CONTACT-BASED STATE ESTIMATION AND POLICY LEARNING FOR ROBOTIC MANIPULATION TASKS

By

Shuai Li

DOCTOR OF PHILOSOPHY

Major Subject: COMPUTER SCIENCE

Examining Committee:

Jeff Trinkle, Dissertation Adviser

Siwei Lyu, Member

Barbara Culter, Member

Stacy Patterson, Member

John Wen, Member

Rensselaer Polytechnic Institute
Troy, New York

June 201
# CONTENTS

LIST OF FIGURES ................................................................. v

ABSTRACT ............................................................................. 1

1. Introduction .................................................................... 1
   1.1 The Problem ............................................................. 2
   1.2 Our Approach .......................................................... 4
   1.3 Contributions ........................................................... 6

2. Previous Works ............................................................. 8
   2.1 State Estimation for Robotic Manipulation ................... 9
      2.1.1 State Estimation Without a Full Model of Multi-Body Dynamics .................................................. 9
      2.1.2 State Estimation With Full Model of Multi-Body Dynamics ......................................................... 11
      2.1.3 Summary ........................................................... 12
   2.2 Policy Learning and Execution for Robotic Manipulation Tasks .......................................................... 12
      2.2.1 Summary ........................................................... 15

3. Background .................................................................... 16
   3.1 The Model of Multi-Body Dynamics ............................... 16
      3.1.1 Newton-Euler Equations and Joint Constraints ............................................................. 17
      3.1.2 Contact Constraints .............................................. 21
      3.1.3 Complementarity Formulation of the Model of Multi-Body Dynamics ........................................... 25
   3.2 Bayesian Filtering Methods ........................................... 29
      3.2.1 Dynamic Bayesian Network and Bayesian Filtering ......................................................... 30
      3.2.2 Kalman Filters ..................................................... 33
      3.2.3 Sequential Monte Carlo Methods ............................................ 38
   3.3 Contact Graphs ........................................................... 48

4. Modeling Options Evaluation .......................................... 52
   4.1 Experiment Setup ....................................................... 57
   4.2 Contact Dynamic Models ............................................. 58
      4.2.1 Rigid Body Model ............................................... 59
      4.2.2 Compliant Body Model ......................................... 59
4.3 Filtering Methods .................................................. 60
  4.3.1 Adding Noise to the State Transition Model ............... 60
  4.3.2 Projected Particle Filter .................................. 62
  4.3.3 Force State Particle Filter .................................. 63
4.4 Evaluation Results and Discussions .............................. 66
  4.4.1 Results and Discussions .................................... 68

5. Contact-Based Rao-Blackwellized Particle Filters for State Estimation .... 78
  5.1 Contact-Based Rao-Blackwellized Particle Filter Design ........ 79
  5.2 Contact-Based Rao-Blackwellized Particle Filter Implementation .... 83
    5.2.1 Stick-Slip Contact Graph and Contact State Sampling ....... 83
    5.2.2 Converting the LCP Model with Known Contact States ....... 90
    5.2.3 Kalman Filter-Based Continuous State Update ............... 91
    5.2.4 Algorithm Summary ...................................... 92
    5.2.5 Additional Problems for Three-Dimensional Manipulation Tasks 93
  5.3 Experiments ..................................................... 99
    5.3.1 Physical Experiment Setup ................................ 99
    5.3.2 Results and Discussions .................................. 102
  5.4 Conclusion ..................................................... 115

6. Contact State-based Policy Learning for Robotic Manipulation Tasks ....... 117
  6.1 Policy Learning for Robotic Manipulation Tasks ............... 118
    6.1.1 Reinforcement Learning .................................. 118
    6.1.2 Manipulation State and Manipulation Graph ............... 122
    6.1.3 Reinforcement Learning with the Manipulation States ..... 127
  6.2 Closed-loop Policy Execution for Robotic Manipulation Tasks .......... 131
  6.3 Experiments ..................................................... 133
    6.3.1 Results and Discussion .................................. 136
  6.4 Conclusion ..................................................... 141

7. Conclusion and Future Work ........................................ 143
  7.1 Conclusion ..................................................... 143
  7.2 Future Work ................................................... 144

APPENDICES

iv
A. Detailed Derivations for Chapter 5 ........................................ 147
   A.1 Contact-Based RBPF Conditional Distribution Derivation .......... 147
   A.2 Converting LCP Models to Linear Models with Known Contact State 148
   A.3 Derivation of the Inverse K Matrix .................................. 150
      A.3.1 Contact Normal Impulse Vector ................................. 151
      A.3.2 Contact Frictional Impulse Vector .............................. 154
      A.3.3 Computing the Inverse of Matrix K .............................. 155

REFERENCES ............................................................................. 157
LIST OF FIGURES

1.1 Robotic manipulations in the DARPA Robotics Challenge. .................. 1

1.2 A robot picks up a package from a shelf in the Amazon Picking Challenge [2]. .......................................................... 3

1.3 A diagram of our proposed solution. ........................................... 5

3.1 Coordinate frames for the $i$th contact between two colliding bodies. $\Lambda_{ji}$ and $\Lambda_{ki}$ are the contact frames attached to the contact points and $\Psi_{im}$ is the signed distance in the contact normal direction between the two contact points of the $i$th contact. $\hat{n}_i$, $\hat{t}_i$ and $\hat{o}_i$ are the contact normal unit vector and the contact tangent vectors respectively. .............. 21

3.2 Friction cone (blue) approximated by a polyhedron (black) with direction vectors $d_j$. The figure shows the limits projected on the contact tangent plane. .................................................. 25

3.3 An example of a multi-body dynamic system, where a robot pushes an object that slides on a plane. The object tilts once it makes contact with the small obstacle on the plane. .............................. 27

3.4 The configuration of the contacts when the object slides on the plane. 28

3.5 A diagram of a simple Dynamic Bayesian Network. $x_t$ represents the vector of the hidden states of the system, and $z_t$ represents the vector of the measurements. The arrowed lines represent dependencies among the states. .................................................. 31

3.6 Visualization of the two steps of a Kalman filter. In the prediction step, the initial distribution (blue) is updated to the predicted distribution (orange). In the updating step, the predicted distribution is combined with the measurement distribution (yellow) and is updated to the posterior distribution (purple). .................. 35

3.7 The sigma points (red dots) for a two-dimensional Gaussian distribution whose contours are projected on the xy-plane. ................. 38

3.8 One iteration of a particle filter. $x_t$ and $x_{t+1}$ are the state vectors at time steps $t$ and $t + 1$, and $z_t$ and $z_{t+1}$ are the corresponding measurement vectors. .................................................. 43

3.9 A diagram of a SLDS model. $r_t$ is the root state variable, $y_t$ is the leaf state variable, and $z_t$ is the measurement variable. .................. 47
3.10 Three types of “principal” contacts between two squares in 2D. Each type consists of two boundary elements being in contact.

3.11 An example of two PCs between two squares. $PC_1$ is an edge-edge PC with edges $c^A$ and $c^B$. $PC_2$ is a vertex-edge PC with vertex $d^A$ and edge $c^B$.

3.12 The boundary elements of two rectangles. $D_i$ and $d_i$ represent the vertices, and $E_i$ and $e_i$ correspond to the edges.

3.13 An example of a partial contact graph of two rectangular objects. The red dot is used to mark one particular vertex of the smaller rectangle.

4.1 The piecewise linear property of the LCP model illustrated with the example from section 3.1.3.1. The green curves represent the trajectories of the object’s center of gravity.

4.2 The five contacts involved in the example introduced in section 3.1.3.1.

4.3 An example of a bi-modal contact state distribution. A square drops onto a triangular obstacle. The square can slide along either side of the obstacle after the contact. Therefore, the distribution of the contact state is bi-modal.

4.4 The graphical representation of the one-dimensional experiment, where a robotic arm moves into contact with a fixed wall.

4.5 The constraint boundary surface/line for our one-dimensional experiment. The state space that consists of the velocity $v_t$ and the position $q_t$ is shown as a two-dimensional plane. The red area indicates the valid half state space.

4.6 The graphical structure of $FSPF$.

4.7 The setup for the physical experiments. The Barrett WAM moves into contact with a wall made of a wooden frame fixed on a table.

4.8 The WMSE results generated by particle filters with state transition models using two contact dynamic models. The experiment data is generated by simulation experiments with the rigid body model. Data points represented as $\triangle$ correspond to filters with the rigid body dynamic model in their state transition models, and data points shown as * correspond to filters with the compliant body dynamic model in their state transition models.
4.9 The WMSE results generated by particle filters with state transition models using two contact dynamic models. The experiment data is generated by simulation experiments with the compliant body dynamic model. Data points represented as \( \triangle \) correspond to filters with the rigid body dynamic model in their state transition models, and data points shown as * correspond to filters with the compliant body dynamic model in their state transition models.

4.10 The WCCE results generated by particle filters with state transition models using two contact dynamic models. The experiment data is generated by simulation experiments with the rigid body dynamic model.

4.12 Screenshots of a rigid body simulation at three time steps, where the green block represents the robotic hand and the black block represents the wall. The experiment starts with a penetration between the robotic hand and the wall.

4.11 The WCCE results generated by particle filters with state transition models using two contact dynamic models. The experiment data is generated by simulation experiments using compliant body dynamic model.

4.13 Screenshots of a compliant body simulation at three time steps, where the green block represents the robotic hand and the black block represents the wall. The experiment starts with a penetration between the robotic hand and the wall.

4.14 The WMSE results generated by particle filters with state transition models using two contact dynamic models. The experiment data is generated by physical experiments.

4.15 The WCCE results generated by particle filters with state transition models using two contact dynamic models. The experiment data is generated by physical experiments.

5.1 The graphic model of the Dynamic Bayesian network of our proposed particle filter.

5.2 A partial stick-slip contact graph of a triangle and a wall. Each rectangle represents a node in the graph. The small picture in each node depicts the contact formation of that node. The top left corner of a node is the index of the node and the top right corner represents the number of contact states sharing that contact formation.
5.3 An example of calculating $\Psi_v$ for a planar dynamic system, which consists of a square and a wall. The superscripts represent the indices of the sigma points. There are two distinct potential contacts: c1 and c2. $\Psi_v$ is calculated as the weighted mean over all the sigma points.

5.4 An example of the sigmoid function ($S$) of the gap distance $\Psi$. The value of the sigmoid function rapidly approaches one as the gap distance moves towards the region $(-\infty, \delta]$, where $\delta$ is a small positive value and can be approximated as $\frac{10}{\pi}$.

5.5 An example of calculating the elements of $v_p$. There are two distinct contact pairs between the robot and the triangular object, and each element of the vector $v_p$ is calculated using the sigmoid function of the gap distance of each contact pair.

5.6 An example of an approximated contact graph laying over the complete contact graph. The green portion is the approximated contact graph.

5.7 Screenshots of two simulation experiments. Each row shows the initial configuration (the picture on the left) and the final configuration (the picture on the right) of one experiment.

5.8 An overview of the three-dimensional experiment setup.

5.9 An overview of the planar experiment setup.

5.10 Physical experiment setup for a planar grasping experiment, where a robotic parallel jaw gripper tries to grasp a triangular object on a plane. The gripper is approximated by a fixed frame and Barrett hand.

5.11 The setup of a physical robotic manipulation experiment, where a Barrett hand tries to grasp a rectangular cube that is placed on the table. The Kinect sensor on the table captures the point cloud of the manipulation process.

5.12 The point cloud of a scene from two viewing angles in a robotic manipulation task, where a Barrett hand tries to grasp a rectangular cube.

5.11 An example of the estimated and ground truth trajectories for one planar grasping experiment, where the robotic gripper grasps the triangular object by moving one finger while keeping the other finger fixed.

5.12 The labeling for the vertices of the triangle in our planar experiment.

5.13 The WMSE of both CPF and contact-based RBPF over ten planar grasping experiments.

5.14 Contact states estimation from contact-based RBPF.
5.15 The WMSE of CPF and contact-based RBPF as their numbers of particles increase from 4 to 160 in the planar grasping experiments.

5.16 The setup of the three axes for our three-dimensional grasping experiments.

5.17 An example of the estimated and ground truth trajectories for one three-dimensional grasping experiment, where the robotic gripper knocked over the rectangular cube that it tried to grasp.

5.18 The WMSE of both CPF and contact-based RBPF over 25 three-dimensional grasping experiments.

5.19 The WMSE of CPF and contact-based RBPF as their numbers of particles increase from 3 to 150 in the three-dimensional grasping experiments.

6.1 A example of a simple reinforcement learning problem. On this grid map, a robot starts from state A and try to find a policy that navigate itself to the goal state (green square). The red squares represent dangerous regions, and the robot will get penalties when it reaches the red squares.

6.2 An example of a partial contact graph of a robotic manipulation task, where a robotic gripper picks up a cup from the table. The green arrows show the steps that a robot can follow so that the task can be completed.

6.3 An example in a cup-picking robotic manipulation task, where the robot (the gripper) is outside the discretization sphere of the object (the cup).

6.4 A robot pushes a rectangle at different locations within the same contact formation. The case on the left will generate a torque that rotates the object clockwise, and the case on the right will rotate the object counterclockwise.

6.5 An example of two contact states for a triangular object. Each small rectangle represents a contact sensor, and the triangle on each contact sensor represents the range of angles in which the sensor can be triggered. $c_1$ is defined as a contact state that has both green contact sensors triggered, and $c_2$ is a contact state with both red contact sensors triggered.

6.6 An example demonstrates a simple directed manipulation graph with a robot (red) and an object (green). The red lines correspond to the graph if the robot moves to the right and the blue lines show the graph when the robot moves to the left.
6.7 An example of a partial manipulation graph of a robotic manipulation task, where a robotic gripper picks up a cup from the table. The green arrows show the steps that a robot can follow so that the task can be completed.

6.8 The setup for our manipulation task.

6.9 An example of projecting a contact with a contact point and contact normal vector (in pink) from the three-dimensional space to a contact point and contact normal vector (in blue) in the two-dimensional plane (the red plane in the figure).

6.10 The setup of the physical experiment.

6.11 The learning curve in terms of the mean Q values for one learning process.

6.12 The average learning curve in terms of the mean Q values of both manipulation state-based (left) and continuous state-based (right) algorithms over 50 learning processes. The shaded areas represent the mean standard deviation of the mean Q values.

6.13 Eight manipulation states examples each representing a case where one of the eight actions is optimal. The red arrows in the figures represent the direction and the magnitude of the action (the velocity in our case) of the robot. The x and y-axes are the axes of the two-dimensional plane.

6.14 Six key frames of a successful physical manipulation experiment, where the robot successfully pushed the object into the hole.

6.15 Six key frames of a failed physical manipulation experiment, where the robot got stuck as a result of the unexpected friction coefficients of the object’s surface.

7.1 A diagram of our approach with algorithms developed in this thesis.

A.1 Two objects come into contact.

A.2 A square is in contact with a vertical wall. The contact impulse is \( p_t \). The mass of the square is \( m \). Because of the angular velocity generated by the contact impulse, the velocity change at the contact is not \( \frac{p_t}{m} \).
List of Algorithms

1. Kalman Filter ......................................................... 35
2. Extended Kalman Filter ........................................... 37
3. Particle Filter ....................................................... 44
4. Particle Filter Update State ...................................... 44
5. Resample Particle Filter .......................................... 46
6. NSPF/NIPF Update State ........................................... 61
7. PPF Update State .................................................... 64
8. FSPF Particle Filter ................................................ 65
9. Calculate the Weighted Gap Vector ............................ 87
10. Sample Contact State ............................................... 89
11. Contact-Based RBPF ............................................... 93
12. Continuous State Particle Filter ............................... 105
13. DynaQ Algorithm ................................................... 121
14. Manipulation Skills Reinforcement Learning Using Manipulation States 131
The robotic manipulation problem is very important in robotic research and applications. In a typical robotic manipulation task, the robot needs to interact with certain objects to accomplish a goal, such as picking up a cup from a table. Without manipulation capabilities, robots will not be able to help humans with their daily tasks such as using tools to fix a broken car. Although the hardware of robots has been significantly improved, the ability to perceive the current state of a robotic manipulation task is still essential for a robot to fully utilize its hardware. However, less effort has been made to address the perception problem for robotic manipulation tasks.

In this thesis, we focus on improving the perception capability for robots and addressing the problem of combining the perception capability with action planning and execution for robotic manipulation tasks. Our proposed approach combines Bayesian filtering methods with accurate models of multi-body dynamics for state estimation in the robotic manipulation tasks. In order to understand the design trade-offs of particle filter applications for the state estimation problems, we evaluate different particle filter modeling options in both simulation and physical experiments. We then propose a contact-based RBPF that samples the discrete contact states and updates the continuous state distribution through Kalman filters. Results show that the contact-based RBPF is more effective and more efficient than the state of the art filters that sample the continuous state space. Finally, we apply reinforcement learning algorithms to learn policies for robotic manipulation tasks with a state space discretized using contact states. This discretized space learning is proven to be more effective than learning with continuous state space. We further propose to combine the learned policies with the contact-based RBPF for online action selection during robotic manipulation tasks.
CHAPTER 1
Introduction

The robotic manipulation problem is very important in robotic research and applications. In a typical robotic manipulation task, the robot needs to interact with certain objects in order to accomplish a goal, such as picking up a cup from a table. Without manipulation capabilities, robots will not be able to help humans with their daily tasks such as using tools to fix a broken car. Recently, the DARPA Robotics Challenge [5] demonstrated some of the most challenging robotic manipulation tasks attempted by robots to date. For example, in Fig. 1.1a, a robot tries to pick up a drill from a shelf. Because of the limits of the robot vision sensors, it has to perform the task without knowing the exact pose of the drill. Fig. 1.1b demonstrates an even harder problem, where the robot needs to use an electric saw to cut a hole in a wall. The robot needs to pull the trigger of the electric saw as well as move the saw, with disturbances from cutting the wall, in a smooth circular trajectory.

(a) A robot tries to pick up a drill [15]. (b) A robot tries to cut a hole in a wall [10].

Figure 1.1: Robotic manipulations in the DARPA Robotics Challenge.

As robots are being integrated into human-dominated environments, their ability to manipulate everyday objects needs to be significantly improved. Although we can re-engineer these objects to make robotic manipulation easier, it is not recommended for several reasons. First, not everything can be re-designed for easy robotic manipulation (e.g., fruits and vegetables). Second, a huge investment has already
been made to engineer objects to be easily manipulated by humans, such as spray bottles, and the cost of redesigning these objects would be too high. To be more useful, a robot should be able to manipulate objects designed for human users. For the aforementioned reasons, many robotic hands have been built to be anthropomorphic. These hands are supported by high-speed controllers acting on readings from tactile, position, and force/torque sensors. In a robotic manipulation task, a robot perceives the states of the task and based on its current perception, it chooses actions for its hardware to execute in the physical world. Although the hardware of robots has been significantly improved, the ability to perceive the current state of a robotic manipulation task is essential for a robot to fully utilize its hardware, and less effort has been made to address the perception problem for robotic manipulation tasks.

\[
\text{Therefore, the focus of this thesis is on improving the perception capability for robots and addressing the problem of combining this capability with action planning and execution for robotic manipulation tasks.}
\]

1.1 The Problem

There are many approaches to executing robotic manipulation tasks. These can be thought of as somewhere between the extreme minimalist and maximalist approaches. The former seeks to design controllers based on large volumes of training data without detailed knowledge of the underlying physical system, while the latter relies on intricate physical models. Regardless of where on the spectrum a method sits, it can benefit from quantitative information such as the poses of the objects and qualitative information such as whether a contact has formed between two objects.

In practical applications, a robot usually cannot have accurate quantitative and qualitative information due to the limitations of the sensors and the unknown conditions of robotic manipulation tasks. Without having such information, many robotic manipulation tasks will fail. As a practical example, consider a robot designed to pick up packages from shelves in an Amazon storage facility (see Fig. 1.2). Once an order comes in, the robot is asked to pick up a package from a shelf. The robot first identifies the target package and estimates its pose on a cluttered shelf
using image sensors. Because of possible occlusions from adjacent packages, the estimate can be inaccurate. Therefore, while the robot attempts to grasp the package, it should keep estimating the pose “on the fly.”

Once the robot has moved its gripper to the desired location and is ready to execute a grasp, it should plan actions for its gripper to execute based on its knowledge of the current state. For example, it is desirable for the robot to apply the appropriate amount of force on the package. Applying too much or not enough force could either break or drop the package. Therefore, the robot needs to know whether or not the package is in contact with the gripper, the shelf, or the surrounding packages in order to execute the grasp. The tactile sensors and the force torque sensors can provide information about these contact states. However, as a result of the electronic noise in the circuits of these sensors, their outputs are usually noisy. As a result, the robot must constantly estimate the contact states and plan actions based on its estimation during the grasping and picking executions.

Figure 1.2: A robot picks up a package from a shelf in the Amazon Picking Challenge [2].

In summary, to successfully perform a robotic manipulation task, a robot should combine measurements from all available sensors (e.g., vision, tactile, and force/torque) with a physical model to accurately estimate the objects’ poses and contact states and take actions based on its estimations in order to complete all grasping and picking tasks. One can view the perception of robotic manipulation tasks as a broad problem that in-
cludes estimating the poses of the objects and predicting contact states. Getting this information is critical for a robot to choose actions during a task. Further, the optimal actions that a robot should take depend on not only the poses of an object but also its contact states. However, current solutions for robotic manipulation tasks still have limitations, such as the lack of accurate dynamic models and capabilities of estimating contact states in the perception algorithms, which make them unsatisfactory for many robotic manipulation applications (see Chapter 2 for further discussions).

1.2 Our Approach

In order to find a better solution to the problem described above, we propose to combine Bayesian filtering methods with a model of multi-body dynamics for state estimation. Based on this filter, we develop algorithms that select optimal actions for a robot in robotic manipulation tasks. Our proposed filtering methods update the joint probability distribution of the objects’ poses and contact states given all past and current sensor data and system inputs. From this joint distribution, our filter infers accurate estimates of the critical information needed for robotic manipulation tasks, such as whether or not contacts have formed between the robotic hand and the object. The robot then maps this estimated states to actions that it will execute to finish a manipulation task. A mapping between the states and the actions is also called a policy in artificial intelligence planning and reinforcement learning. In order to choose actions optimally, a reinforcement learning algorithm is incorporated with states that contain both continuous and discrete contact states. We combine the learned policy with our filtering methods for closed-loop policy execution.

A diagram of our proposed solution can be seen in Fig. 1.3. In the diagram, the actual physical system denotes the physical environment of a manipulation task that consists of all the objects and the robot, $u$ is the vector of inputs that drive the actuators of the robot to interact with the actual physical system, $x$ represents the vector of the objects’ actual state (e.g., poses of the objects) in the physical system, $y$ is the vector of sensor measurements of the actual states, $\hat{x}$ is the vector of estimated state from our filter, $\pi$ is the policy for the manipulation task learned
by our learning algorithm, and a controller combines the policy with the state estimation and calculates inputs to the robot. Our solution focuses on solving the robotic manipulation problem in closed-loop with its core being the improvement of perceiving/estimating the states of the manipulation task.

The accuracy of our filter is essential for the success of a robotic manipulation task. There are two factors that can affect the accuracy of estimation. The first factor is the choice of the multi-body dynamic model used in the Bayesian filtering methods. The second factor is the choice of the Bayesian filtering methods. In order to develop accurate filters with the optimal choice of these two factors, we propose to design filters with different modeling options, i.e., different models of multi-body dynamics combined with different variants of Bayesian filtering methods, and evaluate their performances in simulation and physical experiments (Chapter 4). We further exploited the structure of the Linear Complementarity Problem (LCP) model, which is an accurate model of multi-body dynamics, and developed a filter that simultaneously estimates the poses of the objects and the contact states (Chapter 5). Finally, we proposed algorithms that takes advantage of the estimated states of our proposed filter to learn (reinforcement learning) and execute policies.
(QMDP) for robotic manipulation tasks (Chapter 6).

1.3 Contributions

The primary contribution of this thesis is the development of Bayesian filtering methods that employ a model of multi-body dynamics to solve the state estimation problem for robotic manipulation tasks. With our developed filter, we further designed algorithms that learn policies and execute them with state feedback from the filter. The detailed contributions are listed as follows:

- Modeling options evaluation for state estimation in robotic manipulation problems
  We evaluate multiple modeling options for robotic manipulation state estimation and provide guidance on the application preferences of these modeling options.

- Rao-Blackwellized particle filter for state estimation in robotic manipulation problems
  We develop a Rao-Blackwellized particle filter to exploit the piecewise linear property of the model of multi-body dynamics. The proposed particle filter estimates the contact states with a Monte Carlo sampling method and updates the continuous states with a Kalman filter.

- Contact state estimation through the combination of a contact graph and collision detection algorithms
  We add the sticking/sliding contact states into the traditional contact graph and propose a sampling-based method for contact graph construction for three-dimensional robotic manipulation tasks. This new type of contact graph is used to estimate the discrete contact states in manipulation tasks.

- Policy learning with contact states
  We combine reinforcement learning with contact states. The contact states are used to discretize the state space, and a model-based learning algorithm is
applied directly to learn both a mapping (policy) between the robot’s action and the contact states and the contact graph simultaneously.

- Closed-loop policy execution

We combine our learned policy with our developed Rao-Blackwellized particle filter for closed-loop execution. The belief state is represented in the form of particles, and based on the policy, actions are selected by considering all particles collectively.
CHAPTER 2
Previous Works

One way to categorize the previous work on state and parameter estimations for robotic manipulation tasks is by examining the major goals of the estimation algorithms. Although these state and parameter estimation problems are naturally coupled\(^1\), unlike our proposed approach that solves the coupled problem, most of the previous works focus on solving one problem at a time.

In this thesis work, we studied the state estimation and policy learning and execution problems for robotic manipulation tasks. Our work generally falls into two active research areas for robotic manipulation tasks: state estimation and policy learning and execution. In this chapter, we introduce the major and most recent previous works on these two topics.

For those state estimation works, we further divide them into two groups based on their system dynamics models. The first group contains models that do not use a full model of multi-body dynamics. In this group, the least accurate models are simple random walk models, which assume that the new state is equal to the previous state plus noise. The more accurate models are based on equations of motion derived from Newton’s Laws. Among the models of this type, two of them are static and quasi-static models. The former assumes that the object does not move at all, while the latter assumes that the object moves slowly enough such that dynamic forces, such as inertial forces, are negligible. Finally, in the second group, we discuss works that use a full model of multi-body dynamics, which is discussed in Chapter 3.

For a robotic manipulation task, a policy maps the states of the robot to the optimal actions that it should take. The robot, therefore, should follow a policy in order to finish the task. Many efforts have been made to tackle the policy learning and execution problem for robotic manipulation tasks. Depending on the goal of the works, some of the previous approaches mainly focus on understanding the model of

\(^1\)The quality of the parameter estimations can affect the quality of the state estimation outcomes, and vice versa.
the manipulation process, such as the model of kinematic constraints, while others address the policy learning of a manipulation task with techniques including deep learning and so on.

2.1 State Estimation for Robotic Manipulation

As mentioned above, we divide the works on state estimation for robotic manipulation into two groups, and they are discussed in the following two sections.

2.1.1 State Estimation Without a Full Model of Multi-Body Dynamics

The main difference between our proposed solution and the works in this section is the use of a full model of multi-body dynamics in our method. The detailed differences are discussed here following the review of each work.

In order to estimate the poses of known stationary or movable objects, Petrovaskaya et al. developed an annealing particle filter with a scaling series algorithm. It samples more particles in high probability regions until a certain accuracy is reached [62]. In this method, a static model is used if the object is stationary. A random walk model is used when the object is movable. The main goal of this work is to localize the object using robotic-probing motions. Therefore, unlike our proposed method, this method fails when the dynamic aspects of the object are important, for example, if the object topples.

Hebert et al. tried to solve the state estimation problem of a rectangular block held in a robotic gripper [42] by considering various inputs as measurements including readings from a wrist force/torque sensor, finger joint position sensors, and stereo vision sensors. These measurements are used in an extended Kalman filter to estimate the pose of the block relative to the robotic gripper. Additionally, they used a static multiple model estimator (SMM) to estimate the discrete contact states, such as when the block breaks and makes contact with the gripper. In their method, the motion model of the object is assumed to be a random walk model, so the method cannot estimate dynamic motions of the object, such as when the block falls out of the robotic hand.

Meeussen et al. designed a hybrid filter by incorporating a graph of all possible
contact states and their transitions [46] to estimate the contact states and the pose of the tracked objects. Their method tracks the object’s pose and velocity through a Kalman filter and tracks the contact states through a particle filter. The state transition model for this method is a constant acceleration model. Their method is similar to our approach. However, in the absence of a full model for multi-body dynamics, their method cannot deal with a dynamic motion with contacts, such as the motion of a ball dropping onto the ground.

Chalon et al. developed a particle filter that tracks the pose of an object when it is grasped by a robot gripper [35]. Their method incorporates the kinematic constraints of the robot hand into the state transition model to update the pose of the tracked object. In their method, the constraints of the system are considered to be kinematic, while our proposed method assumes these constraints to be dynamic, i.e., constraints that do consider the dynamic properties (e.g., mass) of the object and its environment.

The work by Koval et al. developed the “manifold particle filter” to support their work in planar push-grasping experiments [36]. An interesting problem that can arise in contact tasks is that as contacts form, the dimension of the valid portion of the configuration space for the system drops. If this issue is not handled properly in the particle propagation process, particle starvation occurs and the filter diverges. The manifold particle filter solves this problem by sampling particles on a pre-computed contact manifold, which includes the configuration space of the tracked object being in contact with the robotic hand. Their method focuses on the planar experiment, and this means that their state space is SE(2), while our method solves the problem in SE(3)\(^2\). Additionally, their approach becomes intractable when the robotic hand changes its configuration during a grasping process since a contact manifold needs to be computed for every set of hand configurations. Therefore, their method only works for planar pushing experiments, while our method solves the general robotic manipulation problem.

\(^2\)SE represents Special Euclidean Group.
2.1.2 State Estimation With Full Model of Multi-Body Dynamics

The works that are reviewed below share the same idea with our approach, which uses a full model of multi-body dynamics. Our method differs from these methods in the way we combine the dynamics model with Bayesian filtering methods.

Duff et al. combined a physics simulator (PhysX) with an RANSAC algorithm to estimate the trajectory of an object’s motion [24]. The inputs to their algorithm are noisy position measurements of the object. The RANSAC algorithm selects inliers and outliers by fitting simulated trajectories to the observed ones. Since their approach fits the entire observed trajectory to the simulated trajectories, the trajectory estimation starts after the whole trajectory is observed and thus their method is an offline method. Therefore, it cannot be used for real-time state estimation.

A follow-up work by Duff et al. used a real-time physics simulator (PhysX) as the basis of the dynamics model of a particle filter so as to track the poses of objects [25]. They demonstrated that this particle filter can track the pose of an object very well even during an occlusion, such as when the camera is blocked by obstacles in the scene. In their method, the simulator is used as a black box. There are two problems with this approach. First, simulators can fail to simulate the dynamics at hard situations [27], such as when a robotic hand grasps an object with a power grasp. Second, this approach fails to exploit the special structures of the model of multi-body dynamics (e.g., the piecewise linear property), and therefore its efficiency and convergence are degraded.

Zhang et al. explored the application of particle filters to estimate positions and orientations of a block during planar push grasping with visual occlusions and tactile sensing [77]. They compared the performances of particle filters with different sources of measurements. In their work, the model of multi-body dynamics is still used as a black box. Although they tried to compare the performances of different variants of particle filters, some of the design trade-offs, such as whether or not should we penalize the particles that violate the physical constraints, were still left out because of the complexity of the experiments.

Zhang et al. developed a general solution by combining a well-developed dynamic Bayesian network with a Rao-Blackwellization particle filter to speed up the
state estimation of dynamic systems [34]. Later, Zhang compared three different particle filters in a push grasping experiment and proposed a relaxed algorithm to break the nonlinear state transition model of the dynamic system into linear equations in a particle filtering implementation [76]. The method used in their work is similar to our approach. However, they did not exploit the piecewise linear property of the model of multi-body dynamics. Specifically, the distribution of the continuous states is still represented as particles, whereas we propose to use a particle filter to sample the contact discrete states and update the distribution of continuous states through Kalman filters. The advantages of our approach are the reduction of the number of particles and the maximum information that is passed by the Kalman filters.

2.1.3 Summary

In summary, to solve the state estimation problem for robotic manipulation tasks, the current approaches are divided into two groups. One group, which is discussed in section 2.1.1, does not consider the dynamics of the objects and can only deal with the quasistatic motions. Although the other group, which is discussed in section 2.1.2, considers the dynamic models of the objects, their approaches either treat the dynamic model of the object as a black box ([24, 25]) or have low efficiency ([77, 34, 76]). Our approach incorporates an accurate model of multi-body dynamics in our state estimation methods. Additionally, we propose to improve the efficiency of our filter by exploiting the piecewise linear property of the multi-body dynamic model.

2.2 Policy Learning and Execution for Robotic Manipulation Tasks

Among the previous works of policy learning and execution for robotic manipulation tasks, some focus on analyzing the model of a manipulation task. Farahat et al. studied the mathematical model of the kinematic constraints for a planar grasping task, where three active polygons try to control the position and orientation of a polygonal workpiece [22]. They derived a closed form solution for the kinematic
constraints using eight fundamental systems of contact constraint equations. This solution provides an efficient way of computing the pose of the workpiece given the pose of the active polygons. In their work, only the kinematic constraints are considered. Therefore, unlike our approach, their method cannot deal with dynamic constraints, such as gravity.

Dafle et al. model a set of dexterous robotic manipulation tasks as sequences of hand-scripted actions [40]. Their hand-scripted actions take advantage of the resources extrinsic to the robotic hand, such as the gravity. Particularly, in their work, they focus on the grasping problem. They design twelve regrasp actions for the robotic hand, and a grasp graph is generated to transition any grasp type to other types. Although the dynamic properties of the object are used in their regrasp actions, the actions are hand-programmed and executed in open loop. Our work considers the dynamic properties in our policy learning process, and we also study the closed-loop execution problem.

There are also previous works on applying learning algorithms to learning the policies for robotic manipulation tasks. Levine et al. [44] developed a method that learns a nonlinear policy through multiple time-varying linear-Gaussian controllers. In their work, multiple time-varying linear-Gaussian controllers are trained with different initial conditions, and a designed framework of guided policy search (GPS) is used to train a complex non-linear policy that can cover highly general manipulation tasks, such as tasks with different target positions. However, unlike our work, their work does not consider the dynamic model of the objects.

Kormushev et al. [43] also developed an algorithm to learn the policy for a robotic pancake-flipping task. They proposed a controller based on a mixture of K proportional-derivative systems, and apply an expectation-maximization (EM) based reinforcement learning algorithm to teach a robot the policy to complete the manipulation task. Although the dynamic properties of the object/pancake are used in the process of learning the policy, their work relies on accurate measurements of the states of the object, while our work considers noisy measurements during the policy execution.

Kumar et al. [52] proposed an algorithm that learns the manipulation skills
for robotic manipulation tasks with learned local dynamic models. In their proposed algorithm, the manipulation skills are represented as time-varying linear-Gaussian controllers, and the dynamics model of the manipulation task is learned with trajectories collected as the robot executes its current policy. They iteratively update their dynamic models and their policy/linear-Gaussian controllers. In order to eliminate the policy divergence as a result of inaccurate learned dynamic models, they also introduce a KL-constraint to their optimization process so that a newly learned controller does not diverge from the controllers learned from the previous iterations. In their policy execution, they assume accurate measurements while we assume noisy measurements. Additionally, instead of an accurate dynamic model of the objects, their method learns the model with trajectory samples, which can lead to bad model approximations.

Dang et al. [18] proposed to learn the policy of a robotic manipulation task from human demonstrations. A manipulation task is described by a task descriptor, which is constructed by series of manipulation primitives. A manipulation primitive contains a pair of vectors: the position vector and the direction vector. After a human demonstration, the demonstrated trajectory is divided into segments, where each segment corresponds one individual manipulation primitive. The robot then learns a chain of manipulation primitives based on the trajectory segmentation. Their approach discretizes a robotics manipulation task with respect to the kinematic relations among rigid bodies and failed to capture the dynamic aspect of the objects in manipulation tasks.

Koval et al. [37] proposed to integrate the contact belief space into a partially observable Markov decision process (POMDP) problem for robotic push-grasping tasks. Similar to our approach, they also discretize the state space with the contact state. In their approach, a state is defined by the distribution of contacts on the robotic manipulator. Additionally, a manifold particle filter [36], which also estimate the contact state, is used in the policy execution. However, unlike our work, their contact state only describes the kinematic relations between the object and the robot, and thus the dynamic properties of an object are not incorporated. As the object’s motions are constrained by the distribution of the contacts forces acting on
dynamic properties of the object (e.g., mass and inertia), our approach considers the dynamics of the object by learning a policy based on the object’s contact states.

2.2.1 Summary

In summary, for most of the existing approaches that solve the policy learning and execution problems for robotic manipulation tasks, the dynamics of the objects are ignored. As a result, the motions of the objects in the tasks are assumed to be pure kinematic ([22, 18, 44]) or quasistatic ([37]). In the works that consider the dynamics of the object, their dynamic models are not accurate ([52]) or explicitly explored ([40, 43]). On the other hand, in this thesis, we propose approaches that apply an accurate model of multi-body dynamics to the policy learning and execution problems for robotic manipulation tasks.
CHAPTER 3

Background

The core feature of our proposed approach is combining an accurate model of multi-body dynamics with Bayesian filtering methods. Models of multi-body dynamics have been actively studied for many years. Their great success in multi-body dynamics simulators, such as Gazebo [6] and Bullet [3], has been used by the robotics community over the past ten years. Bayesian filtering methods have been researched for decades, and their applications include object tracking, Simultaneous Localization and Mapping (SLAM). In addition to the two main components, in order to fully exploit the structure of the multi-body dynamics models, we further incorporate a new type of contact graph to our filtering framework. The rest of this chapter provides a detailed introduction to the key research topics that build the foundations of this thesis.

3.1 The Model of Multi-Body Dynamics

For a robotic manipulation task, the system that consists of the robot and the objects is usually considered as a multi-body dynamic system. One effective approach to model multi-body dynamics is to formulate it as a complementary problem. In a complementarity formulation of a multi-body dynamics model, there are several requirements that it must satisfy [16]. First, it must meet the Newton-Euler equations of motion and a kinematic update law. Secondly, the joint constraints must be included so that the motions of two bodies connected by joints can be described. Finally, it must handle the contact constraints correctly by preventing penetrations among the bodies in the system, enforcing compressive contact forces, and enforcing a friction law.

The Newton-Euler equations of motion and the joint constraints can be described by a set of ordinary differential equations. The contact constraints are usually formulated as complementarity conditions. Putting everything together, the model of multi-body dynamics is a differential complementarity problem (DCP) [47,
and a DCP is defined as:

\[ \dot{u} = g(u, v) \]  

\[ 0 \leq v \perp f(u, v) \geq 0 \]

where \( u \in \mathbb{R}^{n_1} \) and \( v \in \mathbb{R}^{n_2} \) are vectors, \( \perp \) represents orthogonality \( (v^T f(u, v) = 0) \), and \( g(u, v) : \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \to \mathbb{R}^{n_1} \) and \( f(u, v) : \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \to \mathbb{R}^{n_2} \) are functions.

For the rest of this section, we use vector \( q_i \) for the position and orientation of the frame attached to the \( i \)th body’s center of mass, and vector \( \nu_i \) as the velocity twist of the origin of the same frame. A velocity twist is defined as the vector that consists of the linear velocity vector and the angular velocity vector concatenated together shown below:

\[ \nu_i = \begin{bmatrix} v_i \\ \omega_i \end{bmatrix} \]  

where \( v_i \) and \( \omega_i \) are the linear and angular velocities for the \( i \)th body respectively. The generalized coordinate \( q \) and the generalized velocity \( \nu \) of the whole system are formed by concatenating \( q_i \) and \( \nu_i \) respectively [16]. We will also use \( \lambda \) to represent the wrench vector, which is the concatenation of the force vector \( f \) and the torque vector \( \tau \):

\[ \lambda = \begin{bmatrix} f \\ \tau \end{bmatrix} \]  

### 3.1.1 Newton-Euler Equations and Joint Constraints

In the model of multi-body dynamics, Newton-Euler equations and the joint constraints are described as ordinary differential equations, and they form the equality constraints of the model.

\footnote{In some circumstances, we will only consider the forces in \( \lambda \).}
3.1.1.1 Newton-Euler Equations

The Newton-Euler equations describe the relationship between the kinematic motions of a body and its external forces and torques. Newton’s second law of motion states that the acceleration of an object is proportional to the net force on the object, and is inversely proportional to the mass of the object:

\[ F(q_i, \dot{q}_i, t) = m(q_i, t)\ddot{q}_i \]  

(3.5)

Similar to Newton’s second law of motion, Euler’s equation states the relation between the net torque on an object and its angular acceleration as follows:

\[ \tau(q_i, \dot{q}_i, t) = \mathcal{I}(q_i, t)\dot{\omega}_i + \omega_i \times \mathcal{I}(q_i, t)\omega_i \]  

(3.6)

where \( \mathcal{I}(q_i, t) \) is the inertia tensor for the \( i \)th body.

The Newton-Euler equations of motion can be written in a compact matrix representation as below:

\[ M(q, t)\dot{\nu} = \lambda_{vp}(q, \dot{q}, t) + \lambda_{app}(q, t) \]  

(3.7)

where

\[ M = \begin{bmatrix} m_1 & 0 & 0 & 0 & 0 \\ 0 & \mathcal{I}_1 & 0 & 0 & 0 \\ 0 & 0 & \ddots & 0 & 0 \\ 0 & 0 & 0 & m_n & 0 \\ 0 & 0 & 0 & 0 & \mathcal{I}_n \end{bmatrix} \]  

(3.8)

and \( \lambda_{app}(q, t) \) is a vector of the wrenches and joint torques that only depend on the configuration of the body, such as the gravity, and \( \lambda_{vp}(q, \dot{q}, t) \) is the vector of the wrenches that depend on both the configuration and the velocity of the body, such as the second term of the Euler’s equation \( \omega_i \times \mathcal{I}(q_i, t)\omega_i \).

We also define the kinematic update law, which converts the twist \( \nu \) to the
rates of changes in the configuration space $\dot{q}$:

$$\dot{q} = H(q)\nu$$

(3.9)

where

$$H(q) = \begin{bmatrix}
I_{3x3} & 0 & 0 & 0 & 0 \\
0 & J(q_i) & 0 & 0 & 0 \\
0 & 0 & \ddots & 0 & 0 \\
0 & 0 & 0 & I_{3x3} & 0 \\
0 & 0 & 0 & 0 & J(q_n)
\end{bmatrix}$$

(3.10)

and $J(q_i)$ maps the rates of the Euler parameters of the $i$th body to its angular velocity [16].

### 3.1.1.2 Joint Constraints

In a robotics problem, we usually need to consider objects that are connected by joints, e.g., a robot arm. The joint constraints are also known as a class of constraints called *bilateral* constraints, and they can be modeled using algebraic equations. Joint constraints generally consist of constraints on the configuration of the bodies, and can enforce velocity constraints with their derivative terms:

$$\Phi_i(q, t) = 0$$

(3.11)

where $\Phi_i(q, t)$ describes the position-level constraints for joint $i$. The form of $\Phi_i$ can be different from one joint to another, for example, the $\Phi_i$ for a revolute joint will be different from that for a prismatic joint [39]. No matter what form $\Phi_i$ takes, we can always take its derivative to get the velocity constraints:

$$\frac{\partial \Phi_i(q, t)}{\partial q} \dot{q} + \frac{\partial \Phi_i(q, t)}{\partial t} = 0$$

(3.12)
Further, if we combine the kinematic update law from equation (3.9), the velocity constraints shown above can be simplified as below:

\[ G_ib(q, t)\nu + \frac{\partial \Phi_i(q, t)}{\partial t} = 0 \]  

(3.13)

where \( G_ib = \frac{\partial \Phi_i(q, t)}{\partial q} H(q) \) is also known as the bilateral constraint Jacobian matrix.

If there are \( n_b \) bilateral constraints in the system, we can define \( G_b(q, t) = \begin{bmatrix} G_{1b} \\ \vdots \\ G_{n_b} \end{bmatrix} \),

\[ \frac{\partial \Phi(q, t)}{\partial t} = \begin{bmatrix} \frac{\partial \Phi_1(q, t)}{\partial t} \\ \vdots \\ \frac{\partial \Phi_{n_b}(q, t)}{\partial t} \end{bmatrix} \], and the joint constraints for the whole system can be written as:

\[ G_b(q, t)\nu + \frac{\partial \Phi(q, t)}{\partial t} = 0 \]  

(3.14)

### 3.1.1.3 Equality Constraints

We can also use the bilateral Jacobian matrix \( G_b(q, t)^T \) to relate the joint constraint forces to the wrench at the center of the mass for each body in the system. In fact, each column of \( G_b(q, t)^T \) represents a basis vector for the joint constraint forces. As a result, we can write out \( \lambda_b \) as a vector of the magnitudes of all the joint constraint forces and rewrite the Newton-Euler equations in equation (3.7) by adding the joint constraint forces. This puts all the equality constraints in a complementarity formulation of the model of multi-body dynamics shown as below:

\[ M(q, t)\ddot{\nu} = G_b(q, t)\lambda_b + \lambda_{wp}(q, \dot{q}, t) + \lambda_{app}(q, t) \]  

(3.15)

\[ \dot{q} = H(q)\nu \]  

(3.16)

\[ 0 = \Phi(q, t) \]  

(3.17)
3.1.2 Contact Constraints

As mentioned earlier, the contact constraints must prevent penetrations, enforce compressive contact forces, and enforce a friction law. The first two requirements are modeled in the contact normal direction as the contact normal constraints. The last requirement is met by the frictional constraints at the contact point. Particularly, for two bodies that can collide with each other, the coordinate frames of a pair of contact points are shown in Fig. 3.1. We denote the total number of contacts in the system as $n_c$. Suppose that the contact that is shown in Fig. 3.1 is the $i$th contact, then the contact point on each of the object will have a local contact frame $\Lambda_{ji}$ and $\Lambda_{ki}$. $\Psi_{in}$ represents the signed distance in the contact normal direction between the two contact points of the $i$th contact. $\hat{n}_i$, $\hat{i}_i$ and $\hat{o}_i$ are the contact normal unit vector and the contact tangent vectors respectively.

3.1.2.1 Contact Normal Constraints

In the contact normal direction, the signed distance $\Psi_{in}(q,t)$ can be used to model penetration. When $\Psi_{in}(q,t)$ is greater than zero, the two bodies are separate, and if the two bodies are touching, $\Psi_{in}(q,t)$ equals to zero. Penetration is indicated when $\Psi_{in}(q,t)$ becomes negative. Therefore, for a system with no penetration be-
tween bodies, we will enforce:

$$\Psi_n(q, t) \geq 0$$  

(3.18)

where $$\Psi_n(q, t)$$ is the vector containing all the signed distances of all contacts in the system.

A physical interpretation of the compressive contact forces is that the contact forces will not pull two bodies together. If we define $$\lambda_{in}$$ as the magnitude of the contact normal force at contact point $$i$$, and $$\lambda_n$$ is the vector of all the $$\lambda_{in}$$ in the system, then the compressive contact forces condition requires:

$$\lambda_n \geq 0$$  

(3.19)

Additionally, the distance $$\Psi_{in}(q, t)$$ and the contact force $$\lambda_{in}$$ must satisfy an orthogonality relation. When the distance $$\Psi_{in}(q, t)$$ is greater than zero (i.e., bodies are separate), the contact force must be zero. On the other hand, when the contact force $$\lambda_{in}$$ is greater than zero (i.e., bodies are touching), the distance $$\Psi_{in}(q, t)$$ must be zero. This relation can be written as:

$$\Psi_{in}(q, t)^T \lambda_n = 0$$  

(3.20)

Equations (3.18), (3.19) and (3.20) can be compactly represented as a complementarity constraint as below:

$$0 \leq \Psi_n(q, t) \perp \lambda_n \geq 0$$  

(3.21)

Finally, similar to the bilateral constraint, we also define the contact normal constraint Jacobian matrix $$G_{in} = \frac{\partial q_{in}}{\partial q} H(q)$$, which converts contact normal forces to the wrench at the center of the mass for each body. Then similarly, $$G_n$$ is defined as the stack of all the $$G_{in}s$$. 

3.1.2.2 Frictional Constraints

For a multi-body dynamic model with friction forces, its contact constraints must also enforce a model of friction law. Coulomb friction model is one of the most widely used friction models for friction forces between solid surfaces. In this thesis, we adopt the Coulomb friction model for our dynamic model. In the Coulomb friction model, the friction force must remain within a cone (i.e., the friction cone) defined by the magnitude of the contact normal force $\lambda_n$ and the coefficient of friction $\mu$. When the contact is sticking (i.e., no relative motion between the two contact points), the magnitude of the friction force is bounded by $\lambda_n\mu$, and there are no constraints on the direction of the friction force. When the contact is sliding, the magnitude of the friction force is exactly $\lambda_n\mu$, and its direction is opposite to the sliding motion.

Mathematically, we define the friction cone for the $i$th contact as $F(\lambda_{in}, \mu_i)$, and it is expressed as below:

$$F(\lambda_{in}, \mu_i) = \{ (\lambda_{io}, \lambda_{it}) : \mu_i^2 \lambda_{in}^2 - \lambda_{io}^2 - \lambda_{it}^2 \geq 0 \} \quad (3.22)$$

where $\mu_i$ is the coefficient of friction for the $i$th contact, and $\lambda_{io}$ and $\lambda_{it}$ represent the magnitudes of the friction forces along the two axes $\hat{o}_i$ and $\hat{t}_i$.

The directions of the friction forces can be modeled using the maximum dissipation principal, which states that given a magnitude of the contact normal force and a sliding direction, the friction force is the one that maximizes the rate of energy dissipation [16].

$$\begin{align*}
(\lambda_{io}, \lambda_{it}) & \in \arg \max \{ (-\lambda_{io}v_{io} - \lambda_{it}v_{it}) \} \\
\text{where } v_{io} \text{ and } v_{it} & \text{ are the sliding velocities along the two axes in the contact tangent plane. Similarly to the contact normal velocities, we define } \Psi_i = [\Psi_{io}, \Psi_{it}] \text{ as the sliding displacement. Then, we have } v_{io} = G_{io}\nu + \frac{\partial \phi_{io}}{\partial q} \text{ and } v_{it} = G_{it}\nu + \frac{\partial \phi_{it}}{\partial \dot{q}} \text{ with } G_{io} = \frac{\partial \phi_{io}}{\partial q} H(q), G_{it} = \frac{\partial \phi_{it}}{\partial \dot{q}} H(q). \text{ We also call } G_{io} \text{ and } G_{it} \text{ the Jacobian matrices for the friction constraints and they map the friction forces to the wrench at the center of the mass for each body in the system. With further derivation, the maximum}
\end{align*}$$
dissipation principal and the friction cone shown above can be written as below:

\[ 0 = \mu_i \lambda_{in} v_{it} + \lambda_{it} \sigma_i \quad (3.24) \]

\[ 0 = \mu_i \lambda_{in} v_{io} + \lambda_{io} \sigma_i \quad (3.25) \]

\[ 0 \leq \sigma_i \perp \mu_i^2 \lambda_{in} - \lambda_{io}^2 \geq 0 \quad (3.26) \]

where \( \sigma_i = \sqrt{v_{io}^2 + v_{it}^2} \) is the sliding speed at the \( i \)th contact. A physical interpretation of the complementarity constraint above is that when the sliding speed is greater than zero, the friction force must be on the boundary of the friction cone with its magnitude equals to \( \lambda_n \mu \). Conversely, when the sliding speed is zero, the friction force must lie inside the friction cone. A more compact formulation of the frictional constraint of the whole system is shown below:

\[ 0 = (U \lambda_n) \circ v_t + \lambda_t \circ \sigma \quad (3.27) \]

\[ 0 = (U \lambda_n) \circ v_o + \lambda_o \circ \sigma \quad (3.28) \]

\[ 0 \leq \sigma \perp (U \lambda_n) \circ (U \lambda_n) - \lambda_o \circ \lambda_o - \lambda_t \circ \lambda_o \geq 0 \quad (3.29) \]

where \( U \) is a diagonal matrix with the \( i \)th element equals to \( \mu_i \), and \( \circ \) represents the Hadamard product [8]. The formulation of the frictional constraints is in the quadratic (nonlinear) form. Therefore, applying this formulation can have difficulties when we solve the constraints. Therefore, one would like to approximate the constraints with linear equations. Particularly, we approximate the quadratic friction cone with a polyhedron as shown in Fig. 3.2. We choose \( n_d \) direction vectors that positively span the friction cone. For each vector \( d_j \), we define the magnitude of its friction force to be \( (\lambda_{if})_j \) and its sliding displacement to be \( (\Psi_{if})_j \). With the polyhedron approximation, the constraints on the friction cone and the maximum dissipation now can be expressed as:

\[ 0 \leq \lambda_{if} \perp G_{ij} \nu + e \sigma_i + \frac{\partial \Phi_{if}}{\partial t} \geq 0 \quad (3.30) \]

\[ 0 \leq \sigma_i \perp \mu_i \lambda_{in} - e^T \lambda_{if} \geq 0 \quad (3.31) \]

\[ \lambda_{it} \leq \lambda_{io} \]
where $e \in \mathbb{R}^{n_d}$ is a vector of ones, $G_{ij}$ is the Jacobian matrix for all the direction vectors $d_j, j \in 1, \cdots, n_d$. The compact representation of the approximated friction constraints is shown below:

$$0 \leq \lambda_f^T \nu + E\sigma + \frac{\partial \Phi_f}{\partial t} \geq 0$$  \hspace{1cm} (3.33)

$$0 \leq \sigma \perp U\lambda_n - E^T\lambda_f \geq 0$$  \hspace{1cm} (3.34)

$$0 \leq \nu \perp \mathbb{I}$$  \hspace{1cm} (3.35)

where $E$ is diagonal block matrix with the $i$th block equals to $e$.

3.1.3 Complementarity Formulation of the Model of Multi-Body Dynamics

As discussed above, the model of multi-body dynamics can be modeled as a DCP, which includes the Newton-Euler equations of motion, joint constraints as well as the constraints related to contacts. A direct application of the Coulomb friction model on the DCP will yield a quadratic friction cone and a nonlinear DCP. With a polyhedron approximation of the friction cone, we can derive a linear DCP.

The nonlinear DCP or the Nonlinear Complementarity Problem (NCP) can

![Figure 3.2: Friction cone (blue) approximated by a polyhedron (black) with direction vectors $d_j$. The figure shows the limits projected on the contact tangent plane.](image)
be formulated by combining equations (3.15) to (3.17), (3.21) and (3.27) to (3.29) as follows:

\[
\begin{align*}
M(q, t)\dot{\nu} &= G_n(q, t)\lambda_n + G_f(q, t)\lambda_f + G_b(q, t)\lambda_b + \lambda_{vp}(q, \dot{q}, t) + \lambda_{app}(q, t) \\
\dot{q} &= H(q)\nu \\
0 &= \Phi(q, t) \\
0 &= (U\lambda_n) \circ v_t + \lambda_t \circ \sigma \\
0 &= (U\lambda_n) \circ v_o + \lambda_o \circ \sigma \\
0 &\leq \sigma \perp (U\lambda_n) \circ (U\lambda_n) - \lambda_o \circ \lambda_o - \lambda_t \circ \lambda_o \geq 0 \\
0 &\leq \Psi_n(q, t) \perp \lambda_n \geq 0
\end{align*}
\]  

With the linearized friction cone, the linear DCP or Linear Complementarity Problem (LCP) is formulated as below:

\[
\begin{align*}
M(q, t)\dot{\nu} &= G_n(q, t)\lambda_n + G_f(q, t)\lambda_f + G_b(q, t)\lambda_b + \lambda_{vp}(q, \dot{q}, t) + \lambda_{app}(q, t) \\
\dot{q} &= H(q)\nu \\
0 &= \Phi(q, t) \\
0 &\leq \lambda_f \perp G_f^T\nu + E\sigma + \frac{\partial\Phi_f}{\partial t} \geq 0 \\
0 &\leq \sigma \perp E^T\lambda_f \geq 0 \\
0 &\leq \Psi_n(q, t) \perp \lambda_n \geq 0
\end{align*}
\]

Stewart et al. [67] proposed to solve the complementarity formulation of multi-body dynamics through a time-stepping scheme, which is known as the Stewart-Trinkle formulation. The Stewart-Trinkle formulation is one of the most popular and successful methods for solving LCP in simulations. It has been widely used in many simulation engines, such as Bullet [3], and Chrono::engine [4]. In this formulation, the derivatives of the variables are replaced by their discrete-time approximations, and the forces are replaced by impulses over one time step. Let the length of a time step to be \( h \). Then from time step \( t \) to \( t + 1 \), the derivative of the twist \( \dot{\nu} \) can be approximated as \( \dot{\nu} \approx (\nu_{t+1} - \nu_t)/h \). Following the steps in [67], we have the LCP
represented in the Stewart-Trinkle formulation as follows:

\[
\begin{bmatrix}
0 \\
0 \\
(p_n)_{t+1} \\
(p_f)_{t+1} \\
s_{t+1}
\end{bmatrix}
= \begin{bmatrix}
-M & G_b & G_n & G_f & 0 \\
G_b^T & 0 & 0 & 0 & 0 \\
G_n^T & 0 & 0 & 0 & 0 \\
G_f^T & 0 & 0 & 0 & E \\
0 & 0 & U & -E^T & 0
\end{bmatrix}
\begin{bmatrix}
\nu_{t+1} \\
(p_b)_{t+1} \\
(p_n)_{t+1} \\
(p_f)_{t+1} \\
\sigma_{t+1}
\end{bmatrix}
+ \begin{bmatrix}
M \nu_t + p_{app} + p_{vp} \\
\frac{\Psi_t}{h} + \frac{\partial \Phi_t}{\partial t} \\
\frac{\partial \Psi_t}{h} + \frac{\partial (\Psi_n)}{\partial t} \\
\frac{\partial (\Psi_f)}{\partial t} \\
0
\end{bmatrix}
\] (3.49)

\[
q_{t+1} = q_t + H_t \nu_{t+1} h
\] (3.50)

\[
0 \leq \begin{bmatrix}
(p_n)_{t+1} \\
(p_f)_{t+1} \\
s_{t+1}
\end{bmatrix}
= \begin{bmatrix}
(p_n)_{t+1} \\
(p_n)_{t+1} \\
\alpha_{t+1} \\
\sigma_{t+1}
\end{bmatrix}
\geq 0
\] (3.51)

where we evaluate all the Jacobian matrices \(G(\cdot)\) at time step \(t\), \(p(\cdot)\) is the impulse calculated as \(\lambda(\cdot)h\), and \(\rho(\cdot) = \Psi(\cdot)/h\).

### 3.1.3.1 An Example

To illustrate the multi-body dynamic model we introduced above, we give an example of a planar multi-body dynamic system as shown in Fig. 3.3.

![Figure 3.3](image)

Figure 3.3: An example of a multi-body dynamic system, where a robot pushes an object that slides on a plane. The object tilts once it makes contact with the small obstacle on the plane.

In this example, a robot, which is shown as a rectangle on the left, pushes on a triangular object. The object can translate along the x and y-axes and rotate around the z-axis. As a result of the robot’s motion, the object slides on a plane.
Figure 3.4: The configuration of the contacts when the object slides on the plane.

(Fig. 3.3 (a)) and tilts when it hits a small obstacle (Fig. 3.3 (b)). Throughout this thesis, the LCP model is extensively used in our proposed methods, and in this example, we choose to formulate the motions of the object in the form of LCP, which is shown below:

\[
\begin{bmatrix}
0 \\
(p_n)_{t+1} \\
(p_f)_{t+1} \\
s_{t+1}
\end{bmatrix}
= 
\begin{bmatrix}
-M & G_n & G_f & 0 \\
G_n^T & 0 & 0 & 0 \\
G_f^T & 0 & 0 & E \\
0 & U & -E^T & 0
\end{bmatrix}
\begin{bmatrix}
v_{t+1} \\
(p_n)_{t+1} \\
(p_f)_{t+1} \\
\sigma_{t+1}
\end{bmatrix}
\begin{bmatrix}
Mv_t + p_{\text{app}} + p_{\text{cp}} \\
\frac{\partial(p_n)}{\partial h} \\
\frac{\partial(p_f)}{\partial h} \\
0
\end{bmatrix}
\]

(3.52)

\[
q_{t+1} = q_t + H_tv_{t+1}h
\]

(3.53)

\[
0 \leq \begin{bmatrix}
(p_n)_{t+1} \\
(p_f)_{t+1} \\
s_{t+1}
\end{bmatrix} \perp \begin{bmatrix}
(p_n)_{t+1} \\
(p_f)_{t+1} \\
\alpha_{t+1}
\end{bmatrix} \geq 0
\]

(3.54)

In equation (3.52), matrix \(G_b\) is eliminated since no bilateral constraints are involved in this example. The inertia matrix \(M\) is reduced to a \(3 \times 3\) matrix, which is shown below:

\[
M = \begin{bmatrix}
m & 0 & 0 \\
0 & m & 0 \\
0 & 0 & I
\end{bmatrix}
\]

(3.55)

where \(m\) is the mass of the object and \(I\) is the object’s moment of inertia. In this example, if the length of the legs of the triangle is \(l\), \(I = \frac{r}{6}m\). The definitions of all the other variables depend on the poses of the object and the robot. For example,
if the object slides on the plane as shown in Fig. 3.4, the contact normal Jacobian matrix $G_n$ is defined as below:

$$G_n = \begin{bmatrix} n_1 & n_2 & n_3 & n_4 \\ n_1 \times r_1 & n_2 \times r_2 & n_3 \times r_3 & n_4 \times r_4 \end{bmatrix},$$

(3.56)

where the four columns represent the four contacts (two contacts with the robot and two contacts with the plane). The $n_i$ are the contact normal vectors and $r_i$ are the vectors pointing from the contact point to the center of gravity of the object. The values of the $n_i$ and $r_i$ are shown below:

$$n_1 = \begin{bmatrix} 1 & 0 \end{bmatrix}^T, r_1 = \begin{bmatrix} \frac{l}{3} & -l_1 \end{bmatrix}^T$$

(3.57)

$$n_2 = \begin{bmatrix} 1 & 0 \end{bmatrix}^T, r_1 = \begin{bmatrix} \frac{l}{3} & -l_2 \end{bmatrix}^T$$

(3.58)

$$n_3 = \begin{bmatrix} 0 & 1 \end{bmatrix}^T, r_1 = \begin{bmatrix} \frac{l}{3} & \frac{l}{3} \end{bmatrix}^T$$

(3.59)

$$n_4 = \begin{bmatrix} 1 & 0 \end{bmatrix}^T, r_1 = \begin{bmatrix} -\frac{2l}{3} & \frac{l}{3} \end{bmatrix}^T,$$

(3.60)

where $l$ is the length of the triangle’s legs.

### 3.2 Bayesian Filtering Methods

From a statistical perspective, filtering is the task of predicting an unknown probability distribution over the states of a dynamic system, given the system’s inputs and noisy measurements of its state. The Bayesian framework provides a disciplined means to combine information from imprecise prior knowledge about changes in state with noisy measurement data. In a Bayesian filtering method, the unknown probability distribution is updated recursively based on the most recent acquired measurements. For example, in a robotic manipulation problem, the robot starts out with a distribution of the pose of an object. As the robot executes the manipulation task, it constantly updates the belief about the pose using the sensor data (e.g., vision data) collected.

The very foundation of Bayesian filtering is the dynamic Bayesian network (DBN), which is a generative probabilistic model that captures the stochastic de-
dependencies among the states and measurement data over time. There have been many studies on the dynamic Bayesian network and the associated inference procedures in statistics, control, and machine learning literature (see [51] for a complete survey).

There are typically two types of vectors in a Bayesian filtering problem: the state vector and the measurement vector. The state vector contains all the information about the states, and the measurement vector includes the measurements of the states. In a Bayesian filtering problem, the state vector usually cannot be measured exactly with its actual values hidden in the measurements. Therefore, the state vector is often referred to as the hidden state. In the rest of this section, we will represent the state vector as $x$ and the measurement vector as $z$.

### 3.2.1 Dynamic Bayesian Network and Bayesian Filtering

As a generative probabilistic model, Dynamic Bayesian Network (DBN) describes the dependency relations among state variables and measurements over adjacent time steps. The simplest and perhaps the most widely applied example of a DBN is the Hidden Markov Model (HMM), where the state information is represented a single discrete random variable [59, 58, 63]. HMM has been successfully used in many applications such as speech recognition, gesture recognition, and so on. A diagram of a simple DBN is shown in Fig. 3.5, where the subscripts represent the time steps of the variables. In the diagram, $x_t$ represents the vector of hidden states, and $z_t$ represents the vector of measurements. The arrowed lines represent dependencies among the vectors. Putting these vectors in a robotic manipulation problem, $x_t$ can represent the pose of an object, and $z_t$ represents the object pose estimation from the vision sensors.

In a DBN model, we generally make the following two assumptions:

- The dynamic model of the state vector $x$ is Markovian. This indicates that state vector $x_t$ is independent of $x_{t-k}$, where $k \geq 2$, if $x_{t-1}$ is known. The Markov property can be described by equation (3.61).

$$p(x_t | x_{t-1}, x_{t-2}, \ldots, x_0) = p(x_t | x_{t-1})$$  \hspace{1cm} (3.61)
Figure 3.5: A diagram of a simple Dynamic Bayesian Network. $x_t$ represents the vector of the hidden states of the system, and $z_t$ represents the vector of the measurements. The arrowed lines represent dependencies among the states.

- Conditional independence of the measurement vectors. The measurement vector $z_t$ is independent of all the other measurement vectors $z_{t1}$ and state vectors $x_{t1}$, where $t1 \neq t$, if $x_t$ is known. This assumption is summarized in equation (3.62).

$$p(z_t \mid x_T, \ldots, x_0, z_T, \ldots, z_0) = p(z_t \mid x_t) \quad (3.62)$$

The goal of Bayesian filtering is to estimate the probability distribution of the current state vector $x_t$, given all the measurement vectors observed so far $o_{1:t}$. This goal can be written in the mathematical form as shown in equation (3.63).

$$p(x_t \mid z_{1:t}) \quad (3.63)$$

In order to introduce the state vector at the previous time step $x_{t-1}$ into equation (3.63), we write the goal in the form of the marginal distribution of $x_t$ as shown in equation (3.64).

$$p(x_t \mid z_{1:t}) = \int_{x_{t-1}} p(x_t, x_{t-1} \mid z_{1:t})dx_{t-1} \quad (3.64)$$

According to the Bayes’ rule, a conditional distribution can be expressed as its corresponding joint distribution over its marginal distribution as shown in equation
If we apply the Bayes’ rules to equation (3.64), it can be further derived as equation (3.66).

\[ p(x_t \mid z_{1:t}) = \int_{x_{t-1}} p(x_t, x_{t-1} \mid z_{1:t}) dx_{t-1} \]  

\[ = \int_{x_{t-1}} \frac{p(x_t, x_{t-1}, z_{1:t})}{p(z_{1:t})} dx_{t-1} \]  

\[ = \int_{x_{t-1}} \frac{p(z_t \mid x_t, x_{t-1}, z_{1:t-1})p(x_t \mid x_{t-1}, z_{1:t-1})p(x_{t-1} \mid z_{1:t-1})p(z_{1:t-1})}{p(z_{1:t})} dx_{t-1} \]  

\[ = \int_{x_{t-1}} \frac{p(z_t \mid x_t, x_{t-1}, z_{1:t-1})p(x_t \mid x_{t-1}, z_{1:t-1})p(x_{t-1} \mid z_{1:t-1})}{p(z_t \mid z_{1:t-1})} dx_{t-1} \]  

If we further apply the two assumptions from equation (3.61, 3.62) to equation (3.66), equation (3.66) can be derived as shown in equation (3.67), which is also called the Chapman-Kolmogorov equation.

\[ p(x_t \mid z_{1:t}) = \int_{x_{t-1}} p(x_{t-1} \mid z_{1:t-1}) \frac{p(z_t \mid x_t)p(x_t \mid x_{t-1})}{p(z_t \mid z_{1:t-1})} dx_{t-1} \]  

(3.67)

where \( p(z_t \mid z_{1:t-1}) \) can be treated as a constant normalization factor.

Equation (3.67) describes the recursion from \( p(x_{t-1} \mid z_{1:t-1}) \) to \( p(x_t \mid z_{1:t}) \). This equation includes two key components of the goal of Bayesian filtering: the state transition model \( p(x_t \mid x_{t-1}) \) and the measurement model \( p(z_t \mid x_t) \). These two components also correspond to the two major steps of Bayesian filtering: the prediction step and the update step. In the prediction step, a filter predicts the distribution of the state vector at current time step \( x_t \) based on the distribution from last time step \( x_{t-1} \) through state transition model \( p(x_t \mid x_{t-1}) \). In the update step, the predicted distribution is updated with the most recently received measurement through the measurement model \( p(z_t \mid x_t) \). Notice that the marginal distribution \( p(z_t \mid z_{1:t-1}) \) in equation (3.67) does not need to be calculated explicitly but rather it can be treated as a normalization factor.
There are many algorithms designed to solve the Bayesian filtering problem. Based on how the target distribution $p(x_t | z_{1:t})$ is represented, the most popular two groups of approaches are:

- Kalman filters (KF) [50]. $p(x_t | z_{1:t})$ is represented as a Gaussian distribution, and the state transition model together with the measurement model are assumed to be linear.

- Sequential Monte Carlo (SMC) Methods [64]. $p(x_t | z_{1:t})$ is represented as weighted particles using the importance sampling technique.

### 3.2.2 Kalman Filters

The Kalman filter was invented in early 1960 as a technique for filtering and prediction in linear Gaussian systems [45]. A linear Gaussian system has three major components:

- The state transition model $p(x_t | x_{t-1})$ is a linear model with added white Gaussian noise. This is expressed in the equation below:
  \[
  x_t = Ax_{t-1} + Bu_t + \epsilon_t, \tag{3.68}
  \]

where $u_t$ is the input vector (e.g., a vector of joint forces of a robotic arm for a robotic manipulation task), $A$ is a square matrix, $B$ is a matrix with its number of rows equal to the size of $x_t$ and its number of columns equal to the size of the $u_t$, and $\epsilon_t$ is a white Gaussian noise vector with its covariance denoted as $R_t$. As a result, the state transition model can be expressed as a Gaussian distribution as follows:

- The measurement model $p(z_t | x_t)$ is a linear model with added white Gaussian noise.
noise. Its expression is shown in equation (3.70).

\[ z_t = Cx_t + \delta_t \]  

(3.70)

In equation (3.70), \( C \) is a matrix with its number of rows equal to the size of the measurement vector \( z_t \) and its number of columns equal to the size of \( x_t \), and \( \delta_t \) is a white Gaussian noise vector with its covariance denoted as \( Q_t \). Similar to the state transition model, the measurement model \( p(z_t \mid x_t) \) can also be represented by a Gaussian distribution as below:

\[
p(z_t \mid x_t) = \det(2\pi Q_t)^{-\frac{1}{2}} \exp\left\{ -\frac{1}{2} (z_t - Cx_t)Q_t^{-1}(z_t - Cx_t) \right\}
\]

(3.71)

- The initial distribution of the state vector \( x_0 \) is a Gaussian distribution with mean \( \mu_0 \) and covariance \( \Sigma_0 \).

In summary, for the linear Gaussian system, a Kalman filter is the result of linear algebraic operations of the mean and covariance matrices of the two models in equation (3.67): the state transition model \( p(x_t \mid x_{t-1}) \) and the measurement model \( p(z_t \mid x_t) \). The Kalman filter algorithm shown in Algorithm 1 is a direct result of equation (3.67) with the Gaussian assumption of the two models, where line 2 and line 3 perform the prediction step of the Bayesian filtering, and line 5 to line 7 combine the measurement with the prediction and perform the updating step. \( \mu_t \) and \( \Sigma_t \) are the mean vector and the covariance matrix of the Gaussian distribution of the state vector at time step \( t \). In the following content of this chapter, we use \( \mu_t \) and \( \Sigma_t \) to denote the mean vector and covariance matrix for the posterior distribution, and we use \( \bar{\mu}_t \) and \( \bar{\Sigma}_t \) to denote those for the predicted distribution. Fig. 3.6 gives a visualization of the two steps for a Kalman filter, where the prediction step moves the prior distribution to the predicted distribution, and the updating step incorporates the measurement distribution and updates the predicted distribution to the posterior distribution.
Algorithm 1 Kalman Filter

1: function KALMAN FILTER($\mu_{t-1}$, $\Sigma_{t-1}$, $u_t$, $z_t$)
2:     $\bar{\mu}_t = A\mu_{t-1} + Bu_t$
3:     $\bar{\Sigma}_t = A\Sigma_{t-1}A^T + R_t$
4:     $K_t = \bar{\Sigma}_t C^T (C\bar{\Sigma}_t C^T + Q_t)^{-1}$
5:     $\mu_t = \bar{\mu}_t + K_t(z_t - C\bar{\mu}_t)$
6:     $\Sigma_t = (I - K_tC)\bar{\Sigma}_t$
7: return $\mu_t$, $\Sigma_t$
8: end function

Figure 3.6: Visualization of the two steps of a Kalman filter. In the prediction step, the initial distribution (blue) is updated to the predicted distribution (orange). In the updating step, the predicted distribution is combined with the measurement distribution (yellow) and is updated to the posterior distribution (purple).
3.2.2.1 Extended Kalman Filter

In practice, however, many systems do not satisfy the requirements of the linear state transition model and the linear measurement model for a linear Gaussian system, and thus KF cannot be applied to more general scenarios. Many KF-like algorithms have been proposed to resolve the non-linearity problem for Kalman filters. The most famous and popular approach is the extended Kalman filter (EKF). In EKF, instead of assuming a linear state transition model and a linear measurement model, the two models are governed by non-linear functions with white Gaussian noise vectors [45]:

\[
x_t = g(x_{t-1}, u_t) + \epsilon_t \quad (3.72)
\]
\[
z_t = h(x_t) + \delta_t \quad (3.73)
\]

The main idea of EKF is linearization. In EKF, the non-linear functions \(g\) and \(h\) are approximated by linear functions that are tangent to them at the mean of their prior Gaussian distributions. Particularly, EKF takes the linearization via first-order Taylor series expansions of the non-linear functions, and equations (3.72, 3.73) are approximated as follows:

\[
G_t = g'(\mu_{t-1}, u_t) \quad (3.74)
\]
\[
x_t \approx g(\mu_{t-1}, u_t) + G_t(x_{t-1} - \mu_{t-1}) + \epsilon_t \quad (3.75)
\]
\[
\bar{\mu}_t = g(\mu_{t-1}, u_t) \quad (3.76)
\]
\[
H_t = h'(\bar{\mu}_t) \quad (3.77)
\]
\[
z_t \approx h(\bar{\mu}_t) + H_t(x_t - \bar{\mu}_t) + \delta_t \quad (3.78)
\]

where \(\mu_t\) is defined in Algorithm 1, and \(g'(\cdot)\) and \(h'(\cdot)\) are the derivative functions of \(g(\cdot)\) and \(h(\cdot)\).

In equations (3.75, 3.78), we can view the Jacobian matrices \(G_t\) and \(H_t\) as the approximated Kalman filter \(A\) and \(C\) matrices. Then the EKF algorithm follows exactly the same steps as the KF algorithm in Algorithm 1 with the approximated matrices, which is shown in Algorithm 2.
Algorithm 2 Extended Kalman Filter

1: function Extended Kalman Filter($\mu_{t-1}$, $\Sigma_{t-1}$, $u_t$, $z_t$)
2:     $\bar{\mu}_t = g(\mu_{t-1}, u_t)$
3:     $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
4:     $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$
5:     $\mu_t = \bar{\mu}_t + K_t (z_t - h(\bar{\mu}_t))$
6:     $\Sigma_t = (I - K_t H_t) \bar{\Sigma}_t$
7:     return $\mu_t$, $\Sigma_t$
8: end function

3.2.2.2 Unscented Kalman Filter

Another method of tackling the non-linearity issue for Kalman filters is through the unscented transform, and this yields the unscented Kalman filter (UKF) [45]. Similar to EKF, UKF also assumes state transition model and the measurement model are non-linear as defined in equations (3.72, 3.73). However, instead of approximating the non-linear functions $g$ and $h$ by their Taylor series expansions, UKF extracts the so-called sigma points from the Gaussian distribution and passes them through $g$ and $h$. In general, for an $n$-dimensional Gaussian distribution with mean $\mu$ and covariance matrix $\Sigma$, UKF will sample $2n + 1$ sigma points as below:

\[
x^{[0]} = \mu \tag{3.79}
\]
\[
x^{[i]} = \mu + (\sqrt{n + \lambda}) \Sigma_i \quad \text{for } i = 1, \cdots, n \tag{3.80}
\]
\[
x^{[i]} = \mu - (\sqrt{n + \lambda}) \Sigma_i \quad \text{for } i = n + 1, \cdots, 2n \tag{3.81}
\]

where the subscript $i$ indicates the $i$th column of a matrix, $\lambda = \alpha^2 (n + \kappa) - n$ and $\alpha$ and $\kappa$ are both scaling parameters that determine how far the sigma points are spread from the mean. Fig. 3.7 shows an example of the sigma points for a two-dimensional Gaussian distribution. Additionally, there are two weights associated with each sigma point: $w_m$ (for computing the mean) and $w_c$ (for computing the covariance). The definitions for the weights are shown below:

\[
w_m^{[0]} = \frac{\lambda}{n + \lambda} \tag{3.82}
\]
Figure 3.7: The sigma points (red dots) for a two-dimensional Gaussian distribution whose contours are projected on the xy-plane.

\[
\begin{align*}
    w_c^{[0]} & = \frac{\lambda}{n+\lambda} + (1 - \alpha^2 + \beta) \\
    w_m^{[i]} & = w_c^{[i]} = \frac{1}{2(n+\lambda)}
\end{align*}
\]

where \( \beta \) is a parameter to encode additional knowledge about the target distribution underlying the Gaussian representation. If the distribution is Gaussian, then the optimal value for \( \beta \) is 2. For the prediction step, the mean \( \bar{\mu} \) and the covariance matrix \( \bar{\Sigma} \) for the predicted distribution are shown below:

\[
\begin{align*}
    \bar{\mu}_t & = \sum_{i=0}^{2n} w_m^{[i]} g(x_t^{[i]}, u_t) \\
    \bar{\Sigma}_t & = \sum_{i=0}^{2n} w_c^{[i]} (g(x_t^{[i]}, u_t) - \bar{\mu}_t)(g(x_t^{[i]}, u_t) - \bar{\mu}_t)^T + R_t
\end{align*}
\]

I leave out the derivation for the updating step of UKF. Readers are encouraged to read more details about UKF in [45].

3.2.3 Sequential Monte Carlo Methods

Kalman filter and its variations rely on the assumptions that the state and measurement models are linear or can be approximated by linear functions, and the
target distributions are Gaussian. However, these assumptions can sometimes be impractical, especially when the models become highly non-linear and non-smooth, such as the model of multi-body dynamic system in a robotic manipulation task. An alternative approach is the Sequential Monte Carlo (SMC) method, which was proposed in the early 1990s. SMC methods combine the power of Monte Carlo sampling methods with the Bayesian filtering methods. Instead of linearizing the state transition and measurement models, SMC methods sample from the non-linear models and represent the target distribution in the form of particles.

### 3.2.3.1 Importance Sampling

One of the key features for SMC methods is that the target distribution is represented in the form of randomly sampled data points (particles). We consider $f(x)$ as a probability density function, then if we generate $m$ random samples $(x_1, x_2, \cdots, x_m)$ from $f(x)$, $f(x)$ can be approximated by its empirical measure $\hat{f}(x)$ as follows [20]:

$$\hat{f}(x) = \frac{1}{N} \sum_{j=1}^{N} \delta_{x_j}(x)$$

(3.87)

where $\delta_{x_0}(x)$ denotes the Dirac delta function located at $x_0$. Further, in a classical Monte Carlo integration problem, we usually consider an expectation problem as follows [64]:

$$\mathbb{E}_f[h(X)] = \int_X h(x)f(x)dx$$

(3.88)

If we generate $m$ random samples $(x_1, x_2, \cdots, x_m)$ from $f(x)$, then the expectation can be approximated by the empirical average $\hat{h}_m$ as the following:

$$\hat{h}_m = \frac{1}{m} \sum_{j=1}^{m} h(x_j)$$

(3.89)

However, sometimes it is hard to generate samples from $f(x)$, and we seek the help from the importance sampling. In importance sampling, instead of sampling
directly from the target distribution \( f(x) \), we generate samples from a proposal distribution \( g(x) \), which is a distribution that can be sampled easily. We also require the support of \( g(x) \) to be a superset of the support of \( f(x) \), \( \text{supp}(g) \supset \text{supp}(f) \). We can rewrite the expectation problem in equation (3.88) as below:

\[
E_f[h(X)] = \int_X h(x) \frac{f(x)}{g(x)} g(x) dx
\]  

(3.90)

If we draw samples \((x_1, x_2, \ldots, x_m)\) from \( g(x) \), then the expectation can be approximated by the equation below:

\[
\hat{h}_m = \sum_{j=1}^{m} W(x_j) h(x_j)
\]  

(3.91)

\[
w(x_i) = \frac{f(x_i)}{g(x_i)}
\]  

(3.92)

\[
W(x_i) = \frac{w(x_i)}{\sum_{j=1}^{m} w(x_j)}
\]  

(3.93)

where the factor \( \frac{1}{m} \) is eliminated due to the normalization in equation (3.91). In particular, if we choose \( h(x) \) to be a Dirac delta function, the empirical measure of \( f(x) \) is expressed as in equation (3.94).

\[
\hat{f}(x) = \sum_{j=1}^{m} W(x_j) \delta_{x_j}(x),
\]  

(3.94)

where \( w(x) \) and \( W(x) \) have are defined in equations (3.92, 3.93).

Equation (3.94) shows that even with a probability distribution that is hard to draw samples from, we can still approximate the distribution through sampling a proposal distribution, such as a multivariate Gaussian. This idea further develops into the particle filter, which is discussed in the following section.
3.2.3.2 Particle Filter

Another way of expressing the target distribution in equation (3.67) is in the form of its joint distribution as follows:

\[
p(x_{1:t} | z_{1:t}) = p(x_{1:t-1} | z_{1:t-1}) \frac{p(z_t | x_t)p(x_t | x_{t-1})}{p(z_t | z_{1:t-1})} \tag{3.95}
\]

Assuming that our proposal distribution is \(q(x_{1:t})\), then according to importance sampling, with \(m\) samples drawn from \(q(x_{1:t})\), the target distribution can be approximated as:

\[
\hat{p}(x_{1:t} | z_{1:t}) = \sum_{j=1}^{m} W(x_j^{1:t}) \delta_{x_j^{1:t}}(x_{1:t}), \tag{3.96}
\]

where \(W(x_{1:t})\) is defined similarly to equation (3.93). If we take a further look, \(W(x_{1:t})\) can be expressed in relation with \(W(x_{1:t-1})\) as below:

\[
W(x_{1:t}) = \frac{p(x_{1:t} | z_{1:t})}{q(x_{1:t})} = \frac{p(x_{1:t-1} | z_{1:t-1})}{q(x_{1:t-1})} \frac{p(x_{1:t} | z_{1:t})}{p(x_{1:t} | x_{1:t-1})} W(x_{1:t-1}) \tag{3.97}
\]

One natural proposal distribution is to preserve all the samples at time step \(t - 1\), and use the state transition model \(p(x_t | x_{t-1})\) to generate the state vector \(x_t\) [41]. As a result, \(q(x_t | x_{1:t-1})\) equals to \(p(x_t | x_{t-1})\), and \(W(x_{1:t})\) can be further derived as:

\[
W(x_{1:t}) = W(x_{1:t-1}) \frac{p(x_{1:t} | z_{1:t})}{p(x_{1:t-1} | z_{1:t-1})} \frac{p(x_{1:t} | z_{1:t})}{p(x_{1:t} | x_{1:t-1})} \tag{3.98}
\]

where \(\eta = \frac{1}{p(z_t | z_{1:t-1})}\) is the normalization factor.
Based on the derivations above, particle filters have been developed to address Bayesian filtering problems for nonlinear nondifferentiable dynamic systems [38]. A particle filter represents a probability distribution over state space as a set of weighted samples, or “particles”. The weight $W(x)$ of each particle is thus approximately proportional to the probability of the corresponding variables according to the current distribution. Fig. 3.8 shows a highly schematic illustration of one iteration of a particle filter, wherein a multi-dimensional state space is displayed as the horizontal axis. The top graph shows a continuous probability distribution over the state space, given measurements and inputs up to time step $t$. The particle representation of this distribution is shown as circles whose sizes are proportional to their weights.
Since it is a Bayesian filtering method, a particle filter iterates two steps for each time step. In the prediction step, the particles are propagated according to the system dynamic model (the second row of Fig. 3.8). In the correction step, with the observation data, the measurement model (the third row of Fig. 3.8) is used to adjust the weights of the particles to produce a new approximation to the state distribution of the next time step (the fourth row of Fig. 3.8). These two steps represent a Bayesian optimal decision-making process of a particle filter. The relative levels of importance of the model’s predictions and the sensor’s measurements in this process fluctuate in response to the magnitudes of the uncertainties in the dynamic
model (caused by unmodeled dynamics and parameter errors) and the levels of noise in the sensor data.

Algorithms 3 and 4 summarize the steps of a particle filter for one time step. In these algorithms, $x_t$ represents the superset of the vectors of all estimated variables and particles at time step $t$ with $x_t^{[i]}$ representing the vector of the $i$th particle at time step $t$. Similarly, $w_t$ represents the weights of all particles at time step $t$ with $w_t^{[i]}$ representing the weight of the $i$th particle. For each time-step, the particles from the last time step, $x_t$, are fed to the state transition model to get the particles for the next time step $x_{t+1}$. Then $x_{t+1}$ is combined with the measurements through the measurement model, denoted by $P(y_{t+1} \mid x_{t+1})$. Then the weight vector at time step $t + 1$, $\bar{w}_{t+1}$, is updated as $\bar{w}_{t+1}^{[i]} \leftarrow P(z_{t+1} \mid \bar{x}_{t+1}) \cdot w_t^{[i]}$ and then renormalized. The estimation for time-step $t + 1$ is generated by the dot product of $x_{t+1}$ and $w_{t+1}$.

**Algorithm 3** Particle Filter

```plaintext
function Filter($x_t$, $u_t$, $y_{t+1}$, $w_t$)
    $x_{t+1}$, $\bar{w}_{t+1} = Update\_State(x_t, u_t, N, w_t)$
    Normalize $\bar{w}_{t+1} \rightarrow w_{t+1}$
    return $x_{t+1}$, $w_{t+1}$
end function
```

**Algorithm 4** Particle Filter Update State

```plaintext
function Update\_State($x_t$, $u_t$, $N$, $w_t$)
    for $i = 1 \rightarrow N$ do
        $x_{t+1}^{[i]} = State\_Transition\_Model(x_t^{[i]}, u_t)$
        $w_{t+1}^{[i]} = P(y_{t+1} \mid x_{t+1}^{[i]}) \cdot w_t^{[i]}$
    end for
    return $x_{t+1}$, $w_{t+1}$
end function
```

**3.2.3.3 Weight Degeneracy and Particle Resampling**

There is one fundamental difficulty with the particle filter in Algorithms 3 and 4. As we update the weights for all the particles over time, the variance of the weights tends to grow exponentially, which means that weights tend to zero except for one that is close to one. This indicates that only one particle will contribute to
the final estimation result. This problem is known as the weight degeneracy problem and significantly reduces the effect of particle filters [64].

Resampling is an effective technique to alleviate the weight degeneracy problem for particle filters. Specifically, for one time step of a particle filter, after generating the particles $x_{t}^{[i]}$ and their associated weights $w_{t}^{[i]}$, we further select $x_{t}^{[i]}$ with probability $w_{t}^{[i]}$. This process is called resampling as it is sampling from a distribution approximated by samples. If the number of particles is $N$, we simply repeat the resampling process $N$ times. At the end of the resampling processes, we set the weights of all the resampled particles to $\frac{1}{N}$.

Resampling helps to eliminate particles with low weights, and it can benefit particle filters by allocating computational power on the particles with high weights as they carry more information about the target distribution. However, as we set the weights of all particles equal, resampling does introduce some extra variance to the particles as particles with low weights now have equal weights with all the others. Therefore, if the variance of the weights is small, i.e. the differences among the weights of all particles are small, then resampling may be unnecessary [20]. In practice, it is more sensible to perform resampling when the variance of the unnormalized weights is above a certain threshold. The most common measure of the variance of the weights is the so-called Effective Sample Size (ESS) criterion [54], which is defined as below:

$$ESS = \left( \sum_{i=1}^{N} w_{t}^{[i]} \right)^{-1}$$ (3.99)

Based on resampling, the algorithm for a resample particle filter can be described in Algorithm 5, which is very similar to Algorithm 3 except for the resampling part.

3.2.3.4 Rao-Blackwellized Particle Filter

The major advantage of representing a distribution in the form of particles is that high-dimensional integration in Bayesian filtering with non-Gaussian nonlinear models can be approximated by the weighted averages over particles. However, one problem in terms of particle filters is that reliable approximations require the
Algorithm 5: Resample Particle Filter

\begin{algorithm}
\begin{function} {FILTER} {\(x_t, u_t, y_{t+1}, w_t\)}
\(\bar{x}_{t+1}, \bar{w}_{t+1} = \text{UPDATE\_STATE}(x_t, u_t, N, w_t)\)
\[
\left(\sum_{i=1}^{N} w_t^{[i]}\right)^{-1} \rightarrow ESS
\]
\textbf{if} \(ESS \geq \text{Threshold}\) \textbf{then}
\begin{for} {i} = 1 \rightarrow N \end{for}
\text{Draw j with probability } \propto \bar{w}_{t+1}^{[j]}
\[x_{t+1}^{[i]} = \bar{x}_{t+1}^{[j]}, \; w_{t+1}^{[i]} = \frac{1}{N}\]
\textbf{end for}
\textbf{else}
\text{Normalize } \bar{w}_{t+1} \rightarrow w_{t+1}
\[x_{t+1} = \bar{x}_{t+1}, \; w_{t+1} = \bar{w}_{t+1}\]
\textbf{end if}
\textbf{return} \(x_{t+1}, w_{t+1}\)
\end{function}
\end{algorithm}

number of particles to grow exponentially with the dimension of the state space. For a robotic manipulation problem, the state space of tracking a single object can be 12-dimensional, and with multiple objects in the system, the dimension of the overall state space can be extremely high, which leads to a requirement of a significant number of particles. This problem is also called the curse of dimensionality of particle filters [19].

Rao-Blackwellized particle filter or RBPF is a variant of particle filter that can alleviate the curse of dimensionality by only sampling part of the state space with the rest of the state space updated analytically [51]. In particular, if a model has a tractable substructure, which can be marginalized out if some nodes are known, we marginalize out the state variables in this substructure using the Rao-Blackwellized technique [21]. Considering partitioning the state vector \(x_{1:t}\) into two parts \((r_{1:t}, y_{1:t})\), where \(r_{1:t}\) is the root part of the state space, and \(y_{1:t}\) is the leaf part of the state space, an RBPF will sample the root state space using weighted particles just like a regular particle filter, and update the leaf state space, which is tractable, using optimal filtering methods. For each time step, an RBPF typically consists of two major substeps [13]:

- Update the distribution of the root state space with the derivations similar to
Figure 3.9: A diagram of a SLDS model. $r_t$ is the root state variable, $y_t$ is the leaf state variable, and $z_t$ is the measurement variable.

equations (3.96, 3.98) shown as below:

$$
\hat{p}(r_{1:t} \mid z_{1:t}) = \sum_{j=1}^{m} W(r_{1:t}^{j}) \delta_{r_{1:t}}^{j} (r_{1:t}) \tag{3.100}
$$

$$
W(r_{1:t}) = \eta W(r_{1:t-1}) p(z_t \mid r_t) \tag{3.101}
$$

- Update the distribution of the leaf state space based on the samples from the root state space sampling:

$$
p(y_{1:t} \mid z_{1:t}) = \sum_{r_{1:t}} p(y_{1:t} \mid r_{1:t}, z_{1:t}) p(r_{1:t} \mid z_{1:t}) \tag{3.102}
$$

$$
\approx \sum_{r_{1:t}} p(y_{1:t} \mid r_{1:t}, z_{1:t}) \sum_{j=1}^{m} W(r_{1:t}^{j}) \delta_{r_{1:t}}^{j} (r_{1:t})
= \sum_{j=1}^{N} W(r_{1:t}^{j}) p(y_{1:t} \mid r_{1:t}^{j}, z_{1:t})
$$

After the two substeps above, the root and leaf state space can be joined to compose the whole state space, and the joint vector can then be weighted and resampled for the next time step. One model in which RBPF is very effective is the switching linear dynamic system (SLDS) [60]. An example of the SLDS model is shown in Fig. 3.9. In a SLDS model, once the root state variable is known, the rest of the state space simply becomes a linear system and can be optimally estimated.
using Kalman filters. Similarly, in a robotic manipulation task, as will be shown in Chapter 5, the state space can be divided into the contact state space and the continuous state (e.g., poses of the objects) space. The contact state is viewed as the root state variable, and the continuous state becomes the leaf state and is estimated by Kalman filters.

### 3.3 Contact Graphs

In a robotic manipulation task, there are contacts constantly making and breaking between the robot and the objects. For example, when a robot picks up a cup from a table, its gripper will first make contacts with the cup, and then when the cup is picked up, the contacts between the cup and the table will break. Since there are contact forces associated with the contacts, a contact state, which is a set of contacts, imposes a constraint on the motions of the objects. For example, when a cup sits on the table, the contacts between it and the table constrain motions of the cup from falling through the table. One can view the contact states as critical discrete events during a robotic manipulation task. For example, a robot should not try to lift the cup without making contacts with it. There have been many works that develop methods to plan motions and generate control strategies for robotic manipulation tasks based on these discrete contact states (see, for example, [72, 28, 55, 23, 56]).

A contact graph [75, 71] describes the possible transitions of the contact states in a multi-body system. To construct a contact graph, an object’s geometry is first decomposed into a set of boundary elements. The boundary elements of an object are its faces, the boundary elements of a face are the edges and vertices that bound it, and the boundary elements of an edge are its vertices. In a contact graph, the most basic element is a “principal” contact (PC) [75]. A PC is the contact between a pair of boundary elements of two bodies. Examples of the three types of PCs in the two-dimensional case are shown in Fig. 3.10, where the two squares can have an edge-edge PC, an edge-vertex PC or a vertex-vertex PC.

For two 2D objects $A$ and $B$, we define two PCs between them: $PC_1 = \{c^A, c^B\}$ and $PC_2 = \{d^A, d^B\}$, where $c^A$, $c^B$, $d^A$ and $d^B$ are boundary elements on objects $A$
Figure 3.10: Three types of “principal” contacts between two squares in 2D. Each type consists of two boundary elements being in contact.

Figure 3.11: An example of two PCs between two squares. $PC_1$ is an edge-edge PC with edges $c^A$ and $c^B$. $PC_2$ is a vertex-edge PC with vertex $d^A$ and edge $c^B$.

and $B$. $PC_1$ contains $PC_2$, iff one of the following conditions holds:

- $d^A$ is a boundary element of $c^A$, and $d^B$ is a boundary element of $c^B$.
- $d^A$ is a boundary element of $c^A$, and $d^B$ is $c^B$.
- $d^A$ is $c^A$, and $d^B$ is a boundary element of $c^B$.

For example, in fig. 3.11, there exists two PCs between objects $A$ and $B$: an edge-edge PC $PC_1 = c^A, c^B$ and a vertex-edge PC $PC = d^A, c^B$. Following the rules defined above, $PC_1$ contains $PC_2$, since vertex $d^A$ is a boundary element of edge $c^A$ and $c^B$ is $c^B$. 
Figure 3.12: The boundary elements of two rectangles. $D_i$ and $d_i$ represent the vertices, and $E_i$ and $e_i$ correspond to the edges.

A contact graph can be visualized as an undirected graph with each node representing a contact state or a contact formation (CF) [75]. A CF is a set of PCs formed at the same time. An edge connecting two CFs indicates that the two CFs are neighbors, which means that they can transition from one to another without going through any intermediate CFs. For two CFs $CF_i$ and $CF_j$, $CF_i$ contains $CF_j$ iff:

(a) the number of PCs in $CF_i$ is greater than or equal to that of $CF_j$ and

(b) every PC in $CF_j$ either belongs to $CF_i$ or is contained by a PC in $CF_i$.

In a contact graph, $CF_i$ and $CF_j$ are neighbors, iff either of them contains the other.

An example of a partial contact graph between two rectangular objects is shown in Fig. 3.13, where the red dots are used to mark the lower left vertex of the smaller rectangle and the upper left vertex of the bigger rectangle. Fig. 3.12 shows the boundary elements of the two rectangles, where $D_i$ and $d_i$ represent the vertices, and $E_i$ and $e_i$ correspond to the edges. The PCs of each CF in the graph is shown in the table below:

<table>
<thead>
<tr>
<th>CF</th>
<th>PC included</th>
<th>CF</th>
<th>PC included</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_1$</td>
<td>${E_1, e_1}$</td>
<td>$N_5$</td>
<td>${D_1, e_2}$</td>
</tr>
<tr>
<td>$N_2$</td>
<td>${E_1, d_1}$</td>
<td>$N_6$</td>
<td>${E_2, d_2}$</td>
</tr>
<tr>
<td>$N_3$</td>
<td>${D_1, d_1}$</td>
<td>$N_7$</td>
<td>${E_2, e_2}$</td>
</tr>
<tr>
<td>$N_4$</td>
<td>${E_2, d_1}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The edges in the contact graph shown in fig. 3.13 are constructed using the definition of neighbor CFs shown above. For example, $\{E_1, e_1\}$ contains $\{D_1, d_1\}$ as $D_1$ is a boundary element of $E_1$ and $d_1$ is a boundary element of $e_1$, and therefore $N_1$ and $N_3$ are neighbors since the only PC in $N_3$ is contained by the PC in $N_1$. 
Figure 3.13: An example of a partial contact graph of two rectangular objects. The red dot is used to mark one particular vertex of the smaller rectangle.

There have been algorithms developed to compute the contact graph, and they range from analytical methods [71, 75] to random sampling approaches [49].

In the context of state estimation for multi-body dynamic system, the contact state can be used to linearize the model of multi-body dynamics, which we will show in the later chapters of this thesis. Therefore, the contact graph can be used as a powerful tool in the state estimation problem. However, as the contact graph is constructed purely based on kinematic relations, we add dynamic properties, such as the stick-slip status of a contact, to the contact graph so that it can be used in the state estimation problem for robotic manipulation tasks.
CHAPTER 4
Modeling Options Evaluation

We propose to solve the state estimation problem for robotic manipulation tasks by combining an accurate model of multi-body dynamics with Bayesian filtering methods. Because robots take measurements of the manipulation tasks at discrete time steps, we choose to focus on the Stewart-Trinkle formulation of the model of multi-body dynamics, which is discussed in section 3.1.3, in this thesis.

As discussed in Chapter 3, the complementarity problem model treats the contacts as a collection of complementarity conditions. With linearized friction cones, the Linear Complementarity Problem (LCP) model describes the multi-body dynamics as a set of linear equations plus a set of complementarity conditions. Further, as indicated in [65], the LCP model is a piecewise linear model with its “switching points” between the adjacent linear pieces being the contact states. This point is illustrated using the example introduced in section 3.1.3.1 as shown in Fig. 4.1.

Figure 4.1: The piecewise linear property of the LCP model illustrated with the example from section 3.1.3.1. The green curves represent the trajectories of the object’s center of gravity.

Three key frames of the robot pushing the object at three different time steps ($t_1$, $t_2$ and $t_3$) are shown in this figure. The green curves represent the trajectories

---

of the object’s center of gravity. From time step \( t_1 \) to time step \( t_2 \), the object slides on the plane, and from time step \( t_2 \) to \( t_3 \), the object tilts as the robot keeps pushing on it. There are five contacts, which are shown in Fig. 4.2, involved in this process. From time step \( t_1 \) to \( t_2 \), contacts 1 and 2 are sticking contacts, and contacts 3 and 4 are sliding contacts. From time step \( t_2 \) to \( t_3 \), contacts 2, 4 and 5 are all sticking contacts. While the general form of the LCP model of the object’s motions is shown in equations (3.52 - 3.54), with known contact states \(^{1}\) in each period \((t_1 - t_2 \text{ or } t_2 - t_3)\), the complementarity conditions of the LCP model (equation (3.54)) can be eliminated and matrices can be determined as shown below:

\(^{1}\)The friction directions of the sliding contacts are known since the sliding directions are known.
In the matrices above, \( n_i \) represents the contact normal vector of the \( i \)th contact, \( n_{fi} \) represents the contact frictional vector of the \( i \)th contact, and \( r_i \) represents the vector pointing from the \( i \)th contact point to the object’s center of gravity. The definitions of these vectors depend on the poses of the object and the robot. For example, from time step \( t_1 \) to time step \( t_2 \), vector \( n_3 = \begin{bmatrix} 0 & 1 \\ \end{bmatrix}^T \), vector \( n_{f3} = \begin{bmatrix} -1 & 0 \\ \end{bmatrix}^T \) and vector \( r_3 = \begin{bmatrix} \frac{1}{3} \\ \frac{2}{3} \\ \end{bmatrix}^T \), where \( l \) is the length of the triangle’s legs.

While there are multiple choices of Bayesian filtering methods such as the Kalman filter and the Sequential Monte Carlo Methods/particle filter, one needs to choose the method that suits the model of multi-body dynamics. Since the model of multi-body dynamics is highly nonlinear and non-smooth, Kalman filters and extended Kalman filters do not meet the requirements of our problem. As the probability distribution of the states can be multimodal (see an example in Fig. 4.3), unscented Kalman filter will also fail. The combination of a contact state estimator with a switching Kalman filter may be appealing. However, there are no closed-form expressions for the transitions between any two contact states. Furthermore, the probability distribution of the possible contact states is often multimodal during a robotic manipulation task. Fig. 4.3 shows an example for which the current con-
tact state (top picture) is symmetric with respect to the two quantitatively different possible future contact states (bottom pictures). In this case, the probability distribution of the future contact states will be bi-modal with significant probability mass corresponding to each possible future contact state. Precisely because of the above reasons, using a switching Kalman filter along with a contact state estimator will fail in many robotic manipulation tasks when the contact state distribution becomes multimodal.

Figure 4.3: An example of a bi-modal contact state distribution. A square drops onto a triangular obstacle. The square can slide along either side of the obstacle after the contact. Therefore, the distribution of the contact state is bi-modal.

An alternative to estimate the states and parameters would be to use a multi-model Kalman filter, which tracks all the possible models (all possible contact states) with parallel Kalman filters. However, the multi-model Kalman filter becomes infeasible once the number of contact states becomes large. If there are $N$ possible
contact states, which define \( N \) possible models, a multi-model Kalman filter needs to run \( N^2 \) Kalman filters to check all possible model switches at each time step, and the number of Kalman filters grows exponentially with the number of time steps. For a robotic manipulation problem, \( N \) increases rapidly as the complexity of the robotic manipulation task increases. For example, the number of possible contact states can be thousands for a three-dimensional robotic grasping task. As a result, even suboptimal solutions that maintain \( N \) Kalman filters, such as the generalized pseudo-Bayesian of order 1 (GPB1) algorithm [53], are still far from satisfactory. Therefore, it is not practically feasible to use a multi-model Kalman filter in a robotic manipulation problem.

As discussed in section 3.2.3, a particle filter does not place any restrictions on its state transition and measurement models. Therefore, a particle filter is easier to be used with a multi-body dynamics model comparing with the other Bayesian filtering methods. Further still, a Rao-Blackwillized particle filter (RBPF) works very well with a piecewise linear model. As discussed in Chapter 3, in an RBPF, the distribution of the contact state can be estimated using Monte Carlo sampling and the distribution of the continuous state can be computed using Kalman filters. The advantage of using an RBPF over a multi-model Kalman filter is that the particle filter only requires enough particles to represent the distribution of the contact states for one time step. Therefore an RBPF can maintain a much smaller set of Kalman filters comparing with that of a multi-model Kalman filter. Because of these advantages of a particle filter, we limit our choice of the Bayesian filtering methods to particle filters for this thesis effort.

To understand the design trade-offs of this type of particle filters, we evaluate multiple modeling options for the filters in a one-dimensional experiment. These modeling options include combinations of different contact dynamic models having different variants of particle filtering methods. For the contact dynamics models, we have chosen two popular models, namely, the rigid body model and the compliant body model. For the variants of particle filtering methods, we have chosen five particle filtering methods: a particle filter without considering noise in its state transition model, two particle filters with noise added to the outputs of their state
transition models, and two particle filters with noise added to the inputs of their state transition models.

In the remainder of this chapter, we will first introduce the setup of the one-dimensional experiment that we use in this evaluation, then details of the modeling options are discussed, and the discussion is followed by the analyses of the evaluation results.

### 4.1 Experiment Setup

Several researchers have developed particle filters to estimate the states and parameters of objects for robotic manipulation tasks. However, due to the complexity of their experiment setup, the effects of various contact and filtering models, which can have a huge impact on the estimation results, are obfuscated by implementation details. In our work, we propose to analyze the effects of the modeling options in a simpler one-dimensional experiment, where a robotic arm moves into contact with a fixed wall as shown in Fig. 4.4. Since we are most interested in the discrete contact state transitions, we simplify the problem by only considering the states in the approaching direction (the \(x\)-direction) toward the wall. Further, we consider the hand as not only having its own dynamic properties (i.e., position, velocity, etc.), but also the dynamic properties of the arm in the world frame. The dynamics of the robot arm can be expressed as below:

\[
\ddot{\mathbf{q}} = \mathbf{J}^T \lambda + \mathbf{D}(\theta, \dot{\theta})
\]  

(4.1)
where $q$ is the position of the robot hand, $\theta$ is the vector of the joint angles of the robot arm, $\lambda$ is the contact impulse, $\Psi(q)$ is the gap distance between the robotic hand and the wall, $\tilde{M}(\theta)$ is the arm inertia matrix, $\tilde{J}$ is the Jacobian associated with the contact constraints, and $\tilde{D}$ is the vector of all other forces including joint friction, Coriolis, and centripetal forces mapped into the world frame. Equation (4.2) is the complementarity condition explained in section 3.1.2.1. In our evaluation, we further assume that joint friction and the arm velocity are small enough so that the effects of $\tilde{D}$ may be neglected. Also, we isolate the robot hand from the rest of the robot arm and treat the robot as a robotic hand pushed by forces on the wrist. As a result, $\tilde{M}$ becomes a scalar, which equals to the mass of the robotic hand.

We use the following definition in the remaining of this chapter:

$m$: the mass of the robotic hand

$v_t$: the velocity of the robotic hand at time step $t$

$q_t$: the position of the robotic hand at time step $t$

$x_t$: the vector of the state, i.e., $\begin{bmatrix} q_t \\ v_t \end{bmatrix}$, of the robotic hand at time step $t$

$z_t$: the vector of the observation of the state at time step $t$

$u_t$: the vector of the applied force to the robotic hand at time step $t$

$\lambda_t$: the contact impulse between the robotic hand and the wall at time step $t$

$\Psi_t$: the gap distance between the robotic hand and the wall at time step $t$

### 4.2 Contact Dynamic Models

In our evaluation, we consider two widely used contact dynamic models: the rigid body model and the compliant body model, where the objects are allowed to deform in the neighborhood of the contacts.
4.2.1 Rigid Body Model

Similar to the discussion in section 3.1.3, applying the approach in [67] to our problem yields the following discrete-time dynamic model, which takes the form of a linear complementarity problem:

\[ m v_{t+1} = u_t h - \lambda_{t+1} h + m v_t \]  \hspace{1cm} (4.3)

\[ q_{t+1} = v_{t+1} h + q_t \]  \hspace{1cm} (4.4)

\[ \Psi_{t+1} = q_{\text{wall}} - q_{t+1} \]  \hspace{1cm} (4.5)

\[ 0 \leq \lambda_{t+1} \perp \Psi_{t+1} \geq 0. \]  \hspace{1cm} (4.6)

where \( q_{\text{wall}} \) is the position of the wall, \( \perp \) symbol connotes orthogonality. The last condition is known as the normal complementarity condition. It is the conjunction of three conditions: \( \lambda_{t+1} \geq 0, \Psi_{t+1} \geq 0, \) and \( \Psi_{t+1} \lambda_{t+1} = 0^2; \) contact impulses are compressive, no penetration is allowed, and the contact impulse is nonzero if and only if the robot touches the wall (when \( \Psi \) equals to zero).

Similar to equations (3.49 - 3.51), equations (4.3 - 4.6) correspond to a linear complementarity problem (LCP) for our one-dimensional problem with the unknowns \( q_{t+1}, v_{t+1} \) and \( \lambda_{t+1} \). This particular LCP has a unique solution for every possible value of \( h, m, q_t, \) and \( u_t \). The output of this discrete-time dynamic model is the solution to the LCP.

4.2.2 Compliant Body Model

Defining \( \lambda_{t+1} \) as a function of \( \Psi_t \) converts the rigid body model into one with contact compliance. In this evaluation, we use a nonsmooth version of the Voigt-Kelvin contact model, as defined in [66]. Letting \( \delta = -\Psi \) be the depth of penetration and \( K \) and \( C \) be the contact stiffness and damping constants, the contact model becomes:

\[ \delta_t = \max\{0, -\Psi_t\} \]  \hspace{1cm} (4.7)

\[ \dot{\delta}_t = \frac{1}{K} \left[ \max\{0, -\Psi_t\} - \max\{0, -\Psi_{t-1}\} \right] \]  \hspace{1cm} (4.8)

\(^2\Psi \) and \( \lambda \) are scalars in our problem.
Equations (4.7-4.9) replace the complementarity condition equation (4.6) so that the state transition model with compliance becomes the equations (4.3, 4.4, 4.5, 4.7, 4.8, 4.9). This model also has a unique solution, which can be obtained by solving equations (4.7-4.9) for $\lambda_{t+1}$, and then solving equations (4.3, 4.4) for $v_{t+1}$ and $q_{t+1}$. Similar to the rigid body model, the output of the deterministic state transition model is the solution to the equations (4.3, 4.4, 4.5, 4.7, 4.8, 4.9).

### 4.3 Filtering Methods

We also explore different variants of particle filtering methods. In Chapter 3, algorithms 3 and 4 summarize the steps for a particle filter. Among the variants of particle filters that we evaluated in this work, the differences lie in how we propagate (the third line in Algorithm 4) and reweight (the fourth line in Algorithm 4) the particles. If we use the contact dynamics models from section 4.2 as the state transition models, then algorithms 3 and 4 demonstrate a baseline particle filter for the state estimation problem. We call this baseline the basic particle filter (BPF). Based on this BPF, multiple new particle filters are created by changing (or using different) update state functions (algorithm 4). In the following sections, we offer the algorithms of these new particle filters in details.

#### 4.3.1 Adding Noise to the State Transition Model

One source of errors of our particle filters is the inaccuracy of the state transition model. For example, the mass of the robotic hand used in the model can be inaccurate. One common way of alleviating modeling inaccuracies is by adding noise to the state transition model. Equation (4.10) defines the most common way to add noise to deterministic state transition models,

$$x_{t+1} = F(x_t, u_t + \epsilon_t) + \eta_{t+1},$$

where $F$ is the state transition model, $\epsilon_t$ and $\eta_{t+1}$ are noise processes affecting the actuator input and the future state estimation respectively. In this evaluation
work, we assume that the noise processes in equation (4.10) are zero-mean Gaussian processes with given variances.

The pros and cons of the two noise processes are as follows: If only $\eta_{t+1}$ is present, then the distribution of $x_{t+1}$ can be derived from its distribution. For example, if $\eta_{t+1}$ is normally distributed with zero mean and standard deviation $\sigma$, i.e., $\eta_{t+1} = N(0, \sigma^2)$ then $\tilde{x}_{t+1} = N(F(x_t, u_t), \sigma^2)$, where $\tilde{x}_{t+1}$ represents the probabilistic distribution of the future state. Also, adding noise outside the state transition model (model of multi-body dynamics in our case) means that all particles except a subset of measure zero violate the physics constraints captured in the multi-body dynamic model. On the other hand, if we only add noise process $\epsilon_t$ to $u_t$, we assume that the actuators are not perfect effort sources and the resulting $x_{t+1}$ will satisfy the physical constraints. A drawback of this noise process is that the distribution of $x_{t+1}$ usually cannot be determined in closed form.

Based on the two noise processes described by equation (4.10), we designed two particle filters. The first is termed as the noisy input particle filter (NIPF), where we only consider the noise process $\epsilon_t$ for $u_t$. The second is termed as the noisy state particle filter (NSPF). In NSPF, only $\eta_{t+1}$ in the future state is considered. The update state functions of NSPF and NIPF and NSPF are similar with noise added to different places shown with different colors in algorithm 6, where the text contents in red correspond to NSPF, and the blue text contents correspond to NIPF, and the superscripts indicate that the variables belong to the $i$th particle, i.e., $w^{[i]}_t$ is the weight of the $i$th particle at time step $t$.

**Algorithm 6 NSPF/NIPF Update State**

```plaintext
function UPDATE_STATE(x_t, u_t, N, w_t)
for $i = 1 \rightarrow N$ do
    $x_{t+1}^{[i]} = \text{STATE_TRANSITION_MODEL}(x^{[i]}_t, u_t + \epsilon_t) + \eta_{t+1}$
    $w^{[i]}_{t+1} = P(y_{t+1} \mid x_{t+1}^{[i]}), \cdot w^{[i]}_t$
end for
return $x_{t+1}, w_{t+1}$
end function
```
Figure 4.5: The constraint boundary surface/line for our one-dimensional experiment. The state space that consists of the velocity $v_t$ and the position $q_t$ is shown as a two-dimensional plane. The red area indicates the valid half state space.

4.3.2 Projected Particle Filter

As discussed above, one major issue of NSPF is that the noise process $\eta_{t+1}$ added to the future state will cause violations of the physical constraints in the particles. One intuitive way to tackle this problem is to penalize the particles that violate the constraints. There are two questions that we need to answer before we penalize a particle:

- How do we tell whether a particle violates the physical constraints?
- How do we penalize a particle that has such violations?

In this evaluation work, we define a constraint boundary surface, which divides the state space into two half-spaces, i.e., the half-space that satisfies the physical constraints and that does not. Particularly, in our one-dimensional problem, the constraint boundary surface is a line, which is simply $q_t = q_{\text{wall}}$, and the left side of the line is the valid state space (see Fig. 4.5). The general constraint boundary surface can become quite complicated when there are multiple contacts and bilateral constraints in the system. Since the state estimation of a particle can be represented
as a point in the state space, to penalize a particle, we first calculate the distance from the point, which represents the particle in the state space, to the constraint boundary surface. In our problem, this distance is computed as $q_t - q_{wall}$, which is the distance between the point and the constraint half-plane, or the penetration depth. The particle is then “penalized” by reducing its weight monotonically with increasing penetration depths. Since we designed this particle filter that penalizes particles based on the projected distance to the boundary plane of their estimations in the state space, this particle filter is termed as the projected particle filter (PPF).

The statements above applies to the rigid body model naturally. However, for compliant body model, we need a slightly different scheme to penalize the particles. As shown in [66], for a compliant body model, an object consists of a rigid core with a compliant outer shell as the contact zone. Therefore, we define a thickness of the outer shell of the wall and apply a similar constraint boundary surface and a penalization method to the compliant body case. While PPF has the same general steps with those of BPF, it differs from BPF in its update state function, which is summarized in Algorithm 7. In Algorithm 7, function $Projected_{Dist}$ calculates the projected distance from the current state $x_t$ to the constraint boundary surface (a line in our case), $Estimate_{Penalty}$ calculates the penalty weight based on the projected distance $d$, and the superscript indicates that the variable belongs the $i$th particle.

### 4.3.3 Force State Particle Filter

In the physical experiments, we observed that one of the main sources of noise came from the force sensor that collected the input force values. Indeed, this problem can occur in certain cases where the presence of friction in the manipulator mechanism or the actuators is hard to model and control. As a result, we propose to treat the input force $u_t$ as a hidden state which is measured by the force sensor as $u'_t$. In this evaluation, we assume that the force that is applied to the robot hand is constant, and we treat the state transition model for $u_t$ as a random walk model, which is shown below:

$$u_{t+1} = u_t + \epsilon_t,$$  \hspace{1cm} (4.11)
Algorithm 7 PPF Update State

function UPDATE_STATE($x_t$, $u_t$, $N$, $w_t$)
    for $i = 1 \rightarrow N$ do
        if Violate-Constrain($x_t$, $u_t$) is true then
            $d[i] = \text{PROJECTED}\_\text{DIST}(x_t)$
            $\text{AddWeight}[i] = \text{ESTIMATE}\_\text{PENALTY}(d_i)$
        else
            $\text{AddWeight}_i = 1$
        end if
        Sample $\eta_i$ from $\mathcal{N}(0, \sigma^2)$
        $x_{t+1}^{[i]} = \text{STATE}\_\text{TRANSITION}\_\text{MODEL}(x_t^{[i]}, u_t) + \eta_i$
        $w_{t+1}^{[i]} = P(z_{t+1}|x_{t+1}^{[i]}) \cdot \text{AddWeight}[i] \cdot w_t^{[i]}$
    end for
    Normalize $w_{t+1}$
    return $x_{t+1}$, $w_{t+1}$
end function

where $\epsilon_t$ is a white Gaussian noise process, i.e., $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$. Since we consider noise in the force sensor and treat $u_t$ as a state, we term this particle filter the force state particle filter (FSPF). To further illustrate FSPF, we show its graphical structure in fig. 4.6. FSPF also follows the same general steps with those of a BPF. However, since the measurement models of $x_t$ and $u_t$ take different inputs, they must be evaluated separately. As a result, at time step $t$, each particle is first weighted by the measurement of the robotic hand’s position $z_t$, and then it is reweighted by the measurement of the applied force $u_t'$ (i.e., the reading from the force sensor). The algorithm of FSPF is summarized in Algorithm 8, where function Input_Transit represents the state transition model for the input force $u_t$, and the superscript indicates that a variable belongs to the $i$th particle.
Algorithm 8 FSPF Particle Filter

function \textsc{Filter}(x_t, u_t, y_{t+1}, u'_t, w_t)

for \(i = 1 \rightarrow N\) do

Sample \(\epsilon_i\) from \(N(0, \sigma^2)\)

\(\bar{x}_{t+1}^{[i]} = \text{STATE\_TRANSITION\_MODEL}(x_t^{[i]}, u_t)\)

\(\bar{u}_{t+1}^{[i]} = \text{INPUT\_TRANSIT}(u_t) + \epsilon_i\)

\(u^{[i]} w_{t+1}^{[i]} = P(u'_{t+1} | \bar{u}_{t+1}^{[i]}\)\)

\(\bar{w}_{t+1}^{[i]} = P(z_{t+1} | \bar{x}_{t+1}^{[i]}), u^{[i]} w_{t+1}^{[i]}, w_{t}^{[i]}\)

end for

Normalize \(\bar{w}_{t+1}\)

if resample condition satisfied then

for \(i = 1 \rightarrow N\) do

Draw \(j\) with probability \(\propto \bar{w}_{t+1}^{[j]}\)

\(x_{t+1}^{[i]} = \bar{x}_{t+1}^{[j]}, u_{t+1}^{[i]} = \bar{u}_{t+1}^{[j]}\)

\(w_{t+1}^{[i]} = \frac{1}{N}\)

end for

else

\(x_{t+1} = \bar{x}_{t+1}, u_{t+1} = \bar{u}_{t+1}, w_{t+1} = \bar{w}_{t+1}\)

end if

return \(x_{t+1}, w_{t+1}\)

end function
4.4 Evaluation Results and Discussions

For each modeling option, we pair a contact dynamic model with a filtering method, e.g., a rigid body model with NSPF. We evaluated the performances of different modeling options based on two criteria: (1) the accuracy of tracking the position of the robotic hand and (2) the accuracy of predicting contacts. To evaluate the first criterion, we calculate the weighted mean squared errors ($WMSE$) for each of the modeling options. We then evaluate the second criterion using the weighted contact confidence errors ($WCCE$). Both $WMSE$ and $WCCE$ represent the performance of a filter over an entire experiment. If in an experiment, a filter follows the ground truth trajectory exactly, $WMSE$ and $WCCE$ will be zero for the experiment. Smaller values of $WMSE$ indicate better trajectory tracking results, and smaller values of $WCCE$ show better contact estimation performances.

We define $WMSE$ as weighted error of the estimated states made by each particle in a particle filter. The error is accumulated over all time steps and divided by the total number of time steps to get a mean error over the entire experiment. The equation used for calculating a $WMSE$ is shown as equation 4.12,

$$WMSE = \sqrt{\frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{N} w_t^i (\hat{q}_t^i - q_t)^2},$$

where $T$ is the total number of time steps of the trajectory, $N$ is the total number particles, $q_t$ is the ground truth position of the robotic hand at time step $t$, $\hat{q}_t^i$ denotes the estimated position of the robotic hand of particle $i$ at time step $t$, and $w_t^i$ is the weight of particle $i$ at time step $t$. In our experiments, the $WMSE$ is calculated using the position of the robotic hand in meters. Therefore, it is the average estimation error of tracking the position trajectories of the robotic hand. For example, if we use meters as the unit for the position of the robotic hand, a 0.02 difference in the $WMSE$ indicates a 0.02-meters tracking accuracy difference between the filters.

To evaluate the performance of estimating contacts, we define the time-varying contact confidence variable $c(t) \in [0, 1]$. In the ground truth of our one-dimensional experiment, $c(t)$ remains zero before the robot makes contact with the wall and
immediately rises to one once the robot contacts the wall. Therefore, the ideal
response for a contact estimator in this one-dimensional problem is a step function,
which maintains zero before the contact and one after. Equation (4.13) describes
this step function,
\[ c(t) = \begin{cases} 
0, & \text{if } t < t_c \\
1, & \text{if } t \geq t_c 
\end{cases} \]  
(4.13)

where \( t_c \) is the time when the contact happens.

We define \( WCCE \) to show how well our filter tracks the step function shown
above over an entire experiment. We treat each particle in a particle filter as a
contact predictor, and we calculate the estimated contact confidence as the sum of
the weights of all particles that predicts contacts. Similar to the \( WMSE \), \( WCCE \) is
also the mean error over all time steps. The equation to calculate \( WCCE \) is shown
in equation (4.14), where \( \delta^{[i]} \) is a binary function with outputs 0 and 1 indicating
the contact prediction of the \( i \)th particle.

\[
WCCE = \sqrt{\frac{1}{T} \sum_{t=1}^{T} \left( \sum_{i=1}^{N} w_i^{[i]} (\delta^{[i]} - c(t))^2 \right)}
\]  
(4.14)

Since the maximum value of \((\delta^{[i]} - c(t))^2\) is one and the weights \( w_i^{[i]} \) are normalized
in a particle filter, the value of \( WCCE \) varies between zero and one in different
experiments. When a filter performs perfectly in an experiment, its \( WCCE \) will be
zero as the contact estimation of each particle will exactly follow the step function
in equation (4.13). The worst contact estimation performance will be the filter
predicting the exact opposite contact state at each time step, e.g., predicting contact
when the robot is away from the wall. When a filter has the worst possible contact
estimation performance, \( WCCE \) will be one since \((\delta^{[i]} - c(t))^2\) will always be one.

We tested all the modeling options using both simulation and physical ex-
periments. The position of the robot is estimated and measured in meters. We
conducted two simulation experiments, namely, one with the rigid body model and
the other with the compliant body model. In the simulation experiments, the robotic
hand is modeled as a rigid block moved by an external force toward a wall. The physical experiments were conducted with a Barrett WAM arm whose palm moved into contact with a stiff wall with guarded control (see Fig. 4.7). The force/torque sensor at the wrist of the robotic arm is used as the force sensor to measure the input forces. In the experiments, we measured both the positions of the robotic hand and the applied input force. For the simulation experiments, the ground truth positions were acquired directly from the simulation engine, and in the physical experiments, the ground truth positions were calculated by the forward kinematics of the robotic arm using its measured joint angles. In both the simulation and physical experiments, in order to simulate practical situations where only noisy measurements are available, we added white Gaussian noise to the ground truth position trajectories to obtain the noisy measurements. Finally, to produce a fair comparison, we used 500 particles and repeated the experiment ten times for each modeling option.

Figure 4.7: The setup for the physical experiments. The Barrett WAM moves into contact with a wall made of a wooden frame fixed on a table.

4.4.1 Results and Discussions

Based on the discussion in section 4.2, we have two options of the contact dynamic models for the state transition model in our filters: the rigid body model and the compliant body model. We studied the effects of the errors in the contact dynamic models on the performances of our filters. Specifically, we compare the
performances of our filter when its contact dynamic model agrees and disagrees with the contact dynamic model used in simulation experiments. Additionally, we studied the performances of different filtering methods. We conducted two sets of simulation experiments: 1) simulation experiments with a rigid body model 2) simulation experiments with a compliant body model. We also collected data for ten physical experiments. We then calculated $WMSE$ and $WCCE$ of all modeling options for both sets of the simulation experiments and the physical experiments.

Fig. 4.8 and Fig. 4.9 show the performances on tracking the position of the robotic hand for the simulation experiments, where the $x$-axis shows the ID of the experiments and the $y$-axis represents the $WMSE$. As discussed above, the better a filter performs, the lower its $WMSE$ will be. Fig. 4.8 shows the $WMSE$ calculated for ten rigid body simulation experiments. Similarly, Fig. 4.9 shows the $WMSE$ for ten simulation experiments that use compliant body dynamic models. In both figures, all filtering methods with the same contact dynamic model, e.g., all filters with the rigid body dynamic model, perform similarly, i.e., their $WMSE$ stay at similar levels on the $y$-axis. However, in rigid body simulation experiments, Fig. 4.8 shows that filters with the rigid body dynamic model in their state transition models perform better than those with the compliant body dynamic model. The plot shows that on average, in rigid body simulation experiments, filters that choose the rigid body model reduce the tracking errors by 0.04 meters comparing with filters with compliant body dynamic models. For the compliant body simulation experiments in Fig. 4.9, the results, on the other hand, reveal that using the compliant body dynamic model in filters yields better trajectory tracking accuracy/lower $WMSE$. In summary, the plots in both figures show that choosing the correct contact dynamics model, i.e., filters using the same dynamics model with that of the simulation experiments, yields better trajectory tracking accuracy.
Figure 4.8: The WMSE results generated by particle filters with state transition models using two contact dynamic models. The experiment data is generated by simulation experiments with the rigid body model. Data points represented as $\triangle$ correspond to filters with the rigid body dynamic model in their state transition models, and data points shown as $\ast$ correspond to filters with the compliant body dynamic model in their state transition models.
Figure 4.9: The WMSE results generated by particle filters with state transition models using two contact dynamic models. The experiment data is generated by simulation experiments with the compliant body dynamic model. Data points represented as $\triangle$ correspond to filters with the rigid body dynamic model in their state transition models, and data points shown as $\ast$ correspond to filters with the compliant body dynamic model in their state transition models.

Fig. 4.10 and Fig. 4.11 show the WCCE results for simulation experiments with the rigid body or compliant body dynamic models. Similar to the WMSE plots, we also tested all filters over ten experiments. In Fig. 4.10 and Fig. 4.11, the $x$-axis and the $y$-axis represent the experiment ID and the WCCE, respectively. Again, a better contact estimation means lower WCCE in the plots. Both figures show that regardless how the experiment data is generated, i.e., by simulation experiments using the rigid body dynamic model (Fig. 4.10) or the compliant body dynamic model (Fig. 4.11), particle filters with state transition models using the compliant body dynamic model (plots on the right side) always perform better.

In particle filters with state transition models using the rigid body dynamic model (plots on the left side), filters with noise added to the state variable, i.e.,
NSPF and PPF, perform worse than those with noise added to the input forces, i.e., NIPF and FSPF. Filters with no noise added to their state transition models, i.e., BPF, show the best contact estimation results, which shows that adding noise to the filter harms the contact estimation accuracy. For filters with noise added to the state variable (NSPF and PPF), the gap distance between the robot and the wall equals zero in particles that predict in-contact states, and noise added to the state variable can cause penetrations in these particles. In a rigid body dynamic model, penetrations generate significant separation forces that in our case, “shoot” the robotic hand away from the wall as shown in Fig. 4.12. As a result, the contact predictions of these particles switch from in-contact to no-contact within one time step, which reduces the overall contact confidence of the filter. For filters with noise added to the input forces (NFPF and FSPF), the gap distances are also zero in particles that predict in-contact states. Although the noise in the input forces does not generate penetrations, it causes vibrations in the input forces, which also creates vibrations in the position of the robotic hand in these particles. Consequently, particles that predict in-contact states can switch to no-contact predictions, which also decrease the contact confidence of the filter. Therefore, for particle filters using the rigid body dynamic models, adding noise to the state transition model will cause drops of the estimated contact confidence when the robotic hand is in contact with the wall, and thus the contact estimations of these filters are worse than BPF.

For particle filters with the compliant body dynamic model (plots on the right side), all filtering methods perform similarly, and their WCCE are on average lower than those of the particle filters with the rigid body dynamic model. With a compliant body dynamic model, the surface of the wall deforms when the robotic hand makes contact with it. Therefore, for NSPF and PPF, the noise in the state variable generates greater deformations instead of penetrations, and thus the contact forces will not bring about accelerations that separate the robotic hand and the wall (see Fig. 4.13). For NIPF and FSPF, the vibrations of the robotic hand caused by the noise in the input forces will be compensated by the deformations at the contacting surfaces. As a result, the compliance of the compliant body dynamic model compensates the noise in the state transition model, and the filters with the compliant
body dynamic model deliver better contact estimation results.

(a) WCCE generated by particle filters with state transition models using the rigid body dynamic model.

(b) WCCE generated by particle filters with state transition models using the compliant body dynamic model.

Figure 4.10: The WCCE results generated by particle filters with state transition models using two contact dynamic models. The experiment data is generated by simulation experiments with the rigid body dynamic model.
Figure 4.12: Screenshots of a rigid body simulation at three time steps, where the green block represents the robotic hand and the black block represents the wall. The experiment starts with a penetration between the robotic hand and the wall.

Figure 4.11: The WCCE results generated by particle filters with state transition models using two contact dynamic models. The experiment data is generated by simulation experiments using compliant body dynamic model.

Unlike the simulation experiments, where we know the exact dynamic model of the experiments, in the physical experiments, the parameters, such as the mass of the robotic hand, of a dynamic model can be inaccurate. Therefore, we have errors in the contact dynamic models used in our filter regardless of the choice of the dynamic model, i.e., rigid body dynamic model or compliant body dynamic model. As a result, Fig. 4.14 shows that particle filters with different contact
dynamics models perform similarly on estimating the positions of the robotic hand. In addition to inaccurate parameters in the dynamic models of our filters, the input forces measured by the force/torque sensor are noisy in physical experiments. Since the motion of the robotic hand is driven by the input forces, we find that without noise added to the state transition models of our filters, i.e., $BPF$, the noisy input forces diverge the estimated trajectories from the ground truth trajectories, which leads to higher trajectory tracking errors (greater $WMSE$). This is shown in Fig. 4.14, where the $WMSE$ of $BPF$ is higher than the other filtering methods. Fig. 4.14 also indicates that filters with noise added to the state transition model, i.e., $NSPF$, $NIPF$, $PPF$ and $FSPF$, improve the trajectory tracking accuracy (smaller $WMSE$).

Fig. 4.15 shows the results of the contact estimation for the physical experiments. Similar to the simulation experiments, filters using the compliant body dynamic model shows better contact estimation performances than those of the filters using the rigid body dynamic model. For filters with the rigid body dynamic model, $FSPF$ and $NIPF$ perform better than the other filtering methods. Different from the simulation experiments, $BPF$ performs the worst among all filtering methods as a result of its bad trajectory tracking accuracy.
Finally, we draw our conclusions of the evaluation as follows:

- If an accurate dynamic model is available, one will need to apply the accurate
model in their filters to achieve accurate tracking of the poses of the objects in a robotic manipulation task. Errors in the dynamic model used in the filters, such as using a compliant body dynamic model when the manipulation task is executed in a rigid body dynamic system, will lead to bad trajectory tracking results.

• When there are no accurate dynamic models, such as inaccurate parameters in the dynamic models, one need to add noise to the state transition model in order to achieve accurate trajectory tracking results.

• With an accurate dynamic model, adding noise to the state transition model of a filter will hurt its contact estimation accuracy. However, when no accurate dynamic models are available, one need to add noise to the state transition model to achieve accurate contact estimation results.

• Filters with the compliant body dynamic model in their state transition models have better contact estimation accuracy than those with the rigid body dynamic model.

• When the input forces are noisy, filters with noise added to their input forces, i.e., $NIPF$ and $FSPF$, show better performances than filters with noise added to their state variables, i.e., $NSPF$ and $PPF$, at contact estimation. However, their performances at trajectory tracking are similar. We do not find preferences to using $FSPF$ over using $NIPF$ or using $PPF$ over $NSPF$. 
CHAPTER 5
Contact-Based Rao-Blackwellized Particle Filters for State Estimation

In a robotic manipulation task, we are usually interested in two types of states: the continuous state, which includes the poses and velocities of the objects, and the discrete contact state that describes the contacts between the robot and the objects. Although sampling the continuous state space is very intuitive, the number of dimensions of the continuous state space is usually very high. Therefore, it requires the particle filter to have a huge number of particles to approximate the distribution of the continuous state. A large number of particles will increase the computational burden, and on the other hand, an insufficient number of particles will cause divergences between the estimated trajectory and the actual one.

From our evaluation in the previous chapter, an accurate dynamic model is essential for a particle filter to precisely estimate the states in a robotic manipulation task. As mentioned in Chapter 3, the Linear Complementarity Problem (LCP) model is widely recognized as an accurate model of multi-body dynamics and is used in many simulation engines, such as Open Dynamics Engine (ODE) [12]. Therefore, we integrate the LCP model into our particle filter-based state estimation algorithm. Besides the accuracy of the LCP model, as mentioned in Chapter 4, it is also a piecewise linear model with the “switching points” between the adjacent linear pieces being the contact states. Based on this property of the LCP model, we propose to apply the Rao-Blackwellized particle filter (RBPF), which is introduced in section 3.2.3.4, to fully exploit the piecewise linear property of the LCP model, and we term this proposed particle filter contact-based RBPF. Contact-based RBPF samples the discrete contact state space. For each particle in the contact-based RBPF, the LCP model is approximately converted to a single linear model based on its contact state estimation, and the continuous state is updated linearly through Kalman filters.

\footnote{Portions of this chapter previously appear as: S. Li et al., "State Estimation for Dynamic Systems with Intermittent Contact," 2015 IEEE Int. Conf. Robotics and Automation (ICRA), Seattle, WA, 2015, pp. 3709-3715.}
To sample the discrete contact state space, the contact-based RBPF generates the
distribution of the contact state space using both a contact graph and the collision
detection algorithm. Additionally, this particle filter has two advantages:

- Low dimensionality of the sampling space
  Contact-based RBPF samples particles on the discrete contact state space
  whose dimensionality is much lower than that of the continuous state space.
  Sampling low dimensional state space requires a smaller number of particles
  comparing with sampling a high dimensional state space. Therefore, our pro-
  posed particle filter is more efficient than the straightforward particle filter
  that draws samples directly from the state space.

- Accurate distribution of the continuous states
  In contact-based RBPF, the distributions of the continuous states are up-
  dated by Kalman filters. As a result, we approximate the distributions with
  Gaussian mixtures, which deliver better approximation than representing the
  distribution with discrete particles.

### 5.1 Contact-Based Rao-Blackwellized Particle Filter Design

As shown in Chapter 3, Dynamic Bayesian Network (DBN) is a generative
probabilistic model and it can be used to describe the structures of particle filters.
Therefore, in this section, we present the design of our proposed particle filter as a
DBN whose graphic model is shown in Fig. 5.1.
Figure 5.1: The graphic model of the Dynamic Bayesian network of our proposed particle filter.

In Fig. 5.1, for time step \( t \), \( c_t \) is the contact state, \( x_t \) is the continuous state, which includes the poses (\( q_t \)) and velocities (\( v_t \)) of the objects, \( z_t \) is the continuous state measurement, and \( u_t \) is the input to the system. The meanings of the edges of the model are shown below:

- \( 1 \): the linear state transition model for the continuous state
- \( 2 \): the measurement model
- \( 3 \): the dependency between the linear state transition model and the discrete contact state
- \( 4 \): the dependency between the contact state and the continuous state at the previous time step
- \( 5 \): the dependency between two contact states at two consecutive time steps

Among these edges, edges \( 1 \) and \( 3 \) jointly represent a continuous state space transition probability distribution \( p(x_t \mid x_{t-1}, c_t) \), edge \( 2 \) represents a measurement probability distribution \( p(z_t \mid x_t) \), and edges \( 4 \) and \( 5 \) together represent a discrete contact state space transition probability distribution \( p(c_t \mid c_{t-1}, x_{t-1}) \). Although this model appears similar to the switching linear dynamic system model in Fig. 3.9,
there is a critical difference that the discrete states (contact states) in our particle
filter also depends on the continuous states of the previous time step (edge (5)).

Based on Fig. 5.1, we derive the joint distribution of all the variables upto
time step \( t \) as follows:

\[
p(x_{0:t}, c_{0:t}, z_{0:t}, u_{0:t}) = p(x_0)p(c_0) \prod_{l=1}^{t} p(x_l | x_{l-1}, c_l, u_l) \\
\cdots \prod_{l=1}^{t} p(c_l | x_{l-1}, c_{l-1}) \prod_{l=0}^{t} p(z_l | x_l)
\]

(5.1)

In the contact-based RBPF, we are interested in \( p(x_t, c_t | z_{0:t}, u_{0:t}) \), which is the
distribution of the continuous and contact states given all the measurements upto
time step \( t \). Therefore, we further derived the conditional distribution as follows
(please see the Appendix for detailed derivation):

\[
p(x_t, c_t | z_{0:t}, u_{0:t}) = p(x_t | c_t, z_{0:t}, u_{0:t})p(c_t | z_{0:t}, u_{0:t}) \\
\propto p(x_t | c_t, z_{0:t}, u_{0:t})p(z_t | c_t, z_{0:t-1}, u_{0:t}) \\
\sum_{c_{t-1}, x_{t-1}} \int p(c_t | c_{t-1}, x_{t-1})p(x_{t-1} | c_{t-1}, z_{0:t-1}, u_{0:t-1})dx_{t-1} \\
p(c_{t-1} | z_{0:t-1}, u_{0:t-1})
\]

(5.2)

In equation (5.2), \( p(x_t | c_t, z_{0:t}, u_{0:t}) \) represents the linear transition model of the
continuous state given a known contact state. Since each particle updates its con-
tinuous state through a linear model that is converted from an LCP model with
known a contact state, if for each particle, we assume the initial distribution of the
continuous state to be a Gaussian distribution, the distributions, \( p(x_t | c_t, y_{0:t}, u_{0:t}) \),
in the time steps thereafter are also Gaussian. Apart from \( p(x_t | c_t, z_{0:t}, u_{0:t}) \), the
rest of equation (5.2) represents the probability distribution from which we sample
the discrete contact state, and it has three components:

- \( p(c_{t-1} | z_{0:t-1}, u_{0:t-1}) \), which is the distribution of the discrete contact state
  upto time step \( t - 1 \).
\[ \int_{x_{t-1}}^{x_{t-1}} p(c_t \mid c_{t-1}, x_{t-1})p(x_{t-1} \mid c_{t-1}, z_{0:t-1}, u_{0:t-1})dx_{t-1}, \] which is an integral over \( x_{t-1} \).

- \( p(z_t \mid c_t, z_{0:t-1}, u_{0:t}) \), which is the measurement distribution.

While we can get the first component \( p(c_{t-1} \mid z_{0:t-1}, u_{0:t-1}) \) using the particles from time step \( t - 1 \), to compute the second component, we first choose \( p(c_t \mid c_{t-1}, x_{t-1}) \) as our proposal distribution for the discrete contact state. The dependency between \( c_t \) and \( c_{t-1} \) is solved through a contact graph. The dependency between \( c_t \) and \( x_{t-1} \) is solved separately within each particle in the filter. Since the continuous state distribution of a particle at time step \( t - 1 \), \( p(x_{t-1} \mid c_{t-1}, z_{0:t-1}, u_{0:t-1}) \) is a Gaussian distribution, we compute its sigma points [45], and the integral over \( x_{t-1} \) in equation (5.2) can be approximated as the summation of all sigma points as below:

\[
\int_{x_{t-1}} p(c_t \mid c_{t-1}, x_{t-1})p(x_{t-1} \mid c_{t-1}, z_{0:t-1}, u_{0:t-1})dx_{t-1} \\
\approx \sum_{i=1}^{N} p(c_t \mid c_{t-1}, x_{t-1}^{[i]})w^{[i]}, \quad (5.3)
\]

where \( N \) is the number of sigma points, \( x_{t-1}^{[i]} \) is the \( i \)th sigma point and \( w^{[i]} \) is the associated weight. Given a contact graph and \( c_{t-1}, c_t \) can only be chosen from the neighbors of \( c_{t-1} \) in the contact graph. Therefore equation 5.3 can be further derived as:

\[
p(c_t \mid c_{t-1}, x_{t-1}^{[i]}) = \sum_{c_t \in \Omega(c_{t-1})} p(c_t \mid x_{t-1}^{[i]}), \quad (5.4)
\]

where \( \Omega(c_{t-1}) \) is the union of all the neighbors of \( c_{t-1} \) in the contact graph. \( p(c_t \mid x_{t-1}^{[i]}) \) can then be calculated through collision detection algorithms, which we will discuss in the next section.

Finally, the third component \( p(z_t \mid c_t, z_{0:t-1}, u_{0:t}) \) represents the measurement distribution for our particle filter. However, the connection between the continuous state measurement \( z_t \) and the discrete contact state \( c_t \) is indirect. In order to evaluate \( p(z_t \mid c_t, z_{0:t-1}, u_{0:t}) \), we rewrite it by marginalizing over \( x_t \) as
\[ \int p(z_t, x_t \mid c_t, z_{0:t-1}, u_{0:t}) dx_t. \] The marginalization can be further written as \[ \int p(z_t \mid x_t)p(x_t \mid c_t, z_{0:t-1}, u_{0:t}) dx_t. \] \( p(z_t \mid x_t) \) can be evaluated using the continuous state measurement model (usually Gaussian model), and \( p(x_t \mid c_t, z_{0:t-1}, u_{0:t}) \) is the linear transition of the continuous state without considering the measurement at time step \( t \). To evaluate this marginalization, similarly to our approach of evaluating \( p(c_t \mid c_{t-1}, x_{t-1}) \), we first calculate the distribution \( p(x_t \mid c_t, z_{0:t-1}, u_{0:t}) \), and then use the summation over its sigma points to approximate the integral as follows:

\[ \int p(z_t \mid x_t)p(x_t \mid c_t, z_{0:t-1}, u_{0:t}) dx_t \approx \sum_{i=1}^{N} p(z_t \mid x_t^{[i]}) w^{[i]}, \quad (5.5) \]

where \( N \) is the number of the sigma points, \( x_t^{[i]} \) is the value of the \( i \)th sigma point, and \( w^{[i]} \) is the weight of the \( i \)th sigma point.

### 5.2 Contact-Based Rao-Blackwellized Particle Filter Implementation

Based on the design of our contact-based RBPF discussed in the previous section, we introduce the implementation of this particle filter and summarize its pseudocode in this section. The two key pieces of our implementation are the contact state sampling on the stick-slip contact graph and the conversion of the LCP model, which will also be discussed.

#### 5.2.1 Stick-Slip Contact Graph and Contact State Sampling

As introduced in Chapter 3, a contact graph describes the transitions among the contact states in a robotic manipulation task. Each node in a contact graph represents a contact state, which includes one set of contacts.

For a conventional contact graph, a contact state is considered to be pure kinematic, which only considers the geometry information when a contact state is computed. However, as will be shown in section 5.2.2, to convert an LCP model to a linear model, one needs to consider the stick-slip status of a contact as well. Therefore, based on the conventional contact graph, we proposed to add the stick-
slip status to the contact states and call this contact graph the stick-slip contact graph. In a stick-slip contact graph, we preserve the kinematic transitions among the contact states, and each node in the graph is occupied by contact states sharing the same kinematic constraints but with different stick-slip status. Following the notation of the conventional contact graph, we denote the set of kinematic constraints at one node in the stick-slip contact graph as a contact formation. One example of a partial stick-slip contact graph of a wall and a triangular object is shown in Fig. 5.2. In Fig. 5.2, the number on the top left corner of a node represents the index of that node, and the number on the top right corner shows the number of contact states sharing that contact formation. For example, the top node has two contact states: either a sticking or a sliding contact between the top vertex and the wall.

![Figure 5.2: A partial stick-slip contact graph of a triangle and a wall.](image)

Each rectangle represents a node in the graph. The small picture in each node depicts the contact formation of that node. The top left corner of a node is the index of the node and the top right corner represents the number of contact states sharing that contact formation.

At time step $t$, we sample the contact state $c_t$ through distribution $p(c_t | c_{t-1}, x_{t-1})$ as described in equation (5.4). Based on the stick-slip contact graph,
we compute $\Omega(c_{t-1})$, which is the set of all neighbor nodes of the previous contact state $c_{t-1}$, and to sample $c_t$ from $\Omega(c_{t-1})$, we assign probability values to each of the neighbor nodes in $\Omega(c_{t-1})$ by combining a collision detection algorithm with the previous time step continuous state distribution. As discussed in equation (5.3), we first calculate the sigma points of $p(x_{t-1} \mid c_{t-1}, z_{0:t-1}, u_{t-1})$, which is the distribution of the continuous state at time step $t - 1$. With each sigma point, we apply a collision detection algorithm, which calculates the gap for each potential contact. A potential contact indicates a contact that is picked up by the collision detection algorithm but is not necessarily a valid contact. Over all the sigma points, we then enumerate the gap distances for all distinct potential contacts. If a potential contact is discovered by multiple sigma points, its gap distance is computed as the weighted mean of those gap distances. For example, if sigma point 1 with weight $w_1$ contains a potential contact with gap $\Psi_1$ and sigma point 2 with weight $w_2$ also discovers this potential contact with gap $\Psi_2$, then the overall gap distance of this potential contact is calculated as $\Psi = \frac{w_1 \Psi_1 + w_2 \Psi_2}{w_1 + w_2}$. We define $\Psi_v$ as the distances vector that contains the gap distances of all distinct potential contacts over all the sigma points.

Fig. 5.3 shows an example calculating $\Psi_v$ for a planar case, where the dynamic system consists of a square and a wall. Algorithm 9 summarizes the process of calculating $\Psi_v$, where $x_{t-1}$ and $\Sigma_{t-1}$ are the mean vector and the covariance matrix of the distribution of $x_{t-1}$, $\Psi$ is the vector of gap distances of one sigma point, $N$ is the number of sigma points, $w[i]$ and $x_{t-1}[i]$ are the weight and the value of the $i$th sigma points. Function Cal_Sigma_Points calculates the sigma points of a Gaussian distribution given its mean and covariance matrix, and function Update_Gap_Vector updates the gap distance vector $\Psi_v$ by computing the weighted mean distance of each distinct potential contact.

Similar to [76], we generate the probability of each potential contact being in contact through an inverse sigmoid function $S(x) = \frac{1}{1 + e^{-a}}$ of each element of the distance vector ($\Psi_v[i]$). In this inverse sigmoid function, $x$ is the variable and $a$ is a constant. As illustrated in Fig. 5.4, the value of the sigmoid function rapidly approaches one as the gap distance moves towards the region $(-\infty, \delta]$, where $\delta$ is a small positive value and can be approximated as $\frac{10}{a}$. For a distance vector $\Psi_v$, we
Figure 5.3: An example of calculating $\Psi_v$ for a planar dynamic system, which consists of a square and a wall. The superscripts represent the indices of the sigma points. There are two distinct potential contacts: $c_1$ and $c_2$. $\Psi_v$ is calculated as the weighted mean over all the sigma points.

$$\Psi = c_1: (d_1^2 w^2 + d_1^4 w^4 + d_1^5 w^5 + d_1^6 w^6)/(w^2 + w^4 + w^5 + w^6)$$

$$c_2: (d_2^1 w^1 + d_2^2 w^2 + d_2^3 w^3 + d_2^5 w^5 + d_2^7 w^7)/(w^1 + w^2 + w^3 + w^5 + w^7)$$

generate a vector of probabilities $v_p = (p_1, p_2 \ldots p_n)$, where $n$ is the number of the unique potential contacts in $\Psi_v$. Each $p_i$ in $v_p$ corresponds to the contact probability of the $i$th potential contact, which is calculated as $S(\Psi_v^i)$. Please see Fig. 5.5 for an example, where there are two distinct potential contacts between the robot and the triangular object. For each contact formation in the set of neighbor nodes $\Omega(c_{t-1})$, we also generate a probability vector $v_{cf}$. First, the length of $v_{cf}$ is set to be the same with that of $v_p$, which equals to the number of distinct potential contacts. Then, to generate $v_{cf}$ for a contact state, we go through all the potential contacts in $\Psi_v$, and if a potential contact also exists the contact formation, we set the corresponding element in $v_{cf}$ to one otherwise zero. For example, for a contact formation that only contains the first potential contact of $\Psi_v$, the probability vector will be computed as $v_{cf} = (1, 0 \ldots)$. In summary, we have $v_p$ to map all the potential contacts to a vector of possibilities, and correspondingly, we also compute $v_{cf}$ to convert a contact
Algorithm 9 Calculate the Weighted Gap Vector

function CAL_GAP($x_{t-1}$, $\Sigma_{t-1}$)
    for $i = 1 \rightarrow N$
        $x_{[i]}^{t-1}$, $w^{[i]} = \text{CAL_SIGMA_POINTS}(x_{t-1}, \Sigma_{t-1})$
        $\Psi = \text{COLLISION_DETECTION}(x_{[i]}^{t-1})$
        $\Psi_v = \text{UPDATE_GAP_VECTOR}(\Psi_v, w^{[i]} \Psi)$
    end for
    return $\Psi_v$
end function

Figure 5.4: An example of the sigmoid function ($S$) of the gap distance $\Psi$. The value of the sigmoid function rapidly approaches one as the gap distance moves towards the region $(-\infty, \delta]$, where $\delta$ is a small positive value and can be approximated as $\frac{10}{a}$.

formation to a vector of possibilities with the same length. We term $v_p$ and $v_{cf}$ as contact probability vectors as they both map a set of contacts to probability vectors.

As discussed above, we calculate two types of contact probability vectors: $v_p$ and $v_{cf}$. We then calculate the distances between these two vectors as $d = \|v_p - v_{cf}\|$. $d$ is then passed through another inverse sigmoid function to calculate the probability density value of a contact formation as $p_{cf} = S(d)$. For all the contact states in $\Omega(c_{t-1})$, we assign the same probability density values to contact states with the same contact formation (same node in Fig. 5.2).

Notice that, the probability values calculated above only consider the contact
formation (kinematic constraints). In order to sample the contact graph, the stick-slip status also needs to be addressed. However, for a contact formation with \( N \) contacts, there are \( O(2^N) \) contact states sharing identical contact formations in the contact graph, indicating \( O(2^N) \) stick-slip status. This can potentially be a problem when the number of contact pairs (\( N \)) becomes large. To solve this problem, we apply an inverse sigmoid function to the minimum generalized speed in the contact tangential direction to approximate the probability of the contact to be sticking or sliding, which as shown in equation (5.6):

\[
v_f = G_f^T v_{t-1}, \quad p_{si} = S(max\|v_f\|) \tag{5.6}
\]

where \( p_{si} \) corresponds the probability of the \( i \)th contact is sticking, \( G_f \) is the frictional Jacobian matrix defined in equation (3.33), \( v_f \) corresponds to the frictional velocity vector at the contact and \( \|\cdot\| \) computes the norm of a vector. Therefore, as the norm of the frictional velocity vector moving closer to zero, the probability of the contact being sticking contact approaches one. We apply this to each contact in a contact state and compute the probability density value of that contact state’s stick-slip status. For example, for a contact formation with \( N \) contacts, the probability of only the first contact is sliding is \( p_s = (1 - p_{s1}) \prod_{i=2}^{N} p_{si} \). Then we will be able
to limit the number of contact states with the same contact formation to a fixed number for each time step, e.g. always keep the top $m$ highest $p_s$ contact states for each contact formation.

Finally, the overall probability of a contact state (with stick-slip status) is $p_c = p_s p_{cf}$. After normalization, the probability distribution over all contact states in $\Omega(c_{t-1})$ is generated and we sample the contact state $c_t$ from this distribution. Algorithm 10 summarizes our contact states sampling method, where $n_{cf}$ is the number of next possible contact formations, $n_{cf}$ is the number of formations in $\Omega(c_{t-1})$, $CF_i$ is the $i$th contact formation in $\Omega(c_{t-1})$, $n_\Psi$ is the number of elements in the distances vector $\Psi_v$, $\Psi_v[k]$ represents the $k$th element of $\Psi_v$, $p_c$ is a two-dimensional array containing the overall probability of all contact states with the number of rows equal to the number of contact formations in $\Omega(c_{t-1})$ and the number of columns equal to $m$, function Get_Neighbor_Nodes looks up the set of neighbor nodes $\Omega(c_{t-1})$ from the contact graph, function Cal_CF_Prob calculates the contact probability vector $v_{cf}$ for a known contact formation in $\Omega(c_{t-1})$, function Random_Sample samples the contact state space according to the distribution $p_c$.

Algorithm 10 Sample Contact State

```
function SAMPLE_CONTACT_STATE(x_{t-1}, \Sigma_{t-1}, c_{t-1})
    \Psi_v = CAL_GAP(x_{t-1}, \Sigma_{t-1})
    \Omega(c_{t-1}) = GET_NEIGHBOR_NODES(c_{t-1})
    for k = 1 to n_\Psi do
        v_p[k] = S(\Psi_v[k])
    end for
    for i = 1 to n_{cf} do
        v_{cf} = CAL_CF_PROB(CF_i, \Psi_v)
        Calculate the top m highest $p_s$ contact states $p_{cf} = S(\|v_p - v_{cf}\|)$
        for j = 1 to m do
            p_c[i,j] = p_{cf}p_s[j]
        end for
    end for
    return Contact_State = Random_Sample(p_c)
end function
```
5.2.2 Converting the LCP Model with Known Contact States

As discussed at the beginning of the previous chapter, the LCP model is piecewise linear with the contact states being the “switching” points. Therefore, with a known contact state of a multi-body dynamic system, the LCP model shown in equation (3.49) can be converted to a single linear model. Suppose there are \( n_c \) contacts and \( n_f \) directions in each friction cone, then \((p_n)_{t+1}, (\rho_n)_{t+1}, \sigma_{t+1}\) and \(s_{t+1}\) are \( n_c \times 1 \) vectors, and \((p_f)_{t+1}\) and \((\rho_f)_{t+1}\) are \( n_c n_f \times 1 \) vectors. We term the left side of equation (3.51) gap vector and the right side impulse vector in this paper. The perpendicular condition in equation (3.51) implies that for each row, either the element in the gap vector or the one in the impulse vector is zero. Therefore, for each row of the two vectors, one needs to choose one element from the two elements (one element from each vector) to be zero. These choices are determined by the contact state: 1) contact status (in contact/not in contact) of all contacts, 2) the stick-slip status of each contact, 3) the sliding directions of the sliding contacts. As a result, knowing the contact state totally eliminates the complementarity condition in equation (3.51). Finally, we eliminate the bilateral constraints in the process of converting the LCP model and implement it as inputs, which is discussed in section 5.2.5.

Specifically, for a known contact state, with some algebraic manipulations, the last three rows of equation (3.49) can be derived as follows (see the Appendix for more details):

\[
\begin{bmatrix}
0 \\
0 \\
0
\end{bmatrix}
= 
\begin{bmatrix}
(G_n^T)_{\alpha} & 0 & 0 & 0 \\
(G_f^T)_{\alpha} & 0 & 0 & E_{\alpha\alpha} \\
0 & U_{\alpha\alpha} & -(E^T)_{\alpha\alpha} & 0
\end{bmatrix}
\begin{bmatrix}
u_{t+1} \\
((p_{n})_{t+1})_{\alpha} \\
((p_{f})_{t+1})_{\alpha} \\
((\sigma_{t+1})_{\alpha})
\end{bmatrix}
+ 
\begin{bmatrix}
(\frac{y_{n}}{m})_{i_1} \\
0 \\
0
\end{bmatrix}
\]  

(5.7)

In the equations above, \( \alpha \) is the set of indices of rows whose elements are zeros on the left side of equation 3.49. \( A_{\beta\theta} \) is a submatrix of \( A \), whose elements are \((a_{ij})_{i\in\beta, j\in\theta}\).
A_{\theta} is a submatrix of A, whose elements are \((a_{ij})_{\forall i, j \in \theta}\). Further, we would like to convert the LCP model to a linear system in the form of \(v_{t+1} = Av_t + Bu_t + 1\), where \(v_t, u_t\) are the velocity vector and the input vector of the dynamic system at time step \(t\). By combining equation (5.7) with the Newton-Euler equations, we derive \(A, B\) and \(u_t\) of our desired linear system as follows (see the Appendix for more details):

\[
A = M^{-1}(M - HK^{-1}H^T) \tag{5.8}
\]

\[
B = M^{-1} \begin{bmatrix} I - HK^{-1}F & -H \end{bmatrix} \tag{5.9}
\]

\[
H = \begin{bmatrix} (G_n)_{\alpha} & (G_f)_{\alpha} & 0 \end{bmatrix} \tag{5.10}
\]

\[
F = \begin{bmatrix} (\hat{G}_n)_{\alpha} \\ (\hat{G}_f)_{\alpha} \\ 0 \end{bmatrix} \tag{5.11}
\]

\[
K = \begin{bmatrix} (\hat{G}_n G_n)_{\alpha \alpha} & (\hat{G}_n G_f)_{\alpha \alpha} & 0 \\ (\hat{G}_f G_n)_{\alpha \alpha} & (\hat{G}_f G_f)_{\alpha \alpha} & E_{\alpha \alpha} \\ U_{\alpha \alpha} & -(E^T)_{\alpha \alpha} & 0 \end{bmatrix} \tag{5.12}
\]

\[
u = \begin{bmatrix} p_{\text{app}} & \left( \frac{\text{pc}}{h} \right)_{\alpha} & 0 \end{bmatrix}^T \tag{5.13}
\]

\[
\hat{G}_n = G_n^T M^{-1} \tag{5.14}
\]

\[
\hat{G}_f = G_f^T M^{-1} \tag{5.15}
\]

### 5.2.3 Kalman Filter-Based Continuous State Update

For each time step, our particle filter first samples the contact state \(c_t\) following the steps in section 5.2.1. The contact state is then used to convert the LCP model to a linear system as discussed in section 5.2.2. Finally, with the linear state transition model, a Kalman filter update step is directly applied. Notice that the linear state transition model we derived in section 5.2.2 is in the form of \(v_{t+1} = Av_t + Bu\), where \(v_t\) is the velocity vector of the objects. However, it is usually much easier to measure the objects’ poses \((q_t)\) rather than measuring their velocities \((v_t)\) in a robotic manipulation task. Therefore, using the equation of the kinematic update law \(q_{t+1} = q_t + v_{t+1}h\), we further modify the linear state transition model and rewrite it in the form of a linear Gaussian system, which is required by Kalman filters, as
follows:

\[
\begin{bmatrix}
q_{t+1} \\
v_{t+1}
\end{bmatrix} = \tilde{A} \begin{bmatrix}
q_t \\
v_t
\end{bmatrix} + \tilde{B}u + Q
\]

(5.16)

\[
\tilde{A} = \begin{bmatrix}
I & VAh \\
0 & A
\end{bmatrix}
\]

(5.17)

\[
\tilde{B} = \begin{bmatrix}
V Bh \\
B
\end{bmatrix}
\]

(5.18)

\[
z_{t+1} = \begin{bmatrix}
I & 0
\end{bmatrix} \begin{bmatrix}
q_{t+1} \\
v_{t+1}
\end{bmatrix} + R
\]

(5.19)

where $Q$ and $R$ are the noise matrices as defined in equation (3.69, 3.71) and $A$ and $B$ are defined in equations (5.8) and (5.9). Please refer to section 3.2.2 for details about the Kalman filter update step. One should notice that by updating the continuous state through Kalman filters, the errors introduced by the contact state sampling process will be reduced by the innovation term of the Kalman filters.

### 5.2.4 Algorithm Summary

With the two key pieces, i.e., contact state sampling and the conversion of the LCP model, introduced above, we summarize our contact-based RBPF by showing its pseudocode in Algorithm 11. In this pseudocode, $x_t$, $\Sigma_t$, $c_t$, $w_t$ represent sets of variables for all particles, and for each set, the superscript $i$ indicates that the variable is for the $i$th particle in the set. For example, $\Sigma_{t-1}$ represents the set of the covariance matrices of the object pose for all particles at time step $t - 1$, and $\Sigma_t$ represents the covariance matrix for the $i$th particle. The resample condition adopted here is that the effective number of particles $N_{eff} = \frac{1}{\sum_{i=1}^{N} (w_i)^2}$ is less than the threshold $N_{thr} = 0.5N$.

Finally, one should notice that with our contact-based RBPF, the distribution of the continuous state $x_t$ is a Gaussian distribution in each particle, and the distribution of the continuous state over all particles is a Gaussian mixture.
Algorithm 11 Contact-Based RBPF

function Filter($x_{t-1}$, $\Sigma_{t-1}$, $c_{t-1}$, $u_t$, $z_t$, $w_{t-1}$)
    for $i = 1 \rightarrow N$
        $\bar{c}^i_t = \text{Sample Contact State}(x^i_{t-1}, \Sigma^i_{t-1}, c^i_{t-1})$
        $\bar{x}_t^i, \bar{\Sigma}_t^i = \text{Kalman Update}(x^i_{t-1}, c^i_t, u_t, z_t)$
        $\bar{w}_t^i = \text{Weight Update}(\bar{x}_t^i, z_t, w^i_{t-1})$
    end for
    if resample condition satisfied then
        for $i = 1 \rightarrow N$
            Draw $j$ with probability $\bar{w}_t^j$
            $x_t^i = \bar{x}_t^j, c_t^i = \bar{c}_t^j, w_t^i = \frac{1}{N}$
        end for
    else
        $x_t = \bar{x}_t, c_t = \bar{c}_t, w_t = \bar{w}_t$
    end if
    return $x_t, \Sigma_t, c_t, w_t$
end function

5.2.5 Additional Problems for Three-Dimensional Manipulation Tasks

To apply our contact-based RBPF to a three-dimensional robotic manipulation task, a few additional problems need to be addressed. The first problem comes from the contact graph construction. As the kinematic constraints become very complicated in the three-dimensional space, there does not exist an efficient algorithm to calculate the contact graph analytically\(^3\). Additionally, as the robot manipulates the object, the joint angles of the robotic manipulator will be constantly changing, which leads to constantly changing geometries of the manipulator. As a result, since a contact graph is constructed based on kinematic constraints, for each robotic manipulator configuration (each set of joint angles), a contact graph will need to be calculated. Combining the challenges mentioned above, constructing the contact graph for a three-dimensional robotic manipulation task is a very challenging problem. The second problem is related to converting an LCP model to a linear model. In a three-dimensional problem, the contact constraints can become quite complicated. Especially, when there are contacts whose contact normal are close to parallel, the

\(^3\)Although there are algorithms that can be used to compute contact graphs for three-dimensional problems, there have not been works reported to demonstrate their efficiency or feasibility for complicated three-dimensional object.
contact constraints Jacobian matrices $G_f$ and $G_n$ in section 3.1.2 can become ill-conditioned, which leads to a singular $K$ matrix in equation (5.12). Since the inverse of the $K$ matrix is used in calculating matrices $A$ and $B$ in the process of converting the LCP model to linear models, a singular $K$ matrix can lead to bad estimation results. The third problem is caused by the bilateral constraints on the joints of a robot. Since our filter estimates the poses of objects, errors in the estimation of the poses of the robotic links can lead to violations of the kinematic constraints at the joints, e.g., penetrations between adjacent links causing unstable estimation results. In order to tackle these problems, we propose our solutions in the rest of this section.

5.2.5.1 Sampling-Based Method of Constructing Contact Graph

Researchers studying the Partially Observable Markov Decision Process (POMDP) face a similar problem, where the size of the belief space grows exponentially with the dimensionality of the state space[?]. Point-based POMDP algorithms tackle this problem by sampling the belief space and constructing an approximated belief space [31, 29]. In these algorithms, the belief space is approximated by “reachable space” samples collected during an exploration. Inspired by the point-based POMDP algorithms, we propose to alleviate the three-dimensional contact graph construction problem by using a sampling-based method.

In this sampling-based method, we assume that the robot is going to execute a manipulation task wherein the robot will follow a known control policy. We then set up the robotic manipulation task with both the robot manipulator and the object of interest in a simulation $^4$. The robot executes the manipulation task with the same control policy for $N$ times. Each time before the robot executes the task, the initial pose of the object is chosen randomly by sampling the feasible configuration space with a Gaussian distribution. For example, for a simple manipulation task where a Barrett hand [1] tries to grasp a rectangular cube, Fig. 5.7 shows the screen shots (the initial configuration and the final configuration) of two simulation experiments. At each time step, the contact state is computed since the information of contacts

$^4$We used Bullet as our simulation engine.
can be extracted from the simulator. Therefore, the contact state space for a robotic manipulation task is approximated by contact states collected over the $N$ times of the simulated experiments. Besides collecting the contact states, between every two adjacent time steps, we also keep track of the transitions between the two contact states. In summary, over the $N$ times of the experiments, we collect both the contact states and the transitions between them, and namely, we approximately build the contact graph for the manipulation task.

This sampling-based contact graph represents an approximated yet more efficient contact graph comparing with the one computed using analytical algorithms\textsuperscript{5}. The accuracy of our approximated contact graph can be increased by increasing the number of simulated experiments $N$. Also, one should notice that the approximated contact graph is more efficient, as the contact states collected in our approximated contact graph are discovered during the manipulation task thus the redundant con-

\textsuperscript{5}There has not been an analytical algorithm that solves the contact graph construction for three-dimensional problems.
tact states (contact states that the robot will not likely encounter during the task) will not appear in our contact graph. In other words, as shown in Fig. 5.6, our approximated contact graph represents a portion (the green part) of the whole contact graph, which includes contact states that the robot will most likely to encounter during a manipulation task.

![Figure 5.7: Screenshots of two simulation experiments. Each row shows the initial configuration (the picture on the left) and the final configuration (the picture on the right) of one experiment.](image)

5.2.5.2 Approximation of the Inversion of $K$ Matrix

As mentioned above, the $K$ matrix in equation (5.12) becomes singular when the contact Jacobian matrices are ill-conditioned. Since we used the inverse of the $K$ matrix in the process of converting the LCP model in section 5.2.2, a singular $K$ matrix can lead to inaccurate estimation results. Although one can eliminate singular values using singular value decomposition (SVD), discarding the singular
values based the SVD of the whole $K$ matrix imposes problems since the singularity is introduced only by the contact Jacobian matrices. Also, the $K$ matrix contains parameters related to the physical model, e.g., friction coefficients, and taking the inverse of $K$ will lead to the inexplicit expression of these parameters in $K^{-1}$. Therefore, an explicit form of $K^{-1}$ is needed so that the parameter estimation and observability problems can be studied.

Despite we would like to compute the exact explicit form of matrix $K^{-1}$, this inverse matrix is extremely hard to compute because of the unfriendly structure of the matrix $K$. Therefore, we focus on computing an approximated solution. In the LCP model conversion process, the *impulse vector* mentioned in section 5.2.2 can be expressed as follows (please refer to the Appendix for more details):

$$
\begin{pmatrix}
(p_n)_{t+1}_\alpha \\
(p_f)_{t+1}_\alpha \\
(\sigma_{t+1})_\alpha
\end{pmatrix} = -K^{-1}(Fv_t + Jp_{app} + \begin{pmatrix}
\frac{\Psi_n}{h}
0
0
\end{pmatrix}) \quad (5.20)
$$

where $\alpha$ has the same definition with that in equation (5.7). By further inspecting the structure of the equation, we find that the rows at the right side of the $K^{-1}$ matrix will not contribute to the computation of the *impulse vector* and can be ignored. Also, from the LCP conversion process, $\sigma_{t+1}$ does not contribute to the calculation of the continuous state for time step $t+1$, and therefore the bottom rows of the $K^{-1}$ matrix can also be ignored. We directly compute the approximated $K^{-1}$ matrix by considering the Newton’s second law at the contacts (please refer to the Appendix for derivation details). The approximated $K^{-1}$ matrix is shown below:

$$
M_n = \left(G_n^T M^{-1} G_n\right)^{-1} \quad (5.21)
$$

$$
M_f = \left(G_f^T M^{-1} G_f\right)^{-1} \quad (5.22)
$$

$$
K^{-1} = \begin{bmatrix}
M_n & -M_n G_n^T M^{-1} G_f S_{sticking} M_f \\
-S_{sticking} M_f G_f^T M^{-1} G_n M_n + S_{sliding} M_n & S_{sticking} M_f \\
0 & 0 & 0
\end{bmatrix}
$$

where $S_{sticking}$ and $S_{sliding}$ are diagonal selection matrices. If there are $n_c$ contacts,
$S_{\text{sticking}}$ and $S_{\text{sliding}}$ are both $n_c \times n_c$ matrices, and each row of the two matrices corresponds to a contact. For $S_{\text{sticking}}$, the diagonal element at each row is one if the contact is sticking and zero if the contact is sliding. The diagonal elements of $S_{\text{sliding}}$ equal to the friction coefficients at the contact surfaces.

Finally, we expect this approximation to bring in errors in our converted linear model, and we rely on the errors to be eliminated by the Kalman filter update steps.

### 5.2.5.3 Manipulator’s Bilateral Constraints

Although we allow for uncertainties in the object states during the robot manipulation tasks, uncertainties in the relative poses of the adjacent links of a joint on the robot manipulator can cause violations of the kinematic constraints. Also, as can be seen in equation (3.49), the bilateral constraints are enforced by the joint constraints $\Phi_t$ over the time step size $h$. Since $h$ is usually a very small value, an error in the joint constraints, e.g., misaligned joints, can cause unstable estimation results. We notice that in most applications, a robot can acquire the joint angles from its encoder sensors accurately. As a result, the uncertainties in a robotic manipulation task mainly exist in the measurements of poses of the objects. Therefore, instead of estimating the states (poses and velocities) of the links and enforcing the bilateral constraints in our filter, we propose to use the joint angles and the robot’s kinematic constraints to compute their states and use these computed states as the inputs to our particle filter. With some algebraic manipulations, we can modify equations 5.16 and 5.19 as follows:

$$
\begin{pmatrix}
(q_{t+1})_{\gamma} \\
(v_{t+1})_{\gamma}
\end{pmatrix}
= \tilde{A}_{\gamma\gamma}
\begin{pmatrix}
(q_t)_{\gamma} \\
(v_t)_{\gamma}
\end{pmatrix}
+ \begin{pmatrix}
\tilde{A}_{\gamma\tau} & \tilde{B}_{\gamma} & u
\end{pmatrix}
\begin{pmatrix}
(q_t)_{\tau} \\
(v_t)_{\tau}
\end{pmatrix}
+ Q_{\gamma\gamma}
$$

(5.23)

$$
(y_{t+1})_{\gamma} = \begin{bmatrix}
I & 0
\end{bmatrix}
\begin{pmatrix}
(q_{t+1})_{\gamma} \\
(v_{t+1})_{\gamma}
\end{pmatrix}
+ R_{\gamma\gamma}
$$

(5.24)

where $\gamma$ and $\tau$ are the sets of indices of rows that calculate the states of the object and the links of the robot, respectively. The definition of $A_{\alpha\theta}$ remains the same as that in equation 5.10.
5.3 Experiments

We tested our particle filter in both planar and three-dimensional robotic manipulation experiments. In the planar experiments, a parallel jaw gripper grasps a triangular object on a supporting plane as shown in the abstract setup in Fig. 5.9. The object can translate along both x and y-axes and rotate around the z-axis. There are frictional forces between the object and the two gripper fingers as well as between the object and the supporting plane. Initially, the object is in a random pose without touching either of the gripper fingers. As forces are applied to the gripper fingers, the two fingers approach the object along the y-axes and grasp the object. The position and orientation of the object, as well as the position of the two fingers, are measured. Although the experiment is planar, we use the full 3D LCP model in our filter. Also, in this planar experiment, we can compute its contact graph analytically and test the performance of particle filter on both continuous state and discrete contact state estimations.

In the three-dimensional experiments, a robotic hand attempt to grasps on a rectangular cube. The pose of the robotic hand is fixed, and the cube is placed vertically in front of the hand on a plane. The initial position of the cube is Gaussian distributed with the mean position close to the palm of the robotic hand. Please see Fig. 5.8 for an overview of the experiment setup. The robotic hand then grasp the cube by closing its fingers. Since the position of the cube is chosen randomly, there are three outcomes of the grasp: 1) the hand successfully grasps the cube, 2) the hand knocks the cube over, 3) the hand closes its fingers without touching the cube. In the experiments, the pose of the cube is measured by vision sensors. The joint angles of the robotic hand are read from the encoders. As described in section 5.2.5.3, we treat the readings of the joint angles as the inputs to the system.

5.3.1 Physical Experiment Setup

We show the physical experiment setup for both the planar and the three-dimensional experiments in this section. For the planar experiment, to approximate a parallel jaw gripper, we use the Barrett hand on a Barrett WAM as one finger and a fixed frame as the other finger as shown in Fig. 5.10. Since we are only able
Figure 5.8: An overview of the three-dimensional experiment setup.

Figure 5.9: An overview of the planar experiment setup.
Figure 5.10: Physical experiment setup for a planar grasping experiment, where a robotic parallel jaw gripper tries to grasp a triangular object on a plane. The gripper is approximated by a fixed frame and Barrett hand.

to move one finger of this approximate gripper, we call the finger that moves the moving finger and the other finger as the fixed finger. During a grasping experiment, the moving finger moves towards the fixed finger to form a grasp. A table is used as the supporting plane. The pose of the triangular object is tracked by the Natural Point tracking system [11] above the experiment table. The position of the fixed frame is measured before the experiment. The force/torque sensor at the wrist of the robot arm provides the measurements of the forces applied to the gripper finger that is controlled by WAM. The trajectory obtained by the Natural Point tracking system [11] is very accurate, and therefore, is used as the ground truth. To simulate situations where only inaccurate visual data is available, the measurement data is generated by adding a white Gaussian noise to the ground truth.

For the three-dimensional experiment, we use the Barrett hand installed on the Barrett WAM in our lab as the robotic hand. A wooden rectangular cube is put on a table in front of the robotic arm. The pose of the hand is then fixed near the cube with the palm facing towards to the cube. A Microsoft Kinect sensor is installed above the scene of the manipulation task and observes the scene as point cloud data. A point cloud of the rectangular cube is created using the geometry model of the cube. The pose of the cube is then directly estimated from the point
Figure 5.11: The setup of a physical robotic manipulation experiment, where a Barrett hand tries to grasp a rectangular cube that is placed on the table. The Kinect sensor on the table captures the point cloud of the manipulation process.

cloud by the iterative closest point (ICP) algorithm [9], which iteratively register the point cloud of the cube to the point cloud of the scene observed by the Kinect sensor. Please see Fig. 5.12 for an example of the physical experiment setup and Fig. 5.11 for an example of the point cloud of the manipulation scene. Besides the Kinect sensor, we also track the pose of the object using the Natural Point tracking system [11], which is used as the ground truth.

5.3.2 Results and Discussions

We discuss our estimation results for both the planar and the three-dimensional experiments. In the planar experiments, we tested our proposed particle filter against a conventional particle filter, which samples the particles in the continuous state space (pose of the object) and updates the values of the particles through a full multi-body dynamic model. Because of the simplicity of the contact state transitions in this planar experiment, we also compare our contact state estimation with the ground truth contact state transitions, which are acquired by human annotations. In the three-dimensional experiments, we compare the results from our particle filter with the particle filter proposed in [76].
Figure 5.12: The point cloud of a scene from two viewing angles in a robotic manipulation task, where a Barrett hand tries to grasp a rectangular cube.

5.3.2.1 Planar Grasping Experiments

For the planar experiments, we compare our particle filter against a conventional particle filter. In this conventional particle filter, each particle represents a sample from the distribution of the continuous state space. A model of multi-body dynamics is used to update the value of a particle from one time step to the time step next. In this experiment, we use LCP as the dynamic model, and for each particle in this conventional particle, if we call the LCP model formulated at time step $t$ as $LCP_t$, the input to $LCP_t$ as $x_t$, and the solution to $LCP_t$ as $x_{t+1}$, then at time step $t + 1$, $LCP_{t+1}$ is solved by taking $x_{t+1}$ as its input. In other words, one can imagine this conventional particle filter as a collection of simulators with different initial states. We weight the simulators by comparing their outputs to the measurements, and during resampling, replace simulators with low weights by simulators with high weights. We name the conventional particle filter as the continuous particle filter (CPF) since it samples the continuous state space. Algorithm 12 summarizes the CPF, where function Solve_LCP takes the current continuous state as the input and solves for the continuous state for the next time step. The definitions of the other variables in Algorithm 12 stay the same with those in Algorithm 11.
(a) The object’s position on the x-axis.

(b) The object’s position on the y-axis.
Figure 5.11: An example of the estimated and ground truth trajectories for one planar grasping experiment, where the robotic gripper grasps the triangle object by moving one finger while keeping the other finger fixed.

Algorithm 12 Continuous State Particle Filter

```plaintext
function FILTER(x_{t-1}, u_t, z_t, w_{t-1})
    for i = 1 → N do
        \( \tilde{x}^{[i]}_t \) = SOLVE\_LCP(x_{t-1}^{[i]}, u_t)
        \( \tilde{w}^{[i]}_t \) = WEIGHT\_UPDATE(\( \tilde{x}^{[i]}_t \), z_t, w_{t-1}^{[i]})
    end for
    if resample condition satisfied then
        for i = 1 → N do
            Draw j with probability \( \propto \tilde{w}^{[j]}_t \)
            \( x^{[i]}_t = \tilde{x}^{[j]}_t \), \( w^{[i]}_t = \frac{1}{N} \)
        end for
    else
        \( x_t = \tilde{x}_t \), \( w_t = \tilde{w}_t \)
    end if
    return \( x_t, w_t \)
end function
```
In the experiments, we measure the pose of the object. For a planar experiment, there are three variables related to the pose of an object: the two translations along the two axes of the plane and the rotation around the normal direction of the plane. In this experiment, as shown in Fig. 5.9, we define the two axes on the plane as the $x$-axis and the $y$-axis, and the normal direction as the $z$-axis. Fig. 5.11 shows an example of estimated and ground truth trajectories of one planar grasping experiment, where the plots show the trajectories of the object’s positions on the $x$ and $y$-axes and rotations around the $z$-axis. In this example, as the moving finger pushes the object, it translates in the negative $y$ direction and barely changes its position along the $x$-axis. During the grasping process, if we label the vertices of the triangle as in Fig. 5.12, the moving gripper first touches vertex $v_1$ on the triangle causing a rotation in the positive $z$-direction and then contacts form between vertex $v_3$ and the fixed finger, which rotates the object in the negative $z$-direction. These motions are also reflected in the plots of Fig. 5.11.

For the example in Fig. 5.11, our contact-based RBPF (black line) performs better than the conventional particle filter/CPF (purple line). As discussed above, CPF directly samples the continuous state space and updates continuous state distribution through running multiple simulators. Therefore, its performance depends heavily on the accuracy of its simulators. Contact-based RBPF, on the other hand, updates the distribution of the continuous state through Kalman filters, which optimally maintain the balance between the errors in the models and the measurements through the innovation gain (see Chapter 3 for more details). Due to the limits of our measurement equipment and sensors, the parameters, i.e., the shape of the triangle, used in the models of the filters cannot be exact. As a result, CPF, whose precision depends on the accuracy of its dynamic model, performs poorly. Our contact-based RBPF overcomes the parameter errors by optimally updating its continuous state through Kalman filters. In Fig. 5.11, contact-based RBPF starts outperforming CPF right after the robot making contact with the triangle at time step 100. Due to the inaccurate friction coefficients in our dynamic model and the noise in the force/torque sensor, CPF predicts that the triangle rotates and translates much faster than the ground truth shortly after the robot moves into contact.
Figure 5.12: The labeling for the vertices of the triangle in our planar experiment.

with the triangle. Then, as the robot keeps pushing the triangle, CPF predicts the edge \{v_1, v_2\} to be completely in contact with the moving finger while in the actual experiment, only vertex \(v_1\) is in contact due to the friction between the triangle and the supporting plane. This is indicated as the offset between the CPF estimation and the ground truth between time step 100 and time step 300 in Fig. 5.12c. Contact-based RBPF, on the contrary, tracks the trajectory accurately.

In order to make quantitative comparisons between CPF and contact-based RBPF, we also calculate the weighted mean squared errors (WMSE) for their estimated trajectories. Equation (5.25) shows how WMSE is calculated, where \(T\) is the total number of time steps of an experiment, \(N\) is the total number of particles of the filter, \(w\) is the estimated weight of a particle, \(\hat{x}\) and \(x\) are the object’s estimated and ground truth position on the \(x\)-axis, \(\hat{y}\) and \(y\) are the object’s estimated and ground truth position on the \(y\)-axis, \(\hat{\theta}\) and \(\theta\) are the object’s estimated and ground truth rotation around the \(z\)-axis, the superscript and the subscript indicate the index of the particle and the time step that a variable corresponds to, respectively (e.g., \(w_i^t\) is the weight of the \(i\)th particle at time step \(t\)), and \(\beta\) is a scaling constant for the rotation variable so that the errors in different units can be added.

\[
WMSE = \sqrt{\frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{N} w_i^t \left( (\hat{x}_i^t - x_t)^2 + (\hat{y}_i^t - y_t)^2 + \beta (\hat{\theta}_i^t - \theta_t)^2 \right)}
\] (5.25)

WMSE reflects the overall accuracy of the filters by calculating the sum of the average errors of all estimated variables over all time steps. Lower WMSE means
a better tracking accuracy. Fig. 5.13 shows the WMSE calculated for both CPF and contact-based RBPF over ten experiments. To make a fair comparison, we set the number of particles to 60 for both CPF and contact-based RBPF. In Fig. 5.13, although the WMSE of each filter fluctuates over the ten experiments, the WMSE of CPF is on average twice as high as the WMSE of contact-based RBPF, and contact-based RBPF is always more accurate (has lower WMSE) than CPF in these experiments.

Besides better accuracy of tracking the continuous state, our contact-based RBPF also has the advantage of being able to estimate the contact state, which is vital for robotic manipulation tasks, e.g., a robot assembly task where the robot needs to know the contact state in order to manipulate an object precisely. In order to estimate the contact state using our particle filter, one can go through the particles and merge the weights if two particles predict the same contact state. The final contact state estimation will be the contact state with the highest weight. Fig. 5.14 shows the contact state estimation of our particle filter for one experiment. Using the labels from fig. 5.12, this grasping experiment approximately experiences three contact states sequentially: 1) no contacts, 2) one contact between the moving
finger and vertex $v_1$, 3) one contact between the moving finger and vertex $v_1$ and a second contact between the fixed finger and vertex $v_3$. As shown in fig. 5.14, our filter is able to track the contact states accurately with a few time steps lag during the contact state transitions.

**Figure 5.14: Contact states estimation from contact-based RBPF.**

Fig. 5.19 shows the WMSE of CPF and contact-based RBPF as their numbers of particles increase from 4 to 160, and a lower WMSE in this figure means better estimation accuracy. Since contact-based RBPF samples the contact state space whose dimensionality is much lower than the continuous state space, the number of particles needed to obtain an accurate estimation is also lower. As a result, although for a particle filter, increasing the number of particles will usually increase its accuracy, we find that the accuracy of contact-based RBPF does not increase significantly as the number of particles goes beyond 40. This is shown in Fig. 5.19, where the WMSE of contact-based RBPF decreases as the number of particles increases till 40 and maintains the same level afterward. The WMSE of CPF, however, decreases as the number of particles increases.
5.3.2.2 Three-dimensional Grasping Experiment

For the three-dimensional grasping experiments, we compare our particle filter with the particle filter proposed by [76]. Their particle filter samples the continuous state space. The difference between their particle filter and the conventional particle filter is that in each particle, a contact state is estimated based on the outcome of the collision detection, and the value of the continuous state is updated through a linear model, which is obtained by converting the model of multi-body dynamics (the LCP model in their case). Although their filter also converts the model to a linear model to update the continuous state, unlike our filter, ignoring the connections between the contact states in the adjacent time steps in their filter can lead to inaccurate contact state estimation and thus inaccurate linear model. Another difference between our filter and their filter is that their filter samples the continuous state space, which, as will be shown later, requires more particles than our filter to accurately estimate the states.

Figure 5.15: The WMSE of CPF and contact-based RBPF as their numbers of particles increase from 4 to 160 in the planar grasping experiments.
Figure 5.16: The setup of the three axes for our three-dimensional grasping experiments.

We measure the pose of the object in these grasping experiments. In a three-dimensional experiment, there are six variables related to the pose of an object: the three translations along the three axes and the rotations around the three axes. The three axes of the coordinate system in our grasping experiments are setup as shown in Fig. 5.16, where the $x$ and $y$-axes lie on the surface of the supporting table and the $z$-axis is perpendicular to the table.

Similar to the planar grasping experiments, we term the particle filter from [76] as CPF since their filter samples the continuous state space. Fig. 5.17 shows the trajectory estimations using both filters for a grasping experiment. In this experiment, the robotic hand attempts a grasp on the rectangular cube but instead of securely grasping it, the hand accidentally knocks the cube over, which results in a failed grasping experiment. Each plot in Fig. 5.17 shows the ground truth trajectory (green lines) and the estimation results of both filters (blue and magenta lines) for one of the six variables, e.g., the position of the cube along the $z$-axis. As indicated in the ground truth trajectory, the cube was knocked over at around time step 90, and as it fell over, the cube rotated 90 degrees (1.57 radians) around the $x$-axis and its translations along the $x$ and $y$-axes increased while the translation along the $z$-axis decreased.

Although in the CPF proposed in [76], their state transition model is converted
Figure 5.17: An example of the estimated and ground truth trajectories for one three-dimensional grasping experiment, where the robotic gripper knocked over the rectangular cube that it tried to grasp.
from a multi-body dynamic model, its performance still heavily depends on the accuracy of the parameters, e.g. friction coefficients, of the dynamic model. Similar to the planar grasping experiment, our proposed contact-based RBPF, on the other hand, again benefits from the balance between the modeling errors and the measurements using the Kalman filters. Therefore, due to the inaccurate measurements of the parameters of the dynamic model, CPF performs worse than the contact-based RBPF in this grasping experiment example shown in Fig. 5.17. Especially in this example, when the robotic hand touched the cube at time step 90, due to the inaccurate friction coefficients in its dynamic model, CPF predicts the cube to be sliding along the x-axis before falling onto the table. This is shown as the lag in the CPF estimated trajectories in Fig. 5.17b, 5.17c, and 5.17e. Our contact-based RBPF overcomes the modeling errors using its Kalman filters and predicts the cube to fall upon the contact with the robotic hand.

To compare the performances of contact-based RBPF and CPF quantitatively, we also calculate the weighted mean squared errors (WMSE) for estimated trajectories of both filters. Similar to equation (5.25), we calculate the WMSE as shown in equation (5.26), where $T$ is the total number of time steps of an experiment, $N$ is the total number of particles of the filter, $w$ is the estimated weight of a particle, $x$, $y$ and $z$ represent the actual translations along the $x$, $y$ and $z$-axes, $\theta_x$, $\theta_y$ and $\theta_z$ are the actual rotations around the $x$, $y$ and $z$-axes, $\hat{\cdot}$ represents the estimation of a variable, the superscript and the subscript indicate the ID of the particle and the time step that a variable corresponds to, respectively (e.g., $w_t[i]$ is the weight of the $i$th particle at time step $t$), and $\beta$ is a scaling constant for the rotation variable so that the errors in different units can be added.

$$WMSE = \sqrt{\frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{N} w_t[i] \left( (\hat{x}_t[i] - x_t)^2 + (\hat{y}_t[i] - y_t)^2 + (\hat{z}_t[i] - z_t)^2 \right.}$$

$$+ \beta(\hat{\theta}_t[i] - \theta_t)^2 + \beta(\hat{\theta}_t[i] - \theta_t)^2 + \beta(\hat{\theta}_t[i] - \theta_t)^2}$$

(5.26)

This WMSE evaluates the overall accuracy of a filter in a three-dimensional experiment by calculating the sum of the average errors of all six estimated variables.
Figure 5.18: The WMSE of both CPF and contact-based RBPF over 25 three-dimensional grasping experiments.

over all time steps. Similar to the WMSE calculated for the planar experiments, a lower WMSE also indicates a better accuracy of a filter. We compare the WMSEs of both filters in 25 grasping experiments. For each experiment, we run each filter to estimate the trajectory of the cube and calculate the WMSE using equation (5.26). To make a fair comparison, we set the number of particles of both filters to be 120. The WMSEs of both filters over the 25 experiments are shown in Fig. 5.18. In these 25 experiments, although the performance of the filters varies from one experiment to another due to the randomness of the particle filters, contact-based RBPF performs better than CPF with WMSEs on average one-quarter of the WMSEs of CPF. Additionally, since CPF samples the high dimensional continuous state space, its estimation accuracy is heavily limited by the number of particles and it can perform poorly when its samples do not cover the distribution of the state space. As a result, the WMSE of contact-based RBPF has a much lower variance than that of CPF. This indicates that the performance of contact-based RBPF is more stable than that of CPF.

Fig. 5.19 shows the WMSEs of both contact-based RBPF and CPF as their
numbers of particles increase from 3 to 150. Although the accuracy of a particle filter generally improves as the number of particles increases, the WMSE of contact-based RBPF reduces and converges to around 0.2 after its number of particles rises above 40. The WMSE of CPF, on the other hand, decreases on average as its number of particles increases. Also, we notice that the WMSE of CPF is higher than that of contact-based RBPF over all numbers of particles.

5.4 Conclusion

In this chapter, we developed a contact-based RBPF that exploits the piece-wise linear property of the LCP model, which is an accurate model of multi-body dynamics. In this particle filter, we propose to convert the LCP model to a linear model with estimated contact states. Then instead of sampling the high dimensional continuous state space, by applying RBPF to our state estimation problem, we sample the particles in the discrete contact state space and the continuous state is updated through Kalman filters. In order to sample the contact state more effec-
tively, we propose to sample the contact states through a stick-slip contact graph. Additionally, to apply our designed particle filter to three-dimensional robotic manipulation tasks, we propose to construct the stick-slip contact graph through a sampling-based method, analytically approximate the inversion of a matrix in the converted LCP model, and resolve the manipulator’s bilateral constraints by treating them as inputs.

We compared our contact-based RBPF with particle filters that sample the continuous state space in both planar and three-dimensional grasping experiments. The results show that contact-based RBPF is less prone to errors in the dynamic models, such as the friction coefficients. Also, since contact-based RBPF samples the low dimensional contact state space, with the same number of particles, it performs better and is more stable than particle filters that sample the continuous state space.
CHAPTER 6
Contact State-based Policy Learning for Robotic Manipulation Tasks

As mentioned in the previous chapters, the robot’s measurements about the states of the objects usually suffer from noise introduced by the vision sensors or algorithms. With our proposed contact-based RBPF in Chapter 5, a robot is able to estimate the accurate object poses and contact states from noisy measurements. In a robotic manipulation task, a robot then must be able to map the state estimations to optimal actions in order to finish the task. This mapping between measurements to actions is known as the policy for a robotic manipulation task. One of the effective approaches to learning policies is reinforcement learning [68].

Although reinforcement learning has found many successful applications in robotics [43, 32], directly applying it in the manipulation setting still poses considerable challenges. One major problem is that it is difficult to choose an optimal resolution to discretize the state space. Too high a resolution will lead to high data dimension that reduces algorithm efficiency, while too low a resolution may not provide enough information to learn a good policy. Approaches have been proposed to tackle problems with the continuous states, such as adaptive discretizations and approximation functions [30].

If the robot and the object consist of rigid bodies, we can treat executing a robotic manipulation task as a sequence of state changes of a rigid body dynamic system. In the rigid body dynamic system, the robot and the object constantly collide with each other, and the forces generated by these contacts constrain the motions of the object and transition the object from its initial state to the desired state. Having intermittent contacts among objects is one of the main features that distinguish manipulation from other tasks in robotics, such as SLAM. Therefore, it is crucial for the robot to take actions based on the system’s current contact state. However, in a robotic manipulation task, the contact state is usually hard to be measured directly and accurately.
Our proposed contact-based RBPF can provide estimations of the contact state, which can be used as feedback for the robot to choose optimal actions. Therefore, in this chapter, we propose a new approach to solving the contact state-based policy learning and execution problem for robotic manipulation tasks. In this approach, a model-based reinforcement learning algorithm is applied to learn a policy based on the contact states. This learned policy is then combined with the contact-based RBPF using a QMDP algorithm [17], which is an approximation algorithm for solving the Partially Observable Markov Decision Process (POMDP) problem, to execute a manipulation task. Additionally, as pointed out in section 5.2.5, one problem with contact-based RBPF is that the contact graph used in the filter cannot be computed efficiently for three-dimensional manipulation tasks. In our proposed solution to this problem in section 5.2.5, we constructed the contact graph through sampling the contact states with a fixed policy in multiple simulation experiments. However, the policy that a robot executes in the simulation experiments was not well defined. In our proposed approach in this chapter, a robot simultaneously learns the policy and the contact graph through the reinforcement learning process.

6.1 Policy Learning for Robotic Manipulation Tasks

In this section, we introduce our approach to learning contact state-based policies for robotic manipulation tasks. For a contact state-based policy, the state space of the task is discretized with contact states. As a result, the policy is a mapping between the contact states and the actions of the robot. Since we would like to learn the contact graph together with the policy, we propose to apply a model-based reinforcement learning algorithm for the learning process.

6.1.1 Reinforcement Learning

Before we discuss details about our approach, we would like to briefly review the major concepts of reinforcement learning algorithms. Readers are encouraged to read more details about the reinforcement learning algorithm in [33].

A reinforcement learning model consists of a discrete state space $S$, a discrete action space $A$, and a set of reinforcement signals/rewards that reflect the
Figure 6.1: A example of a simple reinforcement learning problem. On this grid map, a robot starts from state A and try to find a policy that navigate itself to the goal state (green square). The red squares represent dangerous regions, and the robot will get penalties when it reaches the red squares.

performance of the robot by taking a sequence of actions. A reinforcement learning algorithm assumes that the control process is stochastic, which means that by taking an action $a_i$ at state $s_i$, the outcome state $s'_i$ will not be deterministic but rather a random state with a distribution over the entire state space. The goal of reinforcement learning is to find a policy $\pi$, which is a mapping between the states and the actions, which maximizes a task performance metric [33].

The transition probabilities between the states in a state space is modeled by a transition function $T(s, a, s')$, and the reward that a state can get by taking an action is modeled by a reward function $R(s, a)$. For a robot that is in state $s$, the expected reward that the robot will get by taking an action $a$ is called the Q value of the tuple $(s, a)$ and it is expressed as $Q(s, a)$. If the transition and reward functions are known, the problem can be modeled as Markov Decision Process (MDP), and the MDP can be solved to obtain the optimal policy. A policy is defined as a table of the states and their corresponding actions. For example, in the example shown in Fig. 6.1, a robot needs to reach the goal state (the green square), in which the robot will receive a huge reward, on a grid map. The red squares represent dangerous states to the robot, and the robot will receive penalties for reaching the dangerous states. An optimal policy for this example is shown in the table below:
In a reinforcement learning setting, it is assumed that one does not have the exact transition function $T$ or the reward function $R$. Depending on whether or not we want to learn the transition and reward functions, reinforcement learning algorithms are grouped as model-based ([69, 70, 57, 61]) and model-free ([73, 74]) approaches. Q-learning is a simple model-free reinforcement learning algorithm. In reinforcement learning algorithms, the Q values can be computed recursively with a known model (i.e., known $T$ and $R$). If the robot takes an action $a$ from state $s$, the Q values are calculated as follows:

$$Q(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(s', a'),$$

(6.1)

where $\gamma$ is the discount factor and $s'$ is the future state by taking action $a$ from state $s$. From the equation above, we can see that a Q value is calculated based on the immediate reward $R(s, a)$ and the discounted maximum expected future reward $\gamma \sum_{s'} T(s, a, s') \max_{a'} Q(s', a')$. Through this equation, the rewards obtained by the future state $s'$ will be backpropagated to the Q value of state $s$. For example, in the simple example shown in Fig. 6.1, the robot will receive a reward if it chooses to move right in state C. This reward will be backpropagated to the moving right action in state B since state C is the future state of moving right from state B. In the Q-learning algorithm, the robot directly learns the Q values of the actions at any given state by exploring the state space without requiring a model ($T$). Instead, the model is approximated by the running averages [14]. The rule of updating the Q values in the Q-learning algorithm is as follows [14]:

$$Q(s, a) \leftarrow Q(s, a) + \alpha (R(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a)),$$

(6.2)

where $s'$ is the state that the robot experienced by taking action $a$ from state $s$, $\alpha$ is the learning rate, $\lambda$ is the discount factor, and $\alpha$ is the exploration rate.
is the learning rate, and $\gamma$ is the discount factor $^6$. As the robot explores the state space, the accuracy of the Q values will be improved and the actions associated with the best Q values are chosen as the optimal actions.

The DynaQ algorithm, which combines model learning with the Q-learning algorithm, is a simple yet effective model-based reinforcement learning algorithm. It updates the Q values using rules of the Q-learning algorithm in equation (6.2) and updates its model ($T$ and $R$) at the same time. Additionally, in order to improve the efficiency of the learning process, the DynaQ algorithm performs $N$ iterations of Q-learning with randomly selected state-action pairs [7] at the end of each step. Algorithm 13 shows the steps of the DynaQ algorithm, where $N_{tot}$ is the total number of iterations of the algorithm.

Algorithm 13 DynaQ Algorithm

```
function UPDATE_STATE($x_t, u_t, N, w_t$)
    Initialize $Q(s, a), T(s, a, s'), R(s, a)$ for all $s \in S$, $a \in A$
    for $i = 1 \rightarrow N_{tot}$ do
       $s \leftarrow$ current state
       $a \leftarrow \arg \max_a Q(s, a)$
       Execute action $a$ and observe outcome state $s'$ and reward $R(s, a)$
       $Q(s, a) \leftarrow Q(s, a) + \alpha (R(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a))$
       $T(s, a, s') \leftarrow Update_Transit_Model(s, a, s')$
    for $j = 1 \rightarrow N$ do
        $s \leftarrow$ randomly choose a previously observed state
        $a \leftarrow$ randomly choose a previously taken action at state $s$
        $s' \leftarrow$ the previously observed outcome state of taking action $a$ from state $s$
        $Q(s, a) \leftarrow Q(s, a) + \alpha (R(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a))$
    end for
end for
end function
```

While there are pros and cons for both model-free and model-based approaches, in this chapter, we have chosen the model-based approach DynaQ to leverage rigorous contact state handling of current multi-body dynamics formulations [67] in the transition function $T$.

$^6$please refer to [14] for more details.
6.1.2 Manipulation State and Manipulation Graph

We propose to discretize the state space for a robotic manipulation task based on contact states. If we modify a contact graph by adding weights to the edges to represent the transition probabilities between the contact states, this contact graph denotes a transition function \( T(s, s') \). In a robotic manipulation task, the actions of a robot also affect the distribution of the contact states. As a result, we further add the actions to the modified contact graph and the resultant contact graph contains directed edges representing transition probabilities between the contact states by taking certain actions. This modified contact graph denotes a transition function \( T(s, a, s') \), which has the same form with the transition function of a reinforcement learning problem. Therefore, with the state space discretized by the contact states, by using a model-based reinforcement learning algorithm, we learn the contact graph/model together with the policy at the same time. In a robotic manipulation task, if we define the initial state and the goal state as two nodes in the contact graph, the sequence of actions for a robot to take can be visualized as a path between the two nodes in the graph. Fig. 6.2 shows an example of a partial contact graph\(^7\) for a simple robotic manipulation task, where a robotic gripper picks up a cup from the table. The green arrows show three steps that the robot can follow in order to pick up the cup.

6.1.2.1 Manipulation State

In a robotic manipulation task, we can divide the state space of the robot into two parts: the state space in which robot makes contacts with the object and that with no contacts between the robot and the object. When there are no contacts between the robot and the object, the optimal actions that a robot should take depend on the relative poses between the robot and the object. Taking the manipulation task shown in Fig. 6.2 as an example, before the robot making any contacts with the cup, depending on whether or not the robot is right above the cup, the robot should either move closer to the cup or close its fingers. Therefore, we define the manipulation state \( m \) that includes both the contact state \( c \) and the

\(^7\)A complete contact graph of the manipulation task will contain more nodes (contact states) and edges (actions).
Figure 6.2: An example of a partial contact graph of a robotic manipulation task, where a robotic gripper picks up a cup from the table. The green arrows show the steps that a robot can follow so that the task can be completed.

Figure 6.2 shows a partial contact graph of a robotic manipulation task, where a robotic gripper picks up a cup from the table. The green arrows show the steps that a robot can follow so that the task can be completed.

The relative pose between the robot and the object $p$:

\[ m = \{c, p\} \]  \hspace{1cm} (6.3)

Discretizing the state space using the manipulation states includes two steps: discretizing the contact state $c$ and discretizing the relative pose $p$. Discretizing the relative pose requires the discretization over the whole Cartesian space in which the robot does not collide with the object. However, this space can be as large as infinity, which leads to an intractable state space. On the other hand, since the key of a robotic manipulation task is the interaction between the robot and the object of interest, the Cartesian space closer to the object is more important than that is further away. Therefore, we propose to discretize the Cartesian space within a sphere whose center is at the geometrical center of the object, and we term this sphere as the discretization sphere. When the robot is outside the discretization sphere, we define the policy to be actions that move the robot into the sphere. Fig. 6.3 shows an example in a robotic manipulation task, where the robot (the gripper) is outside the discretization sphere of the object (the cup), and the optimal action is to move towards the center of the sphere as fast as possible.

The contact state is discrete by itself. However, the nodes in the conven-
Figure 6.3: An example in a cup-picking robotic manipulation task, where the robot (the gripper) is outside the discretization sphere of the object (the cup).

The traditional contact graph described in Chapter 5 and 3 are defined in terms of contact formations, and this definition ignores that contact states with the same contact formation can generate different dynamic behaviors of the object. For example, as shown in Fig. 6.4, although the two contact states have the same contact formation, they generate the opposite torques, which lead the object to rotate in different directions. Therefore, it is also important to further discretize a contact state further within a contact formation using the position and contact normal of a contact. We define our modified contact state as a set of contacts, and two contacts are defined to be the same if their contact locations are within a certain radius $\delta_r$ and contact normals are pointing to similar directions with tolerance $\delta_\theta$. Similar to the contact states proposed in section 5.2.1, we also include the stick-slip status of a contact in our contact states. Specifically, if a contact consists of a contact location $pc$, a contact normal $nc$ and a stick-slip variable $sc$, then two contacts $\{pc_1, nc_1, sc_1\}$ and $\{pc_2, nc_2, sc_2\}$ are defined to be the equal, iff:

\[
\|pc_1 - pc_2\| \leq \delta_r \\
acos(nc_1 \cdot nc_2) \leq \delta_\theta \\
sc_1 = sc_2
\]
Figure 6.4: A robot pushes a rectangle at different locations within the same contact formation. The case on the left will generate a torque that rotates the object clockwise, and the case on the right will rotate the object counter-clockwise.

Figure 6.5: An example of two contact states for a triangular object. Each small rectangle represents a contact sensor, and the triangle on each contact sensor represents the range of angles in which the sensor can be triggered. $c_1$ is defined as a contact state that has both green contact sensors triggered, and $c_2$ is a contact state with both red contact sensors triggered.

Based on our contact state definition above, the object can be view as being covered with contact sensors. Each contact sensor can be trigger by contact forces exerted on a certain object surface area, at a certain range of angles, and with either sticking or sliding status. A contact state $c_i$ is defined as a set of these contact sensors being triggered at the same time. Fig. 6.5 shows two contact states examples (shown as two colors) on a triangular object.
6.1.2.2 Manipulation Graph

With the state space discretized by the previously defined manipulation states, we define the manipulation graph to describe the transition probabilities between the manipulation states. Similar to the modified contact graph shown in Fig. 6.2, in a manipulation graph, a node represents a manipulation state, a weighted edge connecting two manipulation states represents the transition probability between the two states, and actions are attributes of the directed edges. Notice that although each manipulation state can transition to any other manipulation state given enough time, we only need to consider manipulation state transitions within a certain time interval. Since we include the robot’s actions in the graph, this time interval is defined as the period between two action commands (the sample period). For a manipulation state, depending on what action the robot will take, the transition probabilities to other manipulation states can be different. To demonstrate this, Fig. 6.6 shows an example of a simple two-node manipulation graph. In this simple manipulation graph, the robot is able to move to the left and to the right, and the object is on the right side of the robot. We consider two simple manipulation states: \( m_1 \) no contacts between the robot and the object, and \( m_2 \) the robot making contact with the object. If the robot chooses the action of moving right, there will only be transitions from \( m_1 \) to \( m_2 \) and from \( m_2 \) to itself. Similarly, if the robot moves left, transitions that are possible will be from \( m_2 \) to \( m_1 \) and from \( m_1 \) to itself.

The manipulation graph also represents the transition function \( T \) for a robotic manipulation task when we would like to learn actions when there are no contacts between the robot and the object. With the definition of the manipulation graph shown above, it is learned as the transition function using a reinforcement learning algorithm. Additionally, since the manipulation state contains the contact state, as will be shown in section 6.2, the manipulation graph is also used in our proposed particle filter for contact state sampling. In a manipulation graph, the sequence of actions for a robot to take in order to finish a robotic manipulation task can also be visualized as a path in the graph. Fig. 6.7 shows a path of a robotic gripper picking up a cup from a table with an initial pose that is far away from the cup.
Figure 6.6: An example demonstrates a simple directed manipulation graph with a robot (red) and an object (green). The red lines correspond to the graph if the robot moves to the right and the blue lines show the graph when the robot moves to the left.

6.1.3 Reinforcement Learning with the Manipulation States

As mentioned above, we apply the DynaQ algorithm for policy and manipulation graph learning. In our applied DynaQ algorithm, the model and policy are updated at the frequency of the controller of the robot. Since the contact states cannot be measured directly and thus updating the model and the policy would be impossible in physical experiments, in this work, we implement the learning algorithm in simulation experiments, which use LCP model as their dynamic models. In the simulation experiments, the contact states are directly acquired by checking the solution of the LCP model.

Although we treat our problem as an MDP learning problem, in the physical manipulation tasks, the state of the object is measured using noisy sensors, which makes the manipulation task a partially observable Markov decision process (POMDP) problem. However, in the simulation experiments, it is difficult to simulate the noise models of the sensors, e.g., simulating the noise of the Kinect sensor. Therefore, instead of solving the POMDP learning problem directly in our simulation experiments, we solve the MDP learning problem, which assumes that the state of the object is fully observable (no uncertainties in the measurements of the state) and then apply the policy to a physical experiment through a QMDP algorithm (see section 6.2 for more details).

We set the robot to learn the policy through multiple simulation experiments. In these simulation experiments, the inputs and the outputs of our reinforcement
Figure 6.7: An example of a partial manipulation graph of a robotic manipulation task, where a robotic gripper picks up a cup from the table. The green arrows show the steps that a robot can follow so that the task can be completed.

learning algorithm are listed as follows:

- **Inputs:** the manipulation states of the object
  Our algorithm obtains the contacts of the object together with the poses of the object and the robot from the simulation engine. Then the manipulation states are calculated based on its definition in section 6.1.2.1.

- **Outputs:** the policy and the manipulation graph for the manipulation task
  After the learning finishes, our algorithm will be able to provide the policy, which is the mapping between the manipulation states and the robot’s optimal actions. As mentioned in section 6.1.1, we adopt a model-based reinforcement learning algorithm for our problem. Therefore, besides the policy, the algorithm will also output the manipulation graph, which is equivalent to the learning algorithm’s model, of the manipulation task.

Additionally, we set a fixed length of each simulation experiment, and when an experiment reaches its time limit, the robot stops its action regardless the current state of the system. Therefore, each simulation experiment can either be successful
or unsuccessful. We term a simulation experiment as a learning episode. In order to learn a policy, a robot needs to explore the state and action spaces through multiple simulation experiments (i.e., learning episodes). We then term the set of learning episodes as a learning process. At the beginning of a learning process, we randomly select an initial pose of the robot in a starting region, where the robot is not in contact with the object. Then as we start each learning episode, we set the robot to the selected initial pose and let the robot act according to the policy it has discovered so far.

6.1.3.1 Reward Function

In an MDP problem, the transition function describes the transition probabilities among the states and the reward function shows the reward that the robot can get by reaching certain states. In our algorithm, we define the reward function as a summation of two types of rewards. The first and the most important reward is the reward related to the goal of the manipulation task. One example of this reward is reaching the goal state $R_g$. Depending on the goal of the manipulation task, the goal state can be defined differently. For example, for an assembly task where only the final contact state $c_f$ is vital to completing the task, one can assign goal state reward to reaching manipulation states $m_i = c_i, p_i$ whose contact state $c_i$ equals to the final contact state $c_f$. This reward can also include rewards related to the pose of the object. Depending on the requirements of the manipulation task, the form of the reward function can be different. For example, if the pose of the object at its goal state is known to be fixed, the reward function can be written as follows:

$$R_{\text{pose}} = \alpha (p_{\text{target}} - p_{\text{object}})^2 + \beta (o_{\text{target}} - o_{\text{object}})^2$$  \hspace{1cm} (6.7)

where $p_{\text{target}}$ and $o_{\text{target}}$ are the target position and orientation vectors, and $p_{\text{object}}$ and $o_{\text{object}}$ are the object’s position and orientation vectors, $\alpha$ and $\beta$ are weight constants.

The second reward is the exploration reward. We adopted the greedy strategy [33] to choose our exploration rewards. A fixed reward is given to any unvisited state and this exploration reward will decrease as the state is visited more frequently.
Equation (6.8) shows how the exploration reward is calculated for manipulation state $m_i$, where $R_{init}$ is the initial fixed reward for exploration, and $n$ is the number of times that the robot has visited $m_i$.

$$R_{explore}(m_i) = \frac{R_{init}}{n} \quad (6.8)$$

### 6.1.3.2 Our Reinforcement Learning Algorithm

Our reinforcement learning algorithm follows the general steps of the DynaQ algorithm, which is shown in algorithm 13. As mentioned above, we update the model and the Q values at the frequency of the robot’s controller. Therefore, each step in a learning episode can correspond to multiple time steps in a simulation experiment. For example, if the robot is controlled at 10hz and the time step of the simulation experiment is $\frac{1}{60}$ seconds, then each step in a learning episode contains 6 simulation time steps. In a learning episode, at step $l$, the robot takes action $a_l$ and transition from manipulation state $m_l$ to manipulation state $m_{l+1}$. The robot then updates its transition function with the tuple $(m_l, a_l, m_{l+1})$. The Q value $Q(m_l, a_l)$ is updated following equation (6.2). At the end of each step, we also update the Q values for $k$ randomly selected tuples $(m_k, a_k)$. This algorithm is shown in Algorithm 14, where we set a fixed number of episodes ($N_{episodes}$) and a fixed number of steps ($N_{steps}$) for each episode. $M$ is the state space of all manipulation states, $A$ is the action space, $S_{experienced}$ represents the set of all experienced states, $A(m_i)$ returns the actions that have been taken at state $m_i$, $Cal\_Action$ calculates the action to be taken based on the Q value of the current state $m_l$, $Step\_Simulation$ steps the simulation to acquire the manipulation state for the next step, $Cal\_Reward$ calculates the reward for a manipulation state based on the reward functions described in section 6.1.3.1, $Update\_Reward\_Model$ updates the exploration rewards, $Update\_QValues$ updates the Q values with equation (6.2), and $Random\_Sample$ randomly samples the state space or the action space.

Our algorithm returns two variables at the end of a learning process: the transition function $T$, which will be used as the manipulation graph, and the Q values $Q$, which will be used to calculate the optimal actions in the physical experiments.
Algorithm 14 Manipulation Skills Reinforcement Learning Using Manipulation States

\begin{algorithm}
\begin{algorithmic}
\Function{RL\_MANIPULATION}{ }
\State \text{Initialize } Q(m, a), T(m, a, m') and \( R(m, a) \) for all \( m \in M \) and \( a \in A \)
\State \text{Set } \( m_1 = \text{Random\_Initialize\_State} \)
\For{\( i = 1 \rightarrow N_{\text{episodes}} \)}
\For{\( l = 1 \rightarrow N_{\text{steps}} \)}
\State \( a_l = \text{Cal\_Action}(m_l, Q) \)
\State \( m_{l+1} = \text{Step\_Simulation}(m_l, a_l) \)
\State \( r_l = \text{Cal\_Reward}(m_{l+1}) \)
\State \( m_{l+1} \rightarrow S_{\text{experienced}} \)
\State \( T = \text{Update\_Transit\_Model}(m_l, a_l, m_{l+1}) \)
\State \( R = \text{Update\_Reward\_Model}(R, r_j) \)
\State \( Q(m_l, a_l) = \text{Update\_QValues}(Q, R(m_l, a_l)) \)
\EndFor
\EndFor
\State \text{return } T, Q
\EndFunction
\end{algorithmic}
\end{algorithm}

6.2 Closed-loop Policy Execution for Robotic Manipulation Tasks

As discussed in Chapter 5, a contact graph generates a distribution over the contact state, which is used to sample the contact state for our contact-based RBPF to estimate the state. With our previous discussed algorithm, we compute a manipulation graph, which generates a distribution over the manipulation state. To tackle this problem, we convert our manipulation state distribution to a contact state distribution by merging the probability densities of manipulation states that have the same contact states. For example, for a simple distribution over three manipulation states: \( m_1(c_1, p_1), m_2(c_1, p_2), m_3(c_2, p_3) \) with probability density values 0.1, 0.6, 0.3, respectively. We generate the distribution for the contact states by merging \( m_1 \) and \( m_2 \) (since they have the same contact state), and the distribution of over the contact states is computed as \( \{c_1 : 0.7, c_2 : 0.3\} \). On the other hand,
our proposed contact-based RBPF estimates the contact states and the poses of the object. Since our computed policy is based on manipulation states, we then convert the estimations of our particle filter to manipulation states by combining the contact state estimations with the pose estimations.

As discussed previously, in a physical experiment, only noisy measurements of the states can be obtained and the manipulation task becomes a POMDP problem. To illustrate more specifically, rather than knowing the current state $s$ deterministically, in a POMDP problem, one can only assume that a belief state $b$ of the current state is known. A belief of a state $s$ is the probability density value of being at the state and it is also denoted as $b(s)$. Instead of calculating the Q values for all states, POMDP requires a Q function $Q(b)$ of the belief state $b$, which is an NP-hard problem [26]. Since our policy is learned by assuming an MDP problem where the states are known deterministically, our learned policy cannot be directly applied to a physical experiment. To overcome this, we combine our learned policy with a QMDP algorithm. The QMDP algorithm is an effective way to approximate the Q function of a POMDP problem using the Q values of the POMDP’s underlying MDP problem. In a QMDP algorithm, we treat the Q function as a single linear function and calculate the Q value for a belief state $b$ as follows [17]:

$$Q_a(b) = \sum_s b(s)Q(s, a), \quad (6.9)$$

where $Q_a(b)$ represents the Q value for belief state $b$ by taking action $a$.

With our proposed contact-based RBPF, the current belief state $b$ of the manipulation state space is represented in the form of particles. Specifically, the estimation of a particle can be converted to a manipulation state, and the weight of a particle corresponds to the probability of the predicted manipulation state. To compute the belief of a manipulation state $m_i$, we take the sum of the weights of all particles that predict $m_i$ as shown in equation (6.10), where $\Omega_i$ is the set of indices of particles that predict $m_i$.

$$b(m_i) = \sum_{j \in \Omega_i} w[j] \quad (6.10)$$
We, therefore, calculate the Q values of each action for the current belief state following equation (6.9), and the optimal action to take is the action that achieves the best Q value, which is shown below:

\[
a^* = \arg \max_a \sum_{m_i} b(m_i)Q(m_i, a)
\]  

(6.11)

6.3 Experiments

We tested our algorithm physical experiments. In the physical experiments, a robot performs an assembly task, where the goal for the robot is to place a triangular object inside a hole on the ground that fits the shape of the object. Fig. 6.8 shows the setup of the manipulation task.

This triangular object consists of two parallel triangle sides and three rectangle sides. In this manipulation task, the changes of the manipulation states that the robot needs to generate are most significant in the plane parallel to the triangle sides of the object. Therefore, we further simplify the task to a two-dimensional case in our reinforcement learning algorithm. In the two-dimensional setup, we treat the robot as a rectangle, the object as a triangle, and the hole as a triangle hole on the
ground. We also divide the action space of the robot into eight discrete actions, which include two velocity actions along the plus/minus directions of the x and y-axes of the two-dimensional plane. Please refer to Fig. 6.13 for further details.

After a policy is learned, we applied the policy in the three-dimensional experiments by projecting the three-dimensional manipulation states to a two-dimensional plane. We define this plane as a plane $U$ that is parallel to the triangle sides of the object. For a manipulation state $m_i = \{c_i, p_i\}$, it consists of a contact state $c_i$ and a relative pose $p_i$. Therefore, we will need to project $c_i$ and $p_i$ separately onto plane $U$. As described in section 6.1.2.1, a contact state is a set of contacts and a contact is described as a contact location vector, a contact normal vector, and a stick-slip variable. For a contact in $c_j$, in the object’s body frame, we denote its contact location vector as $(x_{cj}, y_{cj}, z_{cj})$ and its contact normal vector as $(x_{nj}, y_{nj}, z_{nj})$. In order to project a contact state $c_i$ onto plane $U$, we then project these two vectors onto plane $U$ as $(x'_{cj}, y'_{cj})$ and $(x'_{nj}, y'_{nj})$. Especially, in our experiments, we align the object’s body frame so that the x and y-axes are both parallel to plane $U$. As a result, the projection can be simplified as follows:

$$
x'_{cj} = x_{cj} \quad \quad (6.12)
$$
$$
y'_{cj} = y_{cj} \quad \quad (6.13)
$$
$$
x'_{nj} = x_{nj} \quad \quad (6.14)
$$
$$
y'_{nj} = y_{nj} \quad \quad (6.15)
$$

The stick-slip status of a contact remains the same after the projection. A contact state $c_i$ is then projected onto the plane by projecting each of its contacts onto plane $U$. A relative pose $p_i$ contains a position vector $(x_{pj}, y_{pj}, z_{pj})$ and a rotation vector $(\alpha_{pj}, \beta_{pj}, \theta_{pj})$. Similar to the projection of a contact, we project the position vector and the rotation vector onto plane $U$ as a position vector $(x'_{pj}, y_{pj})$ and a rotation $\theta_{pj}$. Fig. 6.9 shows an example of projecting a contact onto the plane $U$. Finally, with the projection steps described above, we project the three-dimensional manipulation states estimated by our filter to a two-dimensional manipulation state, and the optimal action is selected using equation (6.11).
Figure 6.9: An example of projecting a contact with a contact point and contact normal vector (in pink) from the three-dimensional space to a contact point and contact normal vector (in blue) in the two-dimensional plane (the red plane in the figure).

In the policy and manipulation graph learning process, we perform the simulation experiments in a two-dimensional RPI-Matlab Simulator \[?\]. The learning follows the exact same steps described in algorithm 14. Specifically, each learning process consists of 60 learning episodes, which means that \(N_{\text{episodes}}\) equals 60. Also, for each learning episode, we set the number of steps \(N_{\text{steps}}\) to 150 and \(k\) to 200. We set the length of a simulation experiment to be 15 seconds with \(\frac{1}{60}\) seconds for a simulation time step. We set the reward of reaching the goal state, i.e., the object getting into the hole, to be 120, and the exploration reward to be 5. In our experiments, we found that on average, 60 learning episodes are enough for our algorithm to converge to a policy and the size of the manipulation graph is about 500. The average running time for a learning process is about 10 minutes.

To test our learned policy the physical experiments, we used the WAM arm in our lab as the robot and a Microsoft Kinect as the vision sensor. The pose of the end effector of the robot is calculated using the forward kinematics of the robotic arm. Similar to the three-dimensional physical experiments in section 5.3.1, we also create a point cloud of the triangular object and obtain the measurements of the pose of the object by the ICP algorithm. The setup of the physical experiment is shown in Fig. 6.10.
Although in our experiments the policy and the manipulation graph are learned in two-dimensional simulations, to learn policies and manipulation graphs in three-dimensional simulations will follow the exact same steps. However, one concern with learning in three-dimensional simulations is that in a learning process, the size of the manipulation graph in three-dimensional simulations may grow much faster than that in two-dimensional simulations. However, we observed from our learning processes that many manipulation states in the graph are only visited by our algorithm as the robot explores the state space and are never visited once an optimal policy is found. As a result, these manipulation states are redundant and can be removed. By removing the redundant manipulation states, even with three-dimensional simulations, the size of the manipulation graph can be constrained.

6.3.1 Results and Discussion

In order to test our proposed algorithm, we ran 50 learning processes, which generated ten policies for the manipulation task mentioned above. To evaluate the progress of the learning processes, for each learning episode, we then calculated the mean of the Q values over all steps. This mean Q value corresponds to the quality of the policy that has been discovered up to a learning episode. For example, the mean Q value of the 30th episode corresponds to the quality of the policy the robot has learned in the first 30 learning episodes. As we assigned positive rewards for the goal manipulation state, the higher a mean Q value is, the better a policy performs. We compare our manipulation state-based learning algorithm with another
DynaQ algorithm that learns policies using a state space discretized with continuous states. In order to make a fair comparison, we set the size of the grids in their discretizations to be the same. For example, we set the grid size along the $x$-axis to be 0.03 meters for both algorithms. Fig. 6.11 shows the learning curve of one learning process in terms of the mean Q values of all its 60 learning episodes. The $x$-axis corresponds to the indices of the learning episodes and the $y$-axis corresponds to the values of the mean Q values. This learning curve shows that for this learning process, with our learning algorithm, the mean Q value converges to a value of 1.18 after 30 episodes of learning, which indicates that our learning algorithm converges to a policy. One should notice that the mean Q values fluctuate in the first few episodes, which corresponds to the exploration of the state space. Depending on the initial state, the performance of a reinforcement learning algorithm varies from one learning process to another. Therefore, comparing the performances of the manipulation state-based learning and the learning with continuous states in one learning process is statistically insufficient. Therefore, we tested both learning algorithms (manipulation state-based and continuous state-based) in 50 learning processes and compares their average performance over these 50 learning processes. Similar to Fig. 6.11 where we show the learning curve of one learning process, we compute the average learning curves of both algorithms over the 50 learning processes. For each learning episode in the average learning curve, we compute the variance of the mean Q values over all the processes. Fig. 6.12 shows the average learning curves of both learning with manipulation states (left) and learning with continuous states (right). The variances of the mean Q values are shown as the shaded areas above and below the learning curve. Although the convergence rates of the two algorithms are approximately the same, the variances of the continuous state-based learning algorithm are much higher than that of the manipulation state-based learning algorithm. Higher variances of the continuous state-based learning indicate that its mean Q values have more fluctuation and thus its performance lacks consistency over the learning processes. Additionally, we define a learning process as a converged process if it satisfies the following two criteria:

- We calculate the average value of the mean Q values of the last ten episodes.
On average, for the last ten episodes, the mean Q value of each episode does not deviate from the average value for more than 0.1, which is shown below:

$$\sum_{i=51}^{60} |Q_i - \bar{Q}| \leq 1,$$

(6.16)

where $Q_i$ is the mean Q value of the $i$th episode and $\bar{Q}$ is the average value of the last ten episodes.

- The learning process converges to a policy that successfully finishes the manipulation task.

Based on the two criteria, we check the convergence of each of the 50 learning processes for both manipulation state-based and continuous state-based algorithms. We find that 41 of the 50 learning processes manage to converge using the manipulation state while only 23 of the 50 learning processes converges using the continuous state. Since we reset the initial pose of the robot for each learning process, we conclude that the manipulation state-based learning algorithm is less sensitive to the initial conditions and performs more consistently.

Eight key states of a policy discovered by our algorithm are shown in Fig. 6.13. In this learning process, the initial position of the robot is above the object. In the
policy learned by our algorithm, the robot navigates to the left side of the object \((a, b)\), and pushes on the upper part of the object \((c, d, e, f, g, h)\). Notice that the contacts formed between the robot and the object can be sliding, which leads to the robot pushing the lower part of the object with more efforts \(^8\) as the object rotates. As a result, the best strategy for the robot to finish this manipulation task is not to keep pushing the object. In our learned policy in Fig. 6.13 \(f, g\), the policy repositions the robot by choosing to move up \((f)\) and further chooses to push the top part of the object \((g)\).

We tested our learned policy in physical experiments using the method discussed in section 6.2 to incorporate our contact-based RBPF discussed in the previous chapter as the state estimator. Six key frames of one successful physical experiment is shown in Fig. 6.14. We also found there are experiments where our robot failed to complete the task due to the friction coefficient in this experiment is higher than the one used in our learning processes. Fig. 6.15 shows six key frames of a failed experiment. In this failed experiment, the robot tried to rotate the object.

---

\(^8\)Since the point of contact between the robot and the object is closer to the center of gravity of the object when the object pushes the lower part, more efforts will be needed to rotate the object.
Figure 6.13: Eight manipulation states examples each representing a case where one of the eight actions is optimal. The red arrows in the figures represent the direction and the magnitude of the action (the velocity in our case) of the robot. The $x$ and $y$-axes are the axes of the two-dimensional plane.

by squeezing one edge of the object (bottom three frames) and failed to achieve the rotation as a result of the unexpected large friction forces. This problem can be solved by combining learning multiple policies using different parameters (e.g., friction coefficient in our experiments) with parameter estimation before executing the manipulation task. Therefore, we believe that parameter estimation is a very important future work based on this thesis, which will be discussed in the next chapter.
6.4 Conclusion

In this chapter, we discussed an algorithm that learns the policies for robotic manipulation tasks with state space discretized using contact states. We define a
manipulation state as a set that includes both the contact state and the relative pose between the robot and the object. A manipulation graph is then defined as a directed weighted graph to represent the transition probabilities between the manipulation states. Further, we include actions in the manipulation graph so that it takes the exact form of a transition function in a reinforcement learning problem. We adopt a model-based reinforcement learning algorithm called DynaQ algorithm to learn the manipulation graph and a policy simultaneously. Finally, we combine our previous proposed contact-based RBPF with the learned policy using a QMDP algorithm for policy execution in actual manipulation tasks. We tested our algorithm in physical experiments. Results show that comparing with continuous state-based learning algorithms, our algorithm is less sensitive to changes in the initial conditions and manages to converge to an optimal policy more consistently. In the physical experiments, our robot successfully finished an assembly task with the policies generated and executed using our algorithms. From the physical experiments, we noticed that the parameters, such as friction coefficients, are important to policy learning. Therefore, we believe that parameter estimation is a very important future work based on this thesis.
CHAPTER 7
Conclusion and Future Work

7.1 Conclusion

In this thesis, we focus on improving the perception capability for robotic manipulation tasks. In our proposed approach, we combine a multi-body dynamic model with a Bayesian filtering method to estimate both the continuous state and the discrete contact state of objects in a manipulation task. Specifically, we choose an accurate model of multi-body dynamics and apply particle filtering methods to estimate both an object’s continuous states (i.e., poses of the object) and its contact states.

To understand the design trade-offs of using particle filters for the state estimation problem of a manipulation task, we evaluate different modeling options, which include different contact dynamic models combined with different particle filtering methods, in a one-dimensional experiment. Further, based on this evaluation, we provide guidance on the choices of modeling options for various robotic manipulation applications.

One important conclusion that we draw from the evaluation of modeling options is that using an accurate model of multi-body dynamics is important for our filter to estimate the states precisely. The Linear Complementarity Problem (LCP) model is an accurate model for multi-body dynamics and has been used in many simulation engines. Therefore, we develop the contact-based Rao-Blackwellized particle filter (RBPF) that takes the LCP model as its model of multi-body dynamics. In the contact-based RBPF, the piecewise linear property of the LCP model is fully exploited by using an RBPF to sample the contact states with a contact graph.

The continuous states are updated through Kalman filters with linear models that are obtained from the LCP model. We compare our proposed contact-based RBPF with particle filters that sample the continuous state space in both two-dimensional and three-dimensional physical experiments. Results show that the contact-based RBPF is less sensitive to errors in the dynamic models, such as the friction coeffi-
cients. Also, since contact-based RBPF samples the low dimensional contact state space, with the same number of particles, it performs better and is more stable than particle filters that sample the continuous state space.

For a robotic manipulation task, a robot needs to take optimal actions based on its perception of the current state of the system. Therefore, we further develop an algorithm that combines a model-based reinforcement learning algorithm with a state space discretized with the contact states. The outputs of this algorithm are a policy that maps a contact state to the optimal action, and a model that corresponds to the contact graph of the manipulation task. With our previously developed contact-based RBPF, we propose to apply the learned policy to robotic manipulation tasks in an online fashion. Specifically, the learned contact graph is reused in the contact-based RBPF for accurate state estimation, and the optimal action is chosen based on a QMDP algorithm that combines the state estimation from the filter with the learned policy. We tested this algorithm in physical experiments. Results show that comparing with continuous state-based learning algorithms, our algorithm is less impacted by changes in the initial conditions and manages to converge to an optimal policy more consistently. In the physical experiments, our robot successfully finished an assembly task with the policies generated and executed using our algorithms.

In summary, in this thesis effort, we develop filters to improve a robot’s perception of the states in a robotic manipulation task. With the developed filters, we further design an algorithm that generates policies that enable a robot to choose optimal actions in closed-loop. In chapter 1, we lay our approach out in Fig. 1.3. With discussions in the previous chapters, we now rewrite this diagram with the developed pieces as shown in Fig. 7.1.

7.2 Future Work

In future work, we consider the following ideas to be promising extensions of this thesis work:

- Develop filters that incorporate the compliant body models. In this thesis, we mainly focus on developing filters that rely on the LCP model, which is a
rigid body dynamic model. However, as pointed out in our evaluation of the modeling options in Chapter 4, the compliant body models perform better at estimating contacts comparing with the rigid body models. Therefore, we believe that incorporating the compliant body models can boost the performance of filters for estimating the contact states.

- Improve the efficiency of the contact-based RBPF. We notice that the major trunk of the running time of the contact-based RBPF is spent on collision detection to build a distribution from which we sample the contact states. Notice that in our particle filter, the continuous state estimations of most particles are similar. Therefore, one can improve the efficiency of the contact-based RBPF by reusing the collision detection result of one particle for other particles.

- Design controllers for contact state transitions. To learn policies for robotic manipulation tasks, we discretize the action space with a fixed set of actions. As discussed in the previous chapters, with known contact states, the LCP model can be converted a single linear model. Since the state space is dis-
cretized with the contact states, one should be able to develop controllers that transition a contact state to another. As a result, one can view the policy searching for robotic manipulation problems as a path searching problem on the contact graph, and the transitions between contact states are deterministic under the designed controllers.

- Develop algorithms for parameter estimation. In the conclusions of Chapter 6, we emphasize the importance of the accuracy of parameters of the multi-body dynamic model for the success of a robotic manipulation task. In the framework of the contact-based RBPF, the LCP model is converted to a linear model with known contact states, and our developed reinforcement learning algorithm is able to learn a policy that enforces a contact state for the dynamic system. One can develop algorithms that identify contact states that can be used for estimating the desired parameters, e.g., sliding contacts on the surface of an object for friction coefficients estimation. Then, a policy can be learned to enforce the identified contact states. As system identification problem for linear systems has been well studied, one can then combine the contact-based RBPF to further solve the parameter estimation problem.

- Improve the reinforcement learning algorithms. In Chapter 6, we found that the parameter differences between the simulator and the physical environment can lead to failed executions in physical experiments. One approach to tackling this problem is to develop parameter estimation algorithms, which is discussed above. On the other hand, one can treat the policies learned through simulation experiments as the seed policies that can be further improved through physical experiments. Therefore, we think that one can develop a reinforcement learning framework on the physical experiments and combine it with the approach discussed in this thesis for better policy learning results.
A.1 Contact-Based RBPF Conditional Distribution Derivation

In the contact-based RBPF, we are interested in \( p(x_t, c_t | z_{0:t}, u_{0:t}) \), which is the distribution of the continuous and contact states given all the measurements up to time step \( t \). Using the Bayes’ rule, we derive this conditional distribution in terms of the distributions that we already obtained. The details of the derivation is as follows:

\[
p(x_t, c_t | z_{0:t}, u_{0:t}, u_{0:t}) = p(x_t | c_t, z_{0:t}, u_{0:t})p(c_t | z_{0:t}, u_{0:t}) \tag{A.1}
\]

\[
= p(x_t | c_t, z_{0:t}, u_{0:t}) \frac{p(c_t, z_t | z_{0:t-1}, u_{0:t})p(z_{0:t-1}, u_{0:t})}{p(z_{0:t}, u_{0:t})} \tag{A.2}
\]

\[
\propto p(x_t | c_t, z_{0:t}, u_{0:t})p(c_t, z_t | z_{0:t-1}, u_{0:t}) \tag{A.3}
\]

\[
= p(x_t | c_t, z_{0:t}, u_{0:t}) \sum_{c_{t-1}} p(c_t, c_{t-1}, z_t | z_{0:t-1}, u_{0:t}) \tag{A.4}
\]

\[
= p(x_t | c_t, z_{0:t}, u_{0:t}) \sum_{c_{t-1}} p(c_t | c_{t-1}, z_{0:t-1}, u_{0:t})
\]

\[
p(z_t | c_t, c_{t-1}, z_{0:t-1}, u_{0:t})p(c_{t-1} | z_{0:t-1}, u_{0:t-1}) \tag{A.5}
\]

\[
= p(x_t | c_t, z_{0:t}, u_{0:t})p(z_t | c_t, z_{0:t-1}, u_{0:t}) \tag{A.6}
\]

\[
\sum_{c_{t-1}} p(c_{t-1} | c_t, z_{0:t-1}, u_{0:t})p(c_{t-1} | z_{0:t-1}, u_{0:t-1})
\]

\[
= p(x_t | c_t, z_{0:t}, u_{0:t})p(z_t | c_t, z_{0:t-1}, u_{0:t})
\]

\[
\sum_{c_{t-1}} \int p(c_{t-1}, x_{t-1} | c_{t-1}, z_{0:t-1}, u_{0:t-1})dx_{t-1}p(c_{t-1} | z_{0:t-1}, u_{0:t-1}) \tag{A.7}
\]

\[
= p(x_t | c_t, z_{0:t}, u_{0:t})p(z_t | c_t, z_{0:t-1}, u_{0:t})
\]

\[
\sum_{c_{t-1}} \int p(c_{t-1} | c_{t-1}, x_{t-1})p(x_{t-1} | c_{t-1}, z_{0:t-1}, u_{0:t-1})dx_{t-1}p(c_{t-1} | z_{0:t-1}, u_{0:t-1}) \tag{A.8}
\]
The result of the derivation above is the same with equation (5.2), which consists of known distributions as explained in section 5.1.

### A.2 Converting LCP Models to Linear Models with Known Contact State

We can further derive equation (5.7) as follows:

\[
\begin{bmatrix}
0 \\
0 \\
0
\end{bmatrix} = \begin{bmatrix}
(G_n^T)\alpha & 0 & 0 & 0 \\
(G_f^T)\alpha & 0 & 0 & E_{\alpha\alpha} \\
0 & U_{\alpha\alpha} & -(E^T)_{\alpha\alpha} & 0
\end{bmatrix} \begin{bmatrix}
v_{t+1} \\
((p_n)_{t+1})\alpha \\
((p_f)_{t+1})\alpha \\
(\sigma_{t+1})\alpha
\end{bmatrix}
\]

\hspace{1cm}+ \begin{bmatrix}
\left(\frac{\Psi_n}{h}\right)\alpha \\
0 \\
0
\end{bmatrix} \quad (A.9)

\[
= \begin{bmatrix}
(G_n^T)\alpha \\
(G_f^T)\alpha \\
0
\end{bmatrix} v_{t+1} + \begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & E_{\alpha\alpha} \\
0 & U_{\alpha\alpha} & -(E^T)_{\alpha\alpha} & 0
\end{bmatrix} \begin{bmatrix}
((p_n)_{t+1})\alpha \\
((p_f)_{t+1})\alpha \\
(\sigma_{t+1})\alpha
\end{bmatrix}
\]

\hspace{1cm}+ \begin{bmatrix}
\left(\frac{\Psi_n}{h}\right)\alpha \\
0 \\
0
\end{bmatrix} \quad (A.10)

Since we want to eliminate the impulse vector from the Newton-Euler function, an express of the impulse vector in terms of the velocity vector needs to be derived. However, the matrix in front of the impulse vector in equation (A.10) is not invert-
ible. To tackle the problem, we rewrite equation (3.49) as follows:

\[
\begin{bmatrix}
0 \\
(p_n)_{t+1} \\
(p_f)_{t+1} \\
s_{t+1}
\end{bmatrix}
=
\begin{bmatrix}
M & I & -G_n & -G_fD & 0 \\
G_n^T & \hat{G}_n & \hat{G}_nG_n & \hat{G}_nG_fD & 0 \\
G_f^T & \hat{G}_f & \hat{G}_fG_n & \hat{G}_fG_f & E \\
0 & 0 & U & -E^T & 0
\end{bmatrix}
\begin{bmatrix}
v_t \\
p_{app} \\
(p_n)_{t+1} \\
(p_f)_{t+1} \\
\sigma_{t+1}
\end{bmatrix}
+
\begin{bmatrix}
-Mv_{t+1} \\
\frac{(\Psi_n)}{h} \\
0 \\
0
\end{bmatrix}
\tag{A.11}
\]

where \(\alpha\) has the same definition with that in equation (5.7), \(\hat{G}_n = G_n^T M^{-1}, \hat{G}_f = G_f^T M^{-1}\) and \(I\) is the identity matrix. Therefore, similar to equation (A.10), we have the following equation:

\[
\begin{bmatrix}
0 \\
0 \\
0
\end{bmatrix}
=
\begin{bmatrix}
(G_n^T)_\alpha \\
(G_f^T)_\alpha \\
0
\end{bmatrix}
v_t
+\begin{bmatrix}
(G_n^T)_\alpha \\
(G_f^T)_\alpha \\
0
\end{bmatrix}
p_{app}
+\begin{bmatrix}
\hat{G}_nG_n \alpha & \hat{G}_nG_fD \alpha & 0 \\
\hat{G}_fG_n \alpha & \hat{G}_fG_fD \alpha & E_{\alpha} \\
U_{\alpha} & -(E^T)_{\alpha} & 0
\end{bmatrix}
\begin{bmatrix}
(p_n)_{t+1} \alpha \\
(p_f)_{t+1} \alpha \\
\sigma_{t+1} \alpha
\end{bmatrix}
+\begin{bmatrix}
\frac{(\Psi_n)}{h} \alpha \\
0 \\
0
\end{bmatrix}
\tag{A.12}
\]

Therefore, we can express the impulse vector in terms of \(v_t\) and \(p_{app}\) as:

\[
\begin{bmatrix}
(p_n)_{t+1} \alpha \\
(p_f)_{t+1} \alpha \\
\sigma_{t+1} \alpha
\end{bmatrix}
= K^{-1}H^T v' + K^{-1}Fp_{app} + K^{-1}
\begin{bmatrix}
\frac{(\Psi_n)}{h} \alpha \\
0 \\
0
\end{bmatrix}
\tag{A.13}
\]

where matrices \(H, K\) and \(F\) are defined in section 5.2.2. Because the zero entries of the impulse vector make no contributions to the final results, we can take equation

\[\text{As mentioned in section 5.2.2, we ignore the bilateral constraints in the derivation.}\]
(A.13) back to the first row of equation (A.11), which yields,

\[ v_{t+1} = M^{-1}(M - HK^{-1}H^T)v_t + M^{-1}[I - HK^{-1}F - H] \begin{bmatrix} p_{app} \\ \left[ \frac{\langle \Psi_n \rangle_h}{h} \right]_a \cdot \\ 0 \\ 0 \end{bmatrix} \]  \hspace{1cm} (A.14)

A.3 Derivation of the Inverse K Matrix

There are two problems with computing the inverse of K matrix inexplicitly (computing the K matrix first and taking the inverse):

- First, one should notice that not every element in matrix K contributes to the final results. Based on equation (A.13), the structure of matrix H, F and \( \left[ \frac{\langle \Psi_n \rangle_h}{h} \right]_a \cdot \) implies that the rightmost rows of matrix \( K^{-1} \) will not be contributing the final results. Additionally, since we plug equation (A.13) to the first row of equation (A.11) to compute the linear model, one can easily see that the bottom rows of matrix \( K^{-1} \) will also not be used because of the similar structure of matrices in the first row of equation (A.11). As a result, computing the inverse of matrix K is not efficient. As matrix K can become singular in some cases, we will have to calculate the inverse of matrix K approximately by discarding the singular eigenvalues. Therefore, discarding the singular eigenvalues based on the whole matrix K also imposes a problem when not every element of matrix K will be used to compute the final results.

- Second, the parameters of the multi-body dynamics system lie in matrix \( K^{-1} \). Therefore, without writing the inverse of matrix K explicitly, we will not be able to identify the observability of the parameters or even estimate the parameters accurately.

Because of the above two shortcomings, it is important to write matrix \( K^{-1} \) in the explicit form. Despite we would like to compute the exact explicit form of matrix \( K^{-1} \), this inverse matrix is extremely hard to compute because of the unfriendly
structure of the matrix $K$. Therefore, we focus on computing an approximate solution to $K^{-1}$.

### A.3.1 Contact Normal Impulse Vector

The contact normal impulses act in the contact normal direction to prevent penetrations between bodies. Therefore, computing the contact normal impulses is very similar to a one-dimensional case, where two objects move into contact with each other. For example, in Fig. A.1, one is moving into contact with another object at time step $t$. For the simplicity of the discussion, we assume that object 2 is fixed. The velocity of the object 1 is $v_t$ and the gap between the two objects is $\Psi$. $p_t$ is an external impulse on object 1. For the next time step $t + 1$, the contact impulse $\lambda_{t+1}$ can be computed as follows:

$$
\lambda_{t+1} = -m(v_t + \frac{p_t}{m} + \frac{\Psi}{h}),
$$

(A.15)

where $h$ is the length of the time step and $m$ is the mass of object 1$^{10}$. From the physics perspective, equation (A.15) implies that the contact normal force should equal to the negative momentum with the velocity being the penetration velocity: $v_t + \frac{p_t}{m} + \frac{\Psi}{h}$.

If we compare equation (A.15) to equation (A.13), one can immediately see the similarities. In fact, for a multi-body dynamics system, the complementarity

---

$^{10}$ $v_t$ and $p_t$ represent a negative velocity and negative impulse in Fig. A.1. Therefore, the plus sign in the equation is correct.
Figure A.2: A square is in contact with a vertical wall. The contact impulse is $p_t$. The mass of the square is $m$. Because of the angular velocity generated by the contact impulse, the velocity change at the contact is not $\frac{p_t}{m}$.

problem is indeed very similar to the above one-dimensional problem, where $H^T v_t$ computes the velocities at each contact, and $F_{p_{app}}$ computes the change of the velocities at the contacts. There are, however, two problems that need to be addressed.

The first problem is that in the one-dimensional case, the mass in front of the brackets in equation (A.15) corresponds to the whole object, but for objects in a higher dimensional system, the mass should correspond to the contacts. Notice that the “mass” at each contact does not equal to the mass of the whole object, see Fig. A.2.

The second problem is that in the one-dimensional case, there can be only one contact, but in a higher dimensional system, there can be multiple contacts at the same time step. The fact that contact impulses can affect each other can make a big difference.

A.3.1.1 Mass at Contact

In order to tackle the first problem mentioned above, we propose to compute the “mass” at each contact. we assume that the “mass” at one contact is $m_c$, the inertia matrix of the whole object is $M$, the contact’s normal Jacobian-vector is $W_n$, and the contact normal impulse is $\lambda_n$. The change of the velocity, $\Delta v$, at the
contact can be computed as follows:

\[ \Delta v = \frac{\lambda_n}{m_c} \]  
(A.16)

Meanwhile, \( \Delta v \) can also be computed as:

\[ \Delta v = W_n^T M^{-1} W_n \lambda_n \]  
(A.17)

In equation (A.17), \( W_n p_n \) computes the contact impulse in the world frame, and \( M^{-1} W_n p_n \) computes the velocity change of the object in the world frame. By comparing equations (A.16) and (A.17), we can derive the "mass" at contact as:

\[ m_c = (W_n^T M^{-1} W_n)^{-1} \]  
(A.18)

**A.3.1.2 Contact Normal Mass Matrix**

To compensate the fact that contact impulses can affect each other, we propose to compute the mass matrix for all contacts at once instead of computing a diagonal matrix with diagonal elements as the "mass" at each contact computed as shown in equation (A.18). If \( \Delta V \) is the vector of velocity changes at all contacts, \( M_c \) is the contact normal mass matrix, \( \Lambda_n \) is the vector of all contact normal impulses, we can compute the normal velocity changes of all the contacts as follows:

\[ \Delta V = M_c \Lambda_n \]  
(A.19)

Also, if \( G_n \) is the Jacobian matrix of all contacts, we can again compute \( \Delta V \) as follows,

\[ \Delta V = G_n^T M^{-1} G_n \Lambda_n \]  
(A.20)

Therefore, the contact normal mass matrix is:

\[ M_c = (G_n^T M^{-1} G_n)^{-1} \]  
(A.21)
The advantage of computing the contact normal mass matrix is that every contact contributes to the computation of the “mass” of other contacts (rows of $M_c$). Therefore, the interactions between contacts are encoded automatically in $M_c$. Finally, the contact normal impulses $\Lambda_n$ can be calculated as:

$$
\Lambda_n = -M_c(G_n^T v_t + G_n^T M^{-1} p_{app} + \frac{\Psi_n}{h})
$$

(A.22)

where $G_n^T v_t + G_n^T M^{-1} p_{app} + \Psi_n$ computes the penetration velocity of all contacts.

### A.3.2 Contact Frictional Impulse Vector

Depending on whether the contact is sliding or sticking, one should compute the frictional impulses differently. If a contact is sliding, the contact frictional impulse should be simple the contact normal impulse times the friction coefficient at the contact. On the other hand, if a contact is sticking, the contact frictional impulse should eliminate the frictional motion momentum (similar to the contact normal impulses).

#### A.3.2.1 Sliding Contact

If we assume that the matrix of friction coefficients is $U$, then the contact frictional impulse for sliding contacts can be computed as follows:

$$
\Lambda_f = U\Lambda_n
$$

(A.23)

#### A.3.2.2 Sticking Contact

For sticking contacts, the contact frictional impulses should equal to the negative momentum with velocities as the frictional velocities. Because of similar reasons with the contact normal mass matrix calculation, we compute the contact frictional mass matrix as below,

$$
M_f = (G_f^T M^{-1} G_f)^{-1}
$$

(A.24)
where \( G_f \) is the Jacobian matrix for the contact frictional directions of all contacts. Therefore, we can calculate the contact frictional impulses for sticking contacts as:

\[
\Lambda_f = -M_f(G_f v_t + G^T_f M^{-1} p_{app})
\]  
(A.25)

### A.3.2.3 Computing the Contact Frictional Impulse Vector

Finally, we introduce two matrices \( S_{sticking} \) and \( S_{sliding} \). These two matrices are diagonal selection matrices. For example, \( S_{sticking} \) will have ones at its diagonal when the rows correspond to sticking contacts and zeros at the rows that correspond to sliding contacts. Then, the contact frictional impulse can be computed as:

\[
\Lambda_f = -S_{sticking} M_f(G_f v_t + G^T_n M^{-1} p_{app}) + S_{sliding} U \Lambda_n
\]  
(A.26)

### A.3.3 Computing the Inverse of Matrix K

From the previous sections, we have computed \( \Lambda_n \) and \( \Lambda_f \) in explicit forms. Although we have considered the correlations among the contact normal/frictional impulses, the correlation between the contact normal impulses and the contact frictional impulses is still missing. We use \( V_{fn} \) and \( V_{nf} \) to represent the vectors of velocities generated by the frictional impulse that are in the normal directions, and generated by the normal impulses that are in the frictional directions, respectively. \( V_{fn} \) and \( V_{nf} \) can be calculated as:

\[
V_{fn} = G^T_n M^{-1} \Lambda_f
\]  
(A.27)

\[
V_{nf} = G^T_f M^{-1} \Lambda_n
\]  
(A.28)

Therefore, we modify the expressions of \( \Lambda_n \) and \( \Lambda_f \) as follows:

\[
\Lambda_n = -M_c(G_n^T v_t + G^T_n M^{-1} p_{app} + \frac{\Psi_n}{h}) - M_c V_{fn}
\]  
(A.29)

\[
\Lambda_f = -S_{sticking} M_f(G_f v_t + G^T_n M^{-1} p_{app}) + S_{sliding} U \Lambda_n
\]  
(A.30)

\[
- S_{sticking} M_f V_{nf}
\]
Finally, by combining equations (A.13, A.29, A.30, A.27, A.28), the explicit form of matrix $K^{-1}$ can be derived as:

$$
K^{-1} = \begin{bmatrix}
M_n & -G_n^T M^{-1} G_f S_{\text{sticking}} M_f & 0 \\
-S_{\text{sticking}} M_f G n^T M^{-1} G_n M_n + S_{\text{sliding}} M_n & S_{\text{sticking}} M_f & 0 \\
0 & 0 & 0
\end{bmatrix}
$$

Notice that we eliminated the contact normal impulse part when computing $V_{fn}$, as adding this term will introduce an oscillation.
REFERENCES


[52] V. Kumar. Optimal control with learned local models: Application to
dexterous manipulation. In Robotics and Automation (ICRA), 2016 IEEE

Theory and Applications. Control and Dynamic Systems. Elsevier Science,
Amsterdam, Netherlands, 1996.


[56] B. J. McCarragher and H. Asada. A discrete event approach to the control of
robotic assembly tasks. In 1993 Proc. IEEE Int. Conf. Robotics and


[59] N. Oliver and E. Horvitz. A comparison of hmms and dynamic bayesian
networks for recognizing office activities. In Int. Conf. User Modeling, pages

[60] V. Pavlovic and J. Rehg. Impact of dynamic model learning on classification

[61] J. Peng and R. J. Williams. Efficient learning and planning within the dyna


