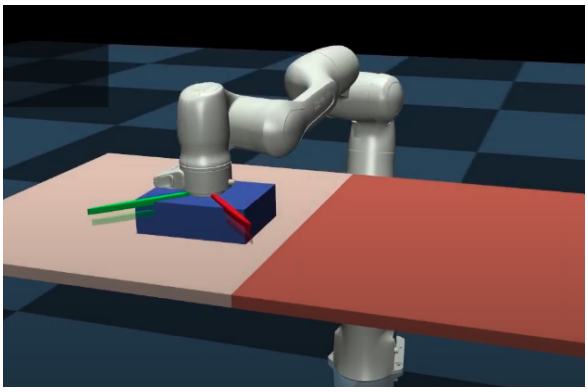


Fig. B.5 Simulated sliding task in Mujoco physics simulator. Friction changes discretely as the robot slides from one surface to the other.

simulation since the physical robot was unavailable due to lack of access to the lab during Covid '19 lockdown. At this point, initial experiments testing the effect of suddenly changing friction on a robot sliding across surfaces had been conducted with the physical robot (see Section 5.3.1), and the behaviour that was expected of the simulator was roughly known.

B.2.1 Friction in Bullet simulator

This experiment was conducted to observe how the end-effector forces and torques are affected in Bullet physics engine when the robot slides between surfaces of different friction. The task was to approach a table and slide on it, while the friction of the surface suddenly changes (such as in Fig. 5.1 (left) and Fig. B.5. However, unlike the response of the physical robot system (see Fig. 5.10), the simulator produced forces and plots that looked very similar to the ideal behaviour of stick-slip model in friction (Fig. B.6).

This was understood to be unrealistic and hence building a framework for dealing with these simulated dynamics effects was decided to be pointless.

B.2.2 Friction in Mujoco simulator

The same experiment was then replicated using Mujoco physics (Fig. B.5). Although the effects looked more realistic (Fig. B.7), the simulator was not ideal for our purposes for several reasons such as:

- Mujoco physics typically runs in a single thread/process and does not allow parallelisation in a straightforward manner. This makes implementing our framework in Mujoco difficult.

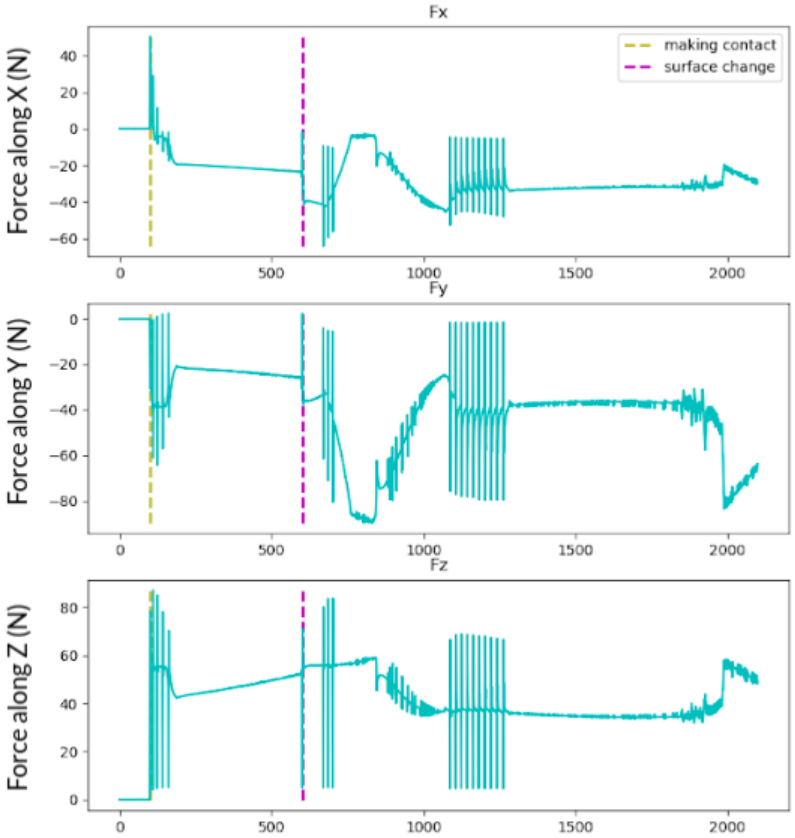


Fig. B.6 End-effector force measured along the axis of motion. The change in sign of forces indicate change in direction of motion. The dotted vertical lines indicate the points where the surfaces change.

- There are inconsistencies between trials even when simulation is run at same speed. Performance can be affected depending on load on computer at the time. (These were tested experimentally, but results are not included here as they are out of scope.)
- Measurement inconsistencies increase when sim speeds are varied. This seems to be because Mujoco does not implement independent clock; sim time is updated by user/code, and would require guaranteed constant loop rate from the client-side to maintain a consistent duration of the simulation steps.

B.2.3 Conclusion

Simulating friction is known to be hard [79]. This meant that discrete dynamic changes due to friction could not be experimented further in simulation. The hybrid framework presented in Chapter 5 was therefore developed using data from the physical robot instead.

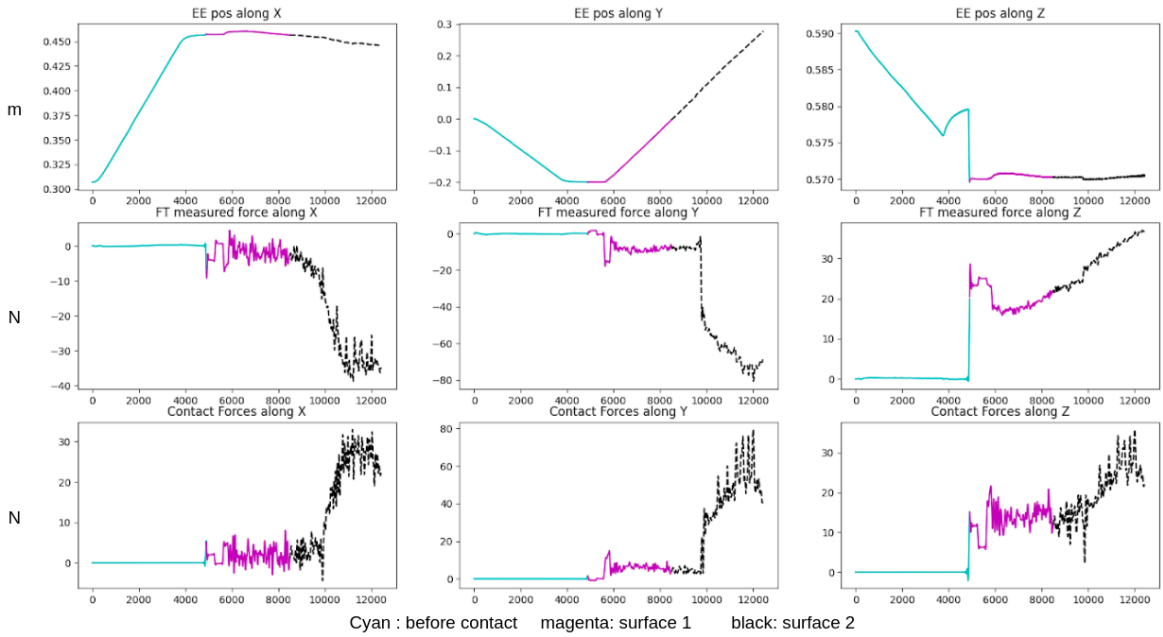


Fig. B.7 End-effector position (left), end-effector forces (middle column), and contact forces from simulator (right) along X (top), Y (middle row), and Z (bottom) axes during the sliding task. **Cyan:** Before contact with surface; **Magenta:** Contact with surface 1; **Black:** In contact with surface 2.

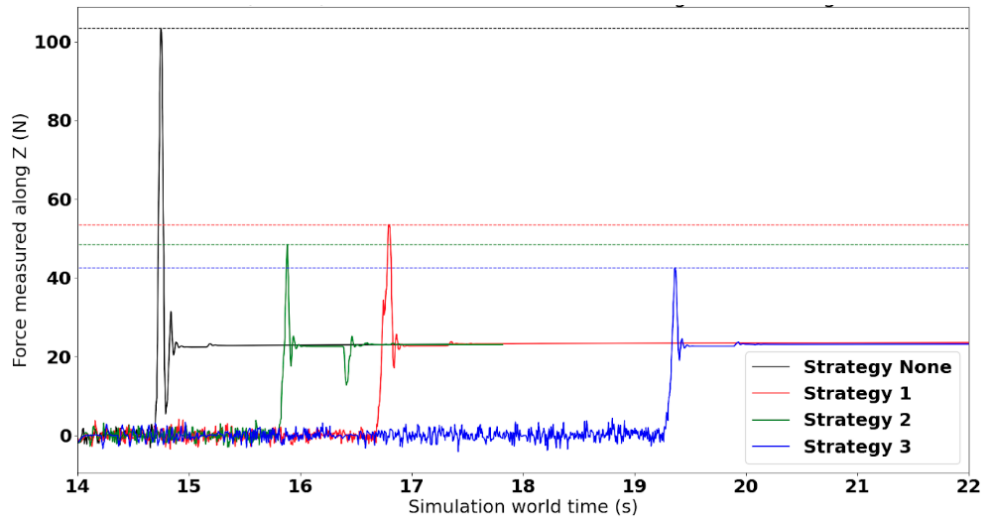
B.3 Impact dynamics experiments

This section presents some of the initial and/or rejected experiments that were conducted during the development of the contact-change-handling module presented in Chapter 6.

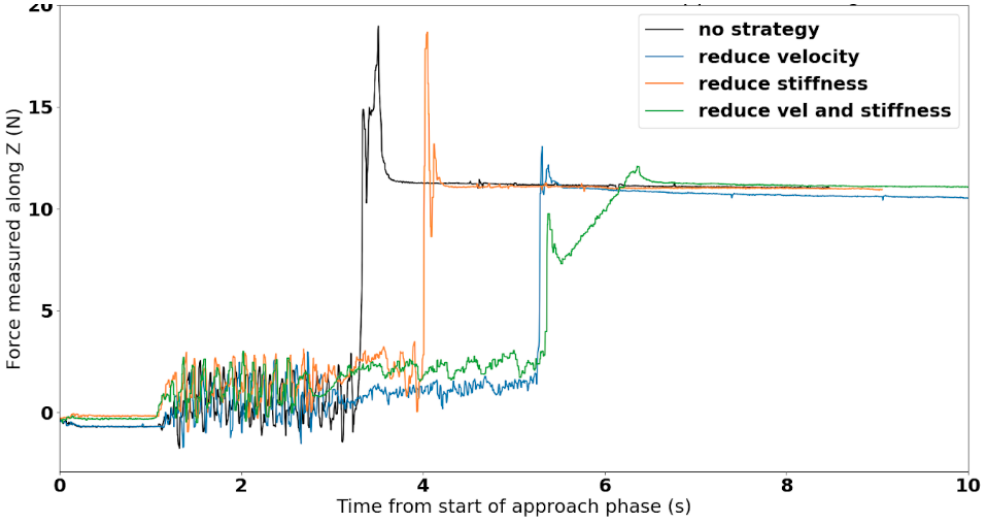
B.3.1 Impact in simulation vs real-world

Experiments to test different transition-phase controllers for handling impact and collisions were first done in simulation to avoid damage to the physical robot. The main strategies that were tested were (i) lowering controller stiffness, (ii) lowering approach velocity, and (iii) a combination of the two.

These strategies when compared in simulation produced different results to physical robot experiments. From Fig. B.8, it is clear that reducing stiffness seams to also reduce the impact force in simulation, which is not the case on a real system. The expected behaviour is seen on the real system (Fig. B.8b) where the impact force is only affected by the approach velocity and not by the controller stiffness. The reasoning for this was explained in Section 6.4.1.



(a) Effect of the strategies during impact in Mujoco simulator. **Strategy 1:** Reduce stiffness; **Strategy 2:** reduce velocity; **Strategy 3:** Reduce velocity and stiffness.



(b) Effect of the strategies during impact on a physical robot

Fig. B.8 Comparing the effect of the 3 strategies (reduce stiffness, reduce velocity, and reduce both) in a physics simulator and on a real robot.

B.3.2 Impact-velocity experiments

The plot in (Fig. B.9) show the linear relation between force on impact and approach velocity. The relation is also linear in simulations (Fig. B.10).

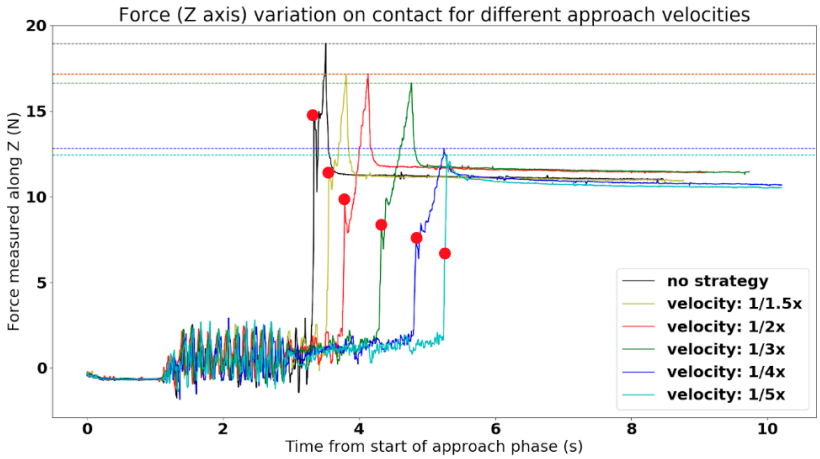


Fig. B.9 Force plots when a robot approaches the same obstacle with different approach velocities.

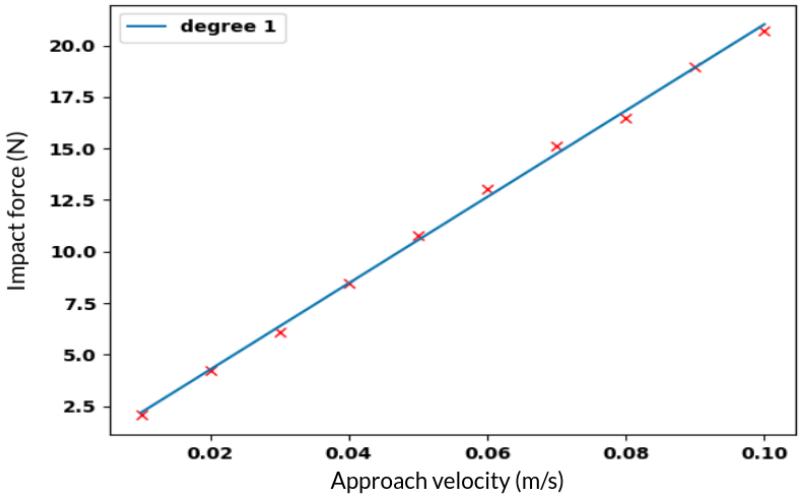


Fig. B.10 The relation between impact force and velocity of approach is much more obvious in simulation. The real world relation is influenced by noise.

B.3.3 Effect of controller stiffness on impact dynamics

Similar experiments were done with a physical robot to study the effect of different controller stiffness on impact dynamics. However, it was observed that although lower stiffness seemed to produce less vibrations on contact, there was no conclusive pattern to formulate a relation between stiffness and jerk/vibration (Fig. B.11). This could also be due to the lower sensitivity of the force-torque sensor or due to controller inadequacies, but the problem was not explored further. It was also clear that reducing stiffness did not seem to have any meaningful relation to the force at impact (Fig. B.12a), but trades of tracking accuracy by bringing delays (Fig. B.12b)

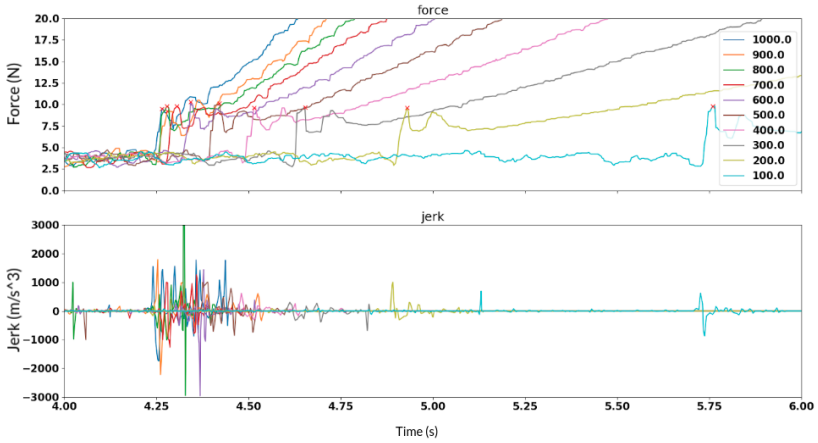
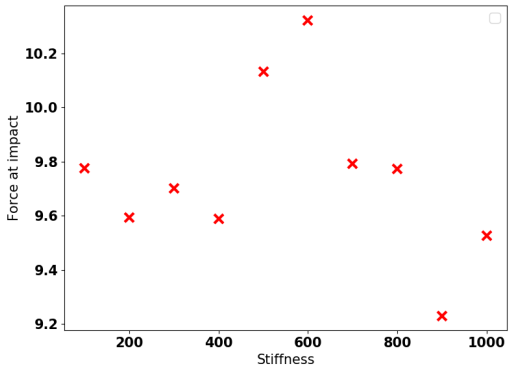
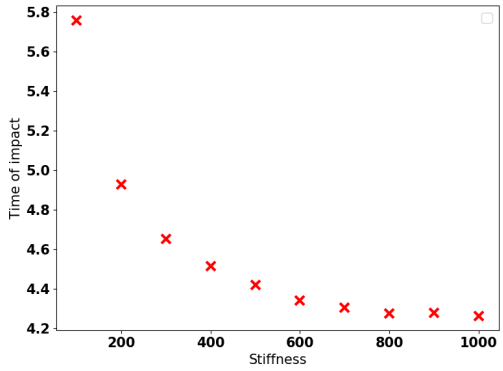


Fig. B.11 Force and jerk plots for a robot making contact with different controller stiffness.



(a) Force vs stiffness. Note that the scale of Y axis is small. The max difference is about $1\,N$, approx $100\,g$, which is the observed resolution of the FT sensor in practice.



(b) Time to impact vs stiffness. Reducing stiffness brings delay in task completion. This is to be expected due to the lag that is introduced by becoming more compliant.

Fig. B.12 Reducing stiffness does not show any meaningful pattern with the change in impact force, but it brings noticeable delays in task completion due to lag in tracking.

B.4 Comparing external FT sensor with internal estimate of robot

The Franka Emika Panda robot provides internal estimation of the end-effector forces and torques that are computed using the robot's internal dynamics model and real-time joint sensor measurements. The experiments in this section were conducted to see how accurate the estimated values were when compared to the measurements from an external FT sensor. The external FT sensor used for comparison was the ATI Gamma 6DoF force-torque sensor.

The ATI sensor is able to provide raw data at 500 Hz, while the internal estimate of the robot is a low-pass-filtered (LPF) estimate that is at the frequency of up to 1000 Hz.

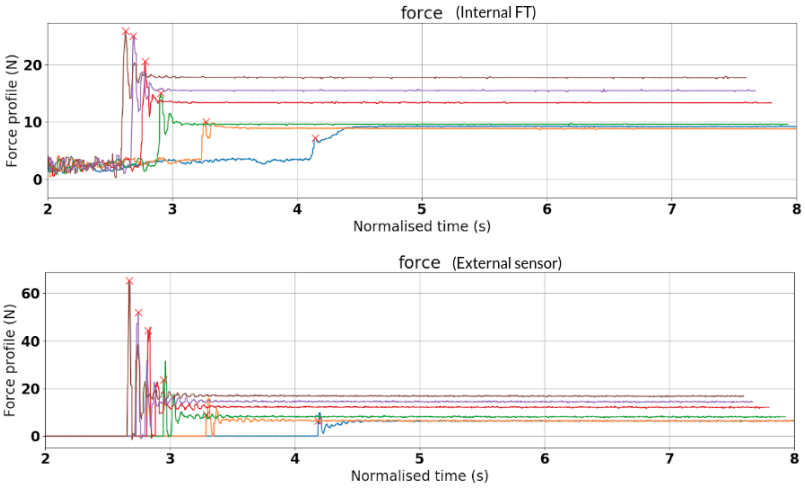


Fig. B.13 Force profiles measured using external FT (top) and internal robot estimates (bottom) for different approach velocities.

Fig. B.14 Task setup for comparing external FT sensor measurements with the robot's internal estimator.

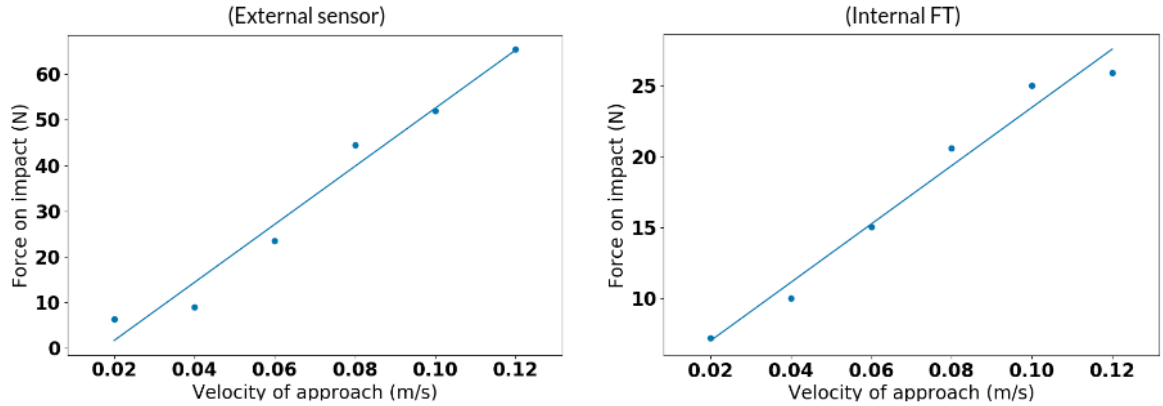


Fig. B.15 Linear fit between impact force and velocity of approach when using measurement from external FT sensor (left) and internal estimate (right).

The experiment was similar to the previous experiment to obtain the relation between impact force and approach velocity. However, this time the robot collided with the FT sensor placed on the table instead of the table itself (Fig. B.14). This way, the impact forces and the subsequent holding forces are measured both by the external FT sensor as well the internal estimator.

It can be seen from Fig. B.13 that the forces measured by the external FT sensor is roughly the same as the internal estimate of the robot except at the impact points. This is

because the internal estimate is smoothed using a low-pass filter which reduces the peaks which occur at impact. However, the measurements are consistent using both modalities when the measured forces are less volatile. This is clear when the robot stays in contact with the surface after impact when the measured forces are equal when using both the sensors.

The effect of LPF smoothing is evident at regions of impact. Even though this produces different peak impact measurements, either of these sensor modalities can be used to successfully build the linear fit between impact force and velocity that we use in Chapter 6 for designing the transition-phase controller for impacts (Fig. B.15).

In the end, we decided to stick with the FT estimates provided by the robot for the following reasons:

- The estimates are close enough to the measured FT for simple tasks and while measuring continuous forces. This is subject to the assumption that the dynamics of all external load is incorporated in the model within the robot controller, or alternatively having a low-weight end-effector or no load at the end-effector.
- Using the external FT sensor would require attaching it to the robot and compensating for the dynamics of the additional load. This is non-trivial and would affect the performance of the AVIC control heavily.
- External FT sensor attached to the end-effector will be affected by the acceleration of the end-effector, and the readings would be unreliable when the robot accelerates. On the other hand, the values estimated by the robot's model should be able to compensate for the motion dynamics while estimating end-effector FT.
- Using the external FT sensor would require writing a low-pass filter to reduce the noise of the raw sensor data (raw noisy data can trigger the 'mode identification' phase even if the robot did not actually change contact mode). This essentially removes all benefits of using the external FT sensor, and would be equivalent to using the internal estimates.
- Using external FT sensor would also require transforming the measured forces to the base frame of the robot so as to use it for control.
- Using the external sensor would require including another computer in the multi-computer setup (see Appendix C.2).

Appendix C

Regarding hardware and implementation

C.1 Hardware used

This appendix gives information about the main hardware used for the experiments in this thesis.

C.1.1 Franka Emika Panda robot

This 7-DoF robot was the main hardware that was used for most real-world experiments presented in this thesis (e.g. Fig. 6.11). The robot can be controlled easily using their web interface, which has a very easy-to-use block-style programming interface. The interface can also be used to provide the dynamics information of custom end-effectors that are attached to the robot, which is then used by the internal controller of the robot for providing dynamics compensation during motion. The company also provides a research suite (C++ library and ROS library) for performing more low-level control from a machine running a real-time Linux kernel. Using the suite, the robot can be controlled using low-level joint torques, position and/or velocity, as well as Cartesian position, velocity and force. The Panda has joint torque sensors which is also used to provide estimates of end-effector wrenches. It is very sensitive in terms of feedback and has safety features that stop the robot when it detects collisions or high forces. It also comes with a detachable position-controllable two-finger gripper.

C.1.2 Rethink Robotics Sawyer robot

The Sawyer is a single-arm robot from Rethink Robotics, which was used for some of our experiments (e.g. Fig. 5.3). This robot consists of serial elastic actuators and has seven degrees of freedom. The robot can be controlled via ROS and provides position, velocity and torque control modes. It also has built-in force-sensing capabilities and the passive compliance makes it safe for human collaborations.

C.1.3 ATI Gamma F/T sensor

The ATI Gamma Multi-Axis Force/Torque Sensor system measures all six components of force and torque acting on its measuring plate. The system consists of a transducer, shielded high-flex cable, and intelligent data acquisition system, Ethernet/DeviceNet interface or F/T controller.

This sensor was used intensively for experiments with the Sawyer robot (see our paper [124]), but was not used much with the Franka mainly for two reasons: (i) the dynamics compensation of the Franka robot required accurate model of the sensor at the end-effector, (ii) the F/T estimates provided by the robot was good enough for our purposes (see Appendix B.4).

C.1.4 Regarding custom end-effectors used

The main tasks considered in the thesis are sliding tasks where the robot has to slide its end-effector on different surfaces. For this purpose, we first used a whiteboard eraser attached to the end-effector (see our paper [124]), then used a custom-cut wooden block (e.g. [175]), and finally a 3D-printed block of similar dimensions (e.g. Fig. 6.11).

The overall framework presented in this thesis works by identifying surfaces of different friction and being able to use appropriate forward models to perform variable impedance control to navigate the surface reliably. The first end-effector used (a real whiteboard eraser) had a soft face for sliding on the table. This was fine for testing the AVIC framework for continuous dynamics.

However, when developing the hybrid framework for PWC dynamics, the soft surface of the eraser often made it difficult to distinguish between surfaces of different friction as it softened the frictional resistance. Therefore, we moved to using a more reliable and solid wooden block. Having a hard face making contact with different surfaces made differentiating between surfaces of different friction easier since this provided more reliable readings from the interactions.

The wooden block was later replaced with a 3D printed model of the block to make it have more perpendicular and flat faces, as well as to avoid the wear and tear that was showing up on the wooden block.

C.2 Practical implementation details

The overall setup used for experiments with the robots typically consisted of a multi-computer setup. One computer typically ran the low-level controller that creates the high-frequency control bridge between the robot and other machines in the network (this was usually done with the *Franka ROS Interface* control node described in Appendix D.1 running on a real-time kernel at $1\,kHz$). This computer also often has the learners (forward models of AVIC, mode detection module, Kalman-filter anticipation module, velocity profile updater, etc.) running on it, unless they are running on a separate machine for efficiency. Another computer is used for running the AVIC controller loop which runs at $500\,Hz$ and listens to target commands (target position, velocity, force, torque, impedance parameters, etc.). These targets are provided by the final computer which usually runs the script for the experiments as well as the logging of experiment data.

C.2.1 Hyper-parameter selection in framework

Controller in AVIC

The main hyperparameters for the controller in the AVIC framework are the parameters for IGMM (Section A.1.2). They are typically set depending on the task and are often relatively straight-forward to tune with minimal trials. They are chosen such that there are not too many components in the model to slow the learning and prediction, while there are enough to capture the variation in a continuous contact mode. The novelty criterion is also selected such that new components are only created if there is enough evidence (the observation is persistent for some time). This way the model is not updated immediately with new information when the contact mode changes; in such cases, the robot will switch to mode identification phase and not use the new information to update the model of the previous mode.

The gain matrices for the impedance controller (stiffness and damping) are left as 3×3 diagonal matrices (separate for translation and rotation) and each diagonal term is typically guided by the accuracy of the predicted end-effector force (or torque for rotational gain matrices) along the corresponding direction. So the translational stiffness along X is high if the accuracy of predicted force along X is low, and so on.

The minimum and maximum stiffness ($\mathbf{K}^{\mathbf{p}}_{free}$ and $\mathbf{K}^{\mathbf{p}}_{max}$) are set as follows: $\mathbf{K}^{\mathbf{p}}_{free}$ is chosen as the stiffness required by the robot to achieve a required tracking accuracy when it makes a similar motion in free space, while $\mathbf{K}^{\mathbf{p}}_{max}$ is the minimum stiffness required to move in the presence of high resistance using a standard task-space fixed gains PD controller. These values are heavily dependent also on the control loop rate as well as the resolution of the trajectory (spacing between consecutive poses in the task-space plan). In practice, however, the $\mathbf{K}^{\mathbf{p}}_{max}$ is often set to a high value that the robot will not have to use in most situations, while $\mathbf{K}^{\mathbf{p}}_{free}$ is set to a low value such as $100 N/m$.

Mode detection module

The mode identification module is active only when the robot detects a sudden discontinuity. When the robot detects a contact change, the mode detection module is activated and collects measurements till it is certain of the identity of the new mode. The parameters for the mode detection module are typically the hyperparameters of the BIRCH clustering algorithm, and includes the maximum number of modes and a measure of 'distance' from existing clusters to identify observations as a new cluster. The maximum number of modes is typically selected as an arbitrary number higher than the actual number of modes. The 'distance', on the other hand, is a highly sensitive hyperparameters that requires some trial-and-error runs with the real system to converge to a value that is good enough to distinguish between contact modes without finding too many modes for small differences in observations.

Another important hyper-parameter is the minimum confidence required to accept the detection as valid. This choice is usually dependent on the sensor used and the total number of modes in the task, and is usually safe to leave somewhere between 55 and 70%. The batch size is also an important parameter that determines the number of observations to be collected before updating the model. Having a higher number helps in removing the noise from the measurements by averaging, while also slowing down the update. This value is typically chosen to be closer to a 100, and can still update the model at over $10 Hz$.

Contact-change-handling module

The Kalman filter-based anticipation model has mainly two noise parameters that determine how noisy the sensor and action models are assumed to be. This is chosen depending on the accuracy of the kinematic model of the robot. In our case, this was usually set to about 0.05 (approximating an error of $5 cm$ in the end-effector position given by forward kinematics).

The desired duration of transition T for switching from the default controller to the transition-phase controller is normally set to a value between $0.5 - -1 s$. The same value is

used for both the controller interpolation and velocity remapping (Eqs. (6.10) and (6.11)) such that the controller switching completes smoothly.

If a linear model is not available, the initial value of the approach velocity in the transition-phase for impacts is set to $0.01\,m/s$. The low stiffness value ($K_{[tr]}^{\mathbf{p},low}$ in Eq. (6.12)) for transition-phase controllers for impacts is set to a low value such as $100N/m$. The learning rate β in Eq. (6.8) is usually set to a low value such as 0.01 unless the slope of the line relating impact to approach velocity is known. If the linear model is available, the learning rate is set to a value that is lower than the slope by 0.005.

C.2.2 Safety considerations

Low-level control

In practice, the AVIC controller is written over a low-level torque controller that has several safety checks and considerations such as:

- Limits on torques, accelerations, velocities, joint positions, and rate limits.
- Network latency checks (for multi-computer setup).
- Safety checks when switching between controllers or control dimensions (for e.g., directions of force control).
- Checks for unexpected collisions and high forces.

The Franka robot's internal control also has its own layer of safety checks for checking if the robot is violating constraints in acceleration, force, torques, etc. There is also a kill-switch which can be used to kill the power to the robot in extreme cases.

Unexpected contacts

While following a trajectory, the user has to define the behaviour when the robot encounters a contact that is not anticipated. It is possible to allow the robot to update its belief about contact locations as demonstrated in Section 6.7.6. Alternatively, the user can also specify that the robot should abort the task or ask for a new plan from the planner in such scenarios. Unexpected impact-less transitions are always handled as demonstrated in Section 6.7.5 by incorporating them into the list of anticipated contact changes for the next trial(s).

The robot aborts the plan if it does not find a contact it expects after attempting to make contact for some distance.

Appendix D

Software developed during research

This section briefly goes over some of the software developed during the course of this research.

D.1 *Franka ROS Interface*

Franka ROS Interface [173] is a ROS/Python interface library for the Franka Emika Panda robot, extending the official *franka-ros* (https://github.com/frankaemika/franka_ros) library to expose more information about the robot, and providing low-level control of the robot using ROS and Python API. It provides utilities for controlling and managing the Franka Emika Panda robot (real and simulated). It also provides interfaces for controlling the joints at low level using position, velocity and torque control. It also has interfaces for the gripper, controller manager, coordinate frames interface, etc. and directly supports motion planning and execution using MoveIt! and ROS Trajectory Action & ActionClient. This package also provides almost complete sim-to-real / real-to-sim transfer of code with the *Panda Simulator* package (see Appendix D.3).

Source code: https://github.com/justagist/franka_ros_interface

Official webpage and API documentation: https://justagist.github.io/franka_ros_interface

D.2 *PandaRobot*

PandaRobot is an extension to *Franka ROS Interface* which unifies several functionalities of the package to provide a single unified interface class with simplified API that can be used directly in Python scripts. The *PandaRobot* package provides simple API methods for low- and high-level motion control of the robot. It also provides functionalities to obtain

real-time kinematics and dynamics information from the robot. It is also fully supported with the *Panda Simulator* (Appendix D.3) package providing almost complete sim-to-real code transfer.

Both the *Franka ROS Interface* and the *PandaRobot* packages are open-source packages hosted on GitHub which are being used by many robot programmers, and have grown to become a collaborative projects with multiple contributors from around the world.

Source code: https://github.com/justagist/panda_robot

Official webpage and API documentation: https://justagist.github.io/panda_robot

D.3 *Panda Simulator*

Panda Simulator [174] is a Gazebo simulator for the Franka Emika Panda robot with ROS interface, providing exposed controllers and real-time robot state feedback similar to the real robot. This software was developed initially to test the framework in a simulated environment before deploying the code on a real robot. The intention was to provide a simulated environment that responds similar to the real robot without having to change the code.

The simulator has low-level controllers (joint position, velocity, torque) available that can be controlled through ROS topics (including position control for gripper) or Python API. It provides real-time robot state (end-effector state, joint state, controller state, etc.) which are available through ROS topics. It is directly controllable using *Franka ROS Interface* and *PandaRobot* APIs providing direct sim-to-real code transfer. It also supports planning using MoveIt for the robot arm as well the gripper.

It is an open-source package hosted on GitHub which is now being used by many robot programmers, and has grown to become a collaborative project with multiple contributors from around the world.

Source code: https://github.com/justagist/panda_simulator

D.4 Packages used for experiments

These are some of the open-source packages developed for performing some of the experiments in this thesis. Some are direct implementations of certain sections/components of the presented overall framework.

D.4.1 AVIC in simulation

This package contains the complete AVIC incremental learner and controller, as well as the custom-built 2D environment used for comparing AVIC with other adaptive control strategies. The package also contains the implementations of the three adaptive control strategies used for the experiments in Section 4.4, along with the scripts for running the experiments in the simulated environments. The package primarily uses Box2D physics.

Source code: https://github.com/justagist/variable_impedance_sim_experiments

D.4.2 Custom velocity profile

This repository contains the code for the custom-built velocity profile (Eq. (6.11)) used in the transition-phase for collisions, along with a few other velocity profiles from literature. It also contains the script used for time-remapping of the position trajectory using any of the available velocity profiles.

Source code: https://github.com/justagist/velocity_profile_for_pos_traj

D.4.3 Kalman filter

Source code: https://github.com/justagist/kalman_filter_python

This repository contains the basic kalman-filter implementation used for creating the contact anticipation model described in Section 6.3.

D.4.4 Covariance ellipsoid

This repository contains the code for the ellipsoid representation used for denoting regions of anticipated mode transitions described in Chapter 6. It can create n-dimensional ellipsoids in a space based on the mean, covariance and confidence level provided.

Source code: https://github.com/justagist/covariance_ellipsoid_python

D.5 Other libraries and packages

D.5.1 Advanced Manipulation and Learning (AML) Library

The AML code stack is an internal code stack used by all research students at the Intelligent Robotics Lab (IRLab), especially by those working on robot manipulation. It was started by Ermano Arruda and Michael Mathew, and is now being primarily developed and maintained by me. The library includes unified interface classes that can be used to control and monitor

all the manipulators (RethinkRobotics Saywer & Baxter, Franka Emika Panda, Kuka LWR) and grippers (DLR Hand, IIT Pisa Softhand, Reflex Takktile 2) in the lab without much change in code. The library also includes physics simulators (Mujoco, Bullet, ODE, Box2D) for these robots as well as several simulated environments for ML experiments. AML also provides implementations or interfacing for several learning algorithms (supervised, unsupervised, reinforcement, deep). It also has a module for several joint-space and task-space high-level controllers that can be used on any of the robots in the lab, as well as utilities for task planning, trajectory optimisation and learning from demonstration. The stack is written in Python and C++. It makes use of ROS (supports versions Kinetic and Melodic fully, and partially Noetic) to create a general package for robot control, manipulation and grasping.

D.5.2 *PybulletRobot*

PybulletRobot is a python interface library for controlling any manipulator in simulation using Bullet Physics engine. It provides a common interface class that can be used to control, monitor and manage any manipulator robot whose URDF model is provided. It includes functionalities to compute forward and inverse kinematics and dynamics, change end-effectors, add other objects in the simulated world, etc.

Source code: https://github.com/justagist/pybullet_robot

D.5.3 *MujocoPanda*

MujocoPanda provides a simulation of the Franka Emika Panda robot using Mujoco physics engine, as well as python interface for controlling the simulated robot. It provides all the functionalities provided by the *PybulletRobot* (see above) interface, allowing users to switch between simulators with minimal code change.

Source code: https://github.com/justagist/mujoco_panda