# Generative and predictive models for robust manipulation


UNIVERSITY OF BIRMINGHAM

**Ermano Arruda**

School of Computer Science

University of Birmingham

A thesis submitted to The University of Birmingham for the degree of

*Doctor of Philosophy*

November 2019

# Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 50,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Ermano Arruda

November 2019

# Acknowledgements

First of all, I would like to deeply thank my supervisor Jeremy Wyatt for being a source of great inspiration and invaluable advice throughout the conduction of my PhD. His encouragement and support made this thesis possible. I will miss our weekly discussions and Jeremy's contagious energy dearly.

I am also very grateful for the helpful discussions and technical knowledge passed onto me by my former co-supervisor Marek Kopicki, from whom I have learnt greatly.

I am truly grateful to Mohan Sridharan for his advice and guidance in the final steps of my thesis, his final pushes were fundamental for the completion of my PhD.

I would also like to thank various colleagues and friends for the great collaborative support and discussions. I thank Michael Mathew as a great friend and collaborator with whom I enjoyed many discussions and joint work in the lab. Special thanks to Maxime Adjigble who helped me uncountable times providing technical advice and help on setting up the robots. I would also like to thank Saif Sidhik for his friendship and advice, his knowledge in electronics and CAD modelling helped me considerably. Akinobu Hayashi who gave me the chance to do an internship in Germany in a subject related to my PhD, our discussions were always insightful and useful.

I am deeply grateful to Diar Karim, who became the brother I never had. His personal and academic advice have been unforgettable teachings for me.

Special thanks to all the dear friends I made. I thank Tom Wyatt and Lander Ellis, my dearest friends with whom I shared so many happy moments even before my PhD. My dear

friend Cristina for the great company and friendship. My kindest thanks to Ilaria Bernardi for her musical spirit and precious friendship. I thank Giulia Covarino for her kindness and friendship, and also for teaching me Italian (grazie mille!). Gianluca Bortoletto and Jade Siu for being such caring friends. Jota Carlo, for being such a cheerful friend. Also big thanks to Afif Batal, Andrea Mazzeo and Rebecca Azpiri for our inspirational jam sessions, music has always pushed me forward even in difficult times. I thank my friends Riccardo Di Guida, Elena Renella, Christos Paterakis for making my first years such a memorable time. My deepest gratitude for everyone I met and crossed paths throughout my PhD.

Above all, I dedicate this thesis to my dear mother, father and my sister. It has been a long journey and I miss you all dearly.

# Abstract

Probabilistic modelling of manipulation skills, perception and uncertainty pose many challenges at different stages of a typical robot manipulation pipeline. This thesis is about devising algorithms and strategies for improving robustness in object manipulation skills acquired from demonstration and derived from learnt physical models in non-prehensile tasks such as pushing. Manipulation skills can be made robust in different ways: first by improving time performance for grasp synthesis, second by employing active perceptual strategies that exploit generated grasp action hypothesis to more efficiently gather task-relevant information for grasp generation, and finally via exploiting predictive uncertainty in learnt physical models. Hence, robust manipulation skills emerge from the interplay of a triad of capabilities: generative modelling for action synthesis, active perception, and finally learning and exploiting uncertainty in physical interactions. This thesis addresses these problems by

- Showing how parametric models for approximating multimodal distributions can be used as a computationally faster method for generative grasp synthesis.
- Exploiting generative methods for dexterous grasp synthesis and investigating how active vision strategies can be applied to improve grasp execution safety, success rate, and utilise fewer camera views of an object for grasp generation.
- Outlining methods to model and exploit predictive uncertainty from learnt forward models to achieve robust, uncertainty-averse non-prehensile manipulation, such as push manipulation.

In particular, the thesis: (i) presents a framework for generative grasp synthesis with applications for real-time grasp synthesis suitable for multi-fingered robot hands; (ii) describes a sensorisation method for under-actuated hands, such as the Pisa/IIT SoftHand, which allows us to deploy the aforementioned grasp synthesis framework to this type of robotic hand; (iii) provides an active vision approach for view selection that makes use of generative grasp synthesis methods to perform perceptual predictions in order to leverage grasp performance, taking into account grasp execution safety and contact information; and (iv) finally, going beyond prehensile skills, provides an approach to model and exploit predictive uncertainty from learnt physics applied to push manipulation. Experimental results are presented in simulation and on real robot platforms to validate the proposed methods.

# Table of contents

**List of figures** **xii**

**List of tables** **xxii**

**Nomenclature** **xxiii**

**1 Introduction** **1**

1.1 Motivation . . . . . 1

1.2 Contributions of this thesis . . . . . 5

1.3 Working scenarios . . . . . 7

1.3.1 Grasping scenario . . . . . 8

1.3.2 Pushing scenario . . . . . 9

1.4 Roadmap . . . . . 10

**2 Robotic manipulation skills and perception** **12**

2.1 Robotic grasping . . . . . 13

2.1.1 Analytic Approaches . . . . . 15

2.1.1.1 Limitations of analytic approaches . . . . . 18

2.1.2 Data-driven Approaches . . . . . 19

2.1.2.1 Discriminative approaches . . . . . 20

2.1.2.2 Generative approaches . . . . . 23

2.2 Active perception for manipulation . . . . . 28
2.3 Non-prehensile manipulation . . . . . 31
2.3.1 Analytic mechanical models of push manipulation . . . . . 33
2.3.2 Learning models for pushing . . . . . 34
2.3.2.1 Estimating predictive uncertainty . . . . . 36
2.4 Summary . . . . . 40

**3 Learning generative and predictive models 44**
3.1 Learning probability densities . . . . . 45
3.1.1 Maximum-likelihood estimation of a Gaussian . . . . . 46
3.1.2 Gaussian mixture models (GMMs) . . . . . 49
3.1.2.1 Expectation-maximisation (EM) algorithm . . . . . 52
3.1.3 Kernel density estimation . . . . . 53
3.1.4 Mixture density networks . . . . . 54
3.2 Gaussian processes . . . . . 57
3.3 Predictive uncertainty: a comparison on toy problems . . . . . 62
3.3.1 Modelling heteroscedasticity . . . . . 63
3.3.2 Modelling meta-uncertainty . . . . . 66
3.4 Summary . . . . . 70

**4 Fundamentals of controlling a robot manipulator 72**
4.1 Robot manipulator design . . . . . 72
4.1.1 Manipulator configuration space . . . . . 73
4.1.2 Manipulator workspace . . . . . 74
4.2 Kinematics . . . . . 75
4.2.1 Forward kinematics . . . . . 75
4.2.1.1 Manipulator Jacobian . . . . . 77

4.2.1.2 Analytical Jacobian . . . . . 79
4.2.2 Inverse kinematics . . . . . 81
4.3 Control . . . . . 83
4.3.1 Joint space control . . . . . 84
4.3.2 Workspace control . . . . . 87
4.4 Stochastic optimal control . . . . . 90
4.5 Motion synthesis . . . . . 91
4.5.1 Motion planning . . . . . 91
4.5.1.1 Sampling-based planning . . . . . 93
4.5.2 Dynamic movement primitives . . . . . 97
4.5.2.1 Learning dynamic movement primitives . . . . . 99
4.6 Summary . . . . . 101

**5 Generative grasp synthesis from demonstration using parametric mixtures 102**
5.1 Representations . . . . . 105
5.1.1 Probability density approximation using parametric mixtures . . . . 106
5.2 Grasp synthesis using parametric contact models . . . . . 109
5.2.1 The object representation . . . . . 109
5.2.2 Contact model . . . . . 109
5.2.3 Hand configuration model . . . . . 111
5.2.4 Grasp synthesis for novel objects . . . . . 112
5.2.4.1 Contact query density . . . . . 112
5.2.4.2 Grasp sampling . . . . . 112
5.2.4.3 Grasp optimisation . . . . . 113
5.3 Grasp synthesis algorithms . . . . . 115
5.3.1 Grasp offline synthesis . . . . . 116
5.3.2 Online grasp synthesis . . . . . 117

5.4 Sensorising the Pisa/IIT SoftHand . . . 119
5.4.1 Sensorisation performance . . . 122
5.4.1.1 Accuracy . . . 124
5.5 Generative grasp synthesis framework . . . 126
5.6 Experiments in simulation . . . 128
5.6.1 Performance comparison . . . 128
5.6.2 Results . . . 130
5.7 Validation on different robot platforms . . . 132
5.7.1 Validation on the Boris platform and the Pisa/IIT SoftHand . . . 132
5.7.2 Validation on the Kinova platform and the KG-3 Gripper . . . 133
5.8 Online grasp synthesis for under-actuated hands . . . 133
5.9 Conclusion and future work . . . 135

**6 Active vision for dexterous grasping 137**
6.1 View Selection . . . 138
6.1.1 Representations . . . 139
6.1.2 Contact Based View Selection . . . 142
6.1.3 Information Gain View Selection . . . 144
6.1.3.1 Information Gain Prediction . . . 147
6.1.3.2 Information Gain View Selection . . . 149
6.1.4 Safety View Planning . . . 149
6.2 Experiments . . . 151
6.2.1 Methodology . . . 152
6.2.2 Results . . . 153
6.3 Conclusions . . . 155

**7 Learning predictive models for uncertainty averse pushing 157**

7.1 Preliminaries . . . 159
7.2 Approach . . . 160
7.2.1 Forward models with predictive uncertainty . . . 161
7.2.2 Uncertainty averse model predictive path integral control . . . 163
7.2.3 An alternative approach to uncertainty averse planning . . . 167
7.3 Experiments . . . 169
7.3.1 Experiment 1 . . . 171
7.3.2 Experiment 2 . . . 171
7.3.3 Experiment 3 . . . 172
7.3.4 Experiment 4: . . . 173
7.4 Results . . . 173
7.5 Conclusion . . . 176

**8 Discussion 178**
8.1 Conclusions . . . 178
8.2 Summary . . . 181
8.3 Future work . . . 183

**References 185**

**Appendix A Groups and rigid body transformations 194**
A.1 Groups . . . 194
A.1.1 *GL(n)* . . . 194
A.1.2 *SO(3)* . . . 195
A.1.2.1 Euler angles . . . 196
A.1.2.2 Unit quaternions . . . 197
A.1.3 *SE(3)* . . . 198
A.2 Representations of pose . . . 199

# List of figures

1.1 The robust manipulation triad. . . . . 5

1.2 The Boris manipulation platform with a set of graspable objects. It consists of two Kuka LWR arms, each arm has 7-DoF. The right arm is equipped with a DLR/HIT Hand II, a five-fingered, fully actuated hand with 20 DoF. The left arm is equipped with an under-actuated Pisa/IIT SoftHand with a single degree of synergetic actuation (1 DoA), but de facto having 19 DoF consisting of passive joints that comply to grasped object geometric shape. The platform is also equipped with three RGBD sensors (Primesense Carmine 1.09), two of these sensors are mounted on each of Boris' wrist, and the third is mounted on its chest. . . . . 8

1.3 The Kinova platform is a 6-DoF arm equipped with a KG-3 under-actuated hand. The hand consists of 3-fingers, each finger has two joints (2 DoF), but only one joint in each finger is controllable (1 DoA per finger). . . . . 9

1.4 The contributed uncertainty-calibrated model and model-predictive path integral control approach for pushing has been developed and evaluated on a Baxter robot platform. It has been equipped with a custom poking end effector on its left arm. The setup also includes an external RGBD sensor (Primesense Carmine 1.09) for object tracking. . . . . 10

2.1 A typical robot manipulator equipped with a multi-fingered hand (DLR-HIT Hand II). It is also equipped with an RGB-D camera sensor mounted on the wrist, with which it is able to acquire information about the object and scene for manipulation purposes. . . . . 13

2.2 A high-level diagram depicting various modules involved in a typical manipulation pipeline for grasp synthesis: 1) *Object information* - Information about the object to be grasped can be readily available as CAD models or acquired via visual perception; 2) *Grasp generation* uses this information to produce grasp hypotheses, this generation mechanism may rely on random sampling of contact points, heuristics or data-driven methods utilising learnt direct mappings from sensory information to grasp parameters; 3) *Grasp evaluation* - Generated grasp hypotheses can be further evaluated using analytic or data-driven techniques defining grasp quality metrics; 4) *Motion synthesis* - Top-ranked grasp according to chosen criteria is selected to be executed, a motion synthesis module takes care of generating a trajectory to realise the grasp action using motion planning, trajectory optimisation, motion primitives or probabilistic techniques for motion generation; 5) *Grasp execution* - The synthesised motion is executed by the robot actuators (using closed or open-loop controllers). . . . . 16

2.3 A forward model predicts the next state of the object given its current state and a push action . . . . . 32

2.4 An inverse model predicts required push action to achieve a desired object motion represented by its current and next desired state. . . . . 32

2.5 Uncertainty sources: *epistemic uncertainty* due to lack of data, *aleatoric uncertainty* originating from intrinsic variability (noise) in the dynamics of a given process that is being modelled. *Aleatoric uncertainty* may be input-dependent (heteroscedastic) or input-independent (homoscedastic). . 38

3.1 Data points represent 272 observations regarding the waiting time between eruptions versus the duration of the eruption for the Old Faithful geyser in Yellowstone National Park, Wyoming, USA [Azzalini and Bowman, 1990]. 49

3.2 Feed-forward neural network diagram of a mixture density network with initial input $\mathbf{h} \in \mathbb{R}^c$. In general a number of $n$ hidden layers can utilised, each hidden layer containing a chosen number of neuron units. . . . . 55

3.3 Prior samples generated from the GP's prior. Each differently coloured curve in dark-blue, green and red represents a different function sampled from the GP prior. The light-blue curve depicts the true underlying function. . . . . 59

3.4 Prior, chart adapted from [Brochu et al., 2010]. The chart shows the estimates for $x^*_{1:3}$ and the corresponding confidence interval for those posterior predictions. . . . . 61

3.5 Posterior sampled functions in dark green, dark blue and red. The true function is shown in light blue. Five measurements were taken from the true function, which are shown as red dots. Note the decrease of uncertainty in the points where data was acquired, depicted by the light-green error bars. . 61

3.6 Figure depicts both ground truth function $y(h) = h$ and corresponding corruputed, heteroscedastic data sampled from Eq. 3.47. The samples were acquired such that each $h_j \sim U(-7.5, 7.5)$ is drawn from a uniform distribution in the interval $(-7.5, 7.5)$, and $y_j \sim y(h_j)$. . . . . 64

3.7 Top: mean and uncertainty predictions of GP. Bottom: mean and uncertainty predictions for the MDN. The shaded grey area corresponds to +/- 3 standard deviations from the mean predictions of each corresponding model. . . . . 65

3.8 The lesioned dataset $\mathcal{D}_m$. It represents an example in which we do not have access to a complete training dataset that covers all the domain of interest for $h$ (lack of knowledge). . . . . 66

3.9 Fitting a GP and an MDN to the lesioned dataset $\mathcal{D}_m$. Top: mean and uncertainty predictions of GP. Bottom: mean and uncertainty predictions for the MDN. The shaded grey area corresponds to +/- 3 standard deviations from the mean predictions of each corresponding model. A single MDN is over-confident with respect to its prections, although it is able to model heteroscedastic aleatoric uncertainty. . . . . 67

3.10 E-MDN diagram. . . . . 68

3.11 E-MDN predictions on the lesioned dataset $\mathcal{D}_m$. Predictions of the E-MDN now offer sensible uncertainty estimates in the regions of missing data. . . . 70

4.1 A general joint space controller. It computes joint torques $\boldsymbol{\tau}$ to actuate the joint motors such that a nominal trajectory $\{\boldsymbol{\theta}_d, \dot{\boldsymbol{\theta}}_d, \ddot{\boldsymbol{\theta}}_d\}$ is followed or tracked. It may use feedback error feedback to correct errors while attempting to track a trajectory. . . . . 84

4.2 A workspace controller is depicted as a combination of an IK solver combined with a general joint space controller. . . . . 87

5.1 An example point cloud of a mug and feature representation. For a given point **p**, the axis in blue is the surface normal pointing outwards the object, the direction of the first principal curvature $k_1$ is depicted as the horizontal red axis, and the direction of the second principal curvature $k_2$ is depicted as the green vertical axis. This right-handed frame is constructed by taking the cross-product between the normal and the first principal curvature direction. Together with the point **p**, these three axis define a rigid body pose, whose position and quaternion representation are given by $\mathbf{v} = (\mathbf{p}, \mathbf{q})$. Therefore, each point of the object has a rigid frame attached in similar fashion, although in the picture we chose to highlight only one to avoid clutter. . . . . 108

5.2 Left: contact model distribution for finger link $L_2$. This finger link has higher affinity to flat surfaces of the object. Right: contact model distribution for finger link $L_1$. This finger link has higher affinity to edge surfaces of the object. . . . . 111

5.3 A resistive flex sensor 4.5" in length. Manufacturer: Spectra Symbol. . . . . 120

5.4 Simplified kinematic diagram highlighting joints responsible for different motions. Diagram adapted from [Catalano et al., 2014]. . . . . 122

5.5 The sensorised Pisa/IIT SoftHand. Each finger flexion is sensed independently with the developed data-glove with update rate of 100Hz. . . . . 122

5.6 Flex sensor performance evaluation on a finger composed of two links, a flex sensor (R1) is placed on the finger surface so as to measure its angular position (top-left). The finger is driven around its revolute joint by a stepper motor with maximum rotation of 90 degrees (top-right). A micro-controller is utilised to convert voltage readings from the flex sensor using an ADC. The micro-controller also has access to the ground-truth commanded angle, both floating point measurements are streamed to the PC for data collection. 123

5.7 Estimated angle using the simple mapping for $\theta_{raw}$ given by Eq. 5.24 and ground-truth motor angular trajectories. . . . . 125

5.8 The sensorised Pisa/IIT SoftHand. The image shows the overlayed SoftHand model on the real robot scene as the hand is commanded from a fully opened to a fully closed posture. Each finger flexion is sensed independently with the developed data-glove with update rate of 100Hz. . . . . 126

5.9 Grasp success rate for the four experimental conditions . . . . . 129

5.10 Set of 60 test objects with varying shapes utilised for the simulated experiments. . 130

5.11 Corresponding total runtime for KDE and GMM-based methods. The total time includes the time for query density estimation, sampling of 200 grasps and optimisation of top 10 grasps using 100 iterations of simulated annealing. The results the GMM-based method is nearly 5 times faster. The GMM-based method requires fewer kernels to approximate the query density and therefore is much faster for likelihood evaluation during optimisation. For the same reason this gain in time performance is also noted during sampling and evaluation. . . . . 131

5.12 Preliminary deployment of KDE and GMM-based grasp synthesis methods on the Boris robot platform. . . . . 132

5.13 The work done has been validated using a Kinova arm equipped with a Gripper KG-3. The goal was to grasp objects on a table using the framework developed in this chapter. . . . . 133

5.14 Deployment of the approach on a Kinova platform equiped with a three-finger KG-3 gripper. . . . . 133

5.15 Diagram depicts an example sequence of the developed grasp synthesis algorithm outlined by Alg. 8. In this instance, the object is mobile and manually relocatted by a human, the grasp synthesis approach continually generates novel grasps to the current sensed object point cloud. It is completely agnostic to pose estimation approaches and converges to sensible grasps as the object is moved on the the table. . . . . 135

6.1 Grasp failure and grasp success. The top row depicts a grasp that failed without active view selection. The bottom row shows a successful grasp that used active view selection. The difference was attributed to the quality of surface reconstruction in proximity to the planned grasp contact points. . . 139

6.2 A grasp example using random view selection baseline. The grasp is successful. However, the grasp tends to be unsafe. It can noted that the grasp trajectory starts pushing the object aside with its fingers far before the final grasp closure is executed in the last picture on the right. This illustrates a common scenario in which grasps usually fail. . . . . 140

6.3 View camera poses forming the set $\Xi$, camera pose highlighted in darker purple belongs to the set of visited poses $V$. The object in the centre is circumscribed by a voxelised cube. Information gain view exploration sculpts this cube when no contacts are found. . . . . 142

6.4 (Left) Example o, showing the contact regions for different finger links highlighted with different colours. Contact-driven vision aims at viewing all these planned contact locations. . . . . 143

6.5 Figure shows a cross-section visualisation of a typical visibility check. Occupied voxels are represented in red, free voxels are shown in green and unknown voxels have dark-blue colour. Once we have defined a viewing frustum to match the real depth camera specifications, a frustum culling procedure was performed in which free voxels were assumed to be transparent, whereas unknown or occupied voxels occlude each other. In this way, only the voxels coloured on the bottom image are taken as visible voxels after the execution of this procedure. . . . . 146

6.6 The 14 objects used for trials. . . . . 152

6.7 Grasp success rate comparison. . . . . 153

6.8 Top three pictures show a failed grasp due to unexpected collision with parts of the object that are not involved in the grasp. Bottom three pictures show a successful and safe grasp selected by our approach. . . . . 154

7.1 Picture shows a high-level diagram of the approach. The forward model is trained and acquired from collected pushing data, and is able to provide uncertainty estimates for in its predictions. The system comprises the robot, the box and the environment. . . . . 159

7.2 Push sequences from two positions with MPPI. The first row shows the results for Box2D (a and b), the second row has the outcomes for E-MDN (c and d) and the final third row shows the results for GP (e and f). The red frames depict the starting and intermediate locations, while the blue frame shows the goal location. In the figures (a) and (c) the goal has the same orientation as but different position from the initial box pose, whereas in (b) and (d) the goal is orientated 90 degrees counter-clockwise with respect to the initial pose of the box. In our experiments, the MPPI strategy using E-MDN as a forward model reached the goal with fewer pushes. . . . . 169

7.3 (a) Predictions of the E-MDN. (b) Predictions of the GP. The top row shows predicted displacement in the X direction. The middle shows the displacements in the Y direction. Theta shows the change in orientation. The dots in red depict the true delta, whilist the dots in green are the predictions on the training data set. Finally, the blue dots portray the prediction on a test set created from a distribution with the same mean and covariance as the original training data set. . . . . 170

7.4 The figure shows acquired simulated data from randomly applied pushes in Box2D. The dataset is subsequently lesioned in different ways (left). The corresponding heat maps depicting the forward model predictive uncertainty in different regions of the state space are also shown (right). Starting locations are represented as green crosses, positions after pushes are depicted as red crosses. . . . . 172

7.5 Uncertainty averse pushing (Alg. 14). Start and goal locations are respectively depicted by the top and bottom grey rectangles. The black dashed boxes show sub-sampled box locations along the trajectory, which is shown in white. In simulation the box is pushed at its center of rotation (CoR), thus it will not rotate. (a) With no uncertainty penalty, the box tends to follow a straight line towards the goal. (b) When penalizing uncertainty, the box avoids it. The heuristic cost was defined such that uncertainty acts as a penalty for 150 pushes, and becomes zero afterwards (i.e. $\gamma = 0$). In this experiment we set $\gamma = 115$, $\mathbf{Q} = diag(0.5, 0.5, 0.5)$, $h = 2.0$, $\rho = 2.0$, $T = 2$, $\Delta t = 0.05$, $N = 10, L = 0$. The multi-coloured lines are forward sampled trajectories from the current box state. Finally, the circle radius depicts the current uncertainty in dynamics. . . . . 174

7.6 Uncertainty averse pushing (Alg. 15). The red dots are the way-points to be followed and are generated by Alg. 15. Alg. 14 then attempts to follow these, producing the actual trajectory (white line). Forward sampled trajectories from the current box state are shown as multi-coloured lines, and the circle radius represents current region uncertainty. . . . . 174

8.1 Modules in a robotic manipulation pipeline to which this thesis have made contributions to. Generative grasp synthesis approaches (orange) (Chapter 5) form the basis for perceptual predictions (yellow) which can be used for devising active perceptual strategies (grey) based on active vision (Chapter 6). Finally, uncertainty-calibrated learnt predictive models (green) allowed us to devise strategies to exploit the knowledge of uncertainty in manipulation action outcomes and devise uncertainty-averse pushing approaches (Chapter 7). 180

# List of tables

5.1 Sensorisation agreement intervals and ground-truth motor angular positions. 124

6.1 Trial Results . . . . . 155

7.1 The cost is a weighted average of the position error (metres) and orientation error (radians). . . . . 173

# Nomenclature

**Acronyms / Abbreviations**

| | |
|---|---|
| AML | Advanced Manipulation Learning Framework |
| CM | Centre of Mass |
| CNN | Convolutional Neural Networks |
| CNS | Central Nervous System |
| CoR | Center of Rotation |
| CWS | Grasp Wrench Space |
| DoF | Degrees of Freedom |
| GP | Gaussian Process |
| HOG | Histogram of oriented gradients |
| IK | Inverse kinematics |
| MDN | Mixture Density Network |
| MLE | Maximum-likelihood |
| ODE | Ordinary Differential Equation |

PRM Probabilistic Roadmap

RMSE Root Mean Square Error

RRT Rapid-Exploring Random Trees

SVM Support-vector Machine

VHGP Variational Heteroscedastic Gaussian Process

# Chapter 1

# Introduction

In this chapter we present the motivation of this research via an intuitive example (Section 1.1). We then proceed to outline the contributions of this thesis (Section 1.2), followed by a description of the work scenarios explored throughout the development of this research (Section 1.3). Finally, we provide a roadmap for the remainder of this thesis (Section 1.4).

## 1.1 Motivation

Bob is sitting at his desk, he wants to drink his tea. In a fraction of a second, he pictures in his head that he could try to grab it by one of the sides, provided the mug is not too hot. Is it too hot? He wonders. Which side is the handle? Slightly uncertain, his eyes almost automatically quickly take a glimpse and find that the tip of the handle is on the opposite side, towards the window in front of his desk. Perhaps best to grasp it by the handle, he almost subconsciously concludes, that will avoid spilling it over his papers on the desk. He knows he may be clumsy sometimes. He needs to push the mug, rotating and quickly poking it so as to make its handle within his reach. He applies controlled pushes and pictures a good way to pick it up. He double checks with one final quick look - no obstacles in sight. He

finally moves his arm to grasp his tea mug. He takes a calm sip while watching the trees being blown by the wind outside. It is a sunny day.

This chain of events happened to Bob in a totally effortless way , it took him less than a second to perform this combined sequence of perceptual and manipulation actions. He imagined various actions to grasp the mug. His visual perceptual system was guided by these grasp hypotheses, looking for the information to make them possible to be executed, while reducing the chances of accidents. He took into account uncertainty in his actions too, not knowing if it was hot, getting his fingers burned could lead to an expected reflex motion that he would like to avoid, he did not want to spill his drink because of that. He pictured other ways of grasping, pushed the object carefully taking that into account, perhaps touching the parts of the mug that he was certain not to be too hot. Finally, he was able to pick it up.

No doubt, drinking a good cup of tea can be a very elegantly orchestrated manipulation task by our brain. In a short amount of time, our central nervous system is able to collaborate with our visual, sensor and motor cortex, actively taking into account and processing many factors to perform a myriad of sensorimotor tasks. It is clear that robots still have much to learn from us.

Endowing robots with these same capabilities requires much more effort. In this thesis we will perform a few steps towards solving these problems. To this end, we will make contributions while exploring probabilistic modelling of manipulation skills, active perception and uncertainty for manipulation. In the process of doing so, we will tackle many challenges at different stages of a typical robot manipulation pipeline, ranging from learning such skills and employing appropriate perception for prehensile (grasping) tasks to learning motion models for non-prehensile manipulation skills under uncertainty, such as pushing.

This thesis is about devising strategies for robustifying object manipulation skills acquired from demonstration and via exploiting learnt physical models in combination with stochastic

optimal control. In particular, we shall focus on *prehensile* and *non-prehensile* object manipulation capabilities, which are instantiated in the contexts of grasping and pushing.

The abilities of imagining *what to do* and predicting *what can happen* to objects are crucial for a robot, because it can then use such hypotheses and predictions to better accomplish a manipulation task in at least two ways. First, imagining *what to do* provides multiple solutions and cues on how to actively seek for the perceptual information necessary for a given task. Second, predicting *what can happen* to an object given motor actions provides cues on how to better act, especially when taking into account uncertainty regarding action outcomes.

These ideas draw inspiration in gathered evidence from cognitive science suggesting that the central nervous system (CNS) predicts the consequences of motor actions in close connection with predictions of perceptual feedback [Miall and Wolpert, 1996; Wolpert and Flanagan, 2001]. Not only that, but our brain also engages in episodic future thinking [Atance and O'Neill, 2001], which serves for various functions such as decision making and spatial navigation. Hence, we are able to self project ourselves to situations that have not yet happened, thus helping in attaining future goals. Ultimately, perception and action are driven by underlying cognitive mechanisms that allow prediction of action consequences in the form of forward dynamic models, as well as the ability to imagine situations that help gather contextual visual and structural information to attain a desired goal. In this way, someone can imagine how to pick up a given object before effectively reaching out to grasp it. Similarly, an experienced dart thrower is able to predict the future state of the dart and modify their throwing action so as to hit the target.

From the interplay between generative and predictive models, we can endow an agent with imagination and foresight, thus enabling the crafting of active perceptual strategies and robust action execution capabilities. Indeed, vision is not a passive skill in the biological world. The human visual system, for instance, is constantly focusing and fixating on certain

points of the scene to acquire the information needed for the current task via *saccadic movements* [Findlay and Gilchrist, 2003]. We argue that an embodied agent such as a robot may likewise mimic such task directed perceptual strategies so as to assist with an underlying manipulation to be performed. Thus, active vision strategies can be employed to positively impact the performance of manipulation tasks, and in particular to positively influence contact information acquisition, grasp performance and execution safety. Finally, going beyond prehensile manipulation skills, the final challenge is how to actively exploit knowledge from learnt physical models. Concretely, we are not only interested in exploiting what is known to an agent, but also taking into account its predictive uncertainty regarding state and action outcomes. Taking into account uncertainty for planning and control is an important factor for achieving robust task execution. In this context, we shall see that it is possible to achieve robust non-prehensile manipulation by harnessing predictive uncertainty, provided with an appropriate combination of uncertainty-calibrated forward models and choice of control strategy.

Together, these components pose many challenges in all stages of a typical robot manipulation pipeline. These challenges are:

**i)** modelling of generative models for grasp synthesis,

**ii)** devising active perception strategies that make use of such generative models to estimate future perceptual measurements, and finally

**iii)** the definition of uncertainty-calibrated forward models for task execution in harmony with suitable control strategies for non-prehensile manipulation.

In this thesis, we will therefore see robust manipulation as collaborative interplay between a triad of capabilities, as depicted in Fig. 1.1. First, we will investigate how generative models for grasp synthesis play the role of a mechanism for imagination, and how we can improve such models to be computationally faster. In turn, such mechanism allows active vision to

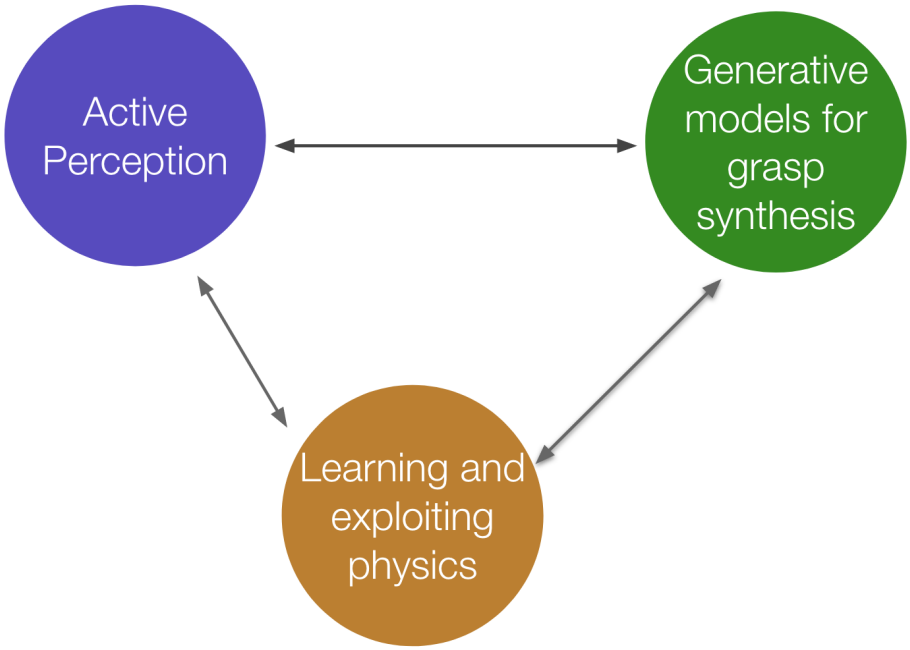


Fig. 1.1 The robust manipulation triad.

take place by reasoning over the future utility gains for achieving generated grasp hypotheses. At last, we shall explore non-prehensile manipulation, in which we will investigate how to exploit uncertainty-calibrated learnt forward models in order to achieve robust pushing.

The remainder of this introductory chapter describes the contributions made in this thesis, accompanied by a list of respective publications. We then describe the domain of testing for the results described in later chapters. Finally, we provide a roadmap describing the structure of this thesis.

## 1.2 Contributions of this thesis

Research in generative models for grasp synthesis is surprisingly limited. On top of generative techniques for grasp synthesis, active perception strategies can be devised to aid manipulation performance. In turn, research in active perception strategies for manipulation is also limited. Finally, going beyond grasping, taking into account predictive uncertainty in learnt forward

models in non-prehensile manipulation skills is still poorly explored. This research has contributed on those fronts by:

**Parametric generative models for grasp synthesis:** Presenting extensions to current grasp synthesis methods considering parametric models for learning multimodal probability densities from human demonstration. We show that parametric models for grasp synthesis are computationally faster and indicate better success rate performance with potential applications for real-time grasp generation. We make available a framework for generative grasp synthesis. These contributions will be covered by **Chapter 5**.

> **Associated publication:** E. Arruda, C. Zito, M. Sridharan, M. Kopicki, J. L. Wyatt (2019). *Generative grasp synthesis from demonstration using parametric mixtures*. Workshop on task-Informed Grasping (TIG-II): From Perception to Physical Interaction, Robotics: Science and Systems (RSS).

**Active perception for dexterous grasping:** Outlining an active vision approach for view selection in order to leverage grasp performance taking into account grasp execution safety and grasp contact information. The approach consists of a task orientated next-best-view planner that seeks to improve contact information and maximise grasp execution safety in terms of collision probability. We also describe an information theoretic view acquisition utility function when no grasps were found due to lack of object geometric information. The developed active vision method improves grasp success rate when compared to a random view selection strategy, while using fewer camera views for grasp planning when compared to a full object reconstruction. It shows comparable grasp success rate performance when compared to a full reconstruction of the object, but using fewer camera views. This work will be discussed in

**Chapter 6**.

> **Associated publication:** E. Arruda, J. L. Wyatt and M. Kopicki (2016). *Active vision for dexterous grasping of novel objects*. In proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 2881–2888.

**Harnessing predictive uncertainty for robust object pushing:** Going beyond prehensile manipulation by proposing that the execution of sensory motor policies can benefit from incorporating predictive uncertainty into the control pipeline. Hence, allowing robust manipulation to take place. We have shown that it is possible to incorporate uncertainty in learnt forward models for non-prehensile manipulation and how to integrate such models in order to obtain uncertainty averse pushing behaviour. In this work, an agent is able to robustly push an object to a target position whilst avoiding regions of high predictive uncertainty in the learnt dynamics. This work will be covered by **Chapter 7**.

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

## 1.3 Working scenarios

This section presents the robot platforms and overall infrastructure utilised to develop and test the algorithms developed in this thesis.

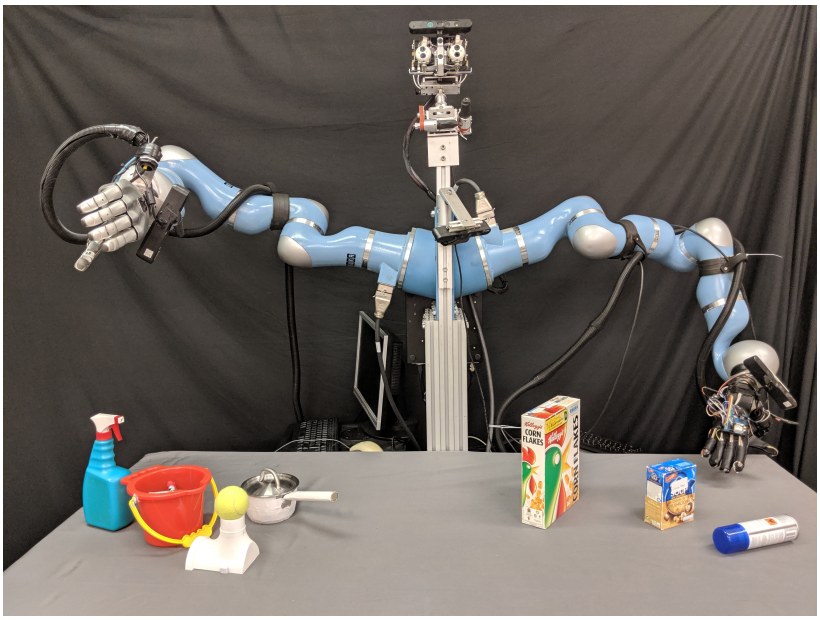

Fig. 1.2 The Boris manipulation platform with a set of graspable objects. It consists of two Kuka LWR arms, each arm has 7-DoF. The right arm is equipped with a DLR/HIT Hand II, a five-fingered, fully actuated hand with 20 DoF. The left arm is equipped with an under-actuated Pisa/IIT SoftHand with a single degree of synergetic actuation (1 DoA), but de facto having 19 DoF consisting of passive joints that comply to grasped object geometric shape. The platform is also equipped with three RGBD sensors (Primesense Carmine 1.09), two of these sensors are mounted on each of Boris' wrist, and the third is mounted on its chest.

### 1.3.1 Grasping scenario

A validation of the approaches described in **Chapter 5** were performed using the left arm of the Boris Platform and the Pisa/IIT SoftHand depicted in Fig 1.2. Validation also took place using the Kinova platorm with a 3-finger gripper KG-3 depicted in Fig 1.3.

The active vision approach for dexterous grasping discussed in **Chapter 6** has been developed and evaluated using the right arm of the Boris platform. The setup is depicted in Fig 1.2.

Fig. 1.3 The Kinova platform is a 6-DoF arm equipped with a KG-3 under-actuated hand. The hand consists of 3-fingers, each finger has two joints (2 DoF), but only one joint in each finger is controllable (1 DoA per finger).

### 1.3.2 Pushing scenario

The uncertainty-averse manipulation approach discussed in **Chapter 7** has been developed and evaluated in the task of pushing using a Baxter robot platform. The setup is depicted in Fig 1.4.

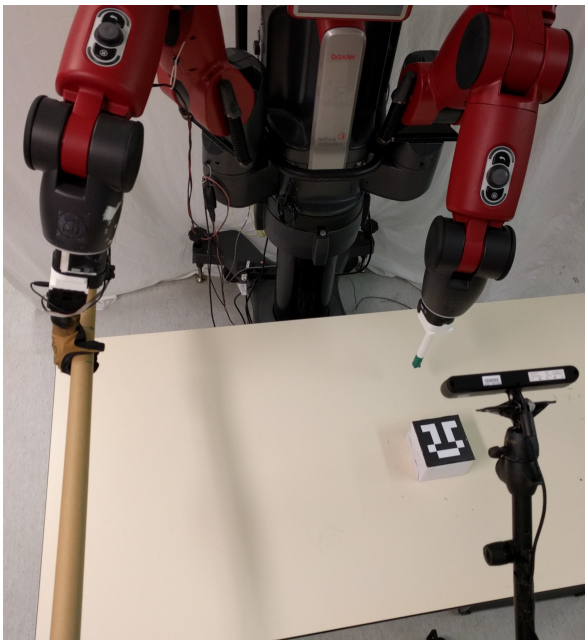

Fig. 1.4 The contributed uncertainty-calibrated model and model-predictive path integral control approach for pushing has been developed and evaluated on a Baxter robot platform. It has been equipped with a custom poking end effector on its left arm. The setup also includes an external RGBD sensor (Primesense Carmine 1.09) for object tracking.

## 1.4 Roadmap

The remaining chapters of this thesis proceed as follows:

**Chapter 2: Robotic manipulation skills and perception**

introduces the state-of-the-art and fundamentals of robotic manipulation, including prehensile and non-prehensile manipulation skills such as grasping and pushing. We also introduce the fundamentals of active perception for manipulation.

**Chapter 3: Learning generative and predictive models**

introduces basic tools and techniques for approximating probability densities and predictive models that are utilised in subsequent chapters.

**Chapter 4: Fundamentals of controlling a robot manipulator**

introduces the basic notions underpinning the design, control and motion synthesis for robot manipulators.

**Chapter 5: Generative grasp synthesis from demonstration using parametric mixtures**

describes the approach and underlying algorithms for learning parametric multimodal distributions as generative models for grasp synthesis from demonstration.

**Chapter 6: Active vision for dexterous grasping**

using generative models for grasp synthesis, such as the one described in the previous chapter, it describes an approach to tackle the problem of improving robot grasp performance using active vision. Two view selection heuristics are proposed: one that allows the robot to explore good quality grasp contact points, and another that permits the robot to investigate its workspace to make sure candidate grasp trajectories will not lead to collisions with unseen parts of the object to be grasped. Results show that this approach yields better grasp success rate when compared to a random view selection strategy, while using fewer camera views for grasp planning.

**Chapter 7: Learning predictive models for uncertainty averse pushing**

goes beyond grasping and describes an approach for learning uncertainty-calibrated forward models for non-prehensile manipulation. These models are then combined with stochastic optimal control techniques, yielding a model-predictive path integral controller to achieve robust pushing.

**Chapter 8: Conclusion**

summarises the work done in this thesis and discusses future work.

# Chapter 2

# Robotic manipulation skills and perception

In this chapter we present a literature survey on prehensile and non-prehensile manipulation skills. We focus our attention on grasp synthesis, active perception and non-prehensile manipulation skills in the context of pushing.

Section 2.1 introduces the problem of grasping and how it has been tackled from both analytical and data-driven perspectives in the literature. It also introduces a typical robotic manipulation pipeline, outlining its main high-level components divided into a perceptual module, a grasp synthesis module, and motion synthesis component for grasp execution.

Section 2.2 introduces the problem of active perception as a natural requirement arising from data-driven methods for grasp synthesis.

Section 2.3 reviews the problem of non-prehensile manipulation. In particular, it focuses on push manipulation and describes the efforts in the literature for learning forward models for pushing.

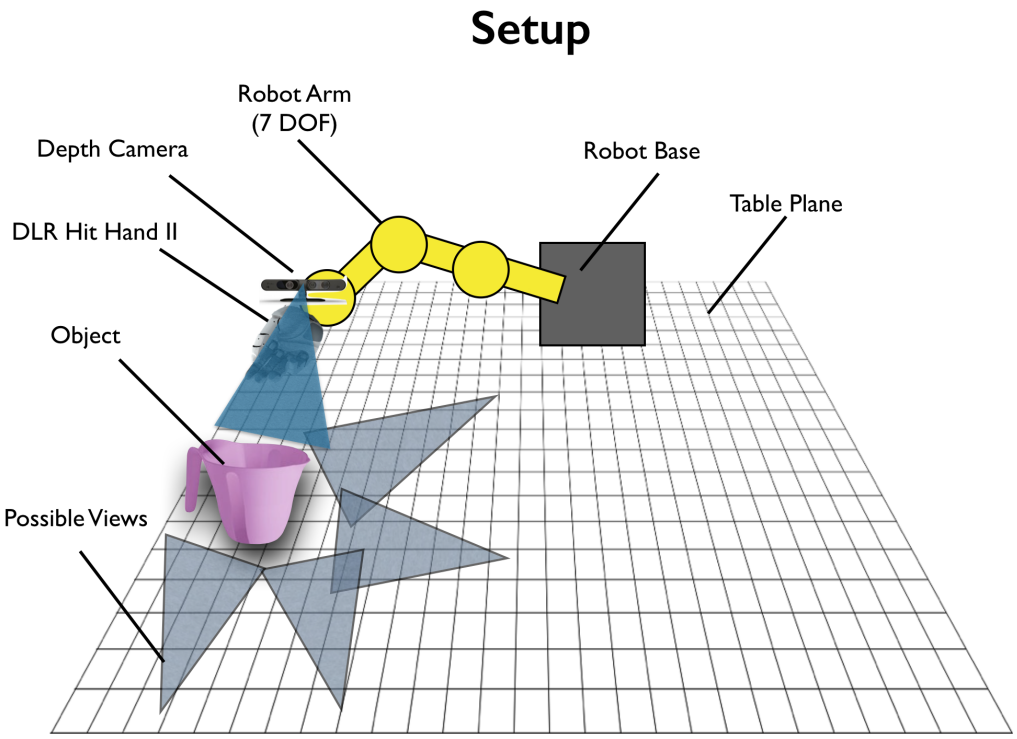


Fig. 2.1 A typical robot manipulator equipped with a multi-fingered hand (DLR-HIT Hand II). It is also equipped with an RGB-D camera sensor mounted on the wrist, with which it is able to acquire information about the object and scene for manipulation purposes.

## 2.1 Robotic grasping

Traditional robot manipulators were originally designed to work in industrial environments. They are typically composed of a robotic arm equipped with a gripper or multi-fingered hand as an end-effector. In addition, the robotic arm may also be equipped with a camera, allowing it to inspect its workspace. An illustration of this setup is shown in Fig. 2.1.

The development of multi-fingered robot hands brought forth the possibility to allow robots to perform more sophisticated manipulation tasks not only in industrial settings, but also domestic tasks, such as cooking and packaging, for instance. In principle, multi-fingered hands may be used in order to generalise across a larger variety of object shapes and sizes, as opposed to having to switch to different gripper sizes, or different types of end effectors. Human and non-human primates make extensive use of multi-fingered dexterous

hands to perform a mesmerising number of manipulation tasks. A plethora of manipulation skills are exercised using our hands ranging from grasping, assembly, in-hand manipulation, folding clothes, sewing, and general tool use, to name a few. In practice, however, even for simple grippers, the problem of grasping arbitrary objects in uncontrolled and unstructured conditions is yet to be fully solved, let alone for multi-fingered dexterous hands.

Hence, understanding how to perform dexterous grasping has been of central interest in recent decades for robotic manipulation research. It is also the subject of avid interest in cognitive sciences. Grasping an object involves the following steps:

i Acquire information about an object of interest to be grasped.

ii Find appropriate contacts on an object that are considered good for the given grasp.

iii Plan a trajectory to take the robot end-effector to a pre-grasp configuration.

iv Close the fingers around the selected contacts in order to achieve grasp closure.

Grasping requires finding the contacts on the object surface and controlling the grasping fingers to apply forces on these contact points such that the object is rendered immobilised in the robotic hand, thus it can be lifted without slipping off the robotic fingers. Algorithms that are able to compute the parameters of the hand to achieve this goal are called grasp synthesis algorithms [Shimoga, 1996].

In recent review by [Bohg et al., 2014] and also similarly to [Sahbani et al., 2012], the authors divide methodologies for grasp synthesis into analytic and data-driven or empirical approaches.

The aim of analytic approaches is to construct force-closure grasps using multi-fingered robot hands that possess certain properties such as dexterity, equilibrium, stability and

dynamical behaviour [Shimoga, 1996]. Good grasps are then ranked according to one or more of these properties. These criteria are measured by reasoning about the forces applied to contact points on the object. In turn, these applied forces generate a corresponding force and a torque with respect to the object's CM, a so called *wrench*. To compute these forces, it is necessary to take into consideration knowledge of friction, mass and also geometrical attributes (shape and size) of both the robot hand and object to be grasped.

In contrast, data-driven approaches follow a separate route. Instead of ranking grasps using analytic mechanics, data-driven methods rely on sampling grasp candidates and subsequently ranking the generated grasps based on specific quality metrics. These metrics are often learnt or heuristic in nature, but may also include classical analytic metrics. The process of grasp generation and ranking is typically based on heuristic strategies or learnt using real or simulated robot experience. A typical pipeline for grasp synthesis is depicted in Fig. 2.2. In the remainder of this chapter we will proceed to describe these two routes.

### 2.1.1 Analytic Approaches

Analytic approaches rely on force analysis (FA) and geometric constraints to generate grasps according to the laws of classical mechanics [Bicchi and Kumar, 2000; Bohg et al., 2014; Sahbani et al., 2012]. The goal is to assert that generated grasps possess properties such as dexterity, equilibrium, stability and dynamical behaviour [Shimoga, 1996]. According to [Liu, 2000; Roa and Suárez, 2015; Shimoga, 1996], these properties can be summarised as follows:

**Disturbance resistance:** the central criteria in analytic methods. If the position of the fingers around the object are able to resist any arbitrary external wrench, then the grasp is said to be in *form-closure*. Form-closure is a state of complete kinematical restraint. This is a pure kinematic property, meaning that the object is fully immobilised solely by carefully chosen positions of the fingers of the robot hand on the object. When not

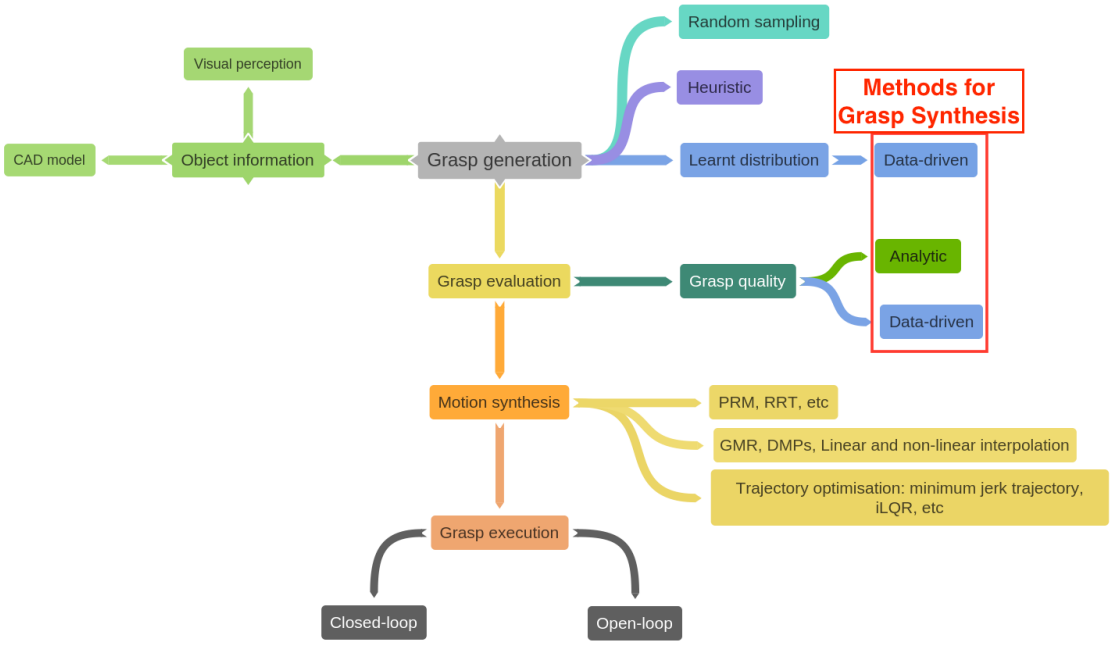


Fig. 2.2 A high-level diagram depicting various modules involved in a typical manipulation pipeline for grasp synthesis: 1) *Object information* - Information about the object to be grasped can be readily available as CAD models or acquired via visual perception; 2) *Grasp generation* uses this information to produce grasp hypotheses, this generation mechanism may rely on random sampling of contact points, heuristics or data-driven methods utilising learnt direct mappings from sensory information to grasp parameters; 3) *Grasp evaluation* - Generated grasp hypotheses can be further evaluated using analytic or data-driven techniques defining grasp quality metrics; 4) *Motion synthesis* - Top-ranked grasp according to chosen criteria is selected to be executed, a motion synthesis module takes care of generating a trajectory to realise the grasp action using motion planning, trajectory optimisation, motion primitives or probabilistic techniques for motion generation; 5) *Grasp execution* - The synthesised motion is executed by the robot actuators (using closed or open-loop controllers).

only the finger contact positions, but the forces applied on the object also render it immobile, then the grasp is said to be in *force-closure*. In force-closure grasps, the fingers are able to withstand or compensate a particular range of external wrenches whilst keeping the object immobilised.

**Dexterity:** given a specified task, a grasp is deemed dexterous when the kinematical relationship between the hand and the object permit the hand to move the object in desired directions according to the task to be performed. When a specific task is not defined, a

grasp is dexterous if the manipulator is able to move the object in any direction after a grasp takes place.

**Equilibrium:** a grasp is in equilibrium when the net force and torques applied on the object by the hand fingers is zero. This includes the forces and torques applied on the object by external disturbances.

**Stability:** a grasp is stable if it returns to equilibrium state after forces and torques from external disturbances seize to be applied.

Based on these desired properties, a variety of grasp quality metrics have been proposed [Roa and Suárez, 2015; Shimoga, 1996]. Geometric-based measures take into account internal angles and area of the grasp polygon formed by grasp contact points and the distance between the centroid of the grasp polygon and the object CM. In particular, [Ferrari and Canny, 1992; Kirkpatrick et al., 1992] introduced the notion of *grasp wrench space* (CWS), that is the space of possible externally applied forces that a grasp can resist without making the object slip off the robot hand. One possible metric quantifies the volume of the GWS [Roa and Suárez, 2015] by taking into account the product of the eigenvalues of the matrix representing the base vectors of the GWS, so called *grasp matrix*. Alternately, the size of this space can be quantified taking into account force constraints, for different choices of contact points, by finding the radius of the largest ball centred at the origin of the wrench space in the object's reference frame. In this sense, a good grasp corresponds to the grasp that can resist the largest magnitude of external perturbation. Variations of this metric have been proposed, taking into account additional force constraints [Mishra, 1995]. Other extensions consider task specific constraints, for example, when the maximum external disturbance limits are known for a given task, the so called *Task Wrench Space* can be defined and quantified in similar fashion [Borst et al., 2004; Pollard, 1996].

#### 2.1.1.1 Limitations of analytic approaches

Analytic approaches provide an elegant formulation to the problem of grasping. However, ranking grasps using force and geometric analysis suffers from many limitations. One of the main issues concerning analytic approaches is the fact that many assumptions need to be made regarding the nature of the contacts involved (hard or soft), force limits, knowledge of object shape and dynamic properties (e.g. friction coefficients, and mass). We list a few assumptions that are often made below:

**i)** The grasp is able to apply infinite forces (no force limits).

**ii)** Assumptions of rigid contacts with fixed contact area.

**iii)** Simplifying assumption of Coloumb's law of friction and exact knowledge friction coefficients (friction is also assumed to be the same for all parts of the object).

**iv)** Object geometry is precisely known, including its surface normals and pose.

**v)** Object physical properties are known in advance, e.g. mass, centre of mass.

**vi)** The object is assumed to be rigid.

**vii)** The hand fingers are assumed to be rigid.

In summary, approaches that rely on force analyses usually require precise and complete knowledge of object and hand models for estimation of *grasp wench space* and its associated metrics. This includes physical properties such as friction, mass and centre of mass. In practice, these properties are impractically hard to estimate. Indeed, Coloumb's Law of Friction provides only a very rough first approximation on how normal forces are translated into frictional forces by means of a coefficient of friction Popov [2010]. Hence, it is often the case that at least one of these assumptions is not held in a typical real world scenario.

Because of these limitations, some evidence has been shown suggesting that metrics based on analytic approaches are not strongly indicative of grasp success when evaluated in the real-world [Balasubramanian et al., 2012; Bekiroglu et al., 2011; Bohg et al., 2014; Goins et al., 2014; Kim et al., 2013]. In addition, such metrics are often computationally costly to compute several times during grasp search. Therefore, although analytic methods are very general, these drawbacks are critical challenges to be faced in real-world scenarios.

We proceed to discuss data-driven approaches, which try to address several of these limitations.

### 2.1.2 Data-driven Approaches

Because of the practical limitations of analytic approaches, researchers have proposed to let robots learn how to grasp from experience. Data-driven approaches do not provide any guarantees regarding the aforementioned criteria of dexterity, equilibrium, stability and dynamical behaviour [Shimoga, 1996]. Instead, the notion of good contacts and grasps is learnt from demonstration, trial and error, or labelled examples [Bohg et al., 2014; Levine et al., 2016]. Using experience acquired in this fashion, two broad approaches can be identified in data-driven grasp synthesis. The first is to learn a grasp evaluation function as a predictor of grasp quality. It represents a mapping from grasp parameters to either a probability of success or a real-valued quality metric. This class of methods is referred to as *discriminative approaches* [Bohg et al., 2014]. The second is to learn a direct mapping from sensory information to the parameters representing a possible grasp on a given object. We refer to this set of methods as *generative approaches*. They more generally represent a mechanism for probabilistic sampling of grasp hypothesis.

#### 2.1.2.1 Discriminative approaches

Discriminative approaches acquire a grasp evaluation function from real or simulated data. An early example of a discriminative approach for grasp synthesis has been proposed by [Saxena et al., 2008a]. In such work, a logistic regressor was learnt from labelled training data consisting of monocular images of objects with annotated patches containing grasp points. The learnt model was then used to predict good contact points for grasping from a monocular input image utilising a variety of local visual features. This was done by using the learnt discriminator as a sliding window classifier to localise good grasp contact points. The authors showed good results in a domestic setting, in which the robot was able to empty a dish washer. Very interestingly, none of the classical criteria in analytic methods were used.

Later work in this direction has been proposed in order to deal with partial shape information [Saxena et al., 2008b], and multiple contact points [Le et al., 2010], also including notions borrowed from analytic metrics such as force-closure, distance to obstacles and distance between object CM and grasping point centre. Additional work was subsequently done to deal with range data [Jiang et al., 2011].

Overall, approaches in this context were either focusing on showing the potential applications for grasping or were directly evaluated in grasping scenarios consisting of predicting good contact points. For instance, [Stark et al., 2008] identified functional affordances on objects, consisting of locations suitable for grasping. Although the approach did not constitute a full grasp synthesis algorithm, the work showed promising results in detecting graspable regions from input monocular images, thus indicating promising prospect applications for robotic grasping. From then on, the major goal in this branch of research revolved around the identification of what object features are sufficiently discriminative to infer or predict the quality of a suitable grasp configuration.

Specific to parallel grippers, the work in [ten Pas and Platt, 2015] used the notion of antipodal grasps (frictionless grasps in force-closure with diametrically opposed contact

points) to generate a large automatically labelled training set. By encoding the projection of the object point cloud inside the gripper closing plane using HOG features, the training set was used to train a support-vector machine (SVM) to classify good and bad grasps. The trained model could then be used to discriminate grasp hypotheses that were more likely to succeed and be executed on the robot, achieving high-precision grasps. This work is a good example of feature engineering for representing grasps. It does, however, require a considerably large number of grasp examples to achieve reasonable performance (over 6500 examples for 18 different objects).

Intense feature engineering using visual information became less of a problem with the recent dissemination of Deep Learning and convolutional neural networks (CNNs) [LeCun et al., 2015] in the manipulation community. However, this came at the cost of data-efficiency. Focusing on parallel grippers, deep learning for grasp detection has been first proposed by [Lenz et al., 2013] as an extension to prior work in [Saxena et al., 2008a,b]. The approach consisted of a two-stage cascade classifier, in which the robot first obtained an RGB-D image of the scene containing objects to be grasped. Using this information, a first stage small deep network was used to score potential grasps in the image. A smaller set of top-ranked candidates was selected for the second stage. Finally, the second stage used a larger deep network to rank the selected promising candidates, producing a single best-ranked grasp. More recently, [Gualtieri et al., 2016] proposed a direct extension of previous work by [ten Pas and Platt, 2015], predicting grasp success using CNNs. The approach showed good performance, with the drawback of requiring an even larger training set containing over 200k examples for a total of 55 different objects.

Utilising vast amounts of data for training discriminative models has become somewhat common practice. An even more extreme example of this is the approach proposed by Levine et al. [2016]. It cast the grasping problem as a continuous hand-eye coordination problem. Using a large scale data collection scheme over the course of two months, a training set of

over 800k grasp attempts across 14 robotic arms was acquired. The gathered training data was subsequently used to train a CNN to predict grasp success from raw input images and torque command inputs. The trained model was used to provide continuous feedback from visual perception to motor commands. At every time step torque commands were chosen so as to maximise grasp success, as predicted by the learnt model, given current visual input. The final system was able to map raw images obtained from a fixed camera to torque commands to control the manipulator for grasping. This approach showed very good results, despite sacrificing data efficiency. Similarly, other methods made use of self-supervision through trial-and-error, as in [Pinto and Gupta, 2016] or simulated experience for learning classical force-closure metrics, such as in [Mahler et al., 2017, 2016].

One shortcoming inherent to most of the aforementioned discriminative approaches is the fact that they commonly require very large amounts of data to be trained. This is due to the nature of the underlying models utilised for learning grasp evaluation functions. A second drawback is that most of the aforementioned methods are tailored for parallel-jaw grippers. We argue that active perceptual strategies for information gathering are slightly more difficult to conceive using those models, because their level of specificity to particular robotic hands, and underlying mechanisms for constructing grasps. Therefore, in this thesis we will instead focus on methods for grasp synthesis with the following properties:

**Data-efficiency:** methods for grasp synthesis that do not require large amounts of data to be learnt. Such approaches are usually made possible via *learning from demonstration* techniques.

**Generality for different robotic hands:** methods that do not make assumptions about the type of robotic hand to be utilised, thus can be applied to hands with arbitrary number of fingers.

In this thesis we compromise that extra flexibility involved that makes the conception of these methods possible may come at the expense of inferior performance when compared to

other state-of-the-art methods. However, they will provide a framework for active perception for dexterous grasping in a way that is not constrained by a particular robotic hardware configuration.

In the next subsection we will describe generative approaches for grasp synthesis. We will see that this class of approaches are more promising candidates that fulfil these properties. Hence, they allow us to form the basic building blocks to devise active perceptual strategies to positively influence grasp performance. This will be described in more details in Chapter 5.

#### 2.1.2.2 Generative approaches

As opposed to learning an evaluation function, generative approaches aim to learn a direct mapping from sensory information to grasp parameters. This mapping is generally modelled as a probability density representing the joint relationship between grasp parameters and object related features. In comparison to previously mentioned methods, relatively less work has been done on learning generative models for grasp synthesis [Bohg et al., 2014]. Generative approaches for grasp synthesis are versatile since the direct mapping from sensory information to grasp parameters can readily provide cues to a perceptual system, implicitly encoding a mechanism of attention for data-gathering. As a mechanism for grasp generation it also allows further optimisation that may also take task requirements into account.

For example, the work done by [Montesano et al., 2008] proposed a method to learn affordances by observing a human performing known manipulation actions on an object producing specific effects. The problem is posed as learning a joint distribution over the set of variables using a Bayesian network framework. The network is composed by nodes encoding object, action and effect features which are observable by the robot during the execution of a task by a human. After a total of 300 trials, the robot is able to learn the structure of the network. Using the learnt model, the robot is then able to play an imitation game, in which it tries to reproduce the same action effects as observed from a human performing different

actions on the object. These are encoded as different affordances (e.g. grasping, tap, and touch). The robot may choose to perform a very different set of motor actions compared to the human demonstrator doing the same task on the object, while achieving the same observed effects.

In [Song et al., 2011] the problem of generating a full task-specific grasp configuration for a given object has been directly tackled. The joint distribution over the set of grasp variables is learnt as a Bayesian network from annotated examples for different tasks. The final learnt model was used to generate grasp configurations conditioned on task and object features. The approach has not been evaluated on a real robot platform. In principle, the method does not require object pose estimation and was able to generate full grasp configurations in simulation.

The work done by [Detry et al., 2011] proposes a method to learn probability densities over grasp poses for a gripper relative to a given object. This learnt density models a *grasp affordance model*, and is acquired by allowing a robot to autonomously gather grasp experience via trial-and-error. As described by the authors, a grasp affordance model is a representation of the success (effect) of generated grasp solutions (action) when applied to an object (environment). The densities are approximated non-parametrically using kernel density estimation (KDE). The approach however depends on object pose estimation, which is a typically hard problem for arbitrarily shaped objects, specially under uncertain and partial reconstruction.

Applied to multi-fingered robot hands, the method proposed by Ben Amor et al. [2012] makes use of human demonstrations to learn reach-to-grasp trajectories. First, contact points for a specific grasp type on a known object are acquired from demonstration. When presented with a new object, its pose is estimated and a subsequent contact warping procedure proposed by the authors takes place. Contact warping consists of mapping known grasp contacts from a known object to a new grasp object based on geometric similarities. Second,

reach-to-grasp trajectories composed of wrist pose and finger trajectories are also acquired from demonstration. To cope with the high dimensionality of the hand configuration space, the authors propose to map hand trajectories into a lower-dimensional latent space using Principal Component Analysis (PCA). Dynamic motion primitives (DMPs) [Schaal, 2006] are learnt over this latent space. In this manner, the synergies between the different fingers which move in synchronised fashion can be modelled, yielding a more human-like motion. Finally, the problem of adapting the motion primitive to new contact points on a presented object is defined as least-square optimisation problem, which computes the final goal for the reach-to-grasp primitive. The end result is a method that is able to learn grasp representations in a data-efficient way using human demonstrations, warp contact points onto new objects, and finally optimise and execute full reach-to-grasp trajectories that are more human-like and applicable to multi-fingered hands. Nonetheless, the approach still requires the knowledge of source and target shapes for pose estimation, implicitly assuming that learnt contacts from an object are from the same or similar object categories.

More recently, generative approaches for parallel grippers using deep learning have also been proposed [Kumra and Kanan, 2017; Morrison et al., 2018; Redmon and Angelova, 2015]. The work in [Redmon and Angelova, 2015] introduces a method for direct regression from RGB-D images to grasp coordinates using CNNs. Grasps are specified by a 2D position, orientation, width and height of a grasping rectangle. The proposed method exhibited real-time performance, synthesising grasps at 13 frames per second. The proposed model was trained and evaluated based on the overlap between the generated grasps with the ground-truth examples made available by Jiang et al. [2011] (Cornell grasping dataset). A conceptually similar approach using CNNs was introduced by [Kumra and Kanan, 2017], albeit without real-time benefits. In this work, an additional model that used both RGB and RGB-D images was proposed for directly computing grasp parameters. Performance evaluation was likewise based on rectangle overlap with ground-truth examples from the Cornell grasping dataset. It

is not clear whether the reported detection accuracy in [Kumra and Kanan, 2017; Redmon and Angelova, 2015] indeed reflects success rate if grasps were to be attempted in simulation or on a real robot platform. However, the authors do expect to run experiments on robot platform as future work. Still tailored for parallel grippers, a more recent approach was proposed by [Morrison et al., 2018]. It consists of a CNN-based model that receives a depth image as input and directly outputs the parameters for an antipodal grasp: a 2D gripper orientation with respect to the object, opening width of the gripper and estimated predicted grasp quality. The method was evaluated using a real robot platform, and also showed real-time grasp synthesis capabilities. Generating grasps in real-time is particularly useful in dynamic environments, when objects may not be static, and providing sensory motor feedback to compensate control inaccuracies.

So far, with the exception of [Ben Amor et al., 2012; Montesano et al., 2008; Song et al., 2011], the discussed approaches either require large amounts of training data, or are tailored to parallel grippers.

An approach that is both data-efficient and general across different types of robotic hands was proposed by Kopicki et al. [2015]. Given a demonstrated grasp, [Kopicki et al., 2015] proposes to efficiently learn two different models in the form of probability densities approximated non-parametrically with KDEs. The first model is called a *contact model*, it represents the relationship between a finger link and nearby contact local surface features of the object. A *contact model* is learnt for every finger link of the robotic hand. Given a new object to be grasped, these models can be used to find locations on the new object surface that may be locally geometrically similar to the ones experienced during demonstration. These are locations over which finger links can be placed. The second model is a *hand configuration model*, it encodes the overall shape of the robotic hand and hence a corresponding grasp type (e.g. pinch, rim, power). Finding new grasp configurations then consists of jointly sampling from these two densities so as to find locations on a new query object where fingers are more

likely to be placed whilst assuming a suitable hand configuration, thus satisfying both learnt models. This approach has been shown to be able to generalise demonstrated grasps to novel query object point clouds, without requiring prior knowledge about object pose, shape (such as CAD mesh models) or dynamic properties such as friction coefficients, mass distribution, among others.

Although the application of KDEs in [Kopicki et al., 2015] is very appealing, one drawback of learning such non-parametric models is the computational time when evaluating the likelihood of grasp samples. If a KDE has $K$ kernels and one wishes to evaluate the likelihood of $N$ data points under the model, the time complexity for computing the data likelihood is $O(KN)$. Noting that for KDEs $K$ is typically large, in the order of hundreds. This is due to the fact that commonly a kernel is placed on every training data point. This large number of kernels makes the final time complexity dependent on the size of the training data set used to approximate a desired density. Spatial data structures such as KD-trees are typically used to speed up computation, although to a limited extent. In contrast, parametric mixtures usually need a smaller number for $K$ to effectively approximate a probability density. Once the model parameters are learnt via Expectation Maximisation (EM) Bishop and Nasrabadi [2007], the runtime for evaluating the likelihood of query data points is not dependent on the size of the data set used for training, but only on the fixed number chosen for $K$. Thus, although the complexity for data likelihood computation is the same in principle for both parametric and non-parametric mixtures; in practice, parametric mixtures are orders of magnitude faster than KDEs since $K$ is rarely greater than 10 for most scenarios.

As one of the contributions of this thesis, we will revisit the work done by [Kopicki et al., 2015] and propose a parametric method to tackle this problem. In Chapter 4, we will propose the following contributed extensions to this model:

**i)** Learning contact models using parametric formulation for faster grasp synthesis.

**ii)** Proposing an algorithm for real-time grasp synthesis using parametric formulation.

**iii)** Incorporating additional constraints for grasp synthesis.

**iv** An open-source framework for generative grasp synthesis, using this class of models.

**v** Validate the approach on different robot platforms with different robotic hands.

Subsequently, we will also use the generative approach developed by [Kopicki et al., 2015] to devise an active vision approach for dexterous grasping. This will be described in more details in Chapter 5.

In this section we have identified that generative models for grasp synthesis have been explored only to a limited extent in the literature. Some of the most recent work is either tailored to specific end-effectors, e.g. parallel grippers, or requires data-intensive methods for learning the underlying models, or both. In contrast, methods based on learning from demonstration are less data-intensive and more easily applicable to multi-fingered hands. In the first two parts of this thesis we shall build upon models based on [Kopicki et al., 2015], making them computationally faster, as we will see in Chapter 4. Later, we will design active perceptual strategies by harnessing such generative mechanisms for grasp synthesis in Chapter 5. In the next section, we proceed to introduce active perception for manipulation.

## 2.2 Active perception for manipulation

Data-driven approaches for grasp synthesis are more deeply connected to object representation and perceptual processing steps such as feature extraction, similarity metrics, object classification and pose estimation, discussed in Section 2.1.2. These natural requirements

expose a natural need for approaches that could actively gather relevant information from different modalities, such as vision and touch, to autonomously satisfy grasp constraints. One of the main goals of this thesis is to address these requirements, more specifically focusing on active vision. In this section we describe active vision, or active perception in general, in the context of grasping and manipulation.

Active vision, or more generally active perception, is defined as the study of modelling and control strategies for perception when the sensor can be moved [Bajcsy, 1988]. The inspiration behind active perception comes from the fact that, like with other animals, the human visual system is active, focusing and fixating on certain points of the scene, searching and probing the world for the information needed for the current task. In order to cope with uncertain and incomplete information, this active aspect is crucial [Bajcsy, 1988]. In fact, by actively directing the visual system, some complex classical computer vision problems such as shape from shading, shape from contour, shape from texture, structure from motion and optical flow, were shown to become simpler and well behaved, given that an agent is now able to integrate visual information over time while performing controlled motion [Aloimonos et al., 1988]. We argue that this is also the case for many other problems, particularly grasping and manipulation in general.

The field of active vision is relatively new, having grown an interest since 1987 with the studies previously cited [Aloimonos et al., 1988; Bajcsy, 1988]. In the context of manipulation, researchers such as Nunez-Varela, and Ballard and Sprague [Ballard, 1991; Nunez-Varela et al., 2012; Yi and Ballard, 2009] pushed the subject forward by reasoning about visually guided hand movements, either for helping manipulation or to provide ways for simplifying common problems in computer vision.

The work done by [Nunez-Varela et al., 2012] investigated the effects on task performance when considering different gaze allocation strategies. In such work, the problem of pick-and-place with object positional uncertainty has been considered in simulation. In order to pick

and place as many objects into containers as possible, the agent has to allocate its perceptual visual system via performing gaze fixations on objects so as to update its underlying belief regarding object locations on a table. Three gaze allocation strategies were devised and compared. The first strategy considers gaze allocation taking into account the prediction of reduction uncertainty in positional uncertainty. By not accounting task specific value, this strategy showed poor task efficiency. To mitigate this, a second strategy was proposed which took into account both the value of performing manipulation actions and predictions of reducing positional uncertainty. The performance of this second model was suboptimal, as it tended to fixate more often over locations with low positional uncertainty and high predicted task value. To address this issue, a final strategy took into account the gain in uncertainty reduction and value, as opposed to maximising absolute expected value. This final strategy represented a better trade-off between taking into account task specific gains and maintaining positional uncertainty under control for better task performance.

Also in the context of grasping and manipulation, researchers have focused on devising strategies for view selection based on recovery of the full shape of the object to be grasped [Chen et al., 2011; Krainin et al., 2011]. Active perception has also been applied to object recognition Atanasov et al. [2014], and to touch for grasping under uncertainty Sommer et al. [2014]; Zito et al. [2013]. However, for most practical manipulation purposes, full object reconstruction is either too costly or simply infeasible. It is also typically redundant, since most of the time only a limited portion of the object surface is in contact during a grasp. These practical considerations were taken into account by a number of studies. For instance, the aforementioned approach proposed by [Kopicki et al., 2015] is able to transfer previously demonstrated grasps to new objects without the need for grasp force analysis, and is able to cope with an incomplete point cloud of the target object. However, grasps still fail in many occasions due to incompleteness of object shape that leads to unexpected collisions. Additional efforts were made by [Hjelm et al., 2014], focusing on task and grasp

transferability from limited training data, i.e. demonstration and partial object point clouds. The work done by [Detry and Piater, 2013], focused on learning grasps by letting the robot autonomously explore and try grasps while at the same time being able to transfer those self-discovered grasps to novel objects. In [Saxena et al., 2008b], efforts were made towards finding stable grasps given limited visibility of object shape from cluttered scenes. The problem of shape incompleteness is dealt with by Bohg et al. [Bohg et al., 2011] by trying to fill the gap between the missing parts of the objects using symmetry assumptions.

Although there has been progress in grasping in the face of partial reconstruction and novel objects, there exists a clear need for active perception so as to decide when and to what extent to fill in the missing information. In addition, we wish to ensure robot safety, avoiding hardware damage due to unexpected collisions.

Inspired by these insights, in Chapter 6 we will propose an active vision approach applied to dexterous grasping. A view selection approach is proposed utilising notions of task utility value related to grasp contact information, acquisition of geometric knowledge about an object and safety of grasp execution based on information theoretic considerations. In order to perform utility predictions for view selection, our proposed method will rely on the generative grasp mechanism proposed by Kopicki et al. [2015].

## 2.3 Non-prehensile manipulation

We can push the subject of manipulation forward by considering non-prehensile manipulation skills. Now, we are no longer interested in caging an object with a robotic hand, but instead in what manipulation actions to use to take an object from an initial to a final desired configuration. In this sense, object pushing can be considered as an instance of unstable grasping.

Grasping an object simplifies the problem of controlling the object state. While under the grip of a robotic hand, the object is constrained to follow any arbitrary motion controlled

by the robot. In non-prehensile manipulation skills, such as pushing, an object seizes to be under direct control of the robot. The outcome of any push action emerges from an intricate relationship between the manipulator, object dynamics and contacting environment. This relationship is complicated to model analytically as it involves many unknown factors, such as friction, object mass, coefficients of restitution, mass distribution, pressure distribution and geometric shape. This leads to greater uncertainty in predictability over manipulation actions. Hence, learning models of object dynamics becomes a crucial component in order to realise a larger class of manipulation skills.

One of the contributions of this thesis is to address the problem of modelling uncertainty in manipulation and how this uncertainty can be utilised to achieve uncertainty-averse pushing, as we will discuss in Chapter 7. Modelling models for pushing is a non-trivial task. These may be *forward models* or *inverse models*:

**Forward models:** predict the motion of an object resultant from a push action (Fig. 2.3).

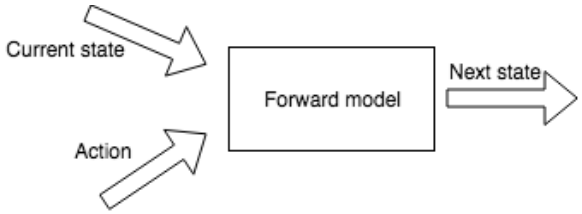


Fig. 2.3 A forward model predicts the next state of the object given its current state and a push action

**Inverse models:** predict the required push action to realise a desired motion (Fig. 2.4).

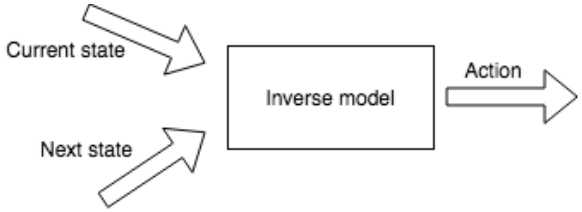


Fig. 2.4 An inverse model predicts required push action to achieve a desired object motion represented by its current and next desired state.

In this section we review relevant work for learning non-prehensile manipulation, focusing specifically on the context of pushing.

### 2.3.1 Analytic mechanical models of push manipulation

The problem of object pushing has been studied in its inception from an analytic perspective. The seminal work by Mason [1982] investigated the mechanics of pushing. Various theorems were proposed to establish the relationship between frictional forces and resultant object motion. In particular, the *voting theorem* was proposed, which yields a method to determine the direction or *sense of rotation* of an object given a push action. In simple terms, the voting theorem can be applied to determine contact points from which an object can be translated or rotated. Later, Goyal et al. [1989] introduced the concept of *limit surface* to draw relationships between frictional load and direction of sliding motion for a rigid body, provided the object support (pressure) distribution is known in advance. The limit surface defines the set of forces that are resisted by the object support friction. An applied force inside the limit surface is resisted by an opposite and equal frictional force. If the applied force lies outside the limit surface, static-equilibrium is then violated, which makes the object accelerate. This concept allows the prediction of the direction of slip motion for an applied force. Due to the large computational time for computing limit surfaces, Lee and Cutkosky [1991] proposed a method to approximate them from data using ellipsoids. The proposed approximation still assumed known pressure distribution. Lynch et al. [1992] utilised these approximations to define the equations of motion and a control system to translate and rotate objects. In Howe and Cutkosky [1996] more approximation methods for limit surfaces were further investigated. The work also provided guidelines on how to choose between the different methods depending on contact geometry, computational cost, pressure distribution and desired accuracy. Unmodelled factors such as pressure distribution and dynamic properties (e.g. friction, mass) yield great uncertainty in push predictions. Peshkin

and Sanderson [1988] proposed to address the uncertainty by sampling all possible pressure distributions, which in turn induce a variety of possible object motions. Dynamic pushing has been addressed by Jia and Erdmann [1999], although under the assumption of frictionless contacts between the pusher and the object. In contrast, Behrens [2013] studies dynamic pushing under the assumption of infinite friction between robot pusher and object.

### 2.3.2 Learning models for pushing

The analytic approaches described in the previous section require modelling and knowledge of parameters such as friction, mass, mass distribution, pressure distribution, coefficients of restitution, and centre of mass (CM) of the manipulated object, manipulator, and other objects in contact. Accurate identification of these parameters is non-trivial. Even if this is solved, approximations used in rigid body physic engines typically yield poor predictions when translated to real scenarios. These are the same shortcomings that yield various limitations for analytic grasp synthesis, as discussed in Section 2.1.1.1.

Hence, an alternative is to learn a model from data [Agrawal et al., 2016; Finn et al., 2016; Kopicki et al., 2017; Pinto and Gupta, 2017], or to use a hybrid approach [Belter et al., 2014; Zhou et al., 2017]. Learning methods divide into data-intensive and data-efficient methods.

Data-intensive methods, such as deep-learning, adopt self-supervision for data collection, allowing the creation of large datasets. In [Agrawal et al., 2016], for example, a *siamese* architecture learns both a forward and an inverse model for pushing. The forward model is used as a regulariser for the inverse model. Once trained, the inverse model can be then used later to directly predict push actions, given current and desired state of the object presented as RGB images as in the scheme shown in Fig 2.4. This work, nonetheless, assumes a discrete action space and does not take into account uncertainty in the predictions of the model.

[Finn et al., 2016] proposed to learn a video predictor model conditioned on robot motor actions. That allowed the prediction of next image frames when given an input motor

command and current image view. This forward model is used in a model predictive control schema to find push actions based on image input. The robot uses video prediction in order to "push" selected pixels from an initial location to a final location in image space. However, this model also lacks knowledge of the predictive uncertainty and has only been shown to achieve single step push manipulations.

[Pinto and Gupta, 2017] combined learnt models for pushing and grasping. The core idea, was to use cross-task training data from both pushing and grasping to improve performance of both models. The work showed that different proportions pushing and grasp data for training data-driven grasp detection models and predictive models for pushing may yield better performance. That indicates that there are features shared to both tasks that lead to better performance. Nonetheless, predictive uncertainty in the learnt models was not explored.

There are also data-efficient approaches to learning push effects [Bauzá and Rodriguez, 2017; Kopicki et al., 2017; Kopicki et al., 2011]. For instance, Kopicki et al. [2017]; Kopicki et al. [2011] proposed to learn modular kernel density regression models from empirically applied pushes. These models are combined so as to make prediction about resultant motion of an object. The work proposes to combine predictions from these models using a *product of experts* criterion Hinton [1999]. In this sense, a final prediction is the result of the product of individual motion predictor density experts. To maximise this product density signifies to satisfy all individually learnt modular constraints. The work also portrayed generalisation features, being able to make good predictions of object motion under changes of object shape. The method was shown to yield better performance than a physics simulator's predictions, however, it still lacked a direct modelling of uncertainty for the output predictions. The model has not yet been used for push planning.

#### 2.3.2.1 Estimating predictive uncertainty

In general, uncertainty in predictive models can be originated from the natural variability of the process dynamics being modelled due to noise or from the lack of knowledge about the process, due to insufficient or missing data. The former type of uncertainty is termed *aleatoric uncertainty* and cannot be reduced by collecting more data, whereas the latter type is referred to as *epistemic uncertainty* or *meta-uncertainty* which can be reduced by collecting more data [Kendall and Gal, 2017; Kiureghian and Ditlevsen, 2009]. Furthermore, the degree of variability or aleatoric uncertainty for a process may be input-dependent (heteroscedastic), or independent of input (homoscedastic). These sources of uncertainty are depicted in Fig. 2.5.

**Heteroscedasticity vs. homoscedasticity: an intuitive example**

In order to better intuitively understand these two types of aleatoric uncertainty it is useful to give an example. For instance, imagine we are trying to predict the height of trees based on trunk diameter. It is reasonable to expect that when a tree has a trunk diameter close to zero its height will also be close to zero, with very small variations among different species of trees. Conversely, the larger the radius of a tree, the more variability in height we could expect to observe on a forest. Thus, naturally, young trees with just a few centimetres of trunk diameter may differ in height by only a couple of centimetres in average. While trees with large but equal trunk diameter instead show great height differences in average, in the order of several metres. Here, we see that our predictive uncertainty rises dependent on the input variable (tree trunk diameter). Hence, we say the population of trees exhibits a heteroscedastic relationship as a function of trunk diameter. If that was not the case, then the population of trees would be homoscedastic with respect to the trunk diameter variable. In nature, however, the heteroscedastic property is what turns out to be the case. And this is also the case in many other arbitrary stochastic prediction problems, including for predictive models of object motion as a function of manipulation actions (e.g. pushing).

The ability to predict both aleatoric uncertainty and meta-uncertainty in dynamics is useful for robot planning as well [Deisenroth and Rasmussen, 2011]. The policy search method PILCO proposed by [Deisenroth and Rasmussen, 2011] made use of Gaussian Processes (GPs) [Williams et al., 2015] to model environment dynamics with estimates of predictive uncertainty. Standard GPs, however, scale poorly with the amount of training data and are not able to model the more complicated relationship of heteroscedasticity in stochastic dynamics. Representations such as Gaussian Mixture Regression scale better

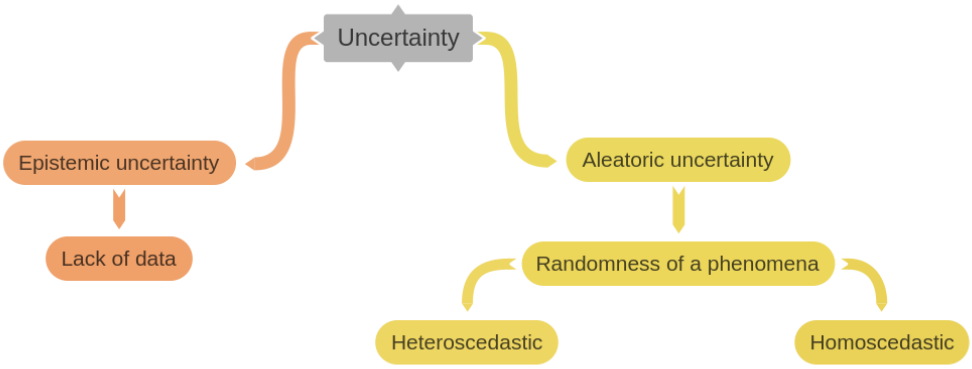


Fig. 2.5 Uncertainty sources: *epistemic uncertainty* due to lack of data, *aleatoric uncertainty* originating from intrinsic variability (noise) in the dynamics of a given process that is being modelled. *Aleatoric uncertainty* may be input-dependent (heteroscedastic) or input-independent (homoscedastic).

and can estimate dynamic uncertainty, but not meta-uncertainty [Da Silva et al., 2012]. A neural network approach to representing aleatoric uncertainty in dynamics is to learn the parameters of a mixture density. This is termed a mixture density network (MDN) [Bishop, 1994]. Neural network models can be extended to add meta-uncertainty due to a lack of data in various ways. One way is to use dropouts as shown by [Gal and Ghahramani, 2016]. Another way is to use an ensemble of MDNs and adversarial training [Lakshminarayanan et al., 2017]. We demonstrate that this latter method can be used to achieve robust pushing under uncertainty in Chapter 7.

In the context of non-prehensile manipulation, modelling uncertainty in predictions for object motion given push actions is important for various reasons. First, empirical data-driven models may be trained with limited amounts of data in limited sections of the workspace of the robot. For instance, depending on the object location it may not be kinematically possible to push an object from certain directions. Data will therefore never be acquired in those regions. Thus, we have uncertainty due to lack of data in certain regions of the state-action space. Making sure that a robot operates in regions where it's more certain about its predictions is therefore critical for safe and robust operation. Second, certain push

actions may be more noisy than others. For example, pushing faster or with more force may produce more unpredictable or uncertain motion due to irregularities of the contact surface, as well as due to dynamic limits of the robot manipulator on following trajectories for pushing under different speed or force profiles. Furthermore, different parts of the object, contacting environment and pusher may possess slightly different frictional properties. This emergent heterogeneity in practical scenarios yields great variability in motion outcomes when applying pushes from different directions on a given object. Hence, uncertainty may also be dependent on the push action, and therefore heteroscedastic. Performing more certain push actions may be more convenient and efficient. For instance, applying a push action that is certain to make an object move 5cm closer to a goal would be more desirable than a push action that may take the object closer by 10cm but with much higher uncertainty. Being risk-averse ensures attainment of a goal in a more consistent fashion. Therefore, modelling heteroscedatic effects in learnt models for pushing is an important and beneficial feature for robust pushing, and for manipulation in general.

Yu et al. [2016] has shown the existence of a wide range of variability involved in empirical data collected for pushing. He has investigated the spatial and temporal variability of friction, the latter due to polishing effects. In addition, the variability of pushing has been investigated for different geometrical shapes, applied forces, supporting surface type, initial contact and push direction. The work showed interesting insights regarding the statistical nature and complexity of push manipulation. For instance, Coloumb friction was shown not to be a good approximation for frictional effects, since the frictional forces were shown to depend on the pushing velocity magnitude (breaking Coloumb's law of friction assumption). Frictional forces were also shown to be anisotropic, as they varied depending on sliding direction. This means that resultant push actions would have different effects when applied from different sides. This insightful work exposes the intricate relationships involved in the dynamics of pushing.

Models that explicitly took into account uncertainty in predicted outcomes for pushing have been more recently investigated by [Arruda et al., 2017; Bauzá and Rodriguez, 2017].

In Bauzá and Rodriguez [2017], using dataset from Yu et al. [2016], forward models were learnt using two different regression methods. The first utilised a standard GP with input independent noise model. The second method utilised a Variational Heteroscedastic Gaussian Process (VHGP) as a model for pushing, which is able model input dependent noise. In this work, the second type of model was shown to outperform the first standard GP method in terms of prediction accuracy of push actions. However, the model was not applied to push planning on a robot platform. In addition, GPs and VHGP do not scale well with large amounts of data. The computational complexity of such models is a cubic function on the size of the data set utilised for training.

In Chapter 7 of this thesis, we will propose to utilise a method to learn a forward model for pushing that is able to scale with large amounts of training data and to yield uncertainty-calibrated predictions. Furthermore, the chosen model is able to capture more complicated statistical properties, such as heteroscedasticity when learning forward models for pushing. In addition, using a suitable choice of control framework we will also show how these predictions can be incorporated so as to achieve uncertainty-averse behaviour in manipulation. The proposed model also shows better performance when compared to standard GP regression method, reinforcing evidence preliminarily shown in related work by Bauzá and Rodriguez [2017]. Results are reported both in simulation and on a real robot platform. Chapter 7 will elaborate on the results obtained in Arruda et al. [2017].

## 2.4 Summary

This chapter presented an outline of ongoing and past efforts from the literature on prehensile and non-prehensile manipulation skills, respectively instantiated in the contexts of grasping and pushing.

Section 2.1 introduced the problem of grasping and identified various limitations concerning analytic methods for grasp synthesis. We saw that analytic approaches require precise geometric and dynamic knowledge of parameters for both objects and robotic hands, which is often impractical. Assumptions such as Coloumb's Law of friction are commonly a very coarse approximation of reality, and indeed often do not hold in practical scenarios. To address various of these shortcomings, data-driven methods for grasp synthesis proposes to let robots learn to grasp from experience. Discriminative approaches learn mappings from grasp parameters to a metric of quality or prediction of grasp success. Nonetheless, they are typically tailored to parallel-jaw grippers and require substantial amounts of training data to achieve good performance. In contrast, generative models for grasp synthesis have not yet been extensively explored in the literature. A great part of recent work on this front is again either tailored to specific end-effectors, such as parallel grippers, require data-intensive methods for learning the underlying models, or both. In contrast, methods based on learning from demonstration are less data-intensive and more easily applicable to multi-fingered hands. The most promising method in this category was proposed by Kopicki et al. [2015]. However, the underlying proposed learning methods have complexity dependent on the size of the training data.

**In Chapter 5** we shall build upon and extend models based on [Kopicki et al., 2015], making them computationally faster. The additional contributions are also proposed: **i**) an open-source framework for task-orientated grasp synthesis, **ii**) a novel algorithm for real-time grasp synthesis.

Furthermore, in Section 2.2 we have seen that comparatively less work has been done to address the problem of actively harnessing visual feedback for grasping. Data-driven approaches are more naturally entwined with object representations and perceptual processing. More specifically, generative methods for grasp synthesis provide the basic building blocks

for devising strategies for active perception, as the process of generating grasps implicitly encodes *affordances* and provides cues for active perception.

**In Chapter 6** this gap is addressed by considering active perception as a natural requirement from data-drive generative grasp synthesis. We shall design active perceptual strategies by harnessing such generative mechanisms for grasp synthesis. The main goals are to positively influence grasp performance and to ensure robot safety, avoiding hardware damage due to unexpected collisions.

Finally, Section 2.3 provided an overview on push manipulation. We have discussed the limitations of analytic techniques deriving forward models which renders them impractical, except for the simplest of controlled scenarios. The limitations arise from the lack of knowledge about physical quantities such as pressure distribution, friction coefficients, and mass, among others. Hence, once again, data-driven methods for learning forward models for pushing have become more widespread in the research community. We have seen evidence of how uncertain predicting push actions can be, and described different sources of uncertainty: meta-uncertainty due lack of data, and aleatoric uncertainty due to intrinsic variability of push dynamics itself. Nonetheless, very few approaches make explicit consideration regarding variability or uncertainty of push action outcomes. Even fewer approaches make explicit use of such knowledge coupled with a suitable strategy to fully specify a push controller or push planning system that harnesses uncertainty for robust push manipulation.

**In Chapter 7**, we shall go beyond prehensile skills and address precisely these issues. To this end, we will describe an approach for learning uncertainty-calibrated forward models for non-prehensile manipulation. These models can then be combined with stochastic optimal control techniques, yielding a model-predictive path integral controller to achieve robust pushing.

We provide a basic overview on various basic concepts that shall assist with the understanding of the later chapters in this thesis. To this end, **in Chapter 3** we proceed to introduce

the basic mathematical tools, representations and techniques for learning probability densities and predictive models with calibrated uncertainty. These techniques form the basis for the techniques and approaches described in subsequent chapters. Finally, in order to cover complementary concepts surrounding robotic manipulation, **in Chapter 4** we introduce the fundamental background for design, control and motion synthesis for robot manipulators.

# Chapter 3

# Learning generative and predictive models

In this chapter we introduce some basic techniques for approximating multimodal probability densities, either for the purpose of generation of samples that resemble the same properties as the original generation mechanism of the collected data, or for the purposes of prediction and regression. For the particular case of predictive models, we focus on models that are able to provide estimates of uncertainty regarding output predictions.

We will start this chapter by providing a brief overview over techniques for modelling multimodal probability densities. We will then discuss the distinction between generative and discriminative models. The former models full joint distribution over the full set of random variables originated from an underlying (possibly unknown) data generation mechanism. The latter are represented by classification and regression problems widely used prediction, we will see that they model a conditional distribution over at least one input variable. We will then see how to incorporate uncertainty in predictive models.

## 3.1 Learning probability densities

In this section we consider the problem of approximating probability densities from empirically acquired data. Under the assumption that a data set of independently and identically distributed (i.i.d) data points have been collected in the form

$$\mathcal{D} = \{\mathbf{x_j} | \mathbf{x_j} \in \mathbb{R}^d\}_{j=1}^N \tag{3.1}$$

where, $N$ is the size of the collected training data set, $d$ is number of dimensions of each data point, in which $\mathbf{x}_j \sim p(\mathbf{x})$ and $p(\mathbf{x})$ is the true underlying distribution that originated the data.

We will be able to derive ways of approximating the original mechanism that has generated the data. This generation mechanism is a probability density, which once learnt will enable us to generate new samples that follow the same original distribution.

It is often the case that a data point $\mathbf{x}_j$ is a vector that can be subdivided in two parts, one input $\mathbf{h}_j$ and an output $\mathbf{y}_j$ which we will be trying to predict as a function of $\mathbf{h}_j$. Thus, $\mathbf{x}_j = (\mathbf{y}_j, \mathbf{h}_j) \in \mathbb{R}^d$ is seen as being composed of two sub-vectors, where $\mathbf{y}_j \in \mathbb{R}^p$ and $\mathbf{h}_j \in \mathbb{R}^q$. Hence, we make the following distinctions:

*Generative models* represent the complete joint distribution of the data $\mathcal{D}$, such that $\mathbf{x}_j \sim p(\mathbf{y}, \mathbf{h})$. This model is useful for generating data resembling to the original distribution.

*Discriminative models* represent the conditional model of the data, i.e. $p(\mathbf{y}|\mathbf{h}_j)$. In some scenarios, it may be only necessary to model the conditional relationship between $\mathbf{y}_j$ and $\mathbf{h}_j$, such that $\mathbf{y}_j \sim p(\mathbf{y}|\mathbf{h}_j)$. This is typically the case when predictions need to be made conditioned on an input variable $\mathbf{h}_j$, such as in regression and classification problems.

A probability density can be approximated either by assuming that it can be described as a function with a specific structure, defined by a set of parameters say $\Theta$, or by allowing the the model to be free to assume any form according to the data. The former class of

methods are called *parametric models*, while the latter are called *non-parametric models*. We proceed to describe techniques for approximating probability densities using parametric and non-parametric models. These techniques will be used later in Chapter 4 for defining generative models for grasping. Finally, we also explore discriminative models that shall be used for prediction in Chapter 6.

### 3.1.1 Maximum-likelihood estimation of a Gaussian

Consider a collection of $d$-dimensional data points $\mathbf{x}_j$ that have been independently drawn from an unknown probability distribution $p(\mathbf{x})$. The collection of samples forms a data set $\mathcal{D} = \{\mathbf{x_j}|\mathbf{x_j} \in \mathbb{R}^d\}_{j=1}^N$, where $\mathbf{x_j} \sim p(\mathbf{x})$. It is convenient to assume that the probability distribution may have some known analytic equation (it can be written in closed-form), but unknown parameters $\Theta$. In this sense, we would like to approximate $p(\mathbf{x})$ by assuming it has a parametric form such that $p(\mathbf{x}|\Theta) \approx p(\mathbf{x})$. For instance, consider a common scenario in which the collected data corresponds to samples drawn from a multivariate Gaussian distribution with unknown parameters given by its mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. The Gaussian distribution is a good choice of distribution for the following reasons:

First, following from the *Central Limit Theorem*, the average of $N$ independent and identically distributed (i.i.d.) random variables in $\mathcal{D}$ approaches a Gaussian distribution as $N$ tends to infinity [Hajek, 2015].

Second, the Gaussian distribution is appealing due its simplicity and small number of parameters. It requires only two parameters to be fully specified, its mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$.

Hence, $p(\mathbf{x}|\Theta)$ can be written as:

$$p(\mathbf{x}|\Theta) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma}|}}, \tag{3.2}$$

where $\Theta = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$. We would like to find the set parameters $\Theta$ that best explain the collected data $\mathcal{D}$. This optimisation problem is stated as follows:

$$\overset{*}{\Theta} = \arg\max_{x} p(\Theta|\mathcal{D}). \tag{3.3}$$

Using Bayes rule, this statement can be rewritten as:

$$p(\Theta|\mathcal{D}) = \eta\, p(\mathcal{D}|\Theta)p(\Theta), \tag{3.4}$$

$$p(\Theta|\mathcal{D}) \propto p(\mathcal{D}|\Theta)p(\Theta), \tag{3.5}$$

where $\eta$ is a normalizing constant, $p(\mathcal{D}|\Theta)$ is the likelihood of the data given a particular choice of parameters $\Theta$. It tells how likely it is to make a measurement or to obtain the collection of samples $\mathcal{D}$ from a probability distribution parametrised by $\Theta$. The term $p(\Theta)$ is the probability distribution that represents our prior knowledge about $\Theta$. Since $\eta$ is a positive constant, it does not influence the maximization in Eq. 3.3. Thus, Eq. 3.4 can be rewritten as Eq. 3.5. Finally, under the assumption that we have no prior knowledge about how $\Theta$ is distributed, that is then equivalent to say $p(\Theta)$ follows a uniform distribution. Thus, Eq. 3.5 can be simplified to

$$p(\Theta|\mathcal{D}) \propto p(\mathcal{D}|\Theta). \tag{3.6}$$

This reduces the problem in Eq. 3.3 to finding the parameters $\Theta$ that yield maximum-likelihood according to our model $p(\mathcal{D}|\Theta)$ as below:

$$\overset{*}{\Theta} = \arg\max_{\Theta} p(\mathcal{D}|\Theta). \tag{3.7}$$

Because we assumed that the data points are independently drawn from the generating distribution, the likelihood of having collected the dataset $\mathcal{D}$ can be written as

$$p(\mathcal{D}|\Theta) = \prod_{j=1}^{N} p(\mathbf{x}_j|\Theta). \tag{3.8}$$

We can take the logarithm of Eq. 3.8, which allows the definition of the following optimisation criterium, so called the log-likelihood of the data:

$$\mathcal{L}(\Theta) = \sum_{j=1}^{N} \log p(\mathbf{x}_j|\Theta). \tag{3.9}$$

Note that the data $\mathcal{D}$ is assumed to be fixed, and thus the log-likelihood function $\mathcal{L}(\Theta)$ is only dependent on $\Theta$.

Recall that every data sample is Gaussian distributed according to Eq. 3.2. Thus, the log-likelihood is concretely expressed by

$$\mathcal{L}(\Theta) = -\frac{N}{2} log(|\mathbf{\Sigma}|) - \frac{Nd}{2}\log(2\pi) - \frac{1}{2}\sum_{j=1}^{N}(\mathbf{x}_j - \boldsymbol{\mu})^{\mathrm{T}}\mathbf{\Sigma}^{-1}(\mathbf{x}_j - \boldsymbol{\mu}) \tag{3.10}$$

In the first term of Eq. 3.10, considering that determinant of an inverse matrix is the inverse of the determinant, we can make all terms expressed in terms of $|\mathbf{\Sigma}^{-1}|$. This will be useful when taking partial derivatives with respect to $|\mathbf{\Sigma}^{-1}|$ in later steps.

$$\mathcal{L}(\Theta) = \frac{N}{2} log(|\mathbf{\Sigma}^{-1}|) - \frac{Nd}{2}\log(2\pi) - \frac{1}{2}\sum_{j=1}^{N}(\mathbf{x}_j - \boldsymbol{\mu})^{\mathrm{T}}\mathbf{\Sigma}^{-1}(\mathbf{x}_j - \boldsymbol{\mu}) \tag{3.11}$$

The final maximum-likelihood estimates for the parameters $\boldsymbol{\mu}$ and $\mathbf{\Sigma}$ are obtained by locating the extrema in Eq.3.11. This is done by taking the partial derivatives of Eq. 3.11 with respect to $\boldsymbol{\mu}$ and $\mathbf{\Sigma}^{-1}$ and equating them to zero:

$$\begin{aligned} \frac{\partial \mathcal{L}(\Theta)}{\partial \boldsymbol{\mu}} &= -\mathbf{\Sigma}^{-1}\sum_{j=1}^{N}\mathbf{x}_j + N\mathbf{\Sigma}^{-1}\boldsymbol{\mu} &= 0 \\ \frac{\partial \mathcal{L}(\Theta)}{\partial \mathbf{\Sigma}^{-1}} &= \frac{N}{2}\mathbf{\Sigma} - \frac{1}{2}\sum_{j=1}^{N}(\mathbf{x}_j - \boldsymbol{\mu})(\mathbf{x}_j - \boldsymbol{\mu})^{\mathrm{T}} &= 0 \end{aligned} \tag{3.12}$$

Rearranging and simplifying the terms yields the maximum likelihood estimates for $\boldsymbol{\mu}$ and $\mathbf{\Sigma}$ which are given by:

$$\begin{aligned} \overset{*}{\boldsymbol{\mu}} &= \frac{1}{N}\sum_{j=1}^{N}\mathbf{x}_j \\ \overset{*}{\boldsymbol{\Sigma}} &= \frac{1}{N}\sum_{j=1}^{N}(\mathbf{x}_j - \overset{*}{\boldsymbol{\mu}})(\mathbf{x}_j - \overset{*}{\boldsymbol{\mu}})^{\mathrm{T}} \end{aligned} \tag{3.13}$$

### 3.1.2 Gaussian mixture models (GMMs)

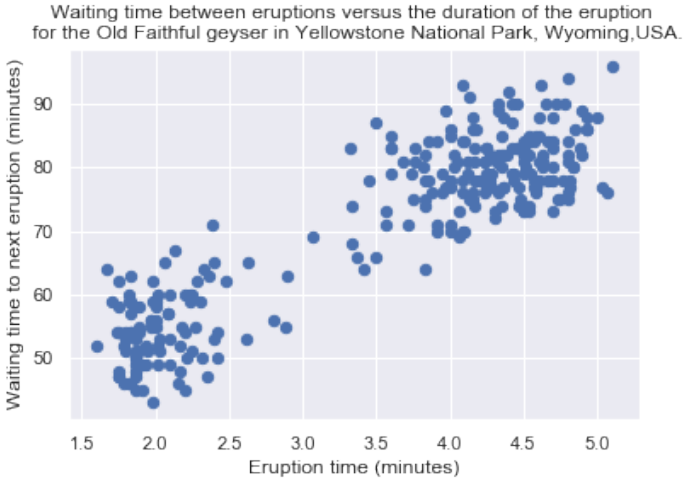


Fig. 3.1 Data points represent 272 observations regarding the waiting time between eruptions versus the duration of the eruption for the Old Faithful geyser in Yellowstone National Park, Wyoming, USA [Azzalini and Bowman, 1990].

The MLE method is a general principle used to estimate optimal parameters for a given model. In the previous section, the model consisted of a single multivariate Gaussian distribution. Nonetheless, assuming the mechanism generating the data can be described by a single Gaussian may be an oversimplification and somewhat limiting. Indeed, we would like to approximate any probability distribution. This can be done by combining multiple Gaussians in a so called mixture distribution expressed as a convex sum of Gaussians densities:

$$p(\mathbf{x}_j|\Theta) = \sum_{k=1}^{K}\pi_k\mathcal{N}(\mathbf{x}_j|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k) \tag{3.14}$$

A Gaussian mixture model (GMM) is able to approximate arbitrary probability densities to an arbitrary degree of accuracy depending solely on the number $K$ of Gaussian components

in the mixture [Bishop, 1994; Bishop and Nasrabadi, 2007]. This mixture is fully specified by $K$ mixing coeficients $\pi_k$, mean vectors $\boldsymbol{\mu}_k$ and covariance matrices $\boldsymbol{\Sigma}_k$. These components form the set of parameters $\Theta = \{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^K$, where $\sum_{k=1}^K \pi_k = 1$, $\boldsymbol{\mu}_k \in \mathbb{R}^d$ and $\boldsymbol{\Sigma}_k \in \mathbb{R}^{d\times d}$ is a positive semi-definite covariance matrix.

Consider again a data set $\mathcal{D}$ of i.i.d samples acquired from some unknown distribution as shown in Fig. 3.1. We would like to find the parameters $\Theta$ for the mixture that best explain the data. In MLE sense, the parameters $\Theta$ can be found as the solution of Eq. 3.9. However this time $p(\mathbf{x}_j|\Theta)$ is given by Eq. 3.14. In addition, a *Lagrange multiplier* is introduced to account for the constraint $\sum_{k=1}^K \pi_k = 1$. We must then maximise the following quantity:

$$\mathcal{L}(\Theta) = \sum_j^N \log\left\{\sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_j|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\right\} + \lambda\left(\sum_{k=1}^K \pi_k - 1\right). \tag{3.15}$$

Maximising Eq. 3.15 expresses itself as a more complex problem to solve in comparison to the single Gaussian example in previous section. The difficulty comes from the summation that appears inside the logarithm. A general algorithm for finding maximum-likelihood parameters for this class of problems has been proposed by Dempster et al. [1977].

We shall proceed to take the partial derivative of Eq. 3.15 with respect to $\boldsymbol{\mu}_k$ and equate it to zero:

$$\begin{aligned}\frac{\partial \mathcal{L}(\Theta)}{\partial \boldsymbol{\mu}_k} &= -\sum_j^N \overbrace{\frac{\pi_k \mathcal{N}(\mathbf{x}_j|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_j|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}}^{\gamma_{jk}} \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_k - \mathbf{x}_j) \\ &= -\sum_j^N \gamma_{jk} \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_k - \mathbf{x}_j)\end{aligned} \tag{3.16}$$

where, the term $\gamma_{jk}$ represents the responsibility assigned to a component $k$ for "explaining" the observation $\mathbf{x}_j$ [Bishop and Nasrabadi, 2007]. Defining $N_k = \sum_j^N \gamma_{jk}$, rearranging the terms and multiplying by $\boldsymbol{\Sigma}_k$, we obtain the expression for $\boldsymbol{\mu}_k$ as

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_j^N \gamma_{jk} \mathbf{x}_j. \tag{3.17}$$

Similarly, taking $\frac{\partial \mathcal{L}(\Theta)}{\partial \boldsymbol{\Sigma}_k^{-1}} = 0$, the partial derivative of Eq. 3.15 with respect to $\boldsymbol{\Sigma}_k^{-1}$, setting it to zero and simplifying yields:

$$\boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_j^N \gamma_{jk} (\mathbf{x}_j - \boldsymbol{\mu})(\mathbf{x}_j - \boldsymbol{\mu})^{\mathrm{T}} \tag{3.18}$$

Finally taking $\frac{\partial \mathcal{L}(\Theta)}{\partial \pi_k} = 0$, yields:

$$\frac{\partial \mathcal{L}(\Theta)}{\partial \pi_k} = \sum_j^N \frac{1}{\pi_k} \frac{\mathcal{N}(\mathbf{x}_j | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_j | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)} + \lambda = 0 \tag{3.19}$$

Multiplying both sides by $\pi_k$ and rearranging gives

$$\pi_k = -\frac{1}{\lambda} \sum_j^N \frac{\pi_k \mathcal{N}(\mathbf{x}_j | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_j | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)} \tag{3.20}$$

Using the fact that $\sum_{k=1}^K \pi_k = 1$ allows us to sum over $k$ on both sides:

$$\sum_k^K \pi_k = -\frac{1}{\lambda} \sum_j^N \underbrace{\left\{ \sum_{k=1}^K \frac{\pi_k \mathcal{N}(\mathbf{x}_j | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_j | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)} \right\}}_{1} = 1 \tag{3.21}$$

$$-\frac{1}{\lambda} \underbrace{\sum_j^N 1}_{N} = 1 \tag{3.22}$$

$$\therefore \lambda = -N \tag{3.23}$$

Notice that using commutation, the summation over $k$ could be moved inside the summation over $N$ which allowed us to conclude $\lambda = -N$.

Plugging this result back into Eq. 3.20:

$$\pi_k = \frac{1}{N} \sum_j^N \frac{\pi_k \mathcal{N}(\mathbf{x}_j | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_j | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)} \tag{3.24}$$

$$\therefore \pi_k = \frac{1}{N} \sum_{j=1}^N \gamma_{jk} \tag{3.25}$$

#### 3.1.2.1 Expectation-maximisation (EM) algorithm

We have now derived the basic equations that will allow us to describe the EM algorithm for learning the parameters of a GMM. The algorithm consists of a two-step iterative process.

**Algorithm 1** The EM algorithm originally proposed by [Dempster et al., 1977] applied to find the maximum-likelihood estimate for the parameters of a Gaussian mixture model. This pseudocode is based on the basic steps described in [Bishop and Nasrabadi, 2007].

**Precondition:** $\mathcal{D}$ (Collected data), $K$ (Number of components)

1: Initialise means $\boldsymbol{\mu}_k$, covariances $\boldsymbol{\Sigma}_k$ and mixing coeficients $\pi_k$
2: $\Theta^{new} \leftarrow \{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^{K}$
3: $\Theta^{old} \leftarrow \Theta^{new}$

4: $\gamma_{jk}(\Theta^{old}) \leftarrow \frac{\pi_k^{old}\mathcal{N}(\mathbf{x}_j|\boldsymbol{\mu}_k^{old},\boldsymbol{\Sigma}_k^{old})}{\sum_{k=1}^{K}\pi_k^{old}\mathcal{N}(\mathbf{x}_j|\boldsymbol{\mu}_k^{old},\boldsymbol{\Sigma}_k^{old})}$
5: $N_k \leftarrow \sum_{j=1}^{N}\gamma_{jk}(\Theta^{old})$
▷ **E-step**

6: $\boldsymbol{\mu}^{new} \leftarrow \frac{1}{N_k}\sum_{j=1}^{N}\gamma_{jk}(\Theta^{old})\mathbf{x}_j$ ▷ Eq. 3.17
7: $\boldsymbol{\Sigma}^{new} \leftarrow \frac{1}{N_k}\sum_{j=1}^{N}\gamma_{jk}(\Theta^{old})(\mathbf{x}_j-\boldsymbol{\mu}^{new})(\mathbf{x}_j-\boldsymbol{\mu}^{new})^{\mathrm{T}}$ ▷ Eq. 3.18
8: $\pi_k^{new} \leftarrow \frac{N_k}{N}$ ▷ Eq. 3.24
▷ **M-step**

9: Evaluate log likelihood $\mathcal{L}(\Theta) = \sum_j^N \log\left\{\sum_{k=1}^{K}\pi_k\mathcal{N}(\mathbf{x}_j|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)\right\}$
10: If stopping criteria is not satisfied, return to step 3. ▷ **Stopping criteria**

11: **return** $\Theta^{new}$ ▷ Estimated GMM parameters

The first step is the expectation or E-step. It calculates the expected responsibilities $\gamma_{jk}$ using the old parameters $\Theta^{old} = \{\pi_k^{old}, \boldsymbol{\mu}_k^{old}, \boldsymbol{\Sigma}_k^{old}\}_{k=1}^{K}$.

The second step is the maximisation or M-step. It uses the responsibilities $\gamma_{jk}$ to find the new maximum-likelihood estimates for the parameters $\Theta^{new} = \{\pi_k^{new}, \boldsymbol{\mu}_k^{new}, \boldsymbol{\Sigma}_k^{new}\}_{k=1}^K$ as derived in Section 3.1.2.

This two steps repeat until a stopping condition is met. The criteria for stopping are met when the data likelihood or the parameters converge, i.e. stop changing with respect to previous iteration, or a maximum number iterations is reached. A pseudocode of this algorithm is given by Alg. 1.

### 3.1.3 Kernel density estimation

The previous section described a method for approximating probability densities using parametric mixtures, such as GMMs. An alternative approach is to consider a non-parametric mixture. Kernel density estimation (KDE) techniques consider every data point as the center of a kernel function. Each data point now becomes a component in the mixture. Thus, a data set $\mathcal{D}$ with $N$ data points $\mathbf{x}_j \in \mathbb{R}^d$ is directly used to approximate its unknown generating density as

$$p(\mathbf{x}) = \sum_{j=1}^{N} w_j \mathcal{K}_{\mathbf{x}}(\mathbf{x}|\mathbf{x_j}) \tag{3.26}$$

where, $\sum_{j=1}^{N} w_j = 1$. A simple choice for $w_j$ is to make each kernel equally important in the mixture, i.e. $w_j = \frac{1}{N}$.

A common choice of kernel is the Gaussian [Bishop and Nasrabadi, 2007; Scott and Sain, 2004].

$$\mathcal{K}_{\mathbf{x}}(\mathbf{x}|\mathbf{x_j}) = \frac{\exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{x_j})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\mathbf{x_j})\right)}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma}|}} \tag{3.27}$$

where $\boldsymbol{\Sigma}$ is a $d \times d$ covariance matrix. For simplicity we can assume that $\boldsymbol{\Sigma}$ is diagonal and spherical, i.e. $\boldsymbol{\Sigma} = \sigma_x^2 \boldsymbol{I}$, where $\boldsymbol{I}$ is the identity matrix. Thus, Eq. 3.27 can be re-written as:

$$\mathcal{K}_x(\mathbf{x}|\mathbf{x_j}) = \frac{1}{(2\pi)^{d/2}\sigma_x^d} \exp\left(-\frac{\|\mathbf{x}-\mathbf{x_j}\|}{2\sigma_x^2}\right) \tag{3.28}$$

With this kernel function, Eq. 3.26 can thus be expressed as

$$p(\mathbf{x}) = C_{norm}(\sigma_x) \sum_{j=1}^{N} \exp\left(-\frac{\|\mathbf{x}-\mathbf{x_j}\|}{2\sigma_x^2}\right) \tag{3.29}$$

where $C_{norm}(\sigma_x) = [N(2\pi)^{d/2}\sigma_x^d]^{-1}$ is a normalisation constant, and $\sigma$ is the kernel bandwidth. The choice of bandwdith may be empirical or follow a "rule-of-thumb" estimation procedure [Scott and Sain, 2004].

Therefore, this density model is constructed by placing a Gaussian in every data point and subsequently dividing by the appropriate normalising $C_{norm}(\sigma_x)$ over all $N$ data points.

Other choices of kernel functions are possible depending on the application. For instance, over the $SO(3)$ space, the space of orientations $\mathbf{q}$ in 3D space, one may achieve better results using a pair of antipodal von Mises-Fisher distributions as kernel functions, forming a Gaussian-like distribution on $SO(3)$ (see [Detry, 2010; Fisher, 1953; Sudderth, 2006]. This kernel is given given by

$$\mathcal{K}_q(\mathbf{q}|\boldsymbol{\mu}_q) = C_4(\boldsymbol{\sigma}_q)\frac{\exp(\sigma_q\boldsymbol{\mu}_q^T\mathbf{q}) + \exp(-\sigma_q\boldsymbol{\mu}_q^T\mathbf{q})}{2} \tag{3.30}$$

where $C_4(\boldsymbol{\sigma}_q)$ is a normalising constant, and $\boldsymbol{\mu}_q^T\mathbf{q}$ is the quaternion dot product.

### 3.1.4 Mixture density networks

We have already seen that multimodal probability densities for real-valued data can be learned in various ways. Popular choices can be non-parametric mixtures such as KDEs or parametric models such as GMMs. In this section we introduce yet another alternative for learning arbitrary densities via gradient descent using mixture density networks (MDNs) [Bishop, 1994]. In such models, the output of the network consists of the parameters of a mixture

conditioned on a suitable choice of input vector $\mathbf{h} \in \mathbb{R}^c$, thus modeling arbitrary multimodal densities over $\mathbf{y} \in \mathbb{R}^d$ as defined in Eq. 3.31 below.

$$p(\mathbf{y}|\mathbf{h}) = \sum_{k=1}^{K} \pi_k(\mathbf{h}) \mathcal{N}(\mathbf{y}|\mu_\mathbf{k}(\mathbf{h}), \sigma_k^2(\mathbf{h})) \tag{3.31}$$

in which, each component is a Gaussian given by

$$\mathcal{N}(\mathbf{y}|\mu_\mathbf{k}(\mathbf{h}), \sigma_k^2(\mathbf{h})) = \frac{1}{(2\pi)^{d/2}\sigma_k(\mathbf{h})^d} \exp\left(-\frac{\|\mathbf{y} - \mu_\mathbf{k}(\mathbf{h}))\|^2}{2\sigma_k(\mathbf{h})^2}\right) \tag{3.32}$$

In Eq. 3.31, the MDN receives a suitable choice of $c$-dimensional feature vector $\mathbf{h} \in \mathbb{R}^c$ as input, and gives as output the parameters of a mixture, in this case a Gaussian mixture model described by $\{\pi_k(\mathbf{h}), \mu_\mathbf{k}(\mathbf{h}), \sigma_k(\mathbf{h})\}_{k=1}^K$. This mixture is depicted by the diagram in Fig. 3.2. If a mixture has to $K$ components, its final layer $\mathbf{h}_n$ must output a total of $(d+2)K$ parameters, i.e. $\mathbf{h}_n \in \mathbb{R}^{(d+2)K}$.

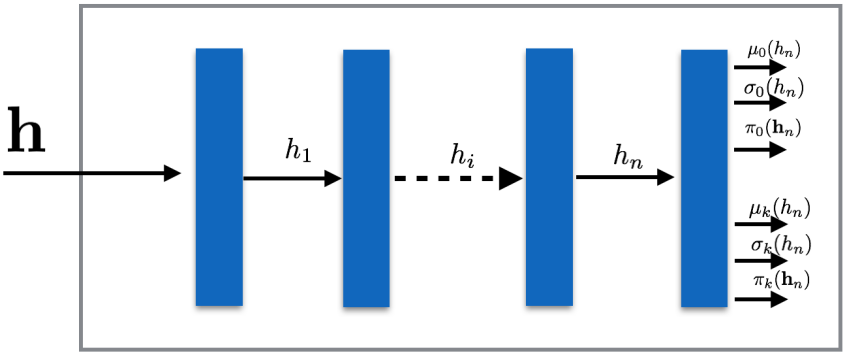


Fig. 3.2 Feed-forward neural network diagram of a mixture density network with initial input $\mathbf{h} \in \mathbb{R}^c$. In general a number of $n$ hidden layers can utilised, each hidden layer containing a chosen number of neuron units.

The final output layer $\mathbf{h}_n$ of an MDN must be treated with special care in order to make sure that the network yields valid parameters for the mixture. For instance, the mixing coeficcients must satisfy the constraint

$$\sum_{k=1}^{K} \pi_k(\mathbf{h}) = 1. \tag{3.33}$$

This can be achieved by using a softmax layer, such that

$$\pi_k = \frac{\exp(\mathbf{h}^{\pi}_{n,k})}{\sum_{k=1}^{K} \exp(\mathbf{h}^{\pi}_{n,k})} \tag{3.34}$$

where $\mathbf{h}^{\pi}_{n,k}$ represents a corresponding section of the output of the network.

Each variance $\sigma_k(\mathbf{h})$ represents a scaling parameter for each component, which must be positive definite. One way to achieve that is by utilising an exponential activation function

$$\sigma_k = \exp(\mathbf{h}^{\sigma}_{n,k}) \tag{3.35}$$

where, again, $\mathbf{h}^{\sigma}_{n,k}$ corresponds to a subsection of the final output layer $\mathbf{h}_n$ dedicated to represent each $\sigma_k$ parameter.

The mean vector location parameter $\mu_{\mathbf{k}}(\mathbf{h}) \in \mathbb{R}^D$ of each component can be directly represented by its corresponding section of the final output layer as

$$\mu_{\mathbf{k}} = \mathbf{h}^{\mu}_{n,k} \tag{3.36}$$

Given a training dataset $\mathcal{D} = \{(\mathbf{y}_j, \mathbf{h}_j)\}_{j=1}^{N}$. The parameters $\Theta$ (network weights) of the MDN can be found using the MLE method, and is given by minimising the negative log-likelihood over the data:

$$\mathcal{L}(\Theta) = \sum_{j=1}^{N} \sum_{k=1}^{K} \pi_k(\mathbf{h_j}) \mathcal{N}(\mathbf{y}_j | \mu_{\mathbf{k}}(\mathbf{h_j}), \sigma_k^2(\mathbf{h_j})) \tag{3.37}$$

The gradient with respect to the network parameters can be found in closed form, and the optimisation is performed using gradient descent techniques, such as the Adam optimiser [Kingma and Ba, 2014]. This allows the parameters to be found via standard back-propagation. Additional implementation details and variations using different kernel components (not necessarily Gaussian) are given by [Bishop, 1994] and [Graves, 2013]. The techniques described in this chapter are used in later chapters this thesis. We make available

an open-source MDN implementation based on the Tensorflow library [Abadi et al., 2016][1]. This same implementation is used later on in Chapter 7.

MDNs are not only capable of modeling arbitrary probability densities, but are also suitable for modelling input-dependent noise, a property called *heteroscedastic*. Intuitively, the model is therefore able to capture situations in which certain input regions can be more noisy than others. Market risk prediction, financial engineering, wind speed and power forecast are example domains in which modeling heteroscedasticity implies better prediction performance in such contexts Mostafa [2011][other]. In Chapter 7 we will see that such models can be put to good use for robotic manipulation applications too.

The ability to model this type of uncertainty has also been demonstrated to work best in the context of pushing by [Bauzá and Rodriguez, 2017], demonstrating that a Variational Heteroscedastic Gaussian Process (VHGP) has better accuracy as a push prediction model than a standard Gaussian Process (GP). However, GPs do not scale well with large amounts of data, and time complexity of predictions is a function of the size of the training set used to train those models.

In Chapter 7 we will use an ensemble of MDNs (E-MDNs) in order to provide uncertainty-calibrated predictions for pushing. We will also combine these predictions using stochastic optimal control framework with path integrals to synthesise uncertainty-averse pushing policies, as a step towards robust pushing.

## 3.2 Gaussian processes

Concisely, a Gaussian process (GP) is an extension of Gaussian distributions to an infinite set of random variables, or an infinite dimensional stochastic process (a.k.a. “random function”), in which any finite subset of dimensions becomes a multivariate Gaussian distribution. Therefore, just as a Gaussian distribution model the distribution of random variables, a GP

[1] Implementation of a general MDN is avalable at: https://github.com/eaa3/aml_dl

models the distribution over random functions [Brochu et al., 2010; Rasmussen, 2006] and is completely specified by a mean function $m(\cdot)$ and a covariance function $k(\cdot)$, such that:

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \tag{3.38}$$

where, $\mathbf{x} \in \mathbb{R}^d$ and $f : \mathbb{R}^d \mapsto \mathbb{R}$.

In essence, we can imagine a random function $f(\mathbf{x})$ as infinite dimensional vector being drawn from an infinite dimensional Gaussian distribution (i.e. GP), a notion which matches directly with multivariate Gaussian distributions for finite dimensional vectors.

By specifying a mean function $m(\cdot)$ and a covariance function $k(\cdot)$, a.k.a. kernel function, we are adding our prior knowledge over the possible set of functions that our *GP* is going to model. Usually, we assume that our mean function is just a zero constant function, $m(\mathbf{x}) = 0$ for all $\mathbf{x}$. Different choices of covariance functions are directly related to assumptions regarding the smoothness of the underlying true function that we are trying to model. A simple and common choice of covariance function is the squared exponential kernel given by Eq 3.39. It is worth mentioning that a wide range of possible kernels exist in the literature, see [Rasmussen, 2006] for a detailed overview, each encoding different assumptions regarding the asymptotic behaviour of the underlying function which we are trying to estimate.

$$k(\mathbf{x}_i, \mathbf{x}_j) = exp(-\frac{1}{2s^2}\|\mathbf{x}_i - \mathbf{x}_j\|^2), \tag{3.39}$$

where $s \in \mathbb{R}^+$ is a non-negative hyper-parameter representing the kernel bandwidth.

Having defined our prior assumptions about the asymptotic behaviour of $f(\cdot)$ as above, given a set of points $x_{1:t}$ one could draw sample functions from our prior $\mathcal{N}(\mathbf{0}, \mathbf{K})$, where the kernel matrix $\mathbf{K}$ is defined as:

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1,\mathbf{x}_1) & k(\mathbf{x}_1,\mathbf{x}_2) & \dots & k(\mathbf{x}_1,\mathbf{x}_t) \\ k(\mathbf{x}_2,\mathbf{x}_1) & k(\mathbf{x}_2,\mathbf{x}_2) & \dots & k(\mathbf{x}_2,\mathbf{x}_t) \\ \dots & \dots & \dots & \dots \\ k(\mathbf{x}_t,\mathbf{x}_1) & k(\mathbf{x}_t,\mathbf{x}_2) & \dots & k(\mathbf{x}_t,\mathbf{x}_t) \end{bmatrix}. \tag{3.40}$$

We are able to draw samples $f_i = f(\mathbf{x}_i)$ from the prior given by 3.38 as below:

$$f_i = m(\mathbf{x}_i) + \mathbf{L} * \mathcal{N}(\mathbf{0},\mathbf{I}) \tag{3.41}$$

Note that $\mathbf{I}$ is the identity matrix, $\mathbf{L}$ is the Cholesky decomposition of $\mathbf{K}$ and $\mathcal{N}(\mathbf{0},\mathbf{I})$ is a normal distributed sample with zero mean and identity covariance. A typical example of sampling four functions from such prior is show in Fig 3.3.

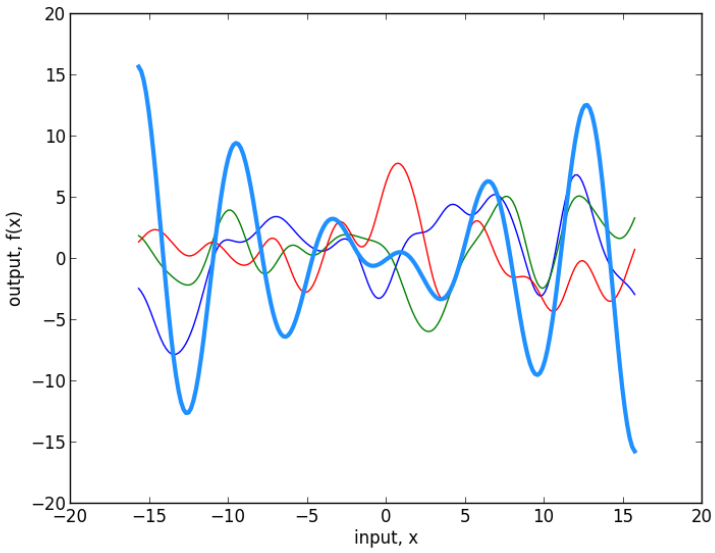


Fig. 3.3 Prior samples generated from the GP's prior. Each differently coloured curve in dark-blue, green and red represents a different function sampled from the GP prior. The light-blue curve depicts the true underlying function.

However, instead of solely relying on our prior knowledge, we are usually interested in making predictions given a set of collected data. The true power of GPs resides in the fact that they allow us to make posterior predictions in closed form given a set of observed

measurements from the underlying function $f$. For this purpose, let $\mathcal{D}_{1:t} = \{\mathbf{x}_{1:t}, f_{1:t}\}$ be the set of observations made by the system so far. We would like to be able to predict $f_{t+1}$ for a given $\mathbf{x}_{t+1}$, in other words we want to estimate $p(f_{t+1}|\mathcal{D}_{1:t}, \mathbf{x}_{t+1})$. Now, according to the properties we described for GPs, we can define the joint distribution of $f_{1:t}$ and $f_{t+1}$ as:

$$\begin{bmatrix} \mathbf{f_{1:t}} \\ \mathbf{f_{t+1}} \end{bmatrix} \sim \mathcal{N}(\mathbf{0}, \begin{bmatrix} \mathbf{K} & \mathbf{k}^T \\ \mathbf{k} & k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) \end{bmatrix}), \tag{3.42}$$

where

$$\mathbf{k} = \begin{bmatrix} k(\mathbf{x}_{t+1}, \mathbf{x}_1) & k(\mathbf{x}_{t+1}, \mathbf{x}_2) & \dots & k(\mathbf{x}_{t+1}, \mathbf{x}_t) \end{bmatrix} \tag{3.43}$$

Finally, as pointed by [Brochu et al., 2010; Rasmussen, 2006], with a more detailed demonstration given by [Bishop, 2006], one can show that:

$$p(f_{t+1}|\mathcal{D}_{1:t}, x_{t+1}) = \mathcal{N}(\mu_t(x_{t+1}), \sigma_t^2(x_{t+1})), \tag{3.44}$$

where

$$\mu_t(x_{t+1}) = \mathbf{k}^T \mathbf{K}^{-1} \mathbf{f_{1:t}}, \tag{3.45}$$

$$\sigma_t^2(\mathbf{x}_{t+1}) = k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) - \mathbf{k}^T \mathbf{K}^{-1} \mathbf{k}. \tag{3.46}$$

Note that, analogously to the way we sampled functions over our prior distribution, we can likewise sample functions over this posterior distribution. We can depict the GP estimates for $\mathbf{x}^*_{1:3}$ for $f$, provided three measurements (shown as dots) in Fig 3.4. It is possible to see now that Gaussian Processes are not an ordinary regression tool. They are able to provide probabilistic predictions, i.e. predictions with a associated measure of uncertainty, for any given query point $\overset{*}{\mathbf{x}}$. A pseudo code outline this prediction procedure is given by Alg 2. In

the next sections we are going to discuss how we can harness many of the good properties that GPs possess not only for typical regression, but also for optimising unknown functions.

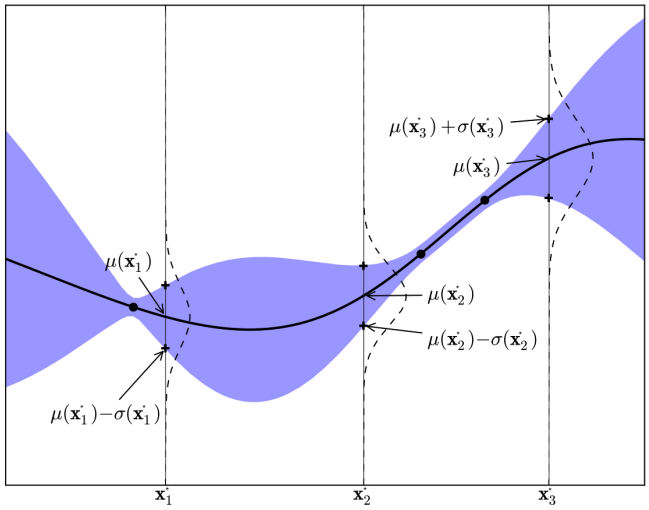


Fig. 3.4 Prior, chart adapted from [Brochu et al., 2010]. The chart shows the estimates for $x_{1:3}^*$ and the corresponding confidence interval for those posterior predictions.

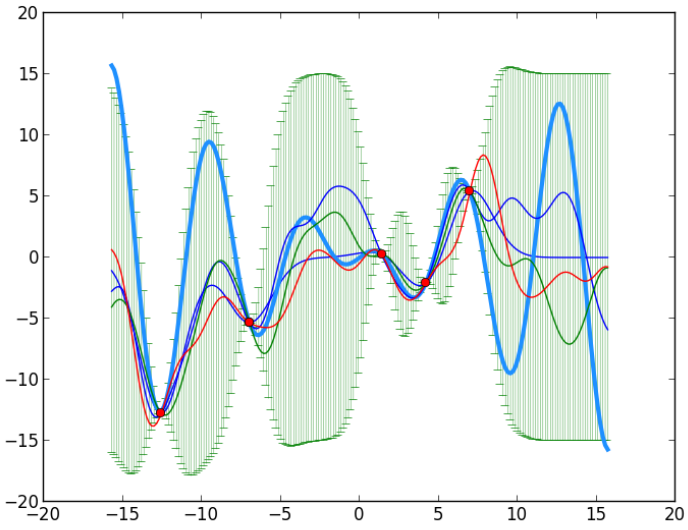


Fig. 3.5 Posterior sampled functions in dark green, dark blue and red. The true function is shown in light blue. Five measurements were taken from the true function, which are shown as red dots. Note the decrease of uncertainty in the points where data was acquired, depicted by the light-green error bars.

**Algorithm 2** Gaussian Process Prediction Algorithm as originally defined in [Rasmussen, 2006].

**Precondition:** $X$ (Inputs), $y$ (targets), $k(\cdot)$ (Covariance function), $\sigma^2$ (noise level), $\overset{*}{\mathbf{x}}$ (test input)

**function** GPPREDICT($\mathbf{X}, \mathbf{y}, k, \sigma_n^2, x^*$)
  $L \leftarrow cholesky(\mathbf{K} + \sigma_n^2 \mathbf{I})$
  $\alpha \leftarrow solve(L, solve(L, \mathbf{y}))$
  $\overline{f}_* \leftarrow \mathbf{k}_*^T \alpha$ ▷ Eq 3.45
  $\mathbf{v} \leftarrow solve(L, \mathbf{k}_*)$
  $\mathbb{V}(\overline{f}_*) \leftarrow k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^T \mathbf{v}$ ▷ Eq 3.46
  $log(p(\mathbf{y}|\mathbf{X})) \leftarrow -\frac{1}{2}\mathbf{y}^T \alpha - \sum_i log(L_{ii}) - \frac{1}{2} log(2\pi)$
  **return** $(\overline{f}_*, \mathbb{V}(\overline{f}_*), log(p(\mathbf{y}|\mathbf{X})))$
**end function**

Note that instead of directly inverting the matrix $(\mathbf{K} + \sigma^2 \mathbf{I})$, the Cholesky decomposition is used to solve the $(\mathbf{K} + \sigma^2 \mathbf{I})^{-1} y$ and $(\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{k}_*$ terms, which are equivalent to the linear systems $\mathbf{K}' \alpha = y$ and $\mathbf{K}' \mathbf{v} = \mathbf{k}_*$ with $\mathbf{K}' = LL = (\mathbf{K} + \sigma^2 \mathbf{I})$, respectively from Eq 3.45 and 3.46. The Cholesky decomposition is used here for numerical stability reasons. The algorithm also computes the marginal probability of the targets given the data, which can be used to optimise the hyper-parameters of the kernel function as discussed in more depth by [Brochu et al., 2010; Rasmussen, 2006].

From now on we are going to refer to GPs as abstract tools for function approximation. In addition to that, we are going to use the ability of GPs to predict posteriors in closed form, providing confidence intervals about its predictions. This extra data are going to allow us to define higher level algorithms on top of the information provided by GPs over the underlying function. These higher level algorithms are using such extra information to make decisions about which should be the next point $\overset{*}{\mathbf{x}}$ that some underlying function $f$ should be evaluated. Indeed, it is often the case that such a function $f$ is very costly to evaluate, so GPs naturally provide a framework for making as fewer call of $f$ as possible to obtain good quality results with fewer, but carefully selected, measurements/evaluations of $f$.

## 3.3 Predictive uncertainty: a comparison on toy problems

Models based on MDNs are versatile and can serve as a powerful tool for learning predictive models. We demonstrate these properties in an intuitive fashion using a few toy examples and

compare them to the standard GP approach described in Section 3.2. MDNs are appealing for robotic applications for many reasons. First, MDNs are cable of approximating arbitrary multimodal probability densities, which can be either be used for prediction or as generative models [Bishop, 1994; Graves, 2013]. Second, MDNs are straightforwardly capable of modeling input-dependent uncertainty, i.e. heteroscedasticity. Third, as preliminarily demonstrated by Lakshminarayanan et al. [2017], when combining a collection of MDNs as an ensemble (E-MDNs, as referred in this thesis) one is also able to have estimates for predictive uncertainty. In Chapter 7 we will demonstrate that these properties will prove to be useful in non-prehensile manipulation tasks, such as pushing, when exploiting predictive uncertainty from learnt predictive models for robust task execution.

### 3.3.1 Modelling heteroscedasticity

As a first instance, consider the following stochastic function corrupted by additive and input-dependent Gaussian noise:

$$y(h) = h + \varepsilon(h) \tag{3.47}$$

where $h \in \mathbb{R}$ and $\varepsilon(h)$ is an input dependent additive Gaussian noise, which for this simple example is defined as $\varepsilon(h) \sim \mathcal{N}(0, \sigma(h))$, with $\sigma(h) = h^2$. Note that the ground truth is a simple linear identity function given by $y(h) = h$. Let us construct a data set $\mathcal{D} = \{(y_j, h_j)\}_{j=1}^{N}$ composed of $N = 200$ samples from Eq. 3.47, as depicted by Fig 3.6.

We utilise the dataset $\mathcal{D}$ to fit a GP model using an exponential kernel with kernel bandwidth $s = 30$, and noise level $\sigma = 1.0$. Furthermore, we utilise the same dataset $\mathcal{D}$ to train an MDN with a single hidden-layer composed of 100 neurons for 40 epochs. By training these two models using $\mathcal{D}$ we obtain the results shown in Fig 3.7, in which the error bars are depicted as the shaded grey area and correspond to 3 standard deviations from the predicted mean for each respective model.

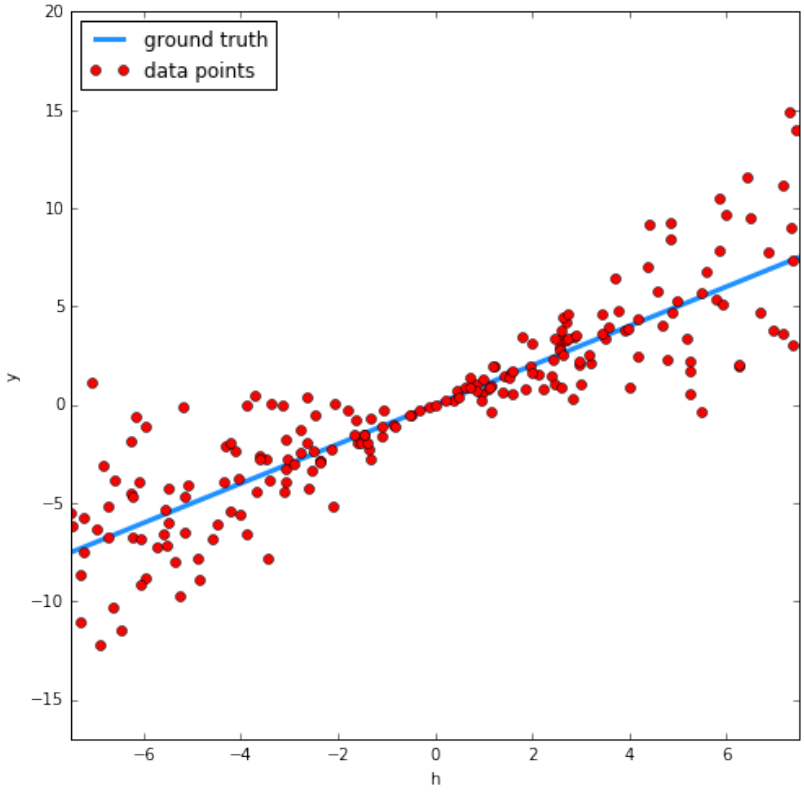


Fig. 3.6 Figure depicts both ground truth function $y(h) = h$ and corresponding corruputed, heteroscedastic data sampled from Eq. 3.47. The samples were acquired such that each $h_j \sim U(-7.5, 7.5)$ is drawn from a uniform distribution in the interval $(-7.5, 7.5)$, and $y_j \sim y(h_j)$.

In this heteroscedastic prediction toy problem notice that the input-dependent stochastic function has generated our training data points $\mathcal{D}$. The learnt MDN model is able to capture the increase in aleatoric uncertainty as a function of $h$, whreas the GP model assumes homoscedastic, or constant aleatoric uncertainty (independent of input $h$), as discussed in Section 2.3.2.1. This is to be expected, since a standard GP is not able to capture input-dependent uncertainty. In this particular example, we also see that the MDN outperforms the GP in terms of the root-mean-squared error (RMSE) with respect to the ground truth

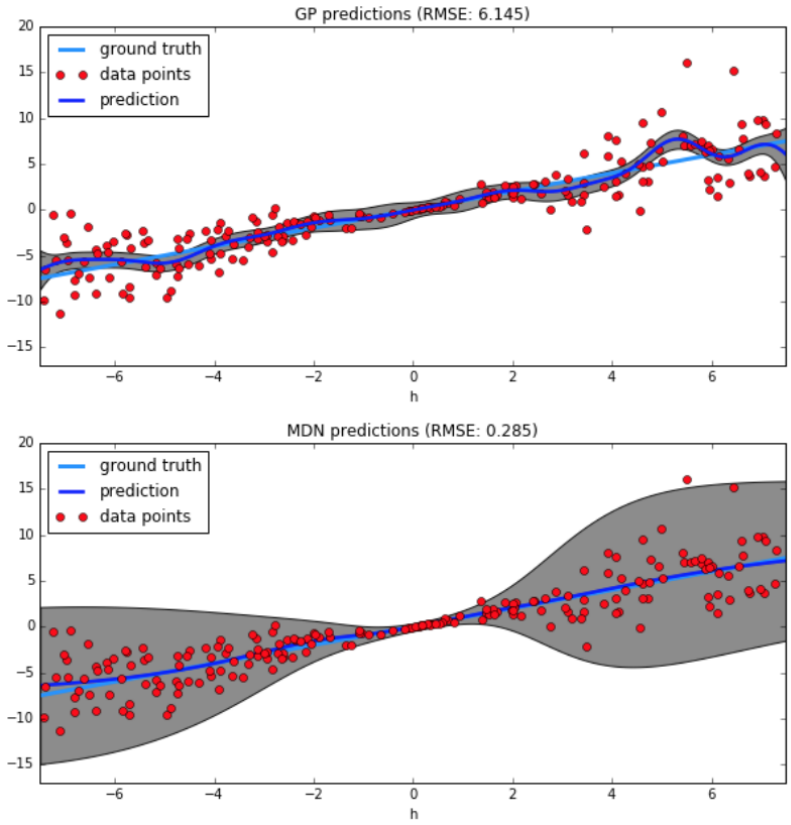


Fig. 3.7 Top: mean and uncertainty predictions of GP. Bottom: mean and uncertainty predictions for the MDN. The shaded grey area corresponds to +/- 3 standard deviations from the mean predictions of each corresponding model.

function[2]. This example exhibited the the capability of MDNs to capture heteroscedastic properties from stochastic processes, whereas a standard GP is unable to.

[2]Although, we note that other authors [Lakshminarayanan et al., 2017] have found cases where minimising a negative log-likelihood objective function (as is the case for learning an MDN) may yield poorer RMSE performance when compared to other regression methods that directly minimise an RMSE loss function.

### 3.3.2 Modelling meta-uncertainty

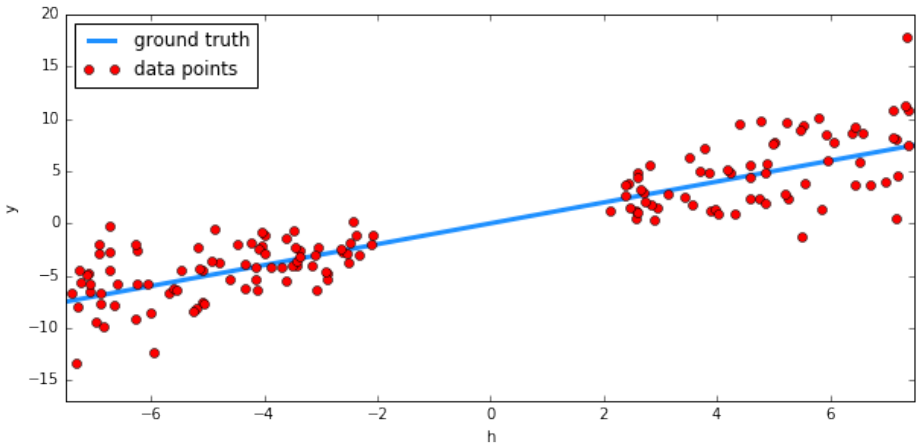


Fig. 3.8 The lesioned dataset $\mathcal{D}_m$. It represents an example in which we do not have access to a complete training dataset that covers all the domain of interest for $h$ (lack of knowledge).

A drawback of MDNs is the fact that they represent a method that in itself is not capable to obtain estimates for meta-uncertainty (uncertainty due to lack of data). In contrast, GPs present an elegant method to estimate this type of uncertainty in a Bayesian fashion [Rasmussen, 2006]. For instance, consider an alternative subset of $\mathcal{D}_m \subset \mathcal{D}$ in which we remove[3] a portion of the data points as depicted in Fig. 3.8.

In principle, making predictions in regions that have not been seen during training may offer risks for robotic applications, since the underlying predictions may be undefined or very far off from the underlying ground truth. Thus, having a notion of uncertainty regarding learnt predictive models is very important so as to take into account predictions that may not be reliable enough.

Consider in Fig. 3.9 the results of fitting the same GP and MDN discussed in Section 3.3.1, but this time using the lesioned dataset $\mathcal{D}_m$. We see that, although MDNs are still able to capture aleatoric uncertainty, they are unable to capture meta-uncertainty. As a consequence, these models tend to be over-confident in their predictions. Thus, in the regions where

[3] In practise, one may not be able to acquire data to cover all the sampling space, e.g. due to kinematic or dynamic limitations of a robot not all state and actions can be attempted (or feasibly explored in a finite amount of time), and thus one may have a limited dataset.

no training data was available its estimated variability or uncertainty was very small. In contrast, the GP model was able to capture meta-uncertainty accordingly, albeit still assuming homoscedastic (aleatoric) uncertainty.

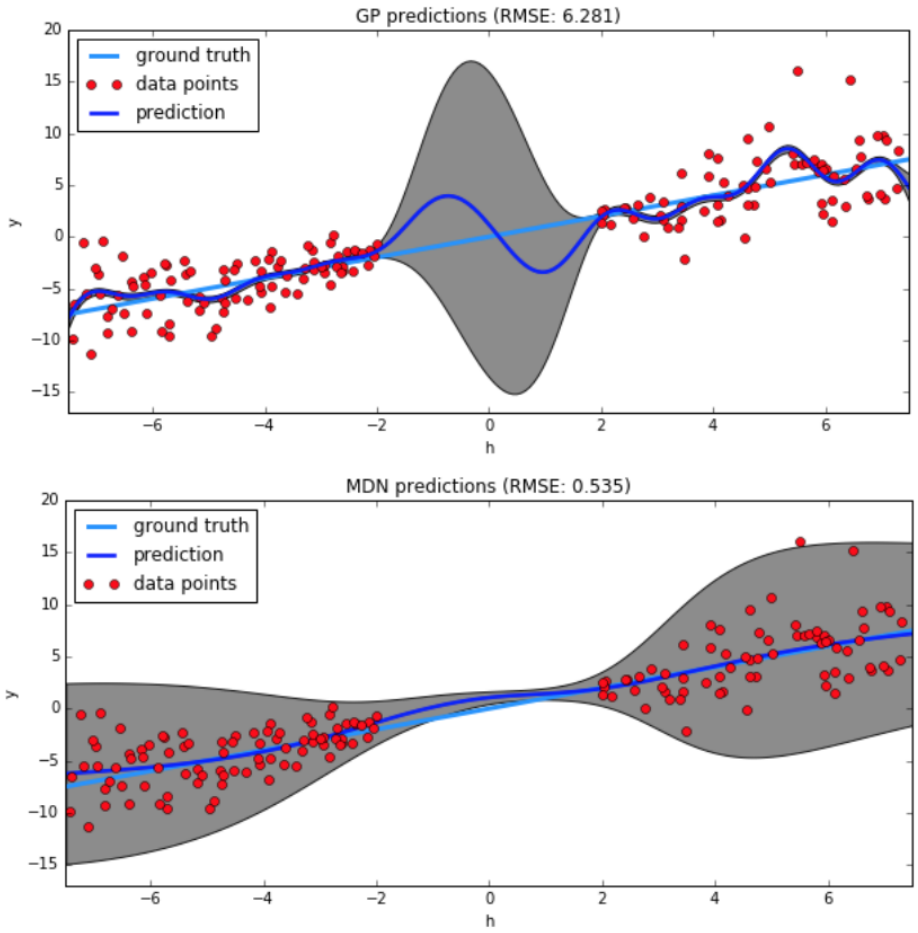


Fig. 3.9 Fitting a GP and an MDN to the lesioned dataset $\mathcal{D}_m$. Top: mean and uncertainty predictions of GP. Bottom: mean and uncertainty predictions for the MDN. The shaded grey area corresponds to +/- 3 standard deviations from the mean predictions of each corresponding model. A single MDN is over-confident with respect to its prections, although it is able to model heteroscedastic aleatoric uncertainty.

In order to be able to provide estimates for predictive meta-uncertainty using MDNs we shall investigate the application of an Ensemble of MDNs (E-MDNs) to this same problem. This approach is based on [Lakshminarayanan et al., 2017]. It consists of a non-Bayesian method to obtain meta-uncertainty estimates from a collection of learnt probability density

models. The E-MDN model consists of a collection of MDNs as depicted in Fig. 3.10. Each member of the ensemble is an MDN, as given by Eq. 3.31.

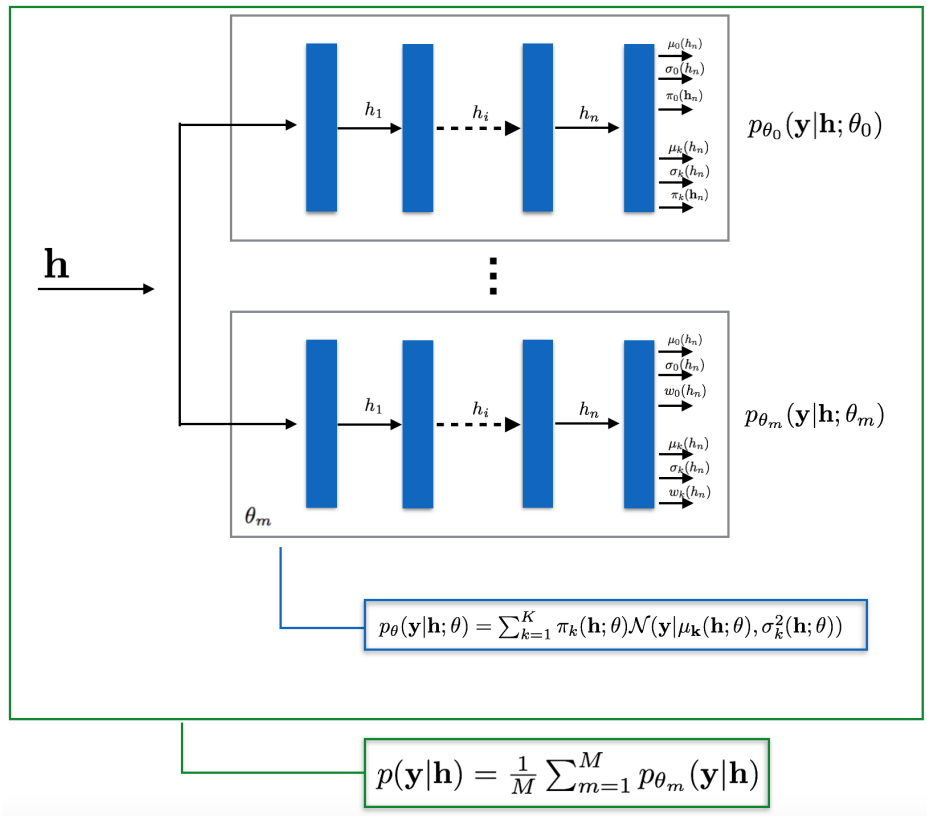


Fig. 3.10 E-MDN diagram.

Thus, an E-MDN can be written as

$$p(\mathbf{y}|\mathbf{h}) = \frac{1}{M}\sum_{m=1}^{M} p_{\theta_m}(\mathbf{y}|\mathbf{h}) \tag{3.48}$$

where, the ensemble contains $M$ members. Each member is an MDN with randomly initialised weights $\theta_m$.

Such model can be trained using the training procedure described by [Lakshminarayanan et al., 2017], which we outline in Alg. 3.

Since the ensemble itself can be seen as a mixture with uniform weights, one simple way of fusing the predictions of each MDN is to compute a Gaussian approximation of the

**Algorithm 3** Training procedure for E-MDN.

**Precondition:** $\mathcal{D}$ (Training data), $\boldsymbol{\varepsilon}$ (adversarial scale), $M$ (number of members in the ensemble)

▷ Let $p_{\theta_m}(\mathbf{y}|\mathbf{h})$ be a neural network that models a probability distribution over $\mathbf{y}$ given $\mathbf{h}$, such as an MDN.

Initialise $\theta_0, \theta_2, \ldots, \theta_m$ randomly ▷ Empirically, we found that initialising with uniformly distributed weights works best

**for** $m = 1$ to $M$ **do**

  Randomly a data point index $i$ such that $(\mathbf{y}_i, \mathbf{h}_i)$

  Generate adversarial example $\mathbf{h}'_i = \mathbf{h}_i + \boldsymbol{\varepsilon} sign(\frac{\partial \mathcal{L}(\theta_m, \mathbf{h}_i, \mathbf{y}_i)}{\partial \mathbf{h}_i})$

  Minimise $\mathcal{L}(\theta_m, \mathbf{h}_i, \mathbf{y}_i) + \mathcal{L}(\theta_m, \mathbf{h}'_i, \mathbf{y}_i)$ w.r.t. $\theta_m$

**end for**

E-MDN prediction whose mean and predictive uncertainty or variance estimates can be given by

$$\hat{\mu} = \frac{1}{M} \sum_{m=1}^{M} \mu_{\theta_m} \tag{3.49}$$

$$\hat{\sigma}^2 = \frac{1}{M} \sum_{m=1}^{M} (\sigma_{\theta_m}^2 + \mu_{\theta_m}^2) - \hat{\mu}^2 \tag{3.50}$$

Using an E-MDN to fit the dataset $\mathcal{D}_m$ yields the uncertainty-calibrated results depicted in Fig.3.11.

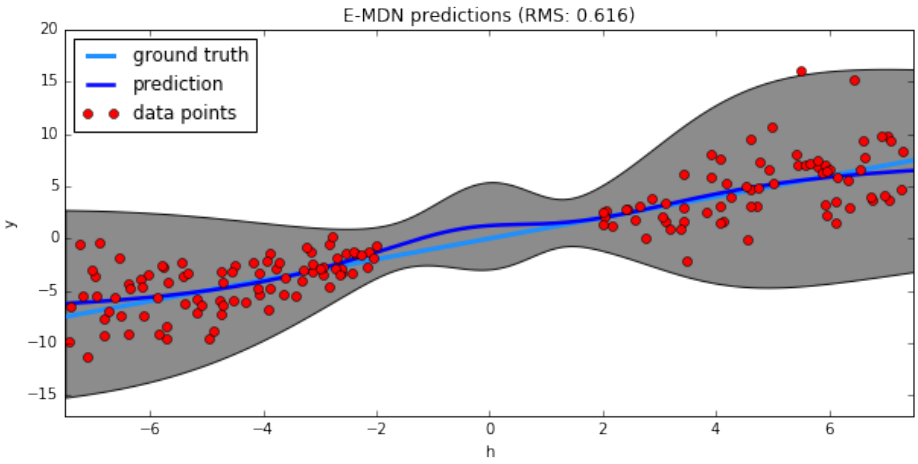


Fig. 3.11 E-MDN predictions on the lesioned dataset $\mathcal{D}_m$. Predictions of the E-MDN now offer sensible uncertainty estimates in the regions of missing data.

## 3.4 Summary

This chapter introduced some basic techniques for approximating probability densities, either for generative modelling or for learning discriminative models used for prediction.

In Section 3.1 the maximum-likelihood estimation (MLE) method has been introduced as general method for parameter estimation. The MLE method was also applied to find parameters of a Gaussian mixture model (GMM) via the EM algorithm. The kernel density estimation (KDE) technique was also introduced as non-parametric alternative to approximate arbitrary probability densities.

Mixture density networks (MDNs) have been described as another method to parametrically approximate multimodal probability densities. This model scales well with large amounts of data, and is able to capture complex statistical properties from the training data, such as *heteroscedasticity* (a type of aleatoric uncertainty initially discussed in Section 2.5). Furthermore, combining MDNs in an ensemble (E-MDNs) allows us to model *meta-uncertainty*. These models shall be explored further in Chapter 7, where E-MDNs are utlised to model both *heteroscedasticity* aleatoric uncertainty and *meta-uncertainty* to achieve robust uncertainty-averse pushing.

In Section 3.1, Gaussian Processes (GPs) have been introduced as a non-parametric technique that is able to capture *meta-uncertainty* (i.e. uncertainty due to lack of knowledge), this type of uncertainty was initially discussed in a concept initially discussed in Section 2.5. This method, however is not able to capture *heteroscedasticity* properties from the data.

# Chapter 4

# Fundamentals of controlling a robot manipulator

In order to perform any manipulation task, a very important part of robot manipulation amounts to understanding basic design characteristics, and how to control a robot manipulator. In this chapter we introduce fundamental concepts regarding kinematics, control and motion synthesis capabilities that serve as basic infrastructure utilised in the algorithms and strategies devised in later chapters of this thesis.

## 4.1 Robot manipulator design

A typical robot manipulator is composed of *links* that are interconnected by articulations represented as *joints*. This mechanical structure thus forms a *kinematic chain*. We will concern ourselves with a typical robot arm that forms an *open kinematic chain*. One end of the chain is fixed to a base, where the arm is mounted. Attached to the other end, one usually finds an *end-effector*, which is typically a tool or gripper specific for a particular manipulation task. A common *end-effector* utilised for grasping is a parallel-jaw gripper with only two fingers, or a multi-fingered hand. Each finger itself can be seen as open kinematic

chain attached to a palm link of a hand. Thus, in this sense a robot hand is composed of smaller robot arms connected in parallel to a palm link. For object pushing, one may have a simple poking rod link attached to the end of a robot arm, allowing it to poke and push objects in its surroundings.

In the mechanical structure of a robot arm, links are assumed to be rigid bodies and joints may be one of the following types[1]:

**Revolute:** this type of joint can be actuated to rotate around an axis. It has a single *degree of freedom* (DoF) and produces a relative rotation motion between two connecting links controlled by an electric motor.

**Prismatic:** this type of joint generates translational motion along a given axis, increasing or decreasing the distance between two connecting links. It also has a single DoF that is actuated by a motor.

The collection of all joints in the chain are identified by joint variables $\theta_i \in \mathbb{R}$.

### 4.1.1 Manipulator configuration space

Each joint variable $\theta_i$ defines one DoF for the motion of a kinematic chain. The total number of DoF for an open kinematic chain consists of the total number of joints in the kinematic chain. Joints are typically limited to move in an interval $\theta_i \in [a_i, b_i] \in \mathbb{R}$. For instance, revolute joints are usually constrained to rotate in the circular interval $0 < b_i - a_i \leq 2\pi$, whereas prismatic joints also possess their translation limits in meters.

As discussed in Appendix A, the *pose* of a rigid body is fully specified by its position and orientation, which is described by a reference frame or a rigid body transformation. Here, we refer to the collection of rigid body poses or frames for each link of the kinematic chain as the

[1] Other types of joints exist, but they can be modelled as a combination of two or more of these basic joints.

*posture* of a robot. The vector $\boldsymbol{\theta} = [\theta_1, \ldots, \theta_m] \in \mathbb{R}^m$ represents a current joint configuration and directly corresponds to the current *posture* of a robot.

When considering the joint limits in the kinematic chain, the configuration space of a robot is the space all possible joint configurations $\boldsymbol{\theta} \in \mathcal{C}$, where $\mathcal{C} \subset \mathbb{R}^m$.

### 4.1.2 Manipulator workspace

It is very important for a manipulator to use its end-effector to reach different positions and orientations in the workspace. If we let the position and orientation of the end-effector be $\mathbf{x} \in SE(3)$, then the *manipulator workspace* consists of the volume in Cartesian space resultant of all possible motions in its configuration space, such that $\mathbf{x} \in \mathcal{W} \subset SE(3)$. This implies the existence of a mapping $fk : \mathcal{C} \mapsto \mathcal{W}$, which takes a configuration $\boldsymbol{\theta}$ to an end-effector *pose*. As in Siciliano et al. [2010], we distinguish the workspace into *reachable* and *dexterous*.

**Reachable workspace:** the set of possible positions that are attainable by the end-effector, without requiring the end-effector to assume a particular orientation. Some positions in the workspace are only reachable when fully stretching the arm. Although they are reachable, they constrain the range of possible orientations with which they can be reached for choice of joint angles $\boldsymbol{\theta}$.

**Dexterous workspace:** the set of all possible positions that are attainable with an arbitrary desired orientation. That is, the set of positions that can be achieved with any required orientation.

## 4.2 Kinematics

We have seen in the previous section the basic components that form the mechanical structure of a robotic arm as an open kinematic chain. In this section we will discuss basic notions on how to define kinematic relationships between a robot configuration space $\mathcal{C}$ and its Cartesian workspace $\mathcal{W}$. In particular, we are interested in the relationship between joint variables and the pose of the manipulator end-effector. There are two types of relatioships:

The forward mapping problem $fk : \mathcal{C} \mapsto \mathcal{W}$ of finding the pose of the end-effetor given a choice joint configuration $\boldsymbol{\theta}$ is called *forward kinematics*. This mapping is typically a many-to-one mapping and can be always solved analytically.

The reverse problem of finding the joint configurations to achieve an end-effector pose is called the *inverse kinematics* problem. This problem is typically a one-to-many mapping and often cannot solved analytically. However, approximate solutions can be found.

### 4.2.1 Forward kinematics

The concrete physical structure of a robot is usually specified by its manufacturer in an openly accessible and understood convention. The Denevit-Hartenberg (DH) convention is a particularly popular choice [Hartenberg and Danavit, 1964; Hartenberg and Denavit, 1955]. This convention defines a set of parameters (DH-parameters) for attaching reference frames to each link of a kinematic chain as a function of its joint variables, or configuration $\boldsymbol{\theta}$.

$$A_{i'}^{i-1} = \begin{bmatrix} \cos\phi_i & -\sin\phi_i & 0 & 0 \\ \sin\phi_i & \cos\phi_i & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

$$A_i^{i'} = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

Here, $a_i$ is the length of the $i^{th}$ link and $\alpha_i$ is degree of torsion in the x-axis between joint $i$ (beggining of $i^{th}$ link) to the next connecting link $i+1$, as given by the design of the robot linkage. In addition for the case of a revolute joint $\phi_i = \theta_i$ and $d_i$ is constant, or alternatively for a prismatic joint $d_i = \theta_i$ and $\phi_i$ is treated as a constant instead.

Thus, combining these two transformations gives us the pose of link $i$ with respect to link $i-1$ as a function of $\theta_i$:

$$A_i^{i-1}(\theta_i) = A_{i'}^{i-1} A_i^{i'} \tag{4.3}$$

Considering an an open kinematic chain constituted by $m$ links connected by $m$ joints allows us to compose these transformations such that

$$T_e^b(\boldsymbol{\theta}) = T_0^b A_1^0(\theta_1) A_2^1(\theta_2) \ldots A_m^{m-1}(\theta_m) T_e^m \tag{4.4}$$

Which yields the position and orientation of the end-effector link with respect to a base frame $b$, where $T_0^b$ is a constant transformation representing the base fixed frame where the arm is mounted. Likewise, $T_e^m$ is a fixed transformation specifying the local position and orientation with respect to link $m$.

Therefore, the forward kinematic mapping using a homogeneous representation for the $SE(3)$ is given by Eq. 4.5 as a function of $\boldsymbol{\theta}$:

$$T_e^b(\boldsymbol{\theta}) = \begin{bmatrix} R_e(\boldsymbol{\theta}) & \mathbf{p}_e(\boldsymbol{\theta}) \\ \mathbf{0} & 1 \end{bmatrix} \tag{4.5}$$

For convenience, we also let $fk : \mathcal{C} \mapsto \mathcal{W}$ be a function that uses Eq. 4.4 and provides a concise representation for the end-effector pose in $SE(3)$. For instance, using position and a quaternion representation, such that $fk(\cdot)$ is given by:

$$\mathbf{x}_e = \begin{bmatrix} \mathbf{p}_e \\ \mathbf{q}_e \end{bmatrix} = fk(\boldsymbol{\theta}) \tag{4.6}$$

where, $\mathbf{x}_e \in \mathbb{R}^7$, with $\mathbf{p}_e \in \mathbb{R}^3, \mathbf{q}_e \in SO(3)$ and $\boldsymbol{\theta} \in \mathcal{C} \subset \mathbb{R}^m$.
Or alternatively, in its minimal representation via using Euler angels:

$$\mathbf{x}_e = \begin{bmatrix} \mathbf{p}_e \\ \boldsymbol{\phi}_e \end{bmatrix} = fk(\boldsymbol{\theta}) \tag{4.7}$$

where, $\mathbf{x}_e \in \mathbb{R}^6$ and $\mathbf{p}_e \in \mathbb{R}^3, \boldsymbol{\phi}_e \in \mathbb{R}^3$.

The DH-parameters is a popular choice of convention. Nonetheless, equivalent solutions for the problem of forward kinematics exist. For instance, an alternative approach would be to use *twists* as in [Murray, 2017].

#### 4.2.1.1 Manipulator Jacobian

One may be interested not only in the current absolute position and orientation of the end-effector as function of its current configuration, but also in the instantaneous forward kinematic relationship between the differential motion of these two variables.

The instantaneous linear velocity $\dot{\mathbf{p}}_e$ and angular velocity $\boldsymbol{\omega}_e$ of the end-effector can be given by the following linear relationship Siciliano et al. [2010]:

$$\mathbf{v}_e = \begin{bmatrix} \dot{\mathbf{p}}_e \\ \boldsymbol{\omega}_e \end{bmatrix} = \mathbf{J}(\boldsymbol{\theta})\dot{\boldsymbol{\theta}} \tag{4.8}$$

where $\mathbf{v}_e \in \mathbb{R}^6$ is a rigid body *twist*.

The Eq. 4.8 represents the differential kinematics equation of a manipulator. In addition, since we are operating in 3D space, $\mathbf{J}(\cdot)$ is a $6 \times m$ matrix, so called *geometric Jacobian* or *manipulator Jacobian* of an open kinematic chain. It defines a linear mapping from instantaneous motion in configuration space to instantaneous motion in workspace. The geometric Jacobian is a function of the manipulator configuration $\boldsymbol{\theta}$, but for compactness we may denote it simply as $\mathbf{J}$ leaving this dependency implicit.

We note that $\mathbf{J}(\boldsymbol{\theta})$ has two components, one for the linear velocity given by

$$\dot{\mathbf{p}}_e = \mathbf{J}_P(\boldsymbol{\theta})\dot{\boldsymbol{\theta}} \tag{4.9}$$

And another for the angular velocity expressed as

$$\boldsymbol{\omega}_e = \mathbf{J}_O(\boldsymbol{\theta})\dot{\boldsymbol{\theta}} \tag{4.10}$$

Thus,

$$\mathbf{J}(\boldsymbol{\theta}) = \begin{bmatrix} \mathbf{J}_P \\ \mathbf{J}_O \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} Jp_0 \\ Jo_0 \end{bmatrix} & \cdots & \begin{bmatrix} Jp_i \\ Jo_i \end{bmatrix} & \cdots & \begin{bmatrix} Jp_m \\ Jo_m \end{bmatrix} \end{bmatrix} \tag{4.11}$$

It turns out, the computation of $\mathbf{J}(\boldsymbol{\theta})$ can be done directly using the DH-parameters of the manipulator (see e.g. [Siciliano et al., 2010] for more details) in the following fashion:

$$\begin{bmatrix} Jp_i \\ Jo_i \end{bmatrix} = \begin{cases} \begin{bmatrix} \mathbf{z}_{i-1} \times (\mathbf{p_e} - \mathbf{p}_{i-1}) \\ \mathbf{z}_{i-1} \end{bmatrix} & \text{if joint } i \text{ is revolute} \\ \begin{bmatrix} \mathbf{z}_{i-1} \\ \mathbf{0} \end{bmatrix} & \text{if joint } i \text{ is prismatic} \end{cases} \tag{4.12}$$

where, $\mathbf{z}_{i-1}$ and $\mathbf{p}_{i-1}$ are the rotational z-axis and position of the previous link $i-1$. They can be given by defining $A_i^0(\boldsymbol{\theta})$ using Eq. 4.3 as the sequence of compositions:

$$A_i^0(\boldsymbol{\theta}) = A_1^0(\theta_1)A_2^1(\theta_2)\dots A_i^{i-1}(\theta_i) \tag{4.13}$$

from which the respective subsections of the transformations can be extracted during the forward kinematics computation as

$$\mathbf{z}_{i-1} = A_{i-1}^0(\boldsymbol{\theta})[0,0,1,0]^T \tag{4.14}$$

$$\mathbf{p}_{i-1} = A_{i-1}^0(\boldsymbol{\theta})[0,0,0,1]^T \tag{4.15}$$

and finally $\mathbf{p}_e$ is given by Eq. 4.4.

This iterative solution essentially computes the linear and angular velocity motion contributions for each link as a function of each joint variable. Thus, it establishes a linear relationship given by $\mathbf{J}$ between differential increments in $\dot{\boldsymbol{\theta}} = \lim_{\Delta t \to 0} \frac{\Delta \boldsymbol{\theta}}{\Delta t}$ and resultant twist $\mathbf{v}_e$ at the end-effector.

#### 4.2.1.2 Analytical Jacobian

As opposed to describing the motion relationship between configuration and end-effector through geometric considerations, one may start investigating the instantaneous behaviour of the minimal representation in Eq. 4.7 by directly taking its analytic time derivative

$$\dot{\mathbf{x}}_e = \begin{bmatrix} \dot{\mathbf{p}}_e \\ \dot{\boldsymbol{\phi}}_e \end{bmatrix} = \frac{\partial fk(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \dot{\boldsymbol{\theta}} \tag{4.16}$$

where, the partial derivative term

$$\mathbf{J}_A(\boldsymbol{\theta}) = \frac{\partial fk(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \tag{4.17}$$

is a $6 \times m$ matrix so called the *analytical Jacobian*. It is called analytical because it is the result of directly differentiating the forward kinematic function $fk(\cdot)$ in Eq. 4.7. It describes a linear relationship instantaneous motion in joint configuration space and corresponding motion in workspace by the end-effector.

As described in [Siciliano et al., 2010], the geometric and analytical Jacobians are related according to a transformation $\mathbf{T}(\boldsymbol{\phi}_e)$ that is dependent on the choice of convention made for the Euler angle representation for orientation part $\boldsymbol{\phi}_e$ of the end-effector pose $\mathbf{x}_e$. Such that

$$\boldsymbol{\omega}_e = \mathbf{T}(\boldsymbol{\phi}_e)\dot{\boldsymbol{\phi}}_e \tag{4.18}$$

If $\boldsymbol{\phi}_e = [\alpha, \beta, \gamma]$ is represented using roll-pitch-yaw angles as in Eq. A.6, then:

$$\mathbf{T}(\boldsymbol{\phi}_e) = \begin{bmatrix} 0 & -\sin\alpha & \cos\alpha\cos\beta \\ 0 & \cos\alpha & \sin\alpha\cos\beta \\ 1 & 0 & -\sin\beta \end{bmatrix}. \tag{4.19}$$

Hence, the relationship between analytical and geometric Jacobians are given by

$$\mathbf{J}(\boldsymbol{\theta}) = \mathbf{A}(\boldsymbol{\phi}_e)\mathbf{J}_A(\boldsymbol{\theta}) \tag{4.20}$$

where,

$$\mathbf{A} = \begin{bmatrix} \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{T}(\boldsymbol{\phi}_e) \end{bmatrix} \tag{4.21}$$

Taking a second look at Eq. 4.19, we see that its determinant is given by $\det(\mathbf{T}(\boldsymbol{\phi}_e)) = -\cos(\beta)$. Hence, it does not have inverse whenever $\cos(\beta) = 0$. This exposes a so called *representation singularity* of $\boldsymbol{\phi}$. Meaning that, although all possible end-effector velocities can be expressed by an appropriate angular velocity vector $\boldsymbol{\omega}_e$, there are angular velocities that cannot be expressed by the time derivative $\dot{\boldsymbol{\phi}}$ when $\cos(\beta) = 0$.

### 4.2.2 Inverse kinematics

In the previous section we described an approach to compute the forward mapping between joint configuration $\boldsymbol{\theta}$ and corresponding end-effector pose. We also described instantaneous forward kinematics relationship from geometric and analytic perspectives.

The inverse problem of finding the joint variables corresponding to a desired end-effector position and orientation is defined as the *inverse kinematics problem* (IK). A differential solution to this problem can be given by investigating the instantaneous motion of the manipulator as given by Eq. 4.8 and Eq. 4.16.

The IK problem can be formalised as an optimisation problem using the *Lagrange multipliers method* by defining the following objective function [Siciliano et al., 2010]:

$$g(\dot{\boldsymbol{\theta}}, \boldsymbol{\lambda}) = \frac{1}{2}\dot{\boldsymbol{\theta}}^T \mathbf{W}\dot{\boldsymbol{\theta}} + \boldsymbol{\lambda}^T(\mathbf{v}_e - \mathbf{J}\dot{\boldsymbol{\theta}}) \tag{4.22}$$

where $\mathbf{W}$ is a $m \times m$ symmetric positive-definite weighting matrix, as such its inverse $\mathbf{W}^{-1}$ also exists.

The necessary conditions have to be satisfied by a solution

$$\left(\frac{\partial g}{\partial \dot{\boldsymbol{\theta}}}\right)^T = \mathbf{0} \qquad \left(\frac{\partial g}{\partial \boldsymbol{\lambda}}\right)^T = \mathbf{0} \tag{4.23}$$

The first condition gives us $\mathbf{W}\dot{\boldsymbol{\theta}} - \mathbf{J}^T\boldsymbol{\lambda} = \mathbf{0}$, which can be simplified to

$$\dot{\boldsymbol{\theta}} = \mathbf{W}^{-1}\mathbf{J}^T\boldsymbol{\lambda} \tag{4.24}$$

The second condition gives us back the constraint $\mathbf{v}_e = \mathbf{J}\dot{\boldsymbol{\theta}}$ (i.e. Eq. 4.8). Substituting Eq. 4.24 back in this constraint we obtain

$$\mathbf{v}_e = \mathbf{J}\mathbf{W}^{-1}\mathbf{J}^T\boldsymbol{\lambda} \tag{4.25}$$

The term $\mathbf{J}\mathbf{W}^{-1}\mathbf{J}^T$ is a $6x6$ square matrix, and by assuming $\mathbf{J}$ is full rank, then this whole term can be inverted, which gives the solution for $\boldsymbol{\lambda}$

$$\boldsymbol{\lambda} = (\mathbf{J}\mathbf{W}^{-1}\mathbf{J}^T)^{-1}\mathbf{v}_e \tag{4.26}$$

Substituting the solution for $\boldsymbol{\lambda}$ back in Eq. 4.24 we obtain

$$\dot{\boldsymbol{\theta}} = \underbrace{\mathbf{W}^{-1}\mathbf{J}^T(\mathbf{J}\mathbf{W}^{-1}\mathbf{J}^T)^{-1}}_{\mathbf{J}_\mathbf{W}^+}\mathbf{v}_e \tag{4.27}$$

The term

$$\mathbf{J}_\mathbf{W}^+ = \mathbf{W}^{-1}\mathbf{J}^T(\mathbf{J}\mathbf{W}^{-1}\mathbf{J}^T)^{-1} \tag{4.28}$$

is the so called *weighted generalised pseudo inverse* of $\mathbf{J}$. In particular, it is a *right pseudo inverse*, because when right-multiplied with $\mathbf{J}$, it yields $\mathbf{I}_{6\times 6} = \mathbf{J}\mathbf{J}_\mathbf{W}^+$.

Since $\mathbf{W}$ is an arbitrary choice of positive-definite and symmetric weighting matrix, it can be simplified to the special case of $\mathbf{J}^T(\mathbf{J}\mathbf{J}^T)^{-1}$ by choosing $\mathbf{W}$ to be the identity $\mathbf{I}_{6\times 6}$. This special case is the *Moore–Penrose inverse* [Siciliano et al., 2010], and we shall denote it by $\mathbf{J}^\dagger = \mathbf{J}^T(\mathbf{J}\mathbf{J}^T)^{-1}$.

A simple iterative algorithm can be finally defined to find a the joint configuration $\boldsymbol{\theta}$ that reaches a desired end-effector position $\mathbf{x}_d$. To achieve this goal, let

$$\mathbf{e} = \mathbf{x}_d - \mathbf{x}_e \tag{4.29}$$

Its time derivative is then

$$\dot{\mathbf{e}} = \dot{\mathbf{x}}_d - \dot{\mathbf{x}}_e \tag{4.30}$$

where, using Eq. 4.16 we can re-write it as

$$\dot{\mathbf{e}} = \dot{\mathbf{x}}_d - \mathbf{J}_A(\boldsymbol{\theta})\dot{\boldsymbol{\theta}}. \tag{4.31}$$

Defining $\dot{\boldsymbol{\theta}}$ as

$$\dot{\boldsymbol{\theta}} = \mathbf{J}^{\dagger}(\dot{\mathbf{x}}_d + \mathbf{K}\mathbf{e}) \tag{4.32}$$

Substituting this expression in Eq. 4.31 and simplifying gives us a homogeneous differential equation describing the dynamics of the error $\mathbf{e}$

$$\dot{\mathbf{e}} + \mathbf{K}\mathbf{e} = \mathbf{0} \tag{4.33}$$

If $\mathbf{K}$ is chosen to be a positive definite matrix, then Eq. 4.33 its solution is an exponential function that decays to zero, i.e. $\lim_{t\to\infty} e = 0$, and thus is *asymptotically stable*.

Therefore, by assuming that goes to a stationary location, i.e. $\mathbf{x}_d = \mathbf{0}$, a solution to this differential equation is simply found via numerical integration. Various integration methods can be applied, in Alg. 4 is equivalent to Euler's method [Boyce et al., 2017].

**Algorithm 4** Differential inverse kinematics

**Precondition:** $\mathbf{x}_d$ (Desired end-effector pose), $\boldsymbol{\theta}_0$ (Initial configuration), $\mathbf{K}$ (Weight matrix), $\varepsilon$ (Tolerance threshold), T (Max iterations)

$t \leftarrow 0$
$\mathbf{e} \leftarrow \mathbf{x}_d - fk(\boldsymbol{\theta}_t)$
**while** $\|\mathbf{e}\| \geq \varepsilon$ or $t > T$ **do**
  $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \mathbf{J}^{\dagger}(\boldsymbol{\theta}_t)\mathbf{K}\mathbf{e}$
  $t \leftarrow t + 1$
  $\mathbf{e} \leftarrow \mathbf{x}_d - fk(\boldsymbol{\theta}_t)$
**end while**
**if** $t \leq T$ **then**
  $\overset{*}{\boldsymbol{\Theta}} \leftarrow \boldsymbol{\Theta}_t$ ▷ Solution found.
**end if**

The algorithm starts from an initial configuration $\boldsymbol{\theta}_0$ and integrates towards $\mathbf{x}_d$, the error will converge to zero, or never converge in case: i) $\mathbf{x}_d$ is out of reach or too far to be reached by the robot arm, ii) joint limits are reached, etc. When a maximum number of iterations $T$ is reached the loop stops.

## 4.3 Control

In the previous section we developed the fundamental notions of kinematics for an open kinematic chain. That allowed us to describe the pose of end-effector given its current

configuration state via forward kinematics. In addition, instantaneous forward kinematics, provided the tools to solve the IK problem of derived to find the joint configurations that take the end-effector to a desired position and orientation. However, we have made no considerations regarding forces, and what motor commands have to be computed in order to effectively control a robot so as to make it assume a desired kinematic state. In this section we give a brief overview on how this can be done.

### 4.3.1 Joint space control

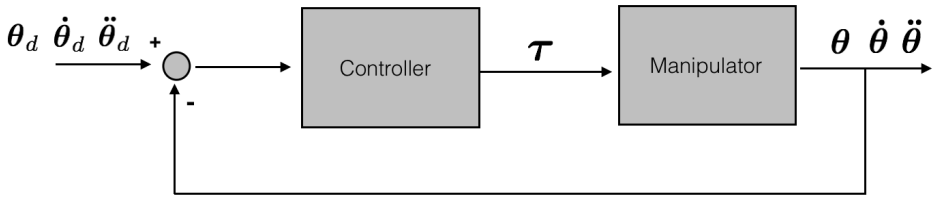


Fig. 4.1 A general joint space controller. It computes joint torques $\boldsymbol{\tau}$ to actuate the joint motors such that a nominal trajectory $\{\boldsymbol{\theta}_d, \dot{\boldsymbol{\theta}}_d, \ddot{\boldsymbol{\theta}}_d\}$ is followed or tracked. It may use feedback error feedback to correct errors while attempting to track a trajectory.

A robot manipulator is in fact a non-linear dynamical system. Each joint is typically actuated by an electric motor that is able to exert enough torque to move its mechanical structure to desired locations. Much like Newton's second law $F = ma$, which describes the resultant force of a point mass with acceleration $a$ and mass $m$, the *manipulator equation* describes the resultant torque from each joint $\boldsymbol{\tau} \in \mathbb{R}^m$ when a multi-linked arm has acceleration $\ddot{\boldsymbol{\theta}}$, velocity $\dot{\boldsymbol{\theta}}$ and configuration $\boldsymbol{\theta}$. Solving recursively the Newton-Euler equations for a robotic manipulator, or alternatively via solving the equivalent Lagrange formulation yields the following equation of motion [Murray, 2017; Siciliano et al., 2010]

$$\mathbf{M}(\boldsymbol{\theta})\ddot{\boldsymbol{\theta}} + \underbrace{\mathbf{C}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) + \mathbf{G}(\boldsymbol{\theta})}_{F(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})} = \boldsymbol{\tau} \tag{4.34}$$

where $M(\boldsymbol{\theta}) \in \mathbb{R}^{m\times m}$ is a state-dependent positive-definite and symmetric inertia matrix, $\mathbf{C}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) \in \mathbb{R}^{m\times m}$ captures the torques arising from Coriolis, centripetal and frictional forces and $\mathbf{G}(\boldsymbol{\theta}) \in \mathbb{R}^m$ represents the torques resulting from the gravitational forces, with $m$ being the number of joints of the manipulator. The terms $\mathbf{C}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})$ and $\mathbf{G}(\boldsymbol{\theta})$ are commonly agglomerated as a single term $F(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})$ for compactness. In this form, Eq. 4.34 is also called the *inverse dynamics equation* of a manipulator, as it evaluates the resultant torques from given manipulator motion $\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}$ and $\ddot{\boldsymbol{\theta}}$. Conversely, the *forward dynamics* equation, which allows us to know the resultant robot motion from applied torques is readily given by

$$\ddot{\boldsymbol{\theta}} = \mathbf{M}(\boldsymbol{\theta})^{-1}(\boldsymbol{\tau} - F(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})) \tag{4.35}$$

The forward dynamics equation of the manipulator system allows it to be simulated via forward integration using a numerical method of choice (e.g. Runge-Kutta, Euler's method, among others [Boyce et al., 2017]).

Given a nominal trajectory $\{\boldsymbol{\theta}_d, \dot{\boldsymbol{\theta}}_d, \ddot{\boldsymbol{\theta}}_d\} \in \mathbb{R}^m$, the goal of joint space control is to find the desired $\boldsymbol{\tau}_d$ which allows us to follow, or track the provided nominal trajectory. It also may or may not use error feedback to compensate for errors while following a trajectory. This general control strategy is depicted by Fig. 4.1.

Ideally, if one were to have a perfect model of the robot dynamics, we could then blindly compute corresponding torque commands $\boldsymbol{\tau}_d$ using the following *open-loop control law*:

$$\boldsymbol{\tau}_d = \mathbf{M}(\boldsymbol{\theta}_d)\ddot{\boldsymbol{\theta}}_d + F(\boldsymbol{\theta}_d, \dot{\boldsymbol{\theta}}_d) \tag{4.36}$$

Having a perfect model of the robot dynamics is usually impractical. As a result, this simple control law does not work well in practise. It is often the case that there are many factors that failed to be modelled accurately enough. The robot may also occasionally be under influence of external forces, and in this case the open-loop control law would completely fail.

A more robust control law makes use of both approximate knowledge of the robot dynamics and feedback to correct for unmodelled errors. The *computed torque control law* [Ortega and Spong, 1988] is given defining $\mathbf{y}_d$ such that:

$$\mathbf{y}_d = \mathbf{K}_p\mathbf{e} + \mathbf{K}_d\dot{\mathbf{e}} + \ddot{\boldsymbol{\theta}}_d \tag{4.37}$$

where $\mathbf{e} = (\boldsymbol{\theta}_d - \boldsymbol{\theta})$, $\mathbf{K}_p$ and $\mathbf{K}_d$ are positive-definite $m \times m$ gain matrices. Letting $\ddot{\boldsymbol{\theta}} = \mathbf{y}_d$ when plugged back to Eq. 4.34 yields

$$\begin{aligned}
\boldsymbol{\tau}_d &= \mathbf{M}(\boldsymbol{\theta})\mathbf{y}_d + F(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) \\
\boldsymbol{\tau}_d &= \underbrace{\mathbf{M}(\boldsymbol{\theta})(\mathbf{K}_p\mathbf{e} + \mathbf{K}_d\dot{\mathbf{e}})}_{\boldsymbol{\tau}_{feedback}} + \underbrace{\mathbf{M}(\boldsymbol{\theta})\ddot{\boldsymbol{\theta}}_d + F(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})}_{\boldsymbol{\tau}_{feedforward}}
\end{aligned} \tag{4.38}$$

note that we identify a feedback term $\boldsymbol{\tau}_{feedback}$ that compensates for errors in dynamics and a feedforward term $\boldsymbol{\tau}_{feedforward}$ that effectively cancels out the non-linear terms captured by $F(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})$ in Eq. 4.34. Indeed this can be shown by plugging Eq. 4.38 in Eq. 4.34, such that

$$\begin{aligned}
\boldsymbol{\tau} &= \boldsymbol{\tau}_d \\
\mathbf{M}(\boldsymbol{\theta})\ddot{\boldsymbol{\theta}} + \cancel{F(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})} &= \mathbf{M}(\boldsymbol{\theta})(\mathbf{K}_p\mathbf{e} + \mathbf{K}_d\dot{\mathbf{e}}) + \mathbf{M}(\boldsymbol{\theta})\ddot{\boldsymbol{\theta}}_d + \cancel{F(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}})} \\
\mathbf{M}(\boldsymbol{\theta})^{-1}\mathbf{M}(\boldsymbol{\theta})\ddot{\boldsymbol{\theta}} &= \mathbf{M}(\boldsymbol{\theta})^{-1}[\mathbf{M}(\boldsymbol{\theta})(\mathbf{K}_p\mathbf{e} + \mathbf{K}_d\dot{\mathbf{e}}) + \mathbf{M}(\boldsymbol{\theta})\ddot{\boldsymbol{\theta}}_d] \\
\ddot{\boldsymbol{\theta}} &= \mathbf{K}_p\mathbf{e} + \mathbf{K}_d\dot{\mathbf{e}} + \ddot{\boldsymbol{\theta}}_d \\
\ddot{\boldsymbol{\theta}} &= \mathbf{K}_p(\boldsymbol{\theta}_d - \boldsymbol{\theta}) + \mathbf{K}_d(\dot{\boldsymbol{\theta}}_d - \dot{\boldsymbol{\theta}}) + \ddot{\boldsymbol{\theta}}_d
\end{aligned} \tag{4.39}$$

We see that the result in Eq. 4.39 essentially means that the originally non-linear dynamics of the manipulator has been simplified by the feed forward term. It is now expressed as a second-order linear differential equation that describes the robot motion in terms of a mass-spring-damper system. For this reason the computed torque control law is also an example of *feedback linearisation* law, that is, it cancels the non-linear terms in the dynamics of the manipulator.

Indeed, when completely disregarding knowledge of the dynamics, an even simpler form of control law can be defined. The idea is to assume attached springs and dampers to each joint which pull each joint to a desired set point given by the desired nominal trajectory.

$$\begin{aligned}\boldsymbol{\tau}_d &= \mathbf{K}_p(\boldsymbol{\theta}_d - \boldsymbol{\theta}) + \mathbf{K}_d(\dot{\boldsymbol{\theta}}_d - \dot{\boldsymbol{\theta}}) \\ \boldsymbol{\tau}_d &= \mathbf{K}_p\mathbf{e} + \mathbf{K}_d\dot{\mathbf{e}}\end{aligned} \tag{4.40}$$

By assuming that our desired set point is stationary (i.e. $\dot{\boldsymbol{\theta}}_d = \mathbf{0}$), a simplified *proportional-derivative* (PD) control law is given by

$$\boldsymbol{\tau}_d = \mathbf{K}_p(\boldsymbol{\theta}_d - \boldsymbol{\theta}) - \mathbf{K}_d\dot{\boldsymbol{\theta}} \tag{4.41}$$

### 4.3.2 Workspace control

As opposed to controlling the robot in configuration space, one may be also interested in directly specifying trajectories in workspace which are typically more meaningful for various tasks. A straightforward solution to that is to add an IK module to the control strategy that is responsible for converting trajectories in workspace $\{\mathbf{x}_d, \dot{\mathbf{x}}_d, \ddot{\mathbf{x}}_d\}$ to configuration space. The diagram in Fig. 4.2 illustrates this type of workspace controller.

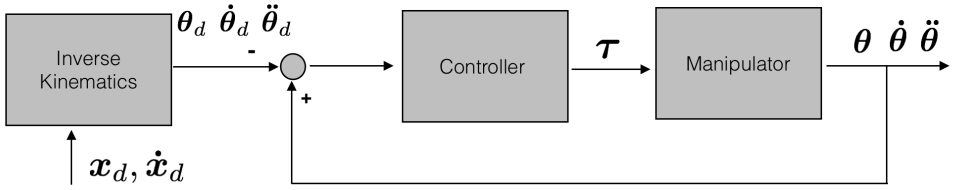


Fig. 4.2 A workspace controller is depicted as a combination of an IK solver combined with a general joint space controller.

However, the approach in Fig. 4.2 suffers from at least two main issues highlighted below

Adding an IK solver in control loop significantly increases the computational load in terms of processing time.

The forward and inverse kinematic mappings represent non-linear relationships of between configuration and workspace coordinates. This means that small errors in workspace may translate to large errors in configuration space. This in turn makes the task of tuning the gain matrices $\mathbf{K}_p$ and $\mathbf{K}_d$ much harder.

To circumvent these problems, it is possible to derive control laws directly in workspace. To achieve this, we may first make use of first and second-order instantaneous kinematic identities:

$$\begin{aligned} \dot{\mathbf{x}}_e &= \mathbf{J}\dot{\boldsymbol{\theta}} \\ \ddot{\mathbf{x}}_e &= \mathbf{J}\ddot{\boldsymbol{\theta}} + \dot{\mathbf{J}}\dot{\boldsymbol{\theta}} \end{aligned} \tag{4.42}$$

As previously defined, it then follows that the term $\ddot{\boldsymbol{\theta}}$ can be given by Eq. 4.35. Thus, making this substitution in Eq. 4.42 yields:

$$\ddot{\mathbf{x}}_e = \mathbf{J}[\mathbf{M}^{-1}(\boldsymbol{\tau} - \mathbf{C}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) - \mathbf{G}(\boldsymbol{\theta})] + \dot{\mathbf{J}}\dot{\boldsymbol{\theta}} \tag{4.43}$$

where we recall $F(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) = \mathbf{C}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) + \mathbf{G}(\boldsymbol{\theta})$.
This expression can be simplified to

$$\ddot{\mathbf{x}}_e = \mathbf{J}\mathbf{M}^{-1}\boldsymbol{\tau} - \mathbf{J}\mathbf{M}^{-1}\mathbf{C}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) - \mathbf{J}\mathbf{M}^{-1}\mathbf{G}(\boldsymbol{\theta}) + \dot{\mathbf{J}}\dot{\boldsymbol{\theta}} \tag{4.44}$$

Utilising the *principle of virtual work* [Siciliano et al., 2010], which is given by $\boldsymbol{\tau} = \mathbf{J}^T\mathbf{F}$ and pre-multiplying all terms by $(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T)^{-1}$ yields the following expression:

$$(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T)^{-1}\ddot{\mathbf{x}}_e = (\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T)^{-1}\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\mathbf{F} - (\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T)^{-1}\mathbf{J}\mathbf{M}^{-1}\mathbf{C}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) \tag{4.45}$$

$$- (\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T)^{-1}\mathbf{J}\mathbf{M}^{-1}\mathbf{G}(\boldsymbol{\theta}) + (\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T)^{-1}\dot{\mathbf{J}}\dot{\boldsymbol{\theta}} \tag{4.46}$$

Note that from Eq. 4.28 we observe that the term $(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T)^{-1}\mathbf{J}\mathbf{M}^{-1}$ is the transposed generalised weighted inverse of $\mathbf{J}$ which is weighted by the positive-definite and symmetric inertia matrix $\mathbf{M}$, i.e. $(\mathbf{J}^+_{\mathbf{M}})^T = (\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T)^{-1}\mathbf{J}\mathbf{M}^{-1}$. Recall further from matrix algebra that $(\mathbf{A}\mathbf{B})^T = \mathbf{B}^T\mathbf{A}^T$. Taking into account these observations:

$$(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T)^{-1}\ddot{\mathbf{x}}_e = \cancel{(\mathbf{J}^+_{\mathbf{M}})^T\mathbf{J}^T}\mathbf{I}\mathbf{F} - (\mathbf{J}^+_{\mathbf{M}})^T\mathbf{C}(\boldsymbol{\theta},\dot{\boldsymbol{\theta}}) - (\mathbf{J}^+_{\mathbf{M}})^T\mathbf{G}(\boldsymbol{\theta}) + (\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T)^{-1}\dot{\mathbf{J}}\dot{\boldsymbol{\theta}} \quad (4.47)$$

Rearranging the terms gives the following expression

$$\underbrace{(\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T)^{-1}}_{\mathbf{M}_x(\boldsymbol{\theta})}\ddot{\mathbf{x}}_e + \underbrace{[(\mathbf{J}^+_{\mathbf{M}})^T\mathbf{C}(\boldsymbol{\theta},\dot{\boldsymbol{\theta}}) - (\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T)^{-1}\dot{\mathbf{J}}\dot{\boldsymbol{\theta}}]}_{\mathbf{C}_\mathbf{x}(\boldsymbol{\theta},\dot{\boldsymbol{\theta}})} + \underbrace{(\mathbf{J}^+_{\mathbf{M}})^T\mathbf{G}(\boldsymbol{\theta})}_{\mathbf{G}_x(\boldsymbol{\theta})} = \mathbf{F} \quad (4.48)$$

Hence, the workspace inverse dynamics equation for a manipulator is given by:

$$\mathbf{M}_x(\boldsymbol{\theta})\ddot{\mathbf{x}}_e + \mathbf{C}_\mathbf{x}(\boldsymbol{\theta},\dot{\boldsymbol{\theta}}) + \mathbf{G}_x(\boldsymbol{\theta}) = \mathbf{F} \quad (4.49)$$

where $\mathbf{F} \in \mathbb{R}^6$ is the vector with generalised forces and torques at the end-effector. At this point, the dynamics of a robot arm is described in terms of the dynamics of its end-effector. The of workspace inverse dynamics in Eq. 4.49 is also analogous to manipulator equation in configuration space given by Eq. 4.34.

For instance, if we let $\mathbf{e} = \mathbf{x}_d - \mathbf{x}$, the analogous computed torque control law in workspace can be given by

$$\mathbf{F}_d = \underbrace{\mathbf{M}_x(\boldsymbol{\theta})(\mathbf{K}_p\mathbf{e} + \mathbf{K}_d\dot{\mathbf{e}})}_{\boldsymbol{\tau}_{feedback}} + \underbrace{F(\boldsymbol{\theta},\dot{\boldsymbol{\theta}})}_{\boldsymbol{\tau}_{feedforward}} \quad (4.50)$$

Likewise, a simple PD controller can be given by:

$$\mathbf{F}_d = \mathbf{K}_p\mathbf{e} + \mathbf{K}_d\dot{\mathbf{e}} \quad (4.51)$$

Although we have derived the equations for controlling and reasoning about the dynamics of the end-effector in workspace, it is worth mentioning that we are only able to command

joint torques of a robot manipulator. Therefore, the computed desired forces need to be converted to torque commands. This can be done by invoking the principle of virtual work. which allows to compute the final desired torques that are sent to the robot actuators via (see [Siciliano et al., 2010] for a derivation of this principle):

$$\boldsymbol{\tau}_d = \mathbf{J}^T(\boldsymbol{\theta})\mathbf{F}_d \tag{4.52}$$

We have now all the basic tools to control a typical robot manipulator. The next step is to explore motion synthesis approaches with applications to various tasks.

## 4.4 Stochastic optimal control

Under the typical assumption of deterministic dynamics, classical *optimal control* deals with finding a set of control commands that are optimal with respect to a cost function. The general framework considers a robot as an agent or an automaton that seeks to minimise a cost function for a fixed or varying time horizon [Kappen, 2011]. There are typically two methods to solve an optimal control problem. The first class of methods make use of the *Hamilton-Bellman-Jacobi* (HJB) formulation. A solution to the problem is found via *dynamic programming*. The second class of methods use the *Pontryagin's Minimum Principle* (PMP), which aim at finding a solution to the ordinary differential equations formed [Kirk, 2012]. These formulations are typically interested in finding a globally optimal solution.

The stochastic optimal control framework also creates an agent in a similar fashion but now the system addressed is subject to noise. Inclusion of noise in the HJB formulation is straight-forward compared to the PMP formulation. But, this class of problems is intractable [Kappen, 2011]. This is due to discretisation of the state-space.

In a variety of contexts, a first order approximation of the dynamics of a system (linear dynamics) is assumed, as well as the definition of a quadratic cost function (both in state and control variables) with linear dependence on zero mean white noise. Problems in which

these assumptions hold can be solved using the standard *linear-quadratric regulator* LQR method in closed form [Stengel, 2012]. A variation to this involving multiplicative noise was solved in [Todorov, 2005]. However, these assumptions are often quite restrictive for robotic applications [Kappen, 2011].

Alternatively, problems with non-linear dynamics, arbitrary state costs and quadratic control costs is solvable using sampling based approaches [Kappen, 2005]. These approaches make use of the exponentiated cost-to-go function that form a partial differential equation (PDE) that can be solved using *path integral* (PI) methods. In this context, the solution to a stochastic optimal control problem can be viewed as a weighted mixture of sub-optimal solutions [Kappen, 2011]. This weighting is sensitive to problem specific features, noise and time horizon (since these parameters affect the cost function).

## 4.5 Motion synthesis

In this thesis, we refer to motion synthesis as the process of generating movements or nominal trajectories that can be then followed by a robot manipulator. This can be generally achieved by means of motion planning algorithms, or alternatively via learning movement primitives from demonstration.

### 4.5.1 Motion planning

An essential problem in robot manipulation is to be able to plan collision free trajectories to avoid hitting obstacles or itself. Two fundamental concepts form the basis of the planning problem:

The concept of *configuration space*, or C-space for short, plays a major role in motion planning. As discussed in Section 4.1.1, the configuration space $\mathcal{C}$ corresponds to the space of all possible unique configurations $\boldsymbol{\theta} \in \mathcal{C}$ of a manipulator. The number $m$ of DoF of a robot corresponds to the dimensionality of its configuration space, thus $\mathcal{C} \subset \mathbb{R}^m$.

The *planning workspace* $\mathcal{W} \subset \mathbb{R}^d$ is also an important element. The planning workspace is an Euclidean space of dimension $d = 2$ for planar space or $d = 3$ for a three-dimensional space. It should not be confused with the manipulator workspace described in Section 4.1.2, which is the space of all possible of positions and orientations (poses) of an end-effector.

Let the configuration-dependent closed set $\mathcal{A}(\boldsymbol{\theta}) \subset \mathcal{W}$ represent the collection of occupied points by the robot at a given configuration $\boldsymbol{\theta} \in \mathcal{C}$. In addition, let the closed set $\mathcal{O} \subset \mathcal{W}$ be the *obstacle regions*, which is composed of a collection of static obstacles in the planning workspace. Both spaces $\mathcal{A}(\boldsymbol{\theta})$ and $\mathcal{O}$ are typically represented by geometric primitives, such as collections of polyhedra, three-dimensional triangles or analytic surfaces.

The *obstacle region* $\mathcal{C}_{obs}$ in C-space is defined as

$$\mathcal{C}_{obs} = \{\boldsymbol{\theta} \in \mathcal{C} | \mathcal{A}(\boldsymbol{\theta}) \cap \mathcal{O} \neq \emptyset\} \tag{4.53}$$

Conversely, the *free space* $\mathcal{C}_{free}$, the set of configurations that avoid collisions and is defined as

$$\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs} \tag{4.54}$$

Given those definitions, the *motion planning problem* is defined as follows [Siciliano and Khatib, 2008]. Given:

1. A *workspace* $\mathcal{W} \subset \mathbb{R}^d$ with $d = 2$ or $d = 3$.

2. An *obstacle region* $\mathcal{O} \subset \mathcal{W}$.

3. A robot $\mathcal{A}(\boldsymbol{\theta})$ defined in $\mathcal{W}$ dependent on the configuration $\boldsymbol{\theta} \in \mathcal{C}$.

4. An initial configuration $\boldsymbol{\theta}_I \in \mathcal{C}_{free}$.

5. A final goal configuration $\boldsymbol{\theta}_G \in \mathcal{C}_{free}$.

Compute a continuous path $\boldsymbol{\tau} : [0,1] \mapsto \mathcal{C}_{free}$, such that $\boldsymbol{\tau}(0) = \boldsymbol{\theta}_I$ and $\boldsymbol{\tau}(1) = \boldsymbol{\theta}_G$.

A classical and simple instance of the path planning problem is known as the *piano mover's problem* [Reif, 1979], and has been shown to be PSPACE-complete.

#### 4.5.1.1 Sampling-based planning

Sampling-based planning algorithms are methods of choice for problems with many degrees of freedom, such as multi-DOF articulated robot arms. They make use of recent advancements in collision detection techniques that allow to determine if $\boldsymbol{\theta} \in \mathcal{C}_{free}$.

A sampling-based planner does not have complete access to $\mathcal{C}_{free}$, they can only query it by means of a collision detector. Such approaches are able to sample different configurations and construct a so called *roadmap* data structure. The constructed roadmap represents an approximation to $\boldsymbol{\tau} : [0,1] \mapsto \mathcal{C}_{free}$. Although sampling-based planners do not guarantee that a collision-free path will be found in finite amount of time, they overcome real-world problems that are impractical for complete approaches [Siciliano and Khatib, 2008].

The constructed roadmap is typically represented by a undirected graph $\mathbf{G}(\mathbf{V},\mathbf{E})$, where the collection of vertices $\mathbf{V}$ is the set of sampled configuration in workspace $\mathcal{W}$ and the set of edges $\mathbf{E}$ consists of collision-free paths connecting two vertices. The different ways of constructing $\mathbf{G}(\mathbf{V},\mathbf{E})$ distinguish themselves into two main categories of approaches:

*Single-query planners* construct a new graph $\mathbf{G}(\mathbf{V},\mathbf{E})$ for every performed planning query. In order to make the construction of $\mathbf{G}(\mathbf{V},\mathbf{E})$ computationally efficient, they typically attempt to minimise how much of $\mathcal{C}$ is explored.

*Multi-query planners* construct a complete graph $\mathbf{G}(\mathbf{V},\mathbf{E})$ at the very beginning of a particular task. The constructed roadmap represents an approximation of $\mathcal{C}_{free}$. Subsequent planning queries do not require the reconstruction of $\mathbf{G}(\mathbf{V},\mathbf{E})$, which assumes all obstacles in the workspace are static.

**Single-query planners**

An example of single-query planner is the Rapid-Exploring Random Trees (RRT) approach proposed by Lavalle et al. [2000]. The idea is to construct the graph $\mathbf{G}(\mathbf{V},\mathbf{E})$ as a tree starting from the initial configuration $\boldsymbol{\theta}_I \in \mathcal{C}_{free}$ and is expanded according to the following procedure:

**1. Initialisation:** initialise $\mathbf{G}(\mathbf{V},\mathbf{E})$ with an empty set of edges $\mathbf{E}$ and include $\boldsymbol{\theta}_I \in \mathcal{C}_{free}$ as an initial vertex in $\mathbf{V}$.

**2. Vertex selection:** let $\boldsymbol{\theta}_{curr}$ be a currently selected vertex from $\mathbf{V}$.

**3. Local path construction:** select a $\boldsymbol{\theta}_{new} \in \mathcal{C}_{free}$ which may or may not be already in $\mathbf{V}$, and try to construct a local path $\boldsymbol{\tau}_{local} : [0,1] \mapsto \mathcal{C}_{free}$, such that $\boldsymbol{\tau}_{local}(0) = \boldsymbol{\theta}_{curr}$ and $\boldsymbol{\tau}_{local}(1) = \boldsymbol{\theta}_{new}$ using a collision detection method. In case of failure, return to step 1.

**4. Edge insertion:** if $\boldsymbol{\tau}_{local}$ exists, then insert it to the set of edges $\mathbf{E}$.

**5. Terminating condition:** if goal has been reached in $\mathbf{G}(\mathbf{V},\mathbf{E})$ then terminate with success, otherwise return to step 2. If some other termination condition has been reached, then terminate with failure.

### Multi-query planners

*Probabilistic Roadmap* (PRM) approaches have been independently devised by various authors [Amato and Wu, 1996; Kavraki and Latombe, 1994; Overmars, 1992] and are an instance of multi-query planning. The goal here is to construct a roadmap represented as a graph $\mathbf{G}(\mathbf{V},\mathbf{E})$ which approximates $\mathcal{C}_{free}$. The life-cycle of a PRM planner has two stages: i) graph construction and ii) query stage. The procedure to construct $\mathbf{G}(\mathbf{V},\mathbf{E})$ can be summarised as follows [Siciliano and Khatib, 2008]:

**Graph construction**

1. **Initialisation:** initialise $\mathbf{G}(\mathbf{V},\mathbf{E})$ as an empty graph.

2. **Sampling of configurations:** sample a configuration a $\boldsymbol{\theta}_s \in \mathcal{C}_{free}$ and add it to $\mathbf{V}$.

3. **Neighboring computation:** let $\rho : \mathcal{C} \times \mathcal{C} \mapsto \mathbb{R}$ be a distance metric between two configurations. Using $\rho(\cdot,\cdot)$ find all neighboring vertices to $\boldsymbol{\theta}_s$ in $\mathbf{G}(\mathbf{V},\mathbf{E})$, such that their are close enough.

4. **Local path construction:** for all $\boldsymbol{\theta}$ that are neighbours of $\boldsymbol{\theta}_s$, attempt to construct a local path $\boldsymbol{\tau}_{local} : [0,1] \mapsto \mathcal{C}_{free}$, such that $\boldsymbol{\tau}_{local}(0) = \boldsymbol{\theta}_s$ and $\boldsymbol{\tau}_{local}(1) = \boldsymbol{\theta}$ using a collision detection method.

5. **Edge insertion:** insert new edge $\boldsymbol{\theta}_s$ to $\mathbf{E}$.

6. **Terminating condition:** if enough vertices have been added to the roadmap, then terminate.

Once the roadmap is constructed, new collision free paths can be queried at the query stage via the following procedure:

**Query stage**

1. **Initialisation:** add the inital and goal configurations $\boldsymbol{\theta}_I, \boldsymbol{\theta}_G \in \mathcal{C}_{free}$ as vertices of $\mathbf{G}(\mathbf{V},\mathbf{E})$.

2. **Graph search:** find a path from $\boldsymbol{\theta}_I$ to $\boldsymbol{\theta}_G$ in $\mathbf{G}(\mathbf{V},\mathbf{E})$ using a graph search algorithm, such as Dijkstra or A*.

### 4.5.2 Dynamic movement primitives

Dynamic movement primitives (DMPs) are a set of differential equations for representing motion, which are typically acquired from demonstration. This approach has been proposed by Schaal [2006], and many robotic applications take advantage of this representation for tackling a wide range of manipulation problems. The idea here is to define a system of differential equations to describe arbitrary trajectories. We shall follow intuitive steps in order to construct this system of differential equations incrementally.

We start by writing down the equation for a simple 1-dimensional spring-damper system:

$$\ddot{\theta} = K_p(g - \theta) - K_d\dot{q} \tag{4.55}$$

where $\theta, g, K_p, K_d \in \mathbb{R}$.

This system has two components that contribute to a particle acceleration, a spring force (recall Hook's law), and a damper (analogous to wind drag, or other instances of fluid resistance).

Note that this is a second order *ordinary differential equation* (ODE). It is ordinary because it has only one independent variable ($\theta$ and its time derivatives $\dot{\theta}$, $\ddot{\theta}$), and is second order because the highest derivative order is 2, i.e. $\ddot{\theta}$. This dynamical system describes a global attractor field to a given goal $g$. It means that the behaviour of a particle dropped in a system under the actions of the forces described in Eq 4.55 will be pulled toward $g$, but also damped (preventing it of going too fast), analogous to a dragging effect.

Also note that our system in Eq 4.55 is not being yet forced to follow a particular trajectory of our choice towards $g$. Thus, what we will do next is to add to our equation an additional term that generates a varying force at different steps. The introduction of this *forcing function* allows us to follow arbitrary trajectories towards the goal $g$. We also introduce a phase variable to rid us from a direct dependency on time. Our system is then rewritten as

$$\begin{cases} \ddot{\theta} = K_p(g-\theta) - K_d\dot{\theta} + \overbrace{f(x;\mathbf{w})}^{\text{forcing function}} \\ \underbrace{\dot{x} = -\beta x}_{\text{Canonical system, or phase variable as opposed to directly depending on time}} \end{cases} \tag{4.56}$$

Where $f(x)$ is typically a function that can be learnt through a variety of non-linear function approximation techniques. For instance, it can be modelled using neural networks (NNs) [ref], Gaussian Processes (GPs) Rasmussen [2006], Support Vector Machines (SVMs) [ref], kernel regression [ref], etc. Here we choose $f(\cdot;\mathbf{w})$ to be a linear combination of radial-basis functions multiplied by the introduced phase variable, i.e.:

$$f(x;\mathbf{w}) = \frac{\sum_{i=0}^{T} w_i \psi_i(x)}{\sum_{i=0}^{T} \psi_i(x)} x, \tag{4.57}$$

and

$$\psi_i(x) = \exp(-h(x-c_i)^2)) \tag{4.58}$$

Now, pay attention to the new equation added to Eq 4.56, the phase variable. That allows us to make the forcing term a function of the so-called canonical system $x$, instead of being directly a function of time. We do that because making the system dependent directly on time can be a bit problematic. For example, we would have to play out trajectories always in the same time interval from which they were originally recorded, but that is virtually unnecessary. Furthermore, we would like the influence of our forcing function to vanish as $t \to T$, thus ensuring global convergence to the goal $g$ (we will not prove convergence properties just yet, but the interested reader can refer to [Schaal, 2006]). Therefore, we constructed the equation for the phase variable $x$ such that its solution is a decaying exponential function [make it more intuitive].

Our next step is to tide things up a bit by *reducing* our second order ODE system into a first order one:

$$\begin{cases} \dot{\theta} = \overbrace{v}^{\text{new variable}} \\ \dot{v} = K_p(g-\theta) - K_d v + f(x;\mathbf{w}) \\ \dot{x} = -\beta x \end{cases} \tag{4.59}$$

Note again that $\dot{x} = \frac{d}{dt}x = -\beta x$, which is solved by directly integrating both sides, yielding the solution: $x = \exp(-\beta t) + C$.

Now, we would like to be able to scale the system temporally and spatially. To achieve temporal scaling, we can simply multiply the response of our attractor system by a factor $\alpha$ [2]:

$$\begin{cases} \alpha\dot{\theta} = v \\ \alpha\dot{v} = K_p(g-\theta) - K_d v + f(x;\mathbf{w}) \\ \alpha\dot{x} = -\beta x \end{cases} \tag{4.60}$$

Note the solution to our canonical system now becomes $x = \exp(-\frac{\beta}{\alpha}t) + C$. It is also now clearer to verify by inspection that $\alpha$ is indeed scaling our time dimension.

Finally, to achieve spatial scaling we will modify our forcing function, scaling it by a factor $\Delta s = g - \theta_0$, i.e. the total offset from start to goal. Our forcing function will look as follows:

$$f(x) = \frac{\sum_{i=0}^{T} w_i \psi_i(x)}{\sum_{i=0}^{T} \psi_i(x)} x \Delta s, \tag{4.61}$$

#### 4.5.2.1 Learning dynamic movement primitives

We now have all the basic equations we need to describe the learning algorithm for DMPs. Given a nominal demonstrated trajectory $\theta^d_{0:T} = [\theta^d_0, \theta^d_1, \ldots, \theta^d_T]$, where $\theta^d_t \in \mathbb{R}$, we want to fit the forcing function to reproduce this motion. We also need to compute, usually

[2] We can intuitively understand this by noting that we are simply scaling the integration time step if we were to solve this system of equations numerically (and we actually will do just that).

**Algorithm 5** Learning and generating a DMP

**Given**:
$\theta^d_{0:T}, \dot{\theta}^d_{0:T}, \ddot{q}^d_{0:T}$: nominal demonstrated trajectory $\theta^d_{0:T} = [\theta^d_0, \theta^d_1, \ldots, q^d_T]$ and its first and second order derivatives $\dot{\theta}^d_{0:T}$ and $\ddot{\theta}^d_{0:T}$;
$\Delta t$: integration step;
$\alpha$: temporal scaling coefficient;
$g = \theta^d_T$: set goal as the state at last time step;
$\Delta s$: spatial scaling, could be set to $\Delta s = g - \theta_0$;
$\beta$: canonical system scaling factor;
$K_p$: spring coefficient;
$K_d$: damping coefficient;

// Numerical solution for phase variable
// Alternatively, this step can be substituted by the closed form solution:
// $x(t) = \exp(-\frac{\beta}{\alpha}t) + C$.

$x_0 = 1$ ▷ Initial condition for phase variable, it decays from 1
**for** $t = 0$ **to** $T-1$ **do**
  $x_{t+1} = x_t + \dot{x}(t)\Delta t$ ▷ Note $\dot{x}(t) = -\frac{\beta}{\alpha}\exp(-\frac{\beta}{\alpha}t)$
**end for**

**for** $t = 0$ **to** $T$ **do**
  $y_t = \frac{\alpha^2 \ddot{\theta}^d_t - K_p(g-\theta^d_t) + \alpha K_d \dot{q}^d_t}{x_t \Delta s}$ ▷ Targets for each time step $t$
**end for**

Solve $\overset{*}{\mathbf{w}} = \arg\min_{\mathbf{w}\in\mathbb{R}^N} J(\mathbf{w})$ ▷ As defined in Eq 4.63

// Generate trajectory using $\overset{*}{\mathbf{w}}$

$\theta_0 = \theta^d_0$, $\dot{\theta}_0 = \dot{\theta}^d_0$
**for** $t = 0$ **to** $T$ **do**
  $\ddot{\theta}_t = \frac{K_p(g-\theta_t) - K_d\alpha\dot{\theta} + f(x_t;\overset{*}{\mathbf{w}})}{\alpha}$ ▷ Rearranged Eq 4.60
  $\dot{\theta}_{t+1} = \dot{\theta}_t + \ddot{\theta}_t \Delta t$ ▷ Output velocity trajectory
  $\theta_{t+1} = \theta_t + \dot{\theta}_t \Delta t$ ▷ Output trajectory
**end for**

numerically, the first and second order derivatives of this given trajectory, yielding $\dot{\theta}^d_{0:T}$ and $\ddot{\theta}^d_{0:T}$. The target points for our forcing function can be found by rearranging Eq 4.60;

performing the adequate substitutions, we can verify that

$$y_t = \frac{\alpha^2 \ddot{\theta}_t^d - K_p(g - \theta_t^d) + \alpha K_d \dot{\theta}_t^d}{x_d \Delta s} \tag{4.62}$$

Eq 4.62 provides our targets for learning the *forcing function*. This way we can find the appropriate parameters for $f(\cdot; \mathbf{w})$ by defining the following objective function to be minimised:

$$J(\mathbf{w}) = \sum_{t=0}^{T} (y_t - f(x_t; \mathbf{w}))^2 \tag{4.63}$$

We want to find $\overset{*}{\mathbf{w}}$ such that:

$$\overset{*}{\mathbf{w}} = \arg\min_{\mathbf{w} \in \mathbb{R}^N} J(\mathbf{w}) \tag{4.64}$$

Subsequently solving this least-squares problem gives us the set of weights $\mathbf{w} = [w_0, \ldots, w_N] \in \mathbb{R}^N$ that encode our demonstration. Fitting the given choice of parametric *forcing function* is a linear regression problem, and can be solved very efficiently. A pseudocode for learning and generating DMPs is outlined by Alg 5.

## 4.6 Summary

This chapter introduced the fundamental notions underlying design and control for robot manipulators.

# Chapter 5

# Generative grasp synthesis from demonstration using parametric mixtures

**In Chapter 2.1** we have identified several limitations pertinent to analytic grasp synthesis. We have seen that these approaches rely on force analysis (FA) and geometric constraints to generate grasps according to the laws of classical mechanics [Bicchi and Kumar, 2000; Curtis and Xiao, 2008]. However, accurate knowledge of quantities such as friction, mass distribution and inertia for both objects and robotic hands is rarely available. These are also non-trivial quantities to estimate for arbitrary objects in unstructured set-ups. Even if system identification was possible, the range of assumptions made in analytic methods do not seem to hold in reality. There has been evidence suggesting that analytic-based metrics may not be strong indicative of grasp success in real scenarios [Balasubramanian et al., 2012; Bekiroglu et al., 2011; Bohg et al., 2014; Goins et al., 2014; Kim et al., 2013]. This has led purely analytic methods to lose popularity over data-driven approaches that sought to overcome these shortcomings.

We have also seen that considerable effort has been put towards tackling the problem of grasping from a data-driven perspective. These approaches often learn a discriminative function to predict grasp quality or a direct mapping from sensory information to grasp parameters in the form of a generative model [Ben Amor et al., 2012; Bicchi and Kumar, 2000; Bohg et al., 2011; Curtis and Xiao, 2008; Detry and Piater, 2013; Gualtieri et al., 2016; Hjelm et al., 2014; Kopicki et al., 2015; Levine et al., 2016; Rietzler et al., 2013; Saxena et al., 2008b; Shimoga, 1996; ten Pas and Platt, 2015].

**Discriminative approaches**

Discriminative approaches for grasp synthesis usually require large amounts of training data. Very often, they are tailored to specific types of end-effectors, such as parallel-jaw grippers. Being data-intensive and specific to a particular type of robotic hand are limiting factors that may constrain the application of such methods in combination with perception strategies, a problem that we shall address later in Chapter 5.

**Generative approaches**

Relatively less work has been done on generative models for grasp synthesis as noted by [Bohg et al., 2014]. Although some recent methods may still be data-intensive and specific to parallel-jaw grippers, other approaches learn generative models from human demonstrations and are both data-efficient and general to multi-fingered robotic hand [Ben Amor et al., 2012; Detry et al., 2011; Kopicki et al., 2015; Montesano et al., 2008; Song et al., 2011].

**In Chapter 3** we described the basic tools for approximating probability densities, either for the purposes of predcition (discriminative models) or for generating data (generative models). We will elaborate on these techniques applied for learning to synthesise grasps from demonstration.

In this chapter, we address the problem of designing generative models for grasp synthesis using parametric representations for probability densities. The presented parametric method is more computationally efficient than its non-parametric counterpart. In addition, the following additional contributions are made: (i) A novel algorithm for real-time grasp synthesis, (ii) An open-source framework for generative grasp synthesis, (iii) A sensorisation method for the under-actuated Pisa/IIT SoftHand, allowing the described grasp synthesis approaches to be deployed for this type of robotic hand.

Parametric mixtures are more concise representations of probability densities and present practical gains in time complexity when evaluating likelihood over a set of query points. This contrasts with the KDE-based methods such as the one utilised by [Kopicki et al., 2015]. For instance, evaluating the likelihood of $N$ data points under a KDE model with $K$ kernels has a time complexity of $O(KN)$. While the time complexity is the same for a parametric model such as a GMM, the number of kernels $K$ is typically very large for KDEs, in the order of hundreds, as commonly a kernel is placed on every training data point. Hence, likelihood evaluation for KDEs has time complexity given as a function of the training data used to approximate a given density. In contrast, a parametric mixture requires fewer kernels, thus a smaller number for $K$ to approximate a probability density. Furthermore, likelihood evaluation is no longer a function of the training data set size. Now, it depends only on the fixed number chosen for $K$. Thus, although the complexity for data likelihood computation is the same in principle for both methods, in practice parametric mixtures can be orders of magnitude faster than KDEs since $K$ is rarely greater than 10 for most scenarios.

As opposed to following a non-parametric approach, we will describe algorithms for grasp synthesis using a parametric mixture formulation for density modelling. We show the benefits for grasp generation time and performance in simulation under different sensor noise conditions, also validate the approach on different robot platforms.

The approaches described in this chapter are part of a general framework for generative grasp synthesis, released as an openly available library module. The framework is openly available[1] and comprises a technical contribution of this thesis.

First, we proceed to describe the basic representations utilised throughout this chapter for modelling probability densities, followed by the description of our approach. We also describe an apporach for sensorising the Pisa/IIT SoftHand, an under-actuated humanoid hand. The sensorisation was required to allow the methods described in this chapter to be applied for this type of robotic hand. Finally, we present our experimental results in simulation and validate the approach and developed framework on different robot platforms.

## 5.1 Representations

We are interested in modelling joint probability densities over rigid body transformations belonging to the $SE(3)$ and arbitrary feature vectors in $\mathbb{R}^n$. Thus, realisations of the random variable we intend to model lie in the space $\mathbf{x} \in SE(3) \times \mathbb{R}^n$. Concretely, the special Euclidean group is the cartesian product of members of the special orthogonal group $SO(3)$ (the group of rotations) and members of $\mathbb{R}^3$ representing translations, thus $SE(3) = \mathbb{R}^3 \times SO(3)$ (see Appendix A). If we choose to represent rotations with unit quaternions, then $\mathbf{x} = (\mathbf{p}, \mathbf{q}, \mathbf{r})$ is a $d = 7 + n$ dimensional vector, where $\mathbf{p} \in \mathbb{R}^3$, $\mathbf{q} \in \mathbb{R}^4$ and $\mathbf{r} \in \mathbb{R}^n$. In this work, we utilise two-dimensional feature vector represented formed by the eigen-values $k_1$ and $k_2$ of the principal curvatures for a given point $\mathbf{p}$ from an object point cloud. In Fig. 5.1 this type of feature is illustrated. Thus, concretely, the feature vector is represented as $\mathbf{r} = [k_1, k_2] \in \mathbb{R}^2$. We will see that we are able to learn probability densities over this manifold using data acquired from demonstration where a data set $\mathcal{D}$ is acquired, such that

$$\mathcal{D} = \{\mathbf{x_j} | \mathbf{x_j} \in \mathbb{R}^3 \times SO(3) \times \mathbb{R}^n\}_{j=1}^{J_D} \tag{5.1}$$

[1]The AML generative grasp synthesis framework is available at http://github.com/eaa3/aml_grasp

where $\mathbf{x}_j \sim \mathbf{pdf}(\mathbf{p}, \mathbf{q}, \mathbf{r})$.

In particular, we will represent quaternions noting their relationship with angular velocities $\boldsymbol{w} \in \mathbb{R}^3$ through the logaritimic and exponential maps defined by Eq. 5.2 and 5.3, as similarly noted by [Ude et al., 2014]. Given a unit quaternion $\mathbf{q} = [\mathbf{q}_v, q_w]^T$, we define its logarithmic map as:

$$\boldsymbol{\omega} = \log(\mathbf{q}) = \begin{cases} \arccos(q_w)\frac{\mathbf{q}_v}{\|\mathbf{q}_v\|} & \mathbf{q}_v \neq \mathbf{0} \\ [0,0,0]^T & otherwise \end{cases} \tag{5.2}$$

And conversely, the exponential map is given by:

$$\mathbf{q} = \exp(\boldsymbol{\omega}) = \begin{cases} [\sin(\|\boldsymbol{\omega}\|)\boldsymbol{\omega}, \cos(\|\boldsymbol{\omega}\|)]^T & \boldsymbol{\omega} \neq \mathbf{0} \\ [0,0,0,1]^T & otherwise \end{cases} \tag{5.3}$$

Using these mappings, we will fit parametric mixtures to model joint distributions over $\mathbf{x} = (\mathbf{p}, \boldsymbol{\omega}, \mathbf{r})$, where the quaternion representation for $\boldsymbol{\omega}$ is readily given by the exponential map $\mathbf{q} = \exp(\boldsymbol{\omega})$.

### 5.1.1 Probability density approximation using parametric mixtures

The density $\mathbf{pdf}(\mathbf{x})$ is approximated using a parametric Gaussian mixture model (GMM) modified so as to internally take care of the appropriate exponential and logarithmic mappings for quaternions. It is then defined as follows:

$$\mathbf{pdf}(\mathbf{x}) = \sum_{j=1}^{K} w^j \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{\boldsymbol{x}}^j, \boldsymbol{\Sigma}_{\boldsymbol{x}}^j) \tag{5.4}$$

For convenience, let $\mathbf{u} = (\mathbf{p}, \mathbf{q})$, i.e. a rigid body transformation. Thus, with $\boldsymbol{\Sigma}_{\boldsymbol{x}}^{j^{-1}} = \boldsymbol{\Lambda}_{\boldsymbol{x}}^{j}$, we denote:

$$\boldsymbol{\mu}_{\boldsymbol{x}}^{j} = (\boldsymbol{\mu}_{\boldsymbol{u}}^{j}, \boldsymbol{\mu}_{r}^{j}), \tag{5.5a}$$

$$\boldsymbol{\Sigma}_{\boldsymbol{x}} = \begin{pmatrix} \boldsymbol{\Sigma}_{uu}^{j} & \boldsymbol{\Sigma}_{ur}^{j} \\ \boldsymbol{\Sigma}_{ur}^{j} & \boldsymbol{\Sigma}_{rr}^{j} \end{pmatrix}, \boldsymbol{\Lambda}_{\boldsymbol{x}}^{j} = \begin{pmatrix} \boldsymbol{\Lambda}_{uu}^{j} & \boldsymbol{\Lambda}_{ur}^{j} \\ \boldsymbol{\Lambda}_{ur}^{j} & \boldsymbol{\Lambda}_{rr}^{j} \end{pmatrix} \tag{5.5b}$$

In this work the density $p(\mathbf{u}|\mathbf{r})$ is a conditional probability density of rigid body transformations given a contact feature $\mathbf{r}$, and is also modelled as a Gaussian mixture model. The parameters of this conditional mixture are obtained in closed form from Eq. 5.4 as (see [Bishop and Nasrabadi, 2007] for an overview):

$$\boldsymbol{\mu}_{u|r}^{j} = \boldsymbol{\mu}_{\boldsymbol{u}}^{j} - \boldsymbol{\Sigma}_{uu}^{j} \boldsymbol{\Lambda}_{ur}^{j} (\mathbf{r} - \boldsymbol{\mu}_{\boldsymbol{r}}^{j}), \tag{5.6a}$$

$$\boldsymbol{\Sigma}_{u|r}^{j} = \boldsymbol{\Sigma}_{uu}^{j} \tag{5.6b}$$

$$p^{j}(\mathbf{u}|\mathbf{r}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{u|r}^{j}, \boldsymbol{\Sigma}_{u|r}^{\boldsymbol{j}}) \tag{5.6c}$$

$$\pi^{j} = \frac{w^{j} \mathcal{N}(\mathbf{r}|\boldsymbol{\mu}_{\boldsymbol{r}}^{j}, \boldsymbol{\Sigma}_{rr}^{j})}{\sum_{j=1}^{K} \mathcal{N}(\mathbf{r}|\boldsymbol{\mu}_{\boldsymbol{r}}^{j}, \boldsymbol{\Sigma}_{rr}^{j})} \tag{5.6d}$$

Therefore, $p(\mathbf{u}|\mathbf{r})$ is expressed as

$$p(\mathbf{u}|\mathbf{r}) = \sum_{j=1}^{K} \pi^{j} p^{j}(\mathbf{u}|\mathbf{r}) \tag{5.7}$$

The final density over features $\mathbf{r}$, i.e. $p(\mathbf{r})$, is also modelled as a Gaussian mixture and is obtained via marginalisation of Eq. 5.4 with respect to $\mathbf{u}$ such that:

$$p^{j}(\mathbf{r}) = \int \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{\boldsymbol{x}}^{j}, \boldsymbol{\Sigma}_{\boldsymbol{x}}^{j}) du = \mathcal{N}(\mathbf{r}|\boldsymbol{\mu}_{\boldsymbol{r}}^{j}, \boldsymbol{\Sigma}_{\boldsymbol{r}}^{j}) \tag{5.8}$$

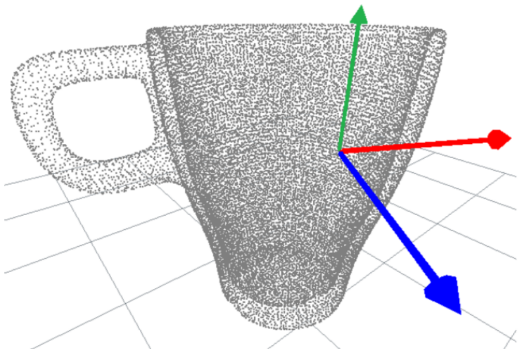

Fig. 5.1 An example point cloud of a mug and feature representation. For a given point $\mathbf{p}$, the axis in blue is the surface normal pointing outwards the object, the direction of the first principal curvature $k_1$ is depicted as the horizontal red axis, and the direction of the second principal curvature $k_2$ is depicted as the green vertical axis. This right-handed frame is constructed by taking the cross-product between the normal and the first principal curvature direction. Together with the point $\mathbf{p}$, these three axis define a rigid body pose, whose position and quaternion representation are given by $\mathbf{v} = (\mathbf{p}, \mathbf{q})$. Therefore, each point of the object has a rigid frame attached in similar fashion, although in the picture we chose to highlight only one to avoid clutter.

Thus, it follows that

$$p(\mathbf{r}) = \sum_{j=1}^{K} w^j p^j(\mathbf{r}). \tag{5.9}$$

The mixture in Eq. 5.4 is learned using the expectation maximisation algorithm (EM) over contact data acquired from a grasp demonstration $\mathcal{D} = \{\mathbf{x_j} | \mathbf{x_j} \in \mathbb{R}^3 \times SO(3) \times \mathbb{R}^n\}$. The mixture formulation is more compact and less memory hungry. Once the density is learnt, we only need to keep its learnt parameters in memory and compute in closed form the additional probability densities in Eq. 5.9 and Eq. 5.7. In addition, it needs fewer kernels to approximate the densities, therefore $K$ is typically smaller than 10. In contrast to the KDE approach, each data point becomes a the center of a kernel, which is of the order of hundreds per contact model learnt.

## 5.2 Grasp synthesis using parametric contact models

Having defined the basic representations for learning distributions using parametric mixtures, we proceed to describe the main components of what we refer to as the parametric grasp synthesis approach.

We will be learning joint probability densities over $\mathbf{x} = (\mathbf{p}, \mathbf{q}, \mathbf{r})$. For compactness we will refer to the position and orientation (pose) of a feature $\mathbf{r}$ as $\mathbf{v} = (\mathbf{p}, \mathbf{q})$, and therefore $\mathbf{x}$ can be written as

$$\mathbf{x} = (\mathbf{v}, \mathbf{r}), \tag{5.10}$$

where $\mathbf{v} \in SE(3)$ and $\mathbf{r} \in \mathbb{R}^n$ is a feature vector.

### 5.2.1 The object representation

At all times, the object intended to be grasped is represented by $O(\mathbf{v}, \mathbf{r})$. Concretely, it consists of the current object point cloud augmented with features $\mathbf{r} = [k_1, k_2]$ representing the principal curvatures at each point of the object. Finally, for a given point $\mathbf{p}$ from the point cloud, using corresponding eigenvectors of the principal curvatures and estimated normal at this given point, one can construct a frame $\mathbf{v} = (\mathbf{p}, \mathbf{q})$. Hence, the object point cloud is then augmented to be a cloud of rigid body transformations associated with a features $\mathbf{r}$ as depicted in Fig. 5.1.

### 5.2.2 Contact model

For a given hand link $L_i$, the conditional contact model density $M_i(\mathbf{u}|\mathbf{r})$ is modelled as a conditional probability density over finger link poses $\mathbf{u} \in SE(3)$ given contact surface features $\mathbf{r} \in \mathbb{R}^n$. This density is proportional to the likelihood of finding a finger link located at $\mathbf{u}$ with respect to the location $\mathbf{v}$ of a contact point with feature vector $\mathbf{r}$. Such model is constructed

from the dataset $\mathcal{D}_O = \{\mathbf{x}_j\}_{j=1}^{K_O}$ as follows. Let $\mathbf{s}_i = (\mathbf{p}_i, \mathbf{q}_i) \in SE(3)$ be the pose of the link $L_i$ in world frame. The relative pose of link $L_i$ with respect to a feature pose $\mathbf{v}_j$ is given by

$$\mathbf{u}_{ij} = (\mathbf{p_{ij}}, \mathbf{q_{ij}}) = \mathbf{v}_j^{-1} \circ \mathbf{s}_i \tag{5.11}$$

Thus we are able to construct the contact model representing a probability distribution of relative poses with respect to contact features $\mathbf{r}_j$ as

$$M_i(\mathbf{u}|\mathbf{r}) \approx p(\mathbf{u}|\mathbf{r}) \tag{5.12}$$

where $p(\mathbf{u}|\mathbf{r})$ is defined in Eq. 5.7 and is obtained via conditioning Eq. 5.4 learnt using the demonstration data set $\mathcal{D}_O$.

Furthermore, it is worth noting that $M_i$ is the contact model for link $L_i$ in proximity during demonstration to a set of features modelled by

$$M_i(\mathbf{r}) \approx p(\mathbf{r}) \tag{5.13}$$

where $M_i(\mathbf{r})$ is obtained via marginalisation of Eq. 5.4 as described by Eq. 5.8. The mixture in Eq. 5.4 is learnt with EM using the demonstration data set $\mathcal{D}_O = \{\mathbf{x}_j\}_{j=1}^{K_O}$, a data set consisting of data points $\mathbf{x}_j = (\mathbf{v}_j, \mathbf{r}_j)$ in proximity to finger link $L_i$.

Let $(\tilde{\mathbf{v}}, \tilde{\mathbf{r}}) \sim O(\mathbf{v}, \mathbf{r})$ be a sample from the object intended to be grasped represented by $O(\mathbf{v}, \mathbf{r})$. It follows that if were to compute the data likelihood of features $\tilde{\mathbf{r}}$ over $O(\mathbf{v}, \mathbf{r})$, then features with high data likelihood reflect higher affinity that this finger link $L_i$ could be positioned over $\tilde{\mathbf{r}}$ located at $\tilde{\mathbf{v}}$. The location of this finger link in world frame is given by $\tilde{\mathbf{s}} = \tilde{\mathbf{v}} \circ \tilde{\mathbf{u}}$, where $\tilde{\mathbf{u}} \sim M_i(\mathbf{u}|\tilde{\mathbf{r}})$, together with its weight representing this affinity $\tilde{w} = M_i(\tilde{\mathbf{r}})$, forming the n-uple $(\tilde{\mathbf{s}}, \tilde{w})$.

Depending on the probability distribution of features during demonstration, different finger links show different affinities, thus modelling different contact distributions. Figure 5.2 depicts this property for two different links of a WSG 50 Schunk gripper. For instance, in Fig.

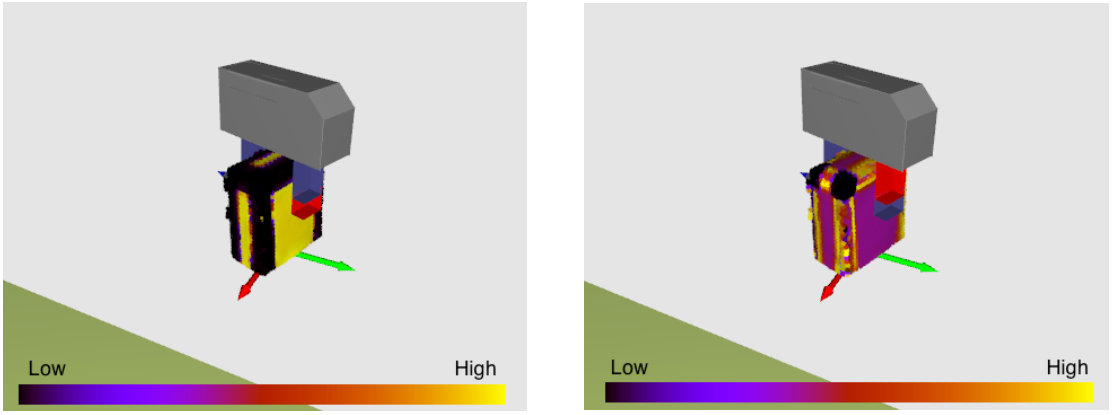


Fig. 5.2 Left: contact model distribution for finger link $L_2$. This finger link has higher affinity to flat surfaces of the object. Right: contact model distribution for finger link $L_1$. This finger link has higher affinity to edge surfaces of the object.

5.2 (left), the link number $L_2$ is closer to flat patches of surface. By evaluating $M_2(\tilde{\mathbf{r}})$ over $(\tilde{\mathbf{v}}, \tilde{\mathbf{r}}) \sim O(\mathbf{v}, \mathbf{r})$, one is able to determine the regions of high and low data likelihood/affinity over the object surface represented by $O(\mathbf{v}, \mathbf{r})$, which in this case is a soup box. In Fig. 5.2 (right), the same process is repeated this time for $L_1$ . Note that $L_2$ prefers flat surfaces, whereas $L_1$ prefers corners, due to its closer proximity to features describing the edges of the object during demonstration. The features we chose to utilise are the same features described by [Kopicki et al., 2015], which are the principal axis of curvature, computed using a PCA method. Thus, each feature $\mathbf{r} \in \mathbb{R}^2$ is a two dimensional vector containing the two largest eigen values approximating the curvatures of a given point of the object point cloud.

### 5.2.3 Hand configuration model

A probability density constructed from a demonstrated grasp trajectory containing starting and final joint configurations of the robot hand. Here, this model is constructed as originally proposed by [Kopicki et al., 2015]. Denoting $\mathbf{h}_e^t$ as the joint angles at the start of a given demonstrated grasp (pre-grasp configuration), and $\mathbf{h}_g^t$ the joint angles at contact with the object. Finally, let the set of hand configurations be $\mathcal{H}_c = \{\mathbf{h}(\gamma) : \gamma \in [-\beta, \beta], \beta \in \mathbb{R}^+\}$,

where $\mathbf{h}(\gamma) = (1-\gamma)\mathbf{h}_g^t + \gamma\mathbf{h}_e^t$, the probability density $C(\mathbf{h}_c)$ is then non-parametrically approximated from this set via a KDE as

$$C(\mathbf{h}_c) = \sum_{\gamma \in [-\beta, \beta]} w(\mathbf{h}_c(\gamma)) \mathcal{N}_D(\mathbf{h}_c | \mathbf{h}_c(\gamma), \sigma_c) \tag{5.14}$$

where $w(\mathbf{h}_c(\gamma)) = \exp(-\alpha \| \mathbf{h}_c(\gamma) - \mathbf{h}_g^t \|)$ and $\alpha \in \mathbb{R}^+$.

## 5.2.4 Grasp synthesis for novel objects

### 5.2.4.1 Contact query density

From the learnt densities described above, given a query point cloud for a new object in its augmented representation $O(\mathbf{v}, \mathbf{r})$, a so called contact query density $Q_i(\mathbf{s})$ can be constructed for each link $L_i$. This probability density is constructed by first sampling $(\tilde{\mathbf{v}}, \tilde{\mathbf{r}}) \sim O(\mathbf{v}, \mathbf{r})$ from the object intended to be grasped. Next, $\tilde{\mathbf{u}}_i \sim M_i(\mathbf{u}|\mathbf{r})$ is sampled from our learnt contact model for finger link $L_i$ in Eq. 5.12, such that a set of $N_{Q_i}$ samples $\tilde{\mathbf{s}}_{ij} = \tilde{\mathbf{v}}_j \circ \tilde{\mathbf{u}}_i = (\tilde{\mathbf{p}}_{ij}, \tilde{\mathbf{q}}_{ij})$ is constructed $\mathcal{D}_{Q_i} = \{\tilde{\mathbf{s}}_{ij}\}_{j=1}^{N_{Q_i}}$ and used to fit a query density defined as

$$Q_i(\mathbf{s}) \approx \sum_{k=1}^{K_{Q_i}} w_k \mathcal{N}(\mathbf{s} | \boldsymbol{\mu}_{\boldsymbol{s}}^k, \boldsymbol{\Sigma}_{\boldsymbol{s}}^k) \tag{5.15}$$

This is density is once again learnt as a GMM using the EM algorithm.

In this fashion, the set of contact query models, one for each finger link and for a demonstrated grasp $g$, is denoted by $\mathcal{Q}_g = \{Q_i(\mathbf{s})\} \forall_i$.

### 5.2.4.2 Grasp sampling

The query density described above allows one to sample finger link poses over the given query point cloud surface. To construct a full hand posture, first a link pose is sampled using Eq. 5.15, i.e. $\tilde{\mathbf{s}} \sim Q_i(\mathbf{s})$. Finally, the full hand posture of a hand is completed using forward

kinematics by sampling the remaining joint angles for the remaining links from the hand configuration model $C(\mathbf{h}_c)$. Many grasp candidates can be generated in this fashion.

#### 5.2.4.3 Grasp optimisation

The generated grasp samples can be later optimised via a derivative-free method using the product of experts likelihood criterion (as introduced by [Hinton, 1999]) as energy function, such as *simulated annealing*.

We denote a grasp solution as the tuple $\mathbf{h} = (\mathbf{h}_w, \mathbf{h}_c)$ containing respectively the hand wrist pose $\mathbf{h}_w \in SE(3)$ and hand joint configuration $\mathbf{h}_c \in \mathbb{R}^D$, such that if $fk(\cdot)$ is the forward kinematic function of the hand, then the hand links in workspace are given by $s_{1:N_L}$, where $N_L$ is the number of links of a robotic hand. Thus, we have

$$s_{1:N_L} = fk(\mathbf{h}), s_l = fk_l(\mathbf{h}). \tag{5.16}$$

The basic optimisation criterion is given by:

$$\mathcal{L}(\mathbf{h}) = C(\mathbf{h}) \prod_{Q_i \in \mathcal{Q}} Q_i(fk_i(\mathbf{h})) \tag{5.17}$$

$$\overset{*}{\mathbf{h}} = \underset{\mathbf{h}=(\mathbf{h}_w,\mathbf{h}_c)}{\arg\max} = \mathcal{L}(\mathbf{h}). \tag{5.18}$$

In [Kopicki et al., 2015] an additional expert is defined $W(\mathbf{h})$ so as to penalise collisions in a soft manner, as it was found to prune the solution space to contain better solutions. Collisions are penalised exponentially by the degree of penetration through the object point cloud by any of the hand links. We have also employed a collision expert in the implementation of this work.

The objective function with the additional collision expert is then given by

$$\mathcal{L}(\mathbf{h}) = W(\mathbf{h})C(\mathbf{h}) \prod_{Q_i \in \mathcal{Q}} Q_i(fk_i(\mathbf{h})). \tag{5.19}$$

And again, the best grasp is found by optimising

$$\overset{*}{\mathbf{h}} = \underset{\mathbf{h}=(\mathbf{h}_w,\mathbf{h}_c)}{\arg\max}\ \mathcal{L}(\mathbf{h}) \tag{5.20}$$

where $W(\cdot)$ is the collision expert, $C(\cdot)$ is the hand configuration expert defined in Eq. 5.14, and $Q_i(\cdot)$ is the contact query expert for link $L_i$ defined by Eq. 5.15. For a robot hand and a given grasp demonstration $g$, there is a set of contact query experts $\mathcal{Q}_g$. This optimisation criterion, therefore, tries to maximise the product of experts [Hinton, 1999], where each expert is a probability density. Each individual expert is responsible for assigning high likelihood to candidate grasps that satisfy just one of the constraints.

An innovation of this work is that we will explore the mixture of experts approach even further, thus allowing our optimisation criterion to incorporate multiple experts to represent additional arbitrary task constraints. This will give the potential to generate grasps specifically tailored for a given task. For this purpose let $\mathcal{E} = \{E_k(\cdot)\}_{k=1}^{N_E}$, such that each $E_i(\cdot)$ assigns a probability to a grasp solution $\mathbf{h}$ that is close to one if the grasp satisfies the constraint represented by $E_i(\cdot)$ and close to zero if the constraint is not satisfied. This set of constraints can be a set of constraints defined a priori by a human designer for a given task or learnt autonomously by the robot manipulator. We augment our optimisation criterion as follows:

$$\mathcal{L}(\mathbf{h}) = W(\mathbf{h})C(\mathbf{h}) \prod_{Q_i \in \mathcal{Q}} Q_i(fk_i(\mathbf{h})) \prod_{E_i \in \mathcal{E}} E_i(\mathbf{h}) \tag{5.21}$$

Furthermore, we define a hard constraint as an expert that returns binary values of one or zero, accepting or rejecting instantly a grasp, thus implementing rejection sampling. Continuous experts whose image are on the continuous interval $[0,1]$ are referred to as soft constraints in this work.

With this extension, we have then created a framework that allows a great variety of constraints to be incorporated into the grasp generation pipeline. For instance, one can think

of experts that imbue dynamic constraints to grasps or specific kinematic constraints such as manipulability, thus paving the way towards task oriented grasp synthesis.

Thus far in this work, the following constraint experts have been defined: i) hard kinematic constraint and ii) soft principal axis alignment constraint. The former expert returns zero if a given grasp solution **h** is not kinematically feasible by the robot platform, and one otherwise. It implements a hard constraint over the grasp generation process, rejecting grasp samples that are not kinematically feasible. The latter expert implements a soft constraint that returns a probability close to one if the wrist frame of the generated grasp **h** is aligned with a given axis provided by the user. This is useful to encourage grasps that always pick an object from the top, for instance.

## 5.3 Grasp synthesis algorithms

The pseudo code for the algorithms for grasp generation are presented by Alg. 6 and Alg. 7. The pseudocode described by Alg. 7 is an offline grasp generation algorithm and has been utilised by the original KDE formulation and the novel GMM-based model. Furthermore, in this work we propose a novel algorithm that allows for real-time grasp synthesis, which is outlined by Alg. 8. The advantage of Alg. 8 is that it can cope with dynamically changing point clouds, such as moving objects.

**Algorithm 6** Pose sampling from $Q_i(s)$ [Kopicki et al., 2015]

**Given**:
$M_i(\cdot)$: Contact model for link link $i$;
$O(\cdot)$: Object model;
$K_{Q_i}$: Number of samples to be generated;

**for** $j = 1$ **to** $K_{Q_i}$ **do**
  Sample $(\tilde{\mathbf{v}}_j, \tilde{\mathbf{r}}_j) \sim O(\mathbf{v}, \mathbf{r})$
  Sample from conditional density $\tilde{\mathbf{u}}_{ij} \sim M_i(\mathbf{u}|\tilde{\mathbf{r}}_j)$
  Compute sample weight $w_{ij} = M_i(\tilde{\mathbf{r}}_j)$
  $\tilde{\mathbf{s}}_{ij} = \tilde{\mathbf{v}}_j \circ \tilde{\mathbf{u}}_{ij}$
  separate $\tilde{\mathbf{s}}_{ij}$ into position $\tilde{\mathbf{p}}_{ij}$ and quaternion $\tilde{\mathbf{q}}_{ij}$
**end for**
return $\{(\tilde{\mathbf{p}}_{ij}, \tilde{\mathbf{q}}_{ij}, w_{ij})\}, \forall_j$

The set of samples generated by Alg. 6 can be approximated non-parametrically using a KDE, as in [Kopicki et al., 2015]. Alternatively, in this work, we propose to approximate it parametrically as a GMM, as given by Eq. 5.15.

### 5.3.1 Grasp offline synthesis

The same derivative free simulated annealing method for grasp optimisation is used with already augmented criterion for additional constraints as follows given by Eq 5.21. The pseudocode outlined by Alg. 7 is a standard offline grasp synthesis algorithm. This algorithm can be used by either KDE-based or the GMM-based method proposed in this work.

**Algorithm 7** Grasp optimisation and selection [Kopicki et al., 2015]

**Given**:
$\mathcal{Q}_g = \{Q_i(\mathbf{s})\}\forall_i$: Query model of grasp $g$ for each link $i$;
$C(\cdot)$: hand configuration model given by Eq 5.14;
$N$: Number of grasp samples to be generated before optimisation;
$\mathcal{K}_{selection}$: iteration mask to when perform grasp selection;
$p$: percentage of grasps to retain after each selection step;

**for** $j = 1$ **to** $N$ **do** ▷ Grasp generation
  Randomly select a query density $Q_i(\mathbf{s}) \in \mathcal{Q}_g$
  Sample pose $\mathbf{s}_i$ of the $i^{th}$ link from $Q_i(\mathbf{s})$ that has been constructed with Alg. 6
  Sample hand configuration $\mathbf{h}_c^j$ from $C(\mathbf{h}_c)$
  Compute complete hand posture defined by wrist pose and hand configuration using forward kinematics $\mathbf{h}_j = (\mathbf{h}_w^j, \mathbf{h}_c^j)$;
**end for**
$\mathcal{H}^1 = \{\mathbf{h}_1, \ldots, \mathbf{h}_N\}$
**for** $k = 1$ **to** $K$ **do** ▷ Grasp optimisation
  **if** $k \in \mathcal{K}_{selection}$ **then** ▷ Selection step
    rank $\mathcal{H}^1$ using Eq 5.21 and retain top $p\%$
  **end if**
  **for** $m = 1$ **to** $|H^k|$ **do**
    $\mathcal{H}_m^k$ perform step of simulated annealing on $\mathcal{H}_m^k$ using Eq 5.17 as objective function.
  **end for**
  $\mathcal{H}^{k+1} = \mathcal{H}^k$
**end for**
rank $\mathcal{H}^{K+1}$ by Eq 5.21
return $\mathcal{H}^{K+1}$

### 5.3.2 Online grasp synthesis

The aforementioned densities can also be used to estimate grasps in real time. This alternative has appealing applications in the case of dynamically changing point clouds, or moving objects. This variation skips the division between generating a query density and subsequently performing optimisation, as described in Alg. 7. Instead, it simultaneously generates queries $\tilde{\mathbf{s}}_{ij} \sim Q(\mathbf{s})$ followed by the grasp completion step $s_{1:N_L} = fk(\tilde{\mathbf{h}})$ with $s_k = \tilde{s}_{ij}$ for $k = j$ and $\tilde{\mathbf{h}} \sim C(\mathbf{h}_c)$. A generated grasp is then immediately evaluated using the criterion in Eq. 5.21 and kept in a priority queue with fixed size $S$. Thousands of grasps are generated in this

**Algorithm 8** Simultaneous grasp generation and optimisation

1: **Given**:
2: $\mathcal{Q}_g = \{Q_i(\mathbf{s})\}\forall_i$: Query model of demonstrated grasp $g$ for each link $i$;
3: $C(\cdot)$: hand configuration model given by Eq 5.14;
4: $K_Q$: Number of pose samples for each query $Q_i$
5: $\mathcal{P}$: priority queue with maximum size $S = |\mathcal{P}|$
6: $S$: Maximum size of priority queue;
7: $Mode(\mathbf{h}_j)$: Counts the number of times a grasp with id $j$ has been at the top of the priority queue;
8:
9: **repeat** ▷ Grasp generation and optimisation
10: | **for** $i = 1$ **to** $|\mathcal{Q}|$ **do**
11: | | Acquire point cloud from depth sensor and construct $O(\mathbf{v}, \mathbf{r})$;
12: | | **for** $j = 1$ **to** $|K_Q|$ **do**
13: | | | Sample pose $\mathbf{s}_j$ of the $i^{th}$ link from $Q_i(\mathbf{s})$ with Alg. 6
14: | | | Sample hand configuration $\mathbf{h}_c^j$ from $C(\mathbf{h}_c)$;
15: | | | Compute complete hand posture $\mathbf{h}_j = (\mathbf{h}_w^j, \mathbf{h}_c^j)$; ▷ Using forward kinematics;
16: | | | Add $(\mathbf{h}_j, \mathcal{L}(\mathbf{h}_j))$ to priority queue $\mathcal{P}$ ▷ With score $\mathcal{L}(\mathbf{h}_j)$ given by Eq 5.21;
17: | | **end for**
18: | **end for**
19: | Let $\overset{*}{\mathbf{h}}_j$ be the top $(\mathbf{h}_j, \mathcal{L}(\mathbf{h}_j))$ from priority queue $\mathcal{P}$;
20: **until** $Mode(\overset{*}{\mathbf{h}}_j) \geq 10$
21: return $\overset{*}{\mathbf{h}}_j$; ▷ Has converged

For reducing the computational load, step 11 constructs a new object model $O(\mathbf{v}, \mathbf{r})$ only when sufficient change with respect to previous acquired point cloud is detected in terms of the difference in number of points.

fashion. Finally, when the same top ranked grasp remains as the best grasp for 10 repeated sampling rounds, this grasp is then chosen for execution. Note that this means that the mode of the grasp distribution has remained the same for many sampling rounds, therefore this is used as a heuristic selection criterion. As it represents the mode of the grasp distribution, thus, the most likely grasp. This procedure is outlined by Alg. 8. Furthermore, note that this accounts for selecting good grasps based on Eq 5.21, also retaining only the $S$ best grasps, since the priority queue has fixed size and therefore the selection process of Alg. 7 is no longer required.

## 5.4 Sensorising the Pisa/IIT SoftHand

The underlying assumption of good kinematic models and joint state encoders is made at the core of the models described in this chapter. We would like to assess the performance of the described approaches when applied to under-actuated hands, as at the time of evaluation of this work this was the available hardware. However, accurate knowledge of finger positions is typically unavailable for soft under-actuated robot hands. In this section we describe our technical efforts to sensorise the Pisa/IIT SoftHand to overcome these limitations.

The Pisa/IIT SoftHand is a humanoid hand, it has 19 DoFs and only one degree of actuation (DoA). The benefit of this type of robotic hand resides in its compliance to object shapes. However, this benefit comes at a cost. Accurate control and knowledge of the hand state is no longer possible. Since there are no encoders to measure the hand configuration $\mathbf{h} \in \mathbb{R}^{19}$, researchers have proposed to use inertial measurement unit (IMU) sensors as a possible approach for measuring the angular state of each finger joint (e.g. [Santaera et al., 2015]). This is done via data fusion algorithms that combine readings from both accelerometer, gyro and magnetometer for attitude estimation. However, this approach has its own drawbacks, as the measurements of earths' magnetic field used for attitude estimation can be sensitive to local distortions caused by nearby sources of magnetic fields, such as motors and metallic structures. Compensating for these distortions is possible via calibration techniques. In practice, nonetheless, the Pisa/IIT SoftHand has a motor in close proximity to the fingers, which represents a strong source of magnetic distortion. In addition, the environment where the robot is located may change completely the local magnetic field in the robot workspace, invalidating any prior calibration. For instance, manipulating metallic objects may also become a problem since they also generate distortions in magnetometer readings.

Hence, in this work we opted for a simpler approach. Since the Pisa/IIT SoftHand is a humanoid hand, in order to measure individual finger flexion we developed a custom

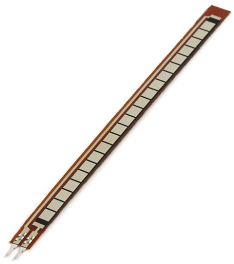

Fig. 5.3 A resistive flex sensor 4.5" in length. Manufacturer: Spectra Symbol.

data-glove to serve as finger encoders. The developed data-glove is composed of 5 resistive flex sensors[2], one for each finger, that serve as angle displacement measurement units, see Fig. 5.3. As a flex sensor bends, its resistance changes which reflects in a corresponding different voltage output. The output voltage as a function of bending angle is then processed by a micro-controller that streams the readings wirelessly to a radio receiver connected to the robot control station at a frequency of 100Hz [3]. The received voltage readings for the five fingers $\mathbf{z} \in \mathbb{R}^5$ (from pinky to the thumb finger) are linearly mapped to normalised values, i.e. $z_i \in [0,1]$. This is done by using prior knowledge of maximum and minimum voltage outputs for each finger given the circuit specification. This normalised output is then scaled to range between the minimum and maximum joint limits, as given by the hand kinematic specification.

For the pinky, ring, middle and index fingers, the flexion joint motion for the proximal, metacarpal and distal interphalangeal joints share the same angles, while the joint corresponding for abduction motion is kept fixed and centred (i.e. abduction is not modelled for those fingers).

[2]Symbolflex sensors (4.5"), see https://www.spectrasymbol.com/product/flex-sensors/
[3]The custom micro-controller has been produced and provided as a courtesy from Obi Robotics Ltd

$$\boldsymbol{\theta}_i = \begin{bmatrix} a_{abd} \\ a^i_{flx} \\ a^i_{flx} \\ a^i_{flx} \end{bmatrix} [z_i] \tag{5.22}$$

Note the expression has been written in matrix multiplication form, where the constants $a_{abd} = 0$ (as we do not not measure abduction motion, only flexion), and $a^i_{flx}$ are conversion scaling factors for the fingers $i \in \{1,2,3,4\}$, from pinky to index finger.

For the thumb, all joints share the same measured angles, including the abduction joint. The thumb ($5^{th}$ finger) is treated as special case, as it is modelled with only three joints for the SoftHand, thus

$$\boldsymbol{\theta}_{thumb} = \begin{bmatrix} b_{abd} \\ b_{flx} \\ b_{flx} \end{bmatrix} [z_5] \tag{5.23}$$

These scaling factors convert the normalised values $z_i$ to minimum and maximum range of the hand joint angles. A single initial calibration procedure in which the robot starts with the hand fully opened, and then controlled to a fully closed state of the hand is enough to acquire these limit mappings. Furthermore, due to the single actuated synergy for this hand model, fingers are indeed coupled, resulting in necessary 5 sensors. However, we have not modelled abduction/adduction finger motion. A diagram highlighting the corresponding joints and respective motion is depicted in Fig 5.4.

The resulting concatenated vector $\mathbf{h} = [\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3, \boldsymbol{\theta}_4, \boldsymbol{\theta}_{thumb}] \in \mathbb{R}^{19}$ comprises the joint configuration of the SoftHand.

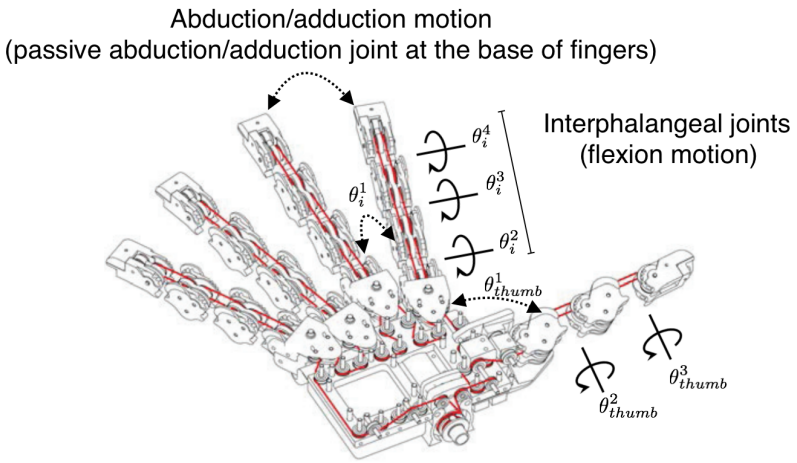


Fig. 5.4 Simplified kinematic diagram highlighting joints responsible for different motions. Diagram adapted from [Catalano et al., 2014].

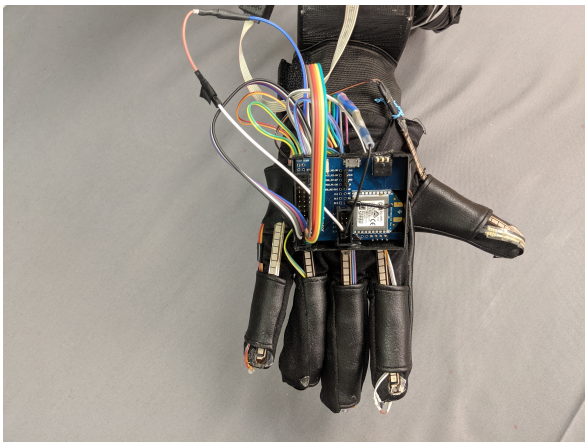

Fig. 5.5 The sensorised Pisa/IIT SoftHand. Each finger flexion is sensed independently with the developed data-glove with update rate of 100Hz.

### 5.4.1 Sensorisation performance

An evaluation setup has been prepared in order to assess the applicability of resistive flex sensors as a plausible method for sensorising the Pisa/ITT SoftHand. We utilised a 3D printed finger composed of two links that are free to rotate around a single axis driven by a stepper motor. A complete diagram of this setup is shown in Fig. 5.6. The figure shows the finger is sensorised using a resistive flex sensor with variable resistance R1, different degrees of flexion yield different levels of voltage output that are read by a micro-controller using

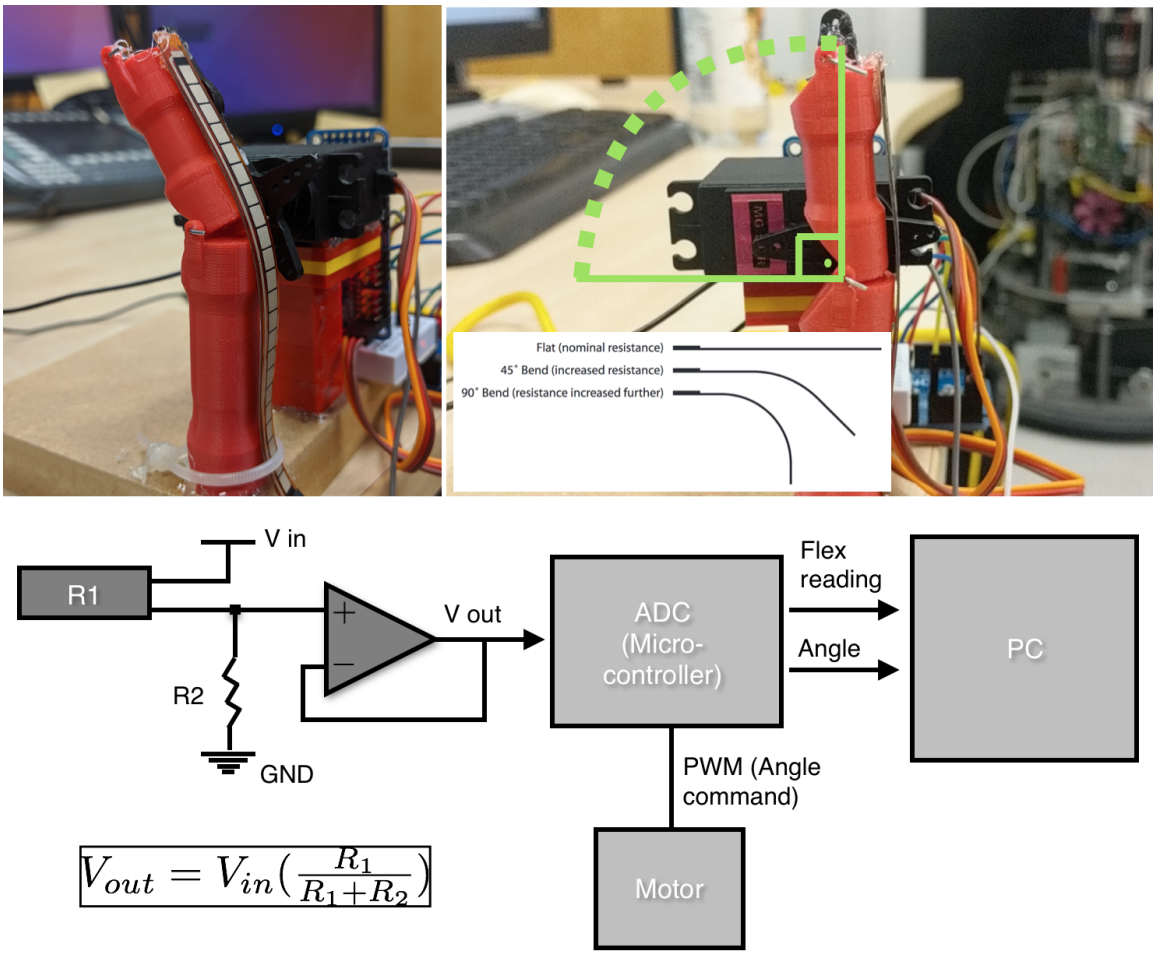


Fig. 5.6 Flex sensor performance evaluation on a finger composed of two links, a flex sensor (R1) is placed on the finger surface so as to measure its angular position (top-left). The finger is driven around its revolute joint by a stepper motor with maximum rotation of 90 degrees (top-right). A micro-controller is utilised to convert voltage readings from the flex sensor using an ADC. The micro-controller also has access to the ground-truth commanded angle, both floating point measurements are streamed to the PC for data collection.

analog-to-digital converter (ADC). The second resistor R2 composing the *voltage divider* circuit is a resistor with fixed nominal resistance of $47K\Omega$. The flex readings were then linearly mapped to an estimated angle given the known rotation limits of the finger in the interval $\beta = [0°, 90°]$ degrees, such that

$$\theta_{raw} = \frac{V_{out} - V_{min}}{V_{max} - V_{min}} 90 \tag{5.24}$$

this equation maps voltage readings to an angle in the known joint limit interval $\beta$.

The micro-controller also has access to the ground-truth commanded angle to the stepper motor, which performed repeated motions between $0^\circ$ and $90^\circ$ in steps of $0.25^\circ$. Using this setup, a total of 10 flexions were performed, which were sampled at $100Hz$. This process generated the trajectory over time steps shown in Fig. 5.7 containing 6864 readings.

An alternative approach for acquiring estimates of flexion angle considered fitting a simple linear model mapping the measured angles given by Eq. 5.24 to the acquired ground-truth angular positions of the stepper motor. This secondary approach can be seen as a calibration procedure in order to improve flexion angle estimates. The model has the following form:

$$\theta_{calibrated} = \alpha_a \theta_{raw} + \alpha_b \tag{5.25}$$

where, the parameters $\alpha_a, \alpha_b \in \mathbb{R}$ were estimated using the dataset depicted in Fig. 5.7 via the MLE method, as previously introduced in Chapter 3.

#### 5.4.1.1 Accuracy

The accuracy of the sensorisation was first evaluated by comparing the raw readings given by Eq. 5.24 with those given the ground-truth commanded angle to the stepper motor. We also considered the performance of calibrated readings given by Eq. 5.25.

In Table. 5.1 we report the mean observed difference (bias), the standard deviation (SD) of the differences and the 95% agreement limit interval over the acquired measurements with respect the ground-truth angle for both raw and calibrated angular mappings. The reported 95% limit of agreement was within $\pm 5$ for the considered sensorised finger.

Table 5.1 Sensorisation agreement intervals and ground-truth motor angular positions.

| Angular mapping | Mean difference (bias) | SD of differences | 95% agreement limit |
|---|---|---|---|
| Raw | -1.85 | 2.45 | -6.75 to 3.05 |
| Calibrated | 0.31 | 2.54 | -4.67 to 5.30 |

Here we obtain good accuracy results with an average error between $\pm 3$, as given by the SD. Very similar accuracy results have been reported on commercial data gloves (e.g. CyberGlove and the DataGlove Williams et al. [2000]), and in other related studies (see Saggio et al. [2015]). Finally, repeatability was measured using intraclass correlation (ICC) over the repeated 10 motions. We achieved $ICC = 0.986$, which is very similar to values of $ICC \approx 1.0$ obtained in related studies and commercial gloves [Saggio et al., 2015; Williams et al., 2000]. The reported error is larger for angles above $80^{\circ}$. This may be still undesirable for certain highly accurate manipulation tasks. Nonetheless, provided that the SoftHand has no other source of angular encoders, we deem this sensorisation method extremely useful for providing a relatively good estimate of the SoftHand joint state.

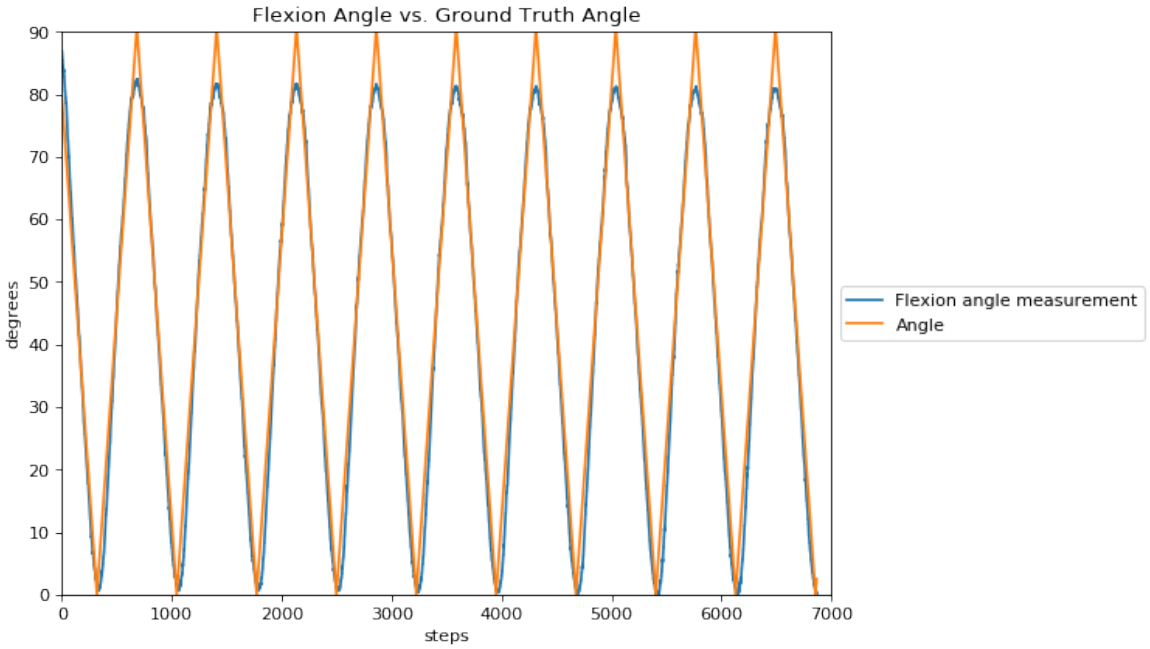


Fig. 5.7 Estimated angle using the simple mapping for $\theta_{raw}$ given by Eq. 5.24 and ground-truth motor angular trajectories.

The sensorised SoftHand offers two benefits. First, it allows us to have a better approximation regarding the location of the hand fingers, making it suitable for approaches described in this chapter. Second, provided the sensorisation for the hand state also permitted the

development of a feedback controller for the hand. A sequence with overlayed kinematic model over the real SoftHand is depicted in Fig 5.8.

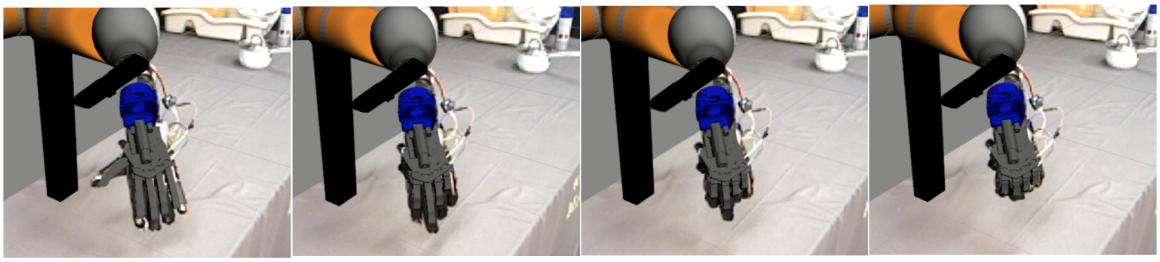

Fig. 5.8 The sensorised Pisa/IIT SoftHand. The image shows the overlayed SoftHand model on the real robot scene as the hand is commanded from a fully opened to a fully closed posture. Each finger flexion is sensed independently with the developed data-glove with update rate of 100Hz.

## 5.5 Generative grasp synthesis framework

The approaches and algorithms described in this chapter have been implemented as part of a general framework for generative grasp synthesis. This framework has been utilised for all experiments and validation in described in this chapter. The code for the developed framework has also been made open-source and constitutes a technical contribution of this thesis. The developed framework offers the following features:

**Interface for demonstration**

A grasp demonstration can be straightforwardly given via loading a pre-recorded object point cloud, and providing at least 3 waypoints a desired robotic hand representing a grasping trajectory for the object.

**Generality acrros different hand kinematics**

The developed approaches can be readily applicable to any robot hand with a given kinematic specification using the Universal Robotic Description Format [Lynch and

Park, 2017]. The goal is to facilitate the deployment of generative grasp synthesis approaches already implemented in the framework as well as future extensions.

**Grasp execution in simulation**

Grasps can be executed in simulation. The framework is able to acquire synthetic point clouds of loaded object models, generate and execute grasps on the given object. The grasp simulation is performed using the Bullet physics engine [Coumans and Bai, 2019].

**Modularity**

The framework is independent of underlying middleware utilised to control a given manipulator. The output grasps are represented by a complete grasp trajectory that can be translated to a desired middleware (such as the Robot Operating System [Koubaa, 2016]). Indeed, a toolbox with appropriate interfaces for standard ROS versions is provided separately[4].

**Flexibility**

The framework allows custom constraints, or experts as defined in Eq. 5.21. For instance, grasps can be rejected using hard kinematic constraints based on kinematic feasibility tailored to specific robot manipulators. These constraints can be defined pragmatically.

Significant effort has been employed implementing this framework, which we deem as technical contribution of this thesis. We proceed to describe preliminary evaluation and validation of the currently implemented approaches.

[4]ROS interfaces are available at https://github.com/eaa3/aml_grasp_ros.git

## 5.6 Experiments in simulation

We perform two separate experiments in simulation. The first experiment compares the success rate between the KDE and GMM-based success rate in simulation (we used Bullet physics [Coumans and Bai, 2019]) in four conditions: A) without depth sensing noise and with optimisation, B) with simulated depth sensing noise and with optimisation, C) without noise and without optimisation and D) with noise and without optimisation. The second experiment shows the runtime performance for each respective method when generating 200 hundred grasps for an object, results are shown in Fig. 5.11.

For all experiments and grasp synthesis methods, the optimisation utilised was the derivative-free simulated annealing method with 100 iterations and maximum initial temperature of 1.0 which decays linearly over the number of iterations.

We adopted the same depth sensor noise model as in [Johns et al., 2016]. The noise model combines additive and lateral shift Gaussian noise applied to every pixel in a simulated depth image:

$$\tilde{z}(u,v) = z(u+\mathcal{N}(0,\sigma_p^2), v+\mathcal{N}(0,\sigma_p^2)) + \mathcal{N}(0,\sigma_d^2), \tag{5.26}$$

where $\sigma_p$ is the standard deviation for the noisy lateral shift of pixels, which has been set to 1 for these experiments. And $\sigma_d$ is the standard deviation for depth measurements, which was set to 1mm.

### 5.6.1 Performance comparison

For the comparison, we use a parallel gripper (WSG 50 Shunk Gripper) and a single demonstrated grasp on a box, from which object the contact models are learnt for both KDE and GMM-based methods. The KDE-based method has utilised one kernel per data point to approximate the densities. The GMM-based contact model has approximated Gaussian mixture densities with $K = 2$, given the bi-modality of the demonstration as depicted in Fig.

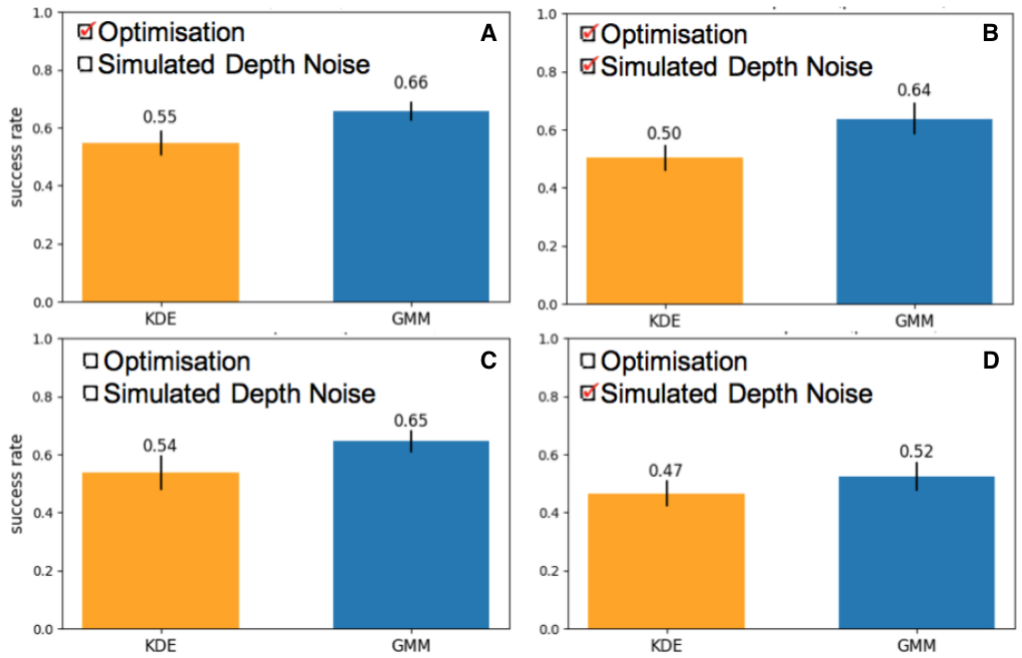


Fig. 5.9 Grasp success rate for the four experimental conditions

5.2 and with the number of kernels $K_Q = 5$ for GMM-based query density approximation from Eq. 5.15.

A total of 60 objects were used for testing the performance of each respective method. Objects are varied, some objects have box-like features, others are curved, cylindrical, convex, non-convex, and of combinations of sizes and shapes as shown in Fig. 5.10. Objects are always placed in the same pose for both approaches, and simulation parameters are kept fixed. For grasp generation, simulated point clouds are acquired from 4 fixed camera poses around the object, thus obtaining a full 3D point cloud of the object. The acquired point clouds are stitched together and post-processed identically for both approaches, including principal curvature feature computation with search radius set to $1cm$.

For each respective approach (KDE and GMM-based), a total number of 200 grasps are generated per object. Sampled grasp solutions are arranged in decreasing order of value as given by Eq. 5.17, in which the highest value represents the best grasp in the sequence. For each approach, the best grasp is executed on each object. Before every execution attempt

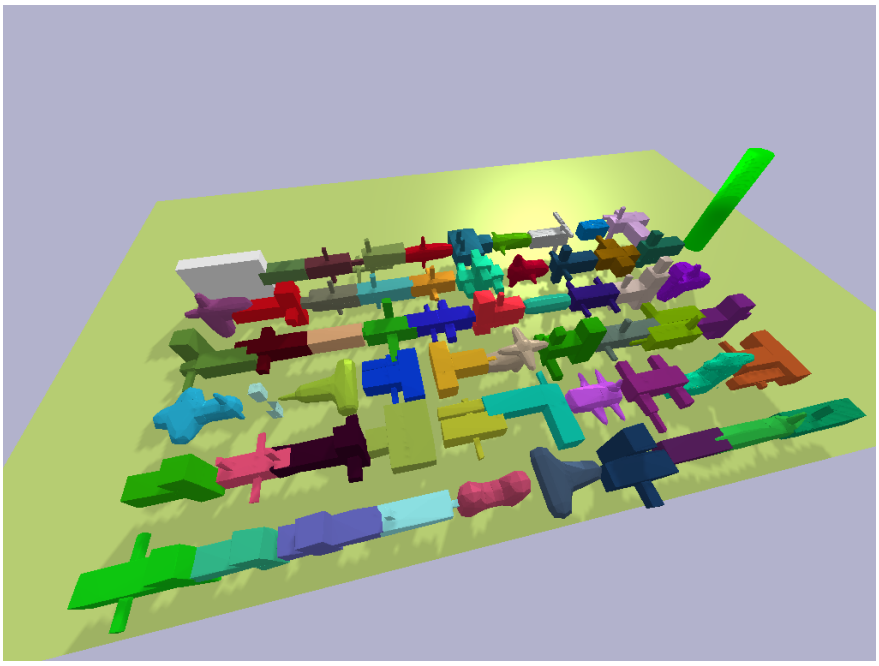

Fig. 5.10 Set of 60 test objects with varying shapes utilised for the simulated experiments.

the object is restarted to the same initial pose. This process is repeated 10 times over the set of 60 object and results are averaged over these 10 trials. We measure success rate as the proportion of successful grasps over the total number of objects.

Finally, a two-tailed paired t-test is performed to assert the significance of the results between KDE and GMM-based methods over the 10 repeated trials.

### 5.6.2 Results

The results in all four different conditions indicate that the GMM-based method shows superior performance over the KDE-based method ($p < 0.05$) in this setup. We see that in the absence of noise, the final optimisation step has little effect in grasp performance as shown by Fig. 5.9. In contrast, optimisation seems to become crucial under noisy conditions, which seems natural. Under the influence of noise, local optimisation with simulated annealing is expected to help to compensate from imperfect sensing, hence still converging to a local or global minimum under the product of experts objective function in Eq. 5.17.

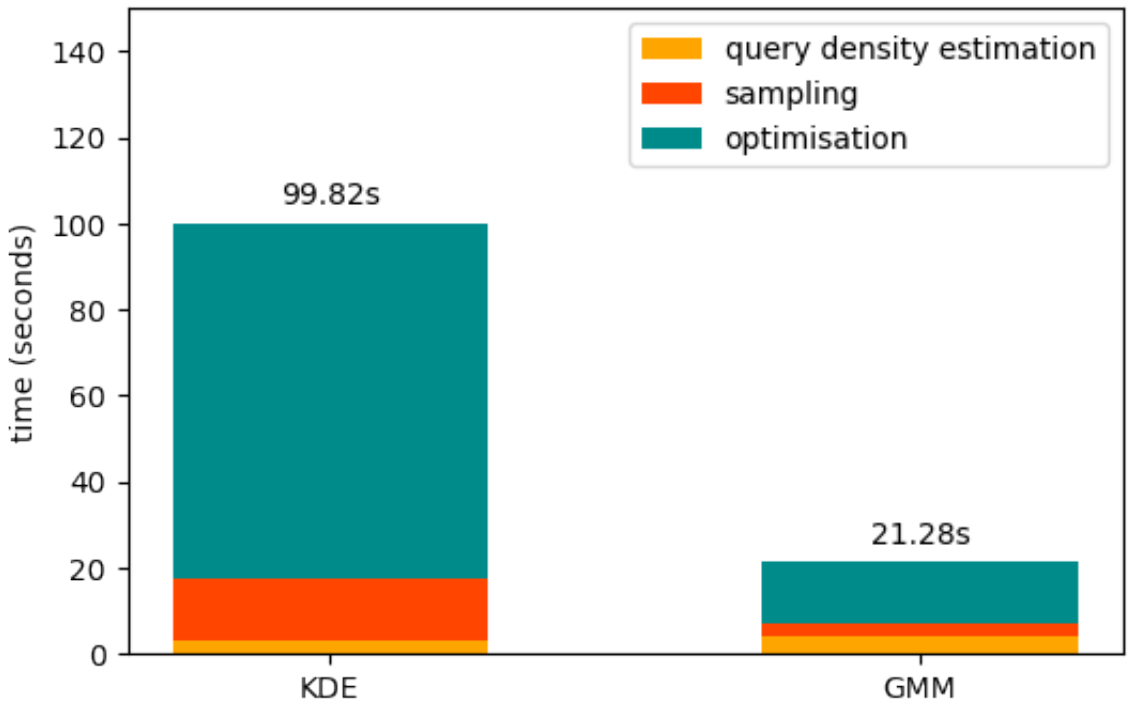


Fig. 5.11 Corresponding total runtime for KDE and GMM-based methods. The total time includes the time for query density estimation, sampling of 200 grasps and optimisation of top 10 grasps using 100 iterations of simulated annealing. The results the GMM-based method is nearly 5 times faster. The GMM-based method requires fewer kernels to approximate the query density and therefore is much faster for likelihood evaluation during optimisation. For the same reason this gain in time performance is also noted during sampling and evaluation.

Perhaps the biggest impact is shown in terms of the total time required for grasp synthesis. The stacked bar chart in Fig. 5.11 shows the total runtime for generating 200 grasps for a given object, highlighting the individual contributions from different steps involved in the grasp synthesis process: i) query density estimation, ii) sampling and ranking and finally iii) optimising top 10 grasps (out of the 200 samples). Due to the fact that the GMM-based method requires fewer kernels to approximate the query density in Eq. 5.15, it directly affects the total runtime which is largely dominated by the number of query likelihood evaluations (Eq. 5.17).

## 5.7 Validation on different robot platforms

Using the developed framework containing the approaches described in this chapter it becomes simpler to deploy grasp synthesis capabilities to a range of robot platforms. In this section we describe some examples.

### 5.7.1 Validation on the Boris platform and the Pisa/IIT SoftHand

Using the sensorisation for the Pisa/IIT SoftHand described in Section 5.4 we have also deployed the developed offline approach given by Alg. 7 approach on a real robot platform as shown by Fig. 5.12. For the robot platform, we made use task-specific kinematic constraints, such as kinematic feasibility and axis alignment that can be straightforwardly added in our implementation using Eq. 5.21 to prune the grasp synthesis process. A summary video on the Boris platform is available on https://youtu.be/9RqZaTAH4Fs.

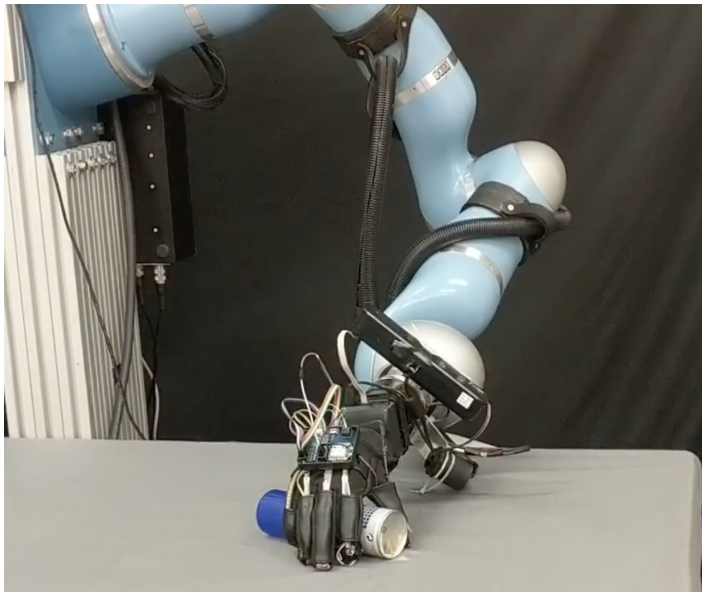

Fig. 5.12 Preliminary deployment of KDE and GMM-based grasp synthesis methods on the Boris robot platform.

Fig. 5.13 The work done has been validated using a Kinova arm equipped with a Gripper KG-3. The goal was to grasp objects on a table using the framework developed in this chapter.

### 5.7.2 Validation on the Kinova platform and the KG-3 Gripper

The developed grasp synthesis framework has also been deployed on the Kinova platform. The hardware utilised for tests consisted of a Kinova arm equipped with a Gripper KG-3 setup at the Honda Research Institute (HRI), as shown in Fig 5.13. Point clouds of the objects placed on table were obtained using an ASUS Xtion Pro.

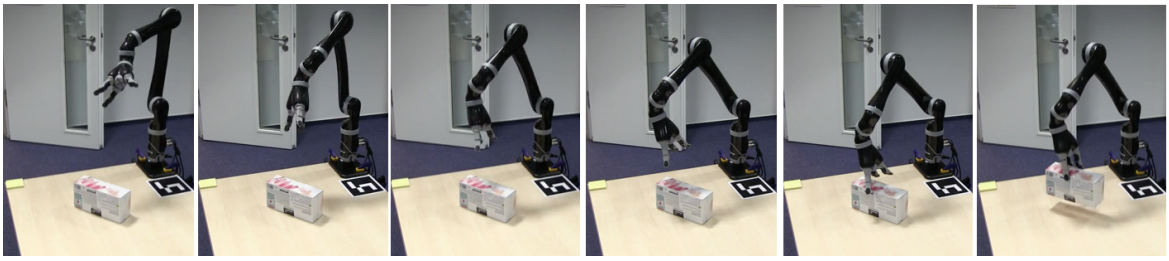

Fig. 5.14 Deployment of the approach on a Kinova platform equiped with a three-finger KG-3 gripper.

## 5.8 Online grasp synthesis for under-actuated hands

Additional validation has also been prelimnarily performed on a real robot platform for Alg. 8. As depicted in Fig. 5.15, using the sensorised data-glove, we were able to perform online

grasp synthesis using the Pisa/IIT SoftHand. The main benefits of the approach outlined by Alg. 8 include its natural ability to cope with dynamically changing point clouds, which encompasses moving objects, while at the same time carrying the benefits of data efficiency and not relying on external pose estimation algorithms[5]. Point clouds acquired from a single external Primesense Carmine 1.09 are processed at a rate of 10Hz. The processing includes, cropping and removing points from the table plane, subsampling utilising voxel size of 0.001m, and principal curvature feature computation. In order to decrease computational load, the current point cloud utilised for grasp synthesis is only effectively updated when significant change occurs. In our setup, a point cloud is considered to have changed significantly if the number of points between the previous and current acquired point clouds differ by at least 50 points. This strategy also allows the grasp synthesis to run undisturbed by slight fluctuations from depth sensing, while at the same time dynamically detecting and generating grasps for the given object in an online fashion. The result is an approach that is able to continually generate grasps and able to cope with moving objects. The generated grasps are kept in a priority queue, ordered according to their corresponding likelihood. A grasp that remains on the top of the priority queue for more than 10 successive updates is then selected to be executed. In Fig. 5.15 we also see that regardless of the object positioning, similar plausible grasps eventually converge.

[5] An example video summarising the approach and outlining the full sequence of generation and grasp execution can be found at https://youtu.be/_0CBrOYXRFw

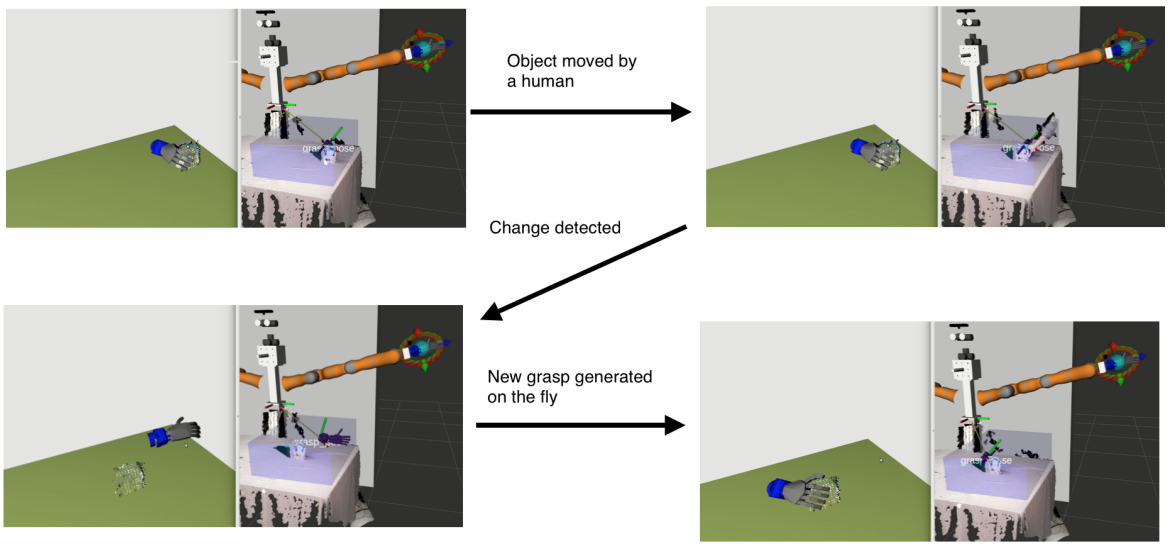


Fig. 5.15 Diagram depicts an example sequence of the developed grasp synthesis algorithm outlined by Alg. 8. In this instance, the object is mobile and manually relocatted by a human, the grasp synthesis approach continually generates novel grasps to the current sensed object point cloud. It is completely agnostic to pose estimation approaches and converges to sensible grasps as the object is moved on the the table.

## 5.9 Conclusion and future work

We proposed an alternative formulation for learning generative models for grasp synthesis using parametric mixtures, such as GMMs. In our experimental setup, the GMM-based method has indicated better performance than related non-parametric approach using KDEs [Kopicki et al., 2015].

However, the obtained simulated results seem to show a lower performance when compared to other state-of-the-art methods. Two factors are responsible for the obtained success rate performance, they are: i) grasps usually fail due to insufficient grip force and ii) the current framework supports a single grasp demonstration. The first, is a inherent limitation of the approach, since it they are purely kinematic grasp synthesis methods, making no considerations about forces to prevent slippage. One way to address this limitation in future work would be to include an additional expert constraint that reasons about such dynamic behaviour and stability criterion, thus making the best of both data-driven and analytic

considerations for grasp synthesis. Fortunately, the developed framework makes it easier to implement such extensions. The second factor, shall be straightforwardly taken into account in next development iterations of the developed framework.

Despite current limitations, the proposed methods and associated framework for grasp synthesis have their own merits in offering a platform for further research. First, we extended the product of experts criterion, allowing us to accommodate task-specific constraints as needed for different deployment contexts. We gave a brief instance of that by including kinematic constraints directly in the evaluation step of the grasp synthesis pipeline on the Boris platform. We also showed that the GMM-based approach offers faster execution times. This promising feature allowed us to propose an alternative algorithm for real-time grasp synthesis, allowing the grasp synthesis strategy to cope with changing point clouds, such as moving objects. Finally, as a technical contribution, the proposed approaches have been implemented as an openly available framework, which was validated on different robot platforms. The deployment also offered a novel sensorisation scheme for the Pisa/IIT SoftHand using a custom made data-glove. As a future work, we intend to perform deeper evaluation of the proposed methods on the deployed robot platforms. In particular, perception has great effect in the final overall performance. In the next Chapter 6, we shall investigate active perception strategies that can be tailored to leverage manipulation performance in the context of generative grasp synthesis.

# Chapter 6

# Active vision for dexterous grasping

In Chapter 2.1, we have seen that grasping of novel objects is a hard problem on which there has been steady progress [Ben Amor et al., 2012; Bohg et al., 2011; Curtis and Xiao, 2008; Detry and Piater, 2013; Hjelm et al., 2014; Kopicki et al., 2015, 2014; Rietzler et al., 2013; Saxena et al., 2008b; Stark et al., 2008]. We now have methods that are able to synthesise dexterous grasps for unfamiliar objects, using incomplete object reconstructions. However, the reliability of grasping declines as the quality and completeness of the reconstruction deteriorates. Given an active vision system, we would like to minimise the number of views taken, while maximising grasping reliability. Using visual feedback to aid manipulation tasks has been explored to a limited extent in the literature, as identified in Chapter 2.2.

One way to approach these limitations would be to guide active vision to reconstruct the complete object, and for a cluttered scene the objects around it. This would be similar to the problem of active SLAM. However, for most practical manipulation purposes, full object reconstruction is either too costly or simply infeasible. It is also typically redundant, since most of the time only a limited portion of the object surface is in contact during a grasp. In this work we instead take the approach of directing gaze only to reconstruct as much of the scene as is necessary to make the executed grasp more reliable. We do this in two ways. First, we use candidate grasps to guide vision. Second, when a good grasp is found we direct vision

to fill in unseen volumes of the workspace so as to make the reach to grasp more reliable. So, in each stage active vision is driven by the task. And only when a grasp is not found at a given time step we fall back on classical active SLAM.

Hence, in this chapter we propose to make use of a generative grasp synthesis method such as the ones discussed in Chapter 5. In particular, we utilised the approach proposed by Kopicki et al. [2015]. The later development of the parametric variation and extensions discussed in Chapter 5 are also compatible with the approach discussed in this chapter. The core idea is that a generative grasp synthesis approach provides cues to an underlying perceptual agent who uses generated grasp hypothesis to make future perceptual predictions taking into account a specific task-related utility function that is contact driven, as well an information-theoretic utility function to maximise information gain, and minimise probability of collisions.

This chapter is organised as follows. First, we proceed to describe our active vision method. This proposed method has two parts, a routine driven by the planned contact points, and a routine driven by the need to ensure a safe reach to grasp trajectory. Second, we present experimental results on 14 novel objects, comparing our method with a randomised view planner, and with a complete reconstruction. Finally, we present our conclusions and discussion of future work.

## 6.1 View Selection

We first sketch our method, and then proceed to the details. The robot begins by taking a single view from a fixed location of the scene. A depth camera mounted on the robot's wrist is used. The robot is then able to choose views, which in turn provide incomplete point clouds of the object. A dexterous grasp planning algorithm is then run, which generates a large number of candidate grasps on the partial point cloud for the object. These grasps will typically assume the existence of graspable surfaces on both sides of a surface defined

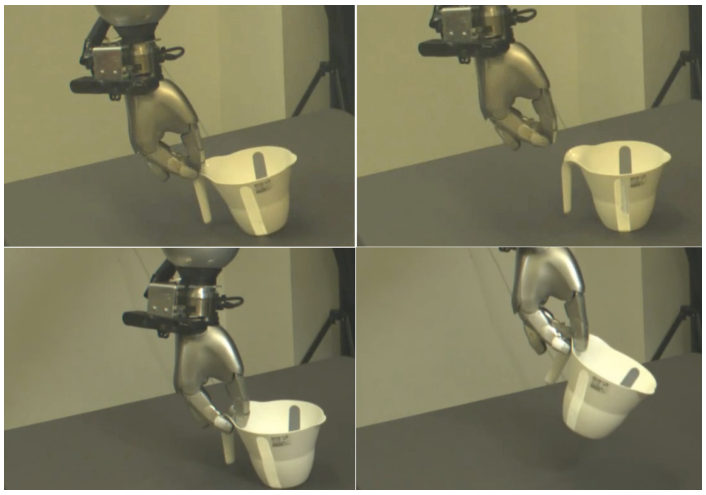

Fig. 6.1 Grasp failure and grasp success. The top row depicts a grasp that failed without active view selection. The bottom row shows a successful grasp that used active view selection. The difference was attributed to the quality of surface reconstruction in proximity to the planned grasp contact points.

by the point cloud. The predicted contact locations are then used to drive the next view. The next view is chosen to maximise the quality of the point cloud at the planned contact locations. If a grasp cannot be found, we employ information gain view planning, using a 3D occupancy map. Once the quality of the relevant surface reconstruction is sufficiently high, or a limit on the number of views is reached, the grasp is fixed. Then the second phase of active vision aims to verify a safe path to the grasp location. To achieve this we again use the 3D occupancy map. This is used to calculate the probability of a collision-free trajectory. Active views for safety are driven to reduce the average entropy in cells through which the candidate reach-to-grasp trajectory passes. This ensures a safe grasp. We now proceed to describe the representations, and the three criteria used to drive active vision at different stages (contact based, information gain, and safety based).

### 6.1.1 Representations

We start by describing the underlying representations used to define our approach. Let $\Xi = [\xi_1, \xi_2, \ldots, \xi_N]$ be a list of possible camera poses, where $\xi_i \in SE(3)$, and $V \subset \Xi$ is the set

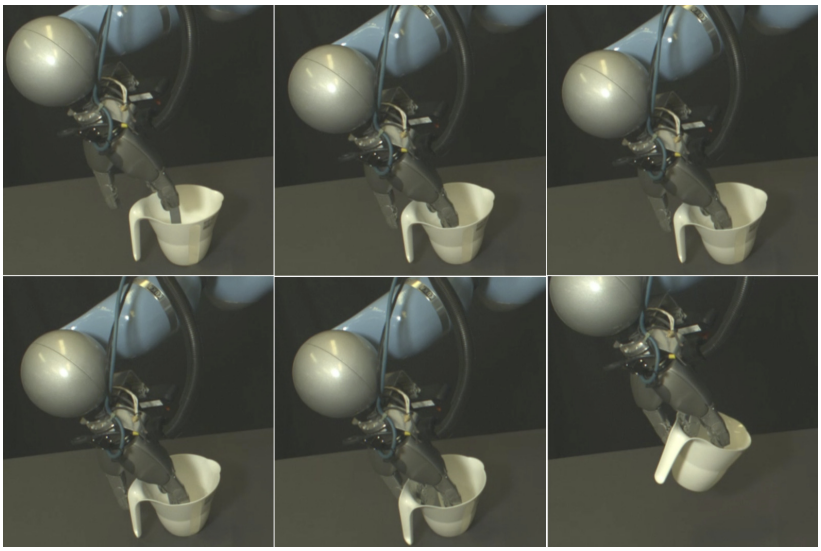

Fig. 6.2 A grasp example using random view selection baseline. The grasp is successful. However, the grasp tends to be unsafe. It can noted that the grasp trajectory starts pushing the object aside with its fingers far before the final grasp closure is executed in the last picture on the right. This illustrates a common scenario in which grasps usually fail.

of already visited camera poses. This list must be finite, and should provide good coverage of the workspace, but in principle views can be added to it on the fly. We describe how we picked this set in Section 6.2. In addition, let $\gamma$ be a point cloud obtained from a certain camera pose $\xi$. We define $\Gamma_t$ as the combined object point cloud, segmented from the table plane after $t$ views have been taken,

$$\Gamma_t = \Gamma_{t-1} \uplus segmented(\gamma), \tag{6.1}$$

i.e., $\Gamma_t$ is the result of segmenting the object point cloud from the table plane in $\gamma$ and integrating this result with our previous obtained object point $\Gamma_{t-1}$.

In addition to the object point cloud, we also maintain a representation of the full robot workspace as a 3D occupancy grid, implemented with an octree. We shall refer to this 3D occupancy grid as $\Lambda$, which is updated after each view and observation $(\xi, \gamma)$. The implementation we use [Hornung et al., 2013] allows us to easily represent known and

**Algorithm 9** Next Best View Exploration

**function** NEXTBESTVIEW($\Xi, \Gamma, \Lambda, G, V, T$)
  $\Omega = \emptyset$ ▷ Most recent found contact points
  $\tau = None$ ▷ Most recent found grasp trajectory
  $stop = false$
  **while** not $stop$ **do**
    $\overset{*}{\xi} = selectNBV(\Xi, \Gamma, \Lambda, V, \Omega, \tau)$
    $V = append(V, \overset{*}{\xi})$ ▷ Appending $\overset{*}{\xi}$ to V.
    $\gamma = capture(\overset{*}{\xi})$ ▷ Point cloud from pose $\overset{*}{\xi}$.
    $\Gamma = \Gamma \uplus segmented(\gamma)$
    $\tau, \Omega = findGrasp(\Gamma, G)$ ▷ Grasp planning with current $\Gamma_t$ based on [Kopicki et al., 2015].
    $\Lambda = updateOcTree(\Lambda, \gamma, \overset{*}{\xi}, \Omega)$
    $T = append(T, \tau)$
    $stop =$ CHECKSTOP($V, T$)
  **end while**
  $\overset{*}{\tau} = \arg\min_{\tau \in T} p(\tau|\Lambda)$
  Return $(V, \overset{*}{\tau}, \Gamma, \Lambda, T)$
**end function**
**function** CHECKSTOP($V, T$)
  Return ($|V| \geq 2$ and $|T| \geq 1$) or $|V| \geq 7$)
**end function**

unknown parts of the robot workspace $\Lambda$ and thus to define the information gain and safety based view planning strategies.

It is possible to find a grasp trajectory $\tau$ by transferring a learnt grasp $G$ to the given object represented by $\Gamma_t$ using the method of Kopicki et al. [2015]. This generates, for any partial or complete point cloud, a set of several hundred candidate grasps. These candidate grasps are generated from a statistical model learned from a single example of a particular grasp type, e.g. a pinch or power grasp. The grasps are ranked in order of likelihood according to the generative model. We pick the first ranked grasp. Then we extract the planned contact points from $\Gamma_t$, yielding a list of contacts $C = [\mathbf{c_1}, \ldots, \mathbf{c_M}]$, where $\mathbf{c_i} = (w_i, \mathbf{p_i}, \mathbf{n_i})$ is composed of a weight $w_i \in \mathbb{R}$, indicating its relevance to the grasp, the contact location $\mathbf{p_i} \in \mathbb{R}^3$, and the surface normal at that point $\mathbf{n_i} \in \mathbb{R}^3$. Points on the surface of the object are relevant to a

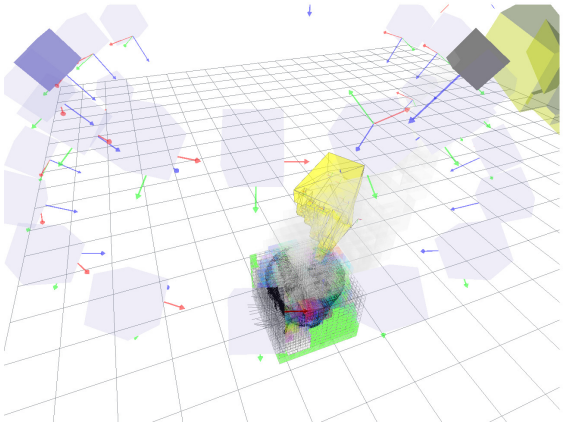

Fig. 6.3 View camera poses forming the set $\Xi$, camera pose highlighted in darker purple belongs to the set of visited poses $V$. The object in the centre is circumscribed by a voxelised cube. Information gain view exploration sculpts this cube when no contacts are found.

grasp if they are close to a planned contact point. The weight $w_i$ or importance of each point falls off exponentially as the distance from the planned finger position increases. Let us also define $\Omega = [(\mathbf{c_1}, z_1), \ldots, (\mathbf{c_M}, z_M)]$ as a list of contact points expanded to include the current quality $z_i$ of the observation of each point from the best view $\xi$ to date. Contact-driven vision prioritises looking at the planned contact points for which there is currently low-quality reconstruction, rather than elsewhere on the object. We now describe contact-based view selection in detail.

### 6.1.2 Contact Based View Selection

We let the viewing direction of a certain view pose $\xi_k \in \Xi$ be the vector $\mathbf{v_k}$, which we always constrain to point towards the object $\Gamma_t$. We define the quality of observation of a contact point $\mathbf{c_i}$ from a given $\xi_k$ as

$$\theta_{ki} = \theta(\xi_k, c_i) = \arccos(min(0, \mathbf{v_k}^T \mathbf{n_i})). \tag{6.2}$$

This models the fact that the depth errors rise as the surface becomes perpendicular to the image plane. Thus when looking at contact point with surface normal $\mathbf{n_i}$, we assign higher

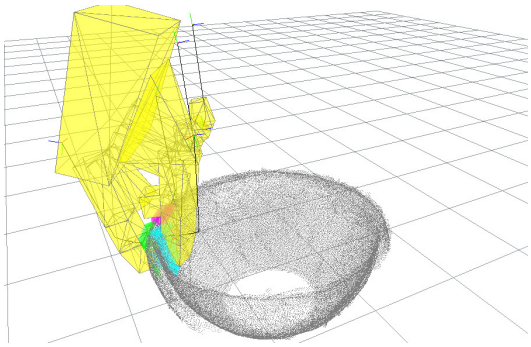

Fig. 6.4 (Left) Example o, showing the contact regions for different finger links highlighted with different colours. Contact-driven vision aims at viewing all these planned contact locations.

values to views in which the image plane normal and the surface normal directly oppose one another, i.e. $\mathbf{n_i}$ and $\mathbf{v_k}$ form an angle of 180 degrees, according to our convention that $\mathbf{v_k}$ always looks towards the object.

Thus, for each element $(\mathbf{c_i}, z_i) \in \Omega$ we store the contact points $\mathbf{c_i}$, and define $z_i$ the best quality of observation to date with respect to all visited poses as

$$z_i = \arg\max_{\xi_j \in V} \theta_{ji}. \tag{6.3}$$

Finally, let $F_\tau = [\mathbf{f_1}, \ldots, \mathbf{f_R}]$ be the list of finger link normals for the finger surfaces that are involved in the grasp. These are calculated for the last time step of the trajectory $\tau$. We then define the value of a particular (untried) view with respect to a particular contact $\mathbf{c_i}$ as

$$\sigma(\xi_k, F_\tau, \mathbf{c_i}) = w_i \sum_{r=1}^{R} max(\theta_{ki}, z_i) \frac{1 - sign(\mathbf{f_r}^T \mathbf{n_i})}{2}. \tag{6.4}$$

This defines high-value views as being those views which gaze head on at contact points. Note that when looking at a certain contact point $\mathbf{c_i}$ we are able either to improve our previous best viewing quality if $\theta_{ki} > z_i$, or leave it as it is. Note also that the multiplying term $\frac{(1-sign(\mathbf{f_r}^T \mathbf{n_i})}{2}$ serves as a switch that yields 0 or 1. This simply ensures that to give information relevant to a particular planned contact the view must be on the side of the point

cloud where the finger will contact. It does this by modelling the geometric relationship that a link must have relative to the object surface, i.e. the surface normal $\mathbf{f_r}$ of a given finger link must point in the opposite direction to the surface normal $\mathbf{n_i}$ of the object at the contact point. Finally, the normalised weight $w_i$ scales this value according to its overall relevance to the grasp as defined by the approach of Kopicki et al. [2015]. It follows that the total utility of a given view $\xi$ is given by

$$u_1(\xi, \Omega, \tau) = \sum_{i=1}^{N} \sigma(\xi, F_\tau, \mathbf{c_i}). \tag{6.5}$$

We are then able to rank the potential views by calculating the total value of a view with respect to all contact points, and picking the view that has the maximum value according to Eq 6.6. The algorithm will thus trade off between being able to gain information about more contacts, and gaining a great deal of information about a single contact. This proves useful when, for example, a single digit is planned to be placed on a back-surface that has not yet been observed.[1] In this case, viewing the back surface will be extremely informative, and will outweigh the value of observations of several finger contacts on the already observed front surface, even though the reconstruction of that front surface is imperfect.

$$\overset{*}{\xi} = \arg \max_{\xi_k \in \Xi - V} u_1(\xi, \Omega, \tau). \tag{6.6}$$

### 6.1.3 Information Gain View Selection

Of course, if no grasp can be found, then grasp driven view selection cannot run. In this case, the robot should look at the workspace around the recovered point cloud. To support this we define an information gain based utility function for view selection. Intuitively, this strategy makes sense, since no contacts were found with the knowledge we have about the object

[1] This ability is due to the fact that for any thin point cloud defined surface, the grasp planning algorithm assumes that this infinitesimally thin surface can be grasped from both sides. A view of the back side will reveal if there is in fact a separately defined back surface.

**Algorithm 10** Select Next Best View Contact Based

**function** NBVCONTACTBASED($\Xi, \Gamma, \Lambda, V, \Omega, \tau$)
  $\overset{*}{\xi} = None$
  **if** $|V| = 0$ **then**
    $\overset{*}{\xi} = head(\Xi)$
  **else if** $\Omega \neq \emptyset$ **then**
    Let $\overset{*}{\xi}$ be selected according to Eq 6.6
  **else**
    Let $\overset{*}{\xi}$ be selected according to Eq 6.14
  **end if**
  Return $\overset{*}{\xi}$
**end function**

shape so far, represented by $\Gamma_t$. Therefore, one should ideally adopt an exploratory behaviour to seek for new parts of the object.

For this purpose, let $bmin(\Gamma_t), bmax(\Gamma_t) \in \mathbb{R}^3$ be the respective minimum and maximum limits of the bounding box that circumscribes the object point cloud $\Gamma_t$. We are then able to extract the set of voxels $\Lambda_{object} \subset \Lambda$ inside this bounding box. If we assume the surface of this voxelised solid box $\Lambda_{object}$ is visible from all cameras, as shown in Fig 6.3. Then we can define a simple strategy to minimise the entropy about the object's shape, by selecting views that are going to have maximum predicted information gain about the voxels in $\Lambda_{object}$. Intuitively, our goal is to select views such that we gradually sculpt the solid cube, in a way that we will eventually reach a constant entropy value for this cube, due to self-occluding parts of the object, from which point no views are going to bring any more information gain.

Our first step to fulfilling this goal is to define a rule with which we can determine the set of visible voxels in $\Lambda_{object}$ visible from a camera pose $\xi$. The visibility test is performed using a typical frustum culling graphics procedure, with a few slight modifications. First, we transform the set of voxels $\Lambda_{object}$ into the camera coordinate system. During the projection phase of the pipeline, we allow many free voxels along the line of sight to be projected onto identical image coordinates, but we do not allow either unknown voxels, nor occupied voxels

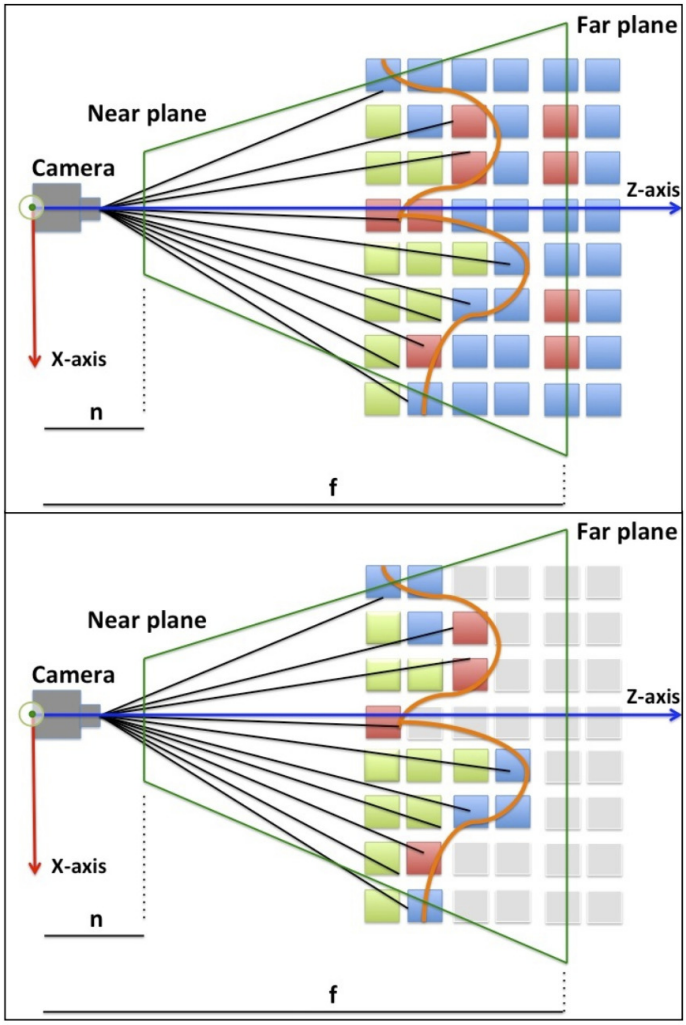


Fig. 6.5 Figure shows a cross-section visualisation of a typical visibility check. Occupied voxels are represented in red, free voxels are shown in green and unknown voxels have dark-blue colour. Once we have defined a viewing frustum to match the real depth camera specifications, a frustum culling procedure was performed in which free voxels were assumed to be transparent, whereas unknown or occupied voxels occlude each other. In this way, only the voxels coloured on the bottom image are taken as visible voxels after the execution of this procedure.

to be projected on top of one another. As a consequence, we find a border in our initial solid cube $\Lambda_{visible}(\xi) \subset \Lambda_{object}$, which contains all free voxels visible on the image plane, together with boundary voxels that might be either unknown or occupied, as shown in Fig 6.5. Thus, $\Lambda_{visible}(\xi) = \{s_1, \ldots, s_D\}$ is defined as our set of voxels of interest for information

**Algorithm 11** Safety Exploration

**function** SAFETYEXPLORATION($\Xi, \Gamma, \Lambda, \tau, V$)
  $stop = false$
  **while** not $stop$ **do**
    $\overset{*}{\xi}, value = safetyNBV(\Xi, \Lambda, \tau)$
    $V = append(T, \overset{*}{\xi})$
    $\gamma = capture(\overset{*}{\xi})$ ▷ Point cloud from pose $\overset{*}{\xi}$.
    $\Gamma = \Gamma \uplus segmented(\gamma)$
    $\Lambda = updateOcTree(\Lambda, \gamma, \overset{*}{\xi}, \Omega)$
    $T = append(T, \tau)$
    $stop =$ CHECKSTOPSAFETY($value$)
  **end while**
  $p = p(\tau|\Lambda)$
  Return $(V, p, \Gamma, \Lambda)$
**end function**
**function** CHECKSTOPSAFETY($value$)
  **if** $value \leq \beta$ **then**
    Return $true$
  **else**
    Return $false$
  **end if**
**end function**

gain prediction. We then follow to describe the information gain prediction for the set of voxels $\Lambda_{visible}(\xi)$.

#### 6.1.3.1 Information Gain Prediction

Let the occupancy probability of a voxel $s_d \in \Lambda_{visible}$ up to our most recent observations $o_{1:t}$ be $p_{s_d} = p(s_d|o_{1:t})$. We can write the entropy of this voxel by viewing it as a Bernoulli random variable with entropy

$$H(s_d) = -p_{s_d}\log(p_{s_d}) - (1-p_{s_d})\log(1-p_{s_d}), \tag{6.7}$$

**Algorithm 12** Safety Exploration View Selection

1: **function** SAFETYNBV($\Xi, \Lambda, \tau$)
2: | $\Lambda_c = findPassingVoxels(\Lambda, \tau)$ ▷ Finding voxels through which the hand is passing
3: | Using $\Lambda_c$, let $\overset{*}{\xi}$ be selected according to Eq 6.14
4: | $value = u_2(\overset{*}{\xi}, \Lambda_c)$
5: | Return $(\overset{*}{\xi}, value)$
6: **end function**

By using a log-odds representation of our occupancy probability such as in [Hornung et al., 2013; Moravec and Elfes, 1985], we can then define the future predicted occupancy probability of $s_d$ as

$$L(s_d|o_{1:t}, o'_{t+1}) = L(s_d|o_{1:t}) + L(s_d|o'_{t+1}), \tag{6.8}$$

where $o'_{t+1} \in O = \{occupied, free\}$ is an imaginary future measurement and $L(s_d|o)$ is also called *inverse sensor model* [Thrun et al., 2005]. The inverse sensor model is defined likewise as in [Hornung et al., 2013] as

$$L(s_d|o) = \begin{cases} L_{occ}, & \text{if } o = occupied. \\ L_{miss}, & \text{otherwise.} \end{cases} \tag{6.9}$$

Note that our occupancy probability converted from log-odds is then

$$p_{sd|o'_t} = p(s_d|o_{1:t}, o'_{t+1}) = 1 - \frac{1}{1 + exp(L(s_d|o_{1:t}, o'_{t+1}))}. \tag{6.10}$$

We make a simplifying assumption that an imaginary measurement has uniform distribution, i.e. $p(occupied) = p(free) = 0.5$. Thus, we define our predicted entropy resulting from an imaginary measurement as the expected value

$$\begin{aligned} H'(s_d) = -\sum_{o' \in O} p(o') \{ & p_{sd|o'} \log(p_{sd|o'}) \\ & + (1 - p_{sd|o'}) \log(1 - p_{sd|o'}) \} \end{aligned} \tag{6.11}$$

Therefore, the information gain of looking at a particular voxel $s_d \in \Lambda_{visible}(\xi)$ from a given view $\xi$ is given by

$$I(\xi, s_d) = H(s_d) - H'(s_d), \tag{6.12}$$

where the average information gain per voxel is given by

$$u_2(\xi, \Lambda_{visible}(\xi), \Gamma_t) = \sum_{s_d \in \Lambda_{visible}} \frac{I(\xi, s_d)}{D}, \tag{6.13}$$

where $D = |\Lambda_{visible}|$ is the number of visible voxels. Note that we refer to the average information gain per voxel since different views have different numbers of visible voxels in their field of view after frustum culling. This makes the predicted information different gain for different views comparable.

#### 6.1.3.2 Information Gain View Selection

Using the definitions aforementioned, when no contacts are available, we are finally able to select next best views according to a maximum information gain strategy via

$$\overset{*}{\xi} = \arg \max_{\xi_k \in \Xi - V} u_2(\xi_k, \Lambda_{visible}(\xi), \Gamma_t). \tag{6.14}$$

### 6.1.4 Safety View Planning

In safety view planning we are interested in estimating the probability of collision prior to the execution of a given trajectory $\tau$, disregarding the collision with the final contact points $\Omega$. Effectively, we estimate the probability of an unexpected collision along the trajectory $\tau$. This is a typical scenario in which the robot hand collides with an unknown part of the object due to the fact that the collision free grasp was originally planned using an incomplete model of the object's shape $\Gamma_t$. In addition, we are also able to access how certain we are regarding this collision estimation by computing the current entropy for this particular trajectory $\tau$.

**Algorithm 13** Grasp Driven Active Sense

**procedure** ACTIVEGRASP($\Xi, G$)
  $\Gamma = \emptyset, \Lambda = \emptyset$
  $V = \emptyset, T = \emptyset$ ▷ List of visited views and found trajectories, respectively. Initially empty
  $grasp = false$
  $\overset{*}{\tau} = None$
  **while** not $grasp$ **do**
    $V, \overset{*}{\tau}, \Gamma, \Lambda = nextBestView(\Xi, \Gamma, \Lambda, G, V, T)$
    $V, p, \Gamma, \Lambda = safetyExploration(\Xi, \Gamma, \Lambda, \overset{*}{\tau}, V)$
    **if** $p \leq \alpha$ **then**
      $grasp = true$
    **else**
      $T = T - \{\overset{*}{\tau}\}$ ▷ Removing $\overset{*}{\tau}$ from our candidate trajectories
    **end if**
  **end while**
  $executeGrasp(\overset{*}{\tau})$
**end procedure**

As such, we select views so as to minimise the entropy of the voxels through which the robot hand is going to pass when following a given grasp trajectory $\tau$. Thus, while a path planning algorithm provides a candidate trajectory, such trajectory is not necessarily collision free, because the information acquired so far from the scene is incomplete. With a safety exploration procedure we are able to add a layer of reasoning about how much we should trust this planned path to be really free of unexpected collisions. This enables us to have a final relatively certain estimation with respect to unexpected collisions that might damage the robot hand, or simply make the grasp fail.

Let the set of voxels through which the hand bounds pass when following a trajectory $\tau$ be $\Lambda_c$. These voxels are retrieved by simulating the hand moving along the trajectory $\tau$ and querying at each time step of this trajectory the voxels the hand is passing through in our voxelised workspace $\Lambda$. Having retrieved those voxels, let $p_{s_c}$ be the probability of occupancy of a given voxel $s_c \in \Lambda_c$. The probability of collision can be calculated by

$$p_{collision}(\tau, \Lambda_c) = 1 - \prod_{s_c \in \Lambda_c} (1 - p_{s_c}). \tag{6.15}$$

For numerical reasons, we prefer to refer to Eq 6.15 using only the product term, representing the probability that all voxels along the trajectory $\tau$ are free, in its logarithmic form as

$$\kappa(\tau, \Lambda_c) = \ln \prod_{s_c \in \Lambda_c} (1 - p_{s_c}) = \sum_{s_c \in \Lambda_c} \ln(1 - p_{s_c}), \tag{6.16}$$

note that $p_{collision}(\tau, \Lambda_c) = 1 - exp(\kappa(\tau, \Lambda_c))$.

Finally, to select views in order to get better estimations for Eq 6.15, we use the same utility function defined in 6.13. Thus if we let $\Lambda_{c_{visible}}(\xi) \subset \Lambda_c$ be the set of visible voxels by a certain view pose $\xi$. Next best views are then selected according to

$$\overset{*}{\xi} = \arg \max_{\xi_k \in \Xi - V} u_2(\xi_k, \Lambda_{c_{visible}}(\xi), \Gamma_t). \tag{6.17}$$

In practice, we allow safety exploration to run while the information gain is above a predefined threshold, i.e. $u_2(\xi_k, \Lambda_{c_{visible}}(\xi), \Gamma_t) > \beta$. If this criteria is not met, the final probability of collision is reported according to Eq 6.15. The trajectory $\tau$ is therefore executed or not based on the probability of collision.

## 6.2 Experiments

Pseudocode for our approach is given in Alg 13, which is divided into two sub-phases. First, a contact-based next best view exploration procedure is run as outlined by Alg 9. In this first phase, at least two views are selected, up to a maximum of 7 views if after the second view no grasp trajectory and contacts are found. The first view is fixed, only subsequent views after this fixed view are selected according to the criteria for contact-based view selection. The second phase of Alg 13 is run in order to estimate the probability of collision of the

most promising candidate trajectory $\overset{*}{\tau}$, selected as the trajectory with the lowest probability of collision prior to the safety view exploration phase, given our current knowledge of the object $\Gamma_t$ and workspace $\Lambda$. This second phase is outlined in Alg 11. Note that the safety exploration phase stops if the currently selected view predicts information gain below a certain threshold $\beta$. If, at the end of the safety exploration phase, we discover that this trajectory $\overset{*}{\tau}$ has collision probability above a certain threshold $\alpha$, we reject $\overset{*}{\tau}$ and cycle back to phase 1, i.e. Alg 9.

### 6.2.1 Methodology

Fig. 6.6 The 14 objects used for trials.

Using Alg 13 we performed trials on a set of 14 novel objects shown in Fig 6.6. A total of 4 trials per object were performed. The set of views were thirty-four roughly evenly spaced locations on two concentric view hemispheres pointing at the centre of the workspace.[2] In our experiments, we compared our algorithm with a random view selection strategy. In other words, we substituted all calls of the selection procedures Alg 10 and Alg 12 by a uniform random view selection scheme. Furthermore, we limited the two phases of this modified randomised approach to be constrained to the same number of views that our algorithm

[2]Note that the algorithm can easily consider views generated on the fly. We restrict the view set here because the full visual SLAM system then required is beyond the scope of this thesis.

performed in both phases. It is also important to note that in our experiments we have set the size of the voxels in our 3D occupancy map $\Lambda$ to be $0.0025m$, for relatively fine precision. Table 6.1 shows the final data for this experiment. In addition, we tested performance on a cluttered scene using an early version of the algorithm that employed only the grasp contact refinement strategy.

All trials were performed with the Boris manipulation platform, which was mounted with a PrimeSense Carmine 1.09 depth camera on its wrist. The robot hand utilised was the DLR-HIT2 hand [Liu et al., 2008]. A grasp was considered successful if the robot could lift the object without letting it fall or slip.

### 6.2.2 Results

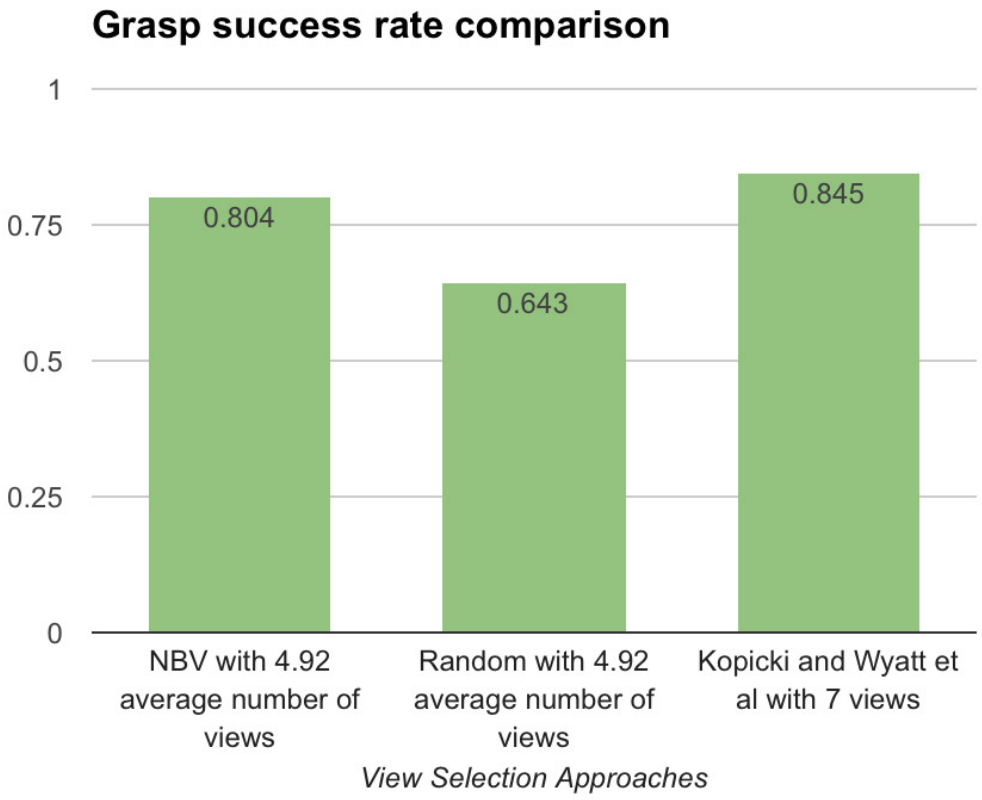


Fig. 6.7 Grasp success rate comparison.

The results shown in Table 6.1 show the contrast between the two approaches. We first note that the success rate of our proposed view selection approach was 80.4%, whereas the modified randomised approach showed a success rate of only 64.3%. A closer look at Table 6.1 reveals that random exploration tended to yield unsafe grasps, under the same view number constraints as our active view selection approach. This indicates that random view selection would probably need to cycle back to generate new grasp trajectory candidates more times, which seems a natural consequence of its sub-optimal exploratory behaviour. One such example is highlighted by Fig 6.8, in which the final trajectory executed with a probability of collision 1.0 and, indeed, makes the robot hand collide with a part of the mug not involved in the grasp, finally failing for safety reasons. We also note that our collision probability appears to be over-sensitive, the random approach also succeeded for various cases in which the probability of collision was 1.0. Nonetheless, even for successful grasps as the one depicted in Fig 6.2, grasps with probability 1.0 tended to collide prematurely with different parts of the object. In addition, we also noted that for the case of the toothpaste, the trivial solution of a grasp with as few collisions as possible might yield grasps with very poor grip. This indicates future work towards a middle ground between these two extremes.

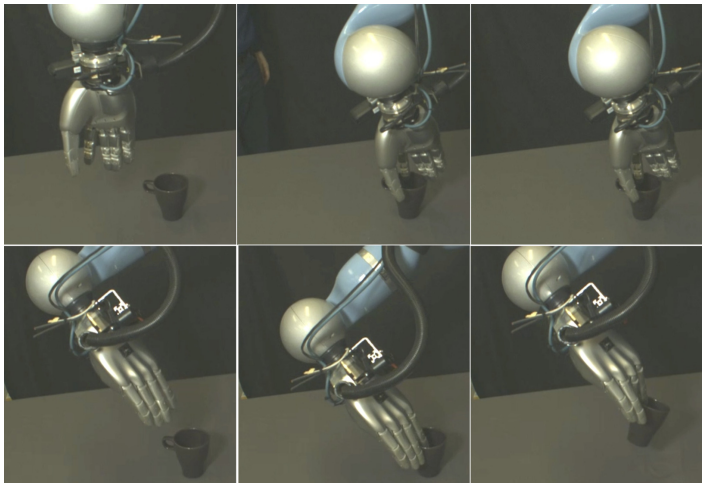

Fig. 6.8 Top three pictures show a failed grasp due to unexpected collision with parts of the object that are not involved in the grasp. Bottom three pictures show a successful and safe grasp selected by our approach.

As shown by Table 6.1 and in Fig 6.7, our approach had competitive success rate to prior work done by [Kopicki et al., 2015]. In our experiments, our approach used on average 4.92 views for grasp planning, as compared with 7 in [Kopicki et al., 2015]. Additional views were only used to assess safety [3].

Finally, we tested the robot on a cluttered scene with three novel objects. This single trial completed successfully [4].

Table 6.1 Trial Results

| | Phase 1 Average View Count | Phase 2 Average View Count | Average Success Rate per Object | | Average Collision Probability per Object | | Average Grasp Views |
|---|---|---|---|---|---|---|---|
| Objects | NBV & Random | NBV & Random | NBV | Random | NBV | Random | t of $\Gamma_t$ |
| bowl small | 2.67 | 3.67 | 1 | 1 | 0.0006166666667 | 0.6667388333 | 3.17 |
| bucket | 4.17 | 3.34 | 1 | 1 | 0.0000525 | 0.5037015 | 6.34 |
| chopsticks | 3 | 3.67 | 1 | 0.75 | 0.005303 | 0.0002226666667 | 5.67 |
| coke | 2.5 | 4.5 | 0.75 | 0.25 | 0.01711816667 | 0.6674898333 | 3.67 |
| cup1 | 3.17 | 4.34 | 0.75 | 0.5 | 0.001250833333 | 0.8333813333 | 6.17 |
| dustpan | 3 | 3 | 1 | 1 | 0.005206333333 | 1.0 | 4 |
| glass2 | 3.5 | 3.17 | 0.25 | 0 | 0.6666955 | 1.0 | 4 |
| guttering | 3 | 3.67 | 1 | 0.5 | 0.005433166667 | 0.8333818333 | 4 |
| jug | 3.34 | 4.34 | 1 | 1 | 0.07424566667 | 0.833678 | 6.34 |
| mrmuscle | 2.67 | 3.17 | 1 | 0.75 | 0.002464 | 1.0 | 4.5 |
| mug1 | 2.84 | 3.67 | 0.25 | 0.25 | 0.003175666667 | 0.6672291667 | 5.34 |
| rennie | 3 | 3.17 | 1 | 1 | 0.004263666667 | 0.5001438333 | 4.17 |
| stand2 | 4.67 | 2.34 | 0.5 | 0.5 | 0.003219666667 | 0.666765 | 4.67 |
| toothpaste | 3.84 | 4 | 0.75 | 0.5 | 0.003282333333 | 0.3352755 | 6.84 |
| | | Overall Success Rate | 0.804 | 0.643 | | Overall Grasp Views | 4.92 |

## 6.3 Conclusions

In this chapter, we have proposed an effective approach for view selection comprising two stages. The first stage guides gaze by planned contacts, seeking a low noise point cloud near those contacts. If no contact points are available, we guide gaze so as to minimise the entropy of the 3D occupancy grid map around our object cloud $\Gamma_t$. After candidate grasps

[3] A video illustrating our approach can be found at https://youtu.be/uBSOO6tMzwA

[4] See https://youtu.be/hnGgsWtKzPw.

are found, we gaze to assess the safety of the reach-to-grasp trajectory candidate. We have thus demonstrated that a generative grasp synthesis approach does provide sensible cues for guiding perceptual actions. This was made possible by first defining a task specific utility function that seeks to minimise noisy point cloud readings from a depth sensor in the vicinity of generated grasp contacts. Second, in the absence of contacts, an additional utility function has been defined so as to minimise entropy about the object shape around a given point cloud. And finally, a safety utility function is used to make predictions and assess the safety of a grasp execution in terms of collision probability along a given grasp trajectory. We showed that this yields a better success rate compared to a random strategy and competitive success rate to [Kopicki et al., 2015], while using fewer views for grasp planning. We have also demonstrated the system works well in cluttered scenes, and is able to clear a table with objects placed closed together, subsequently placing them in a designated container.

In future work, we would like to explore methods not only for predicting perceptual utility values, by also to predict the consequences of manipulation actions themselves, such as grasp actions. Indeed, in the following Chapter 7, we will perform steps in this direction by investigating models and control strategies that take into account uncertainty in manipulation actions.

# Chapter 7

# Learning predictive models for uncertainty averse pushing

In the previous **Chapter 6**, we have seen that generative grasp synthesis approaches allowed us to make perceptual predictions with respect to task (contact-based) and information theoretic utility functions for geometric knowledge acquisition. That allowed us to reduce the number of views for grasp synthesis, as well as maximise safety in terms of probability of collision with undiscovered parts of the workspace. Grasp actions were then executed safely and slightly more efficient in terms of number of views for grasp generation.

In this chapter, we investigate the idea of prediction even further not only for predicting perceptual outcomes, but also by reasoning about the uncertainty of outcomes in manipulation actions themselves. As a first step, we focus particularly on an elementary task of object pushing. We conjecture that the models and control strategies described in this chapter can also be adapted and generalised to guide the execution of other manipulation actions, such as grasping.

Manipulating objects via non-prehensile actions, such as pushing, is a well-known problem in robotics [Cosgun et al., 2011; Dogar and Srinivasa, 2011]. Planning these push-manipulations requires a forward model. There are many ways to express and acquire such a

model, from analytic mechanics to machine learning, as well as hybrid techniques. There are several open problems, of which we address two. First, as discussed in Chapter 2.3, push manipulation typically does not take account of all the types of uncertainty in the predictions of the forward model. Second, when using purely learned models, push planning has only been demonstrated for single pushes, not for complex push sequences. In this work we present a combined solution to these problems.

We have seen in Chapter 2.3.2.1 that uncertainty in prediction come from two sources. First, it can arise from small variations in physical properties, such as shape and friction, that are hard to measure, but which significantly alter action effects. These effects are typically modelled as aleatoric uncertainty, which can be in simpler cases homoscedastic (does not dependent on input variable) or in more complex scenarios it can be heteroscedastic (input dependent). Second, in a learned forward model it can arise from a paucity of data. This type of uncertainty is termed meta-uncertainty. In this paper, we use two different learning methods to explicitly predict a distribution over push outcomes, including both types of uncertainty. In particular, we propose a model that versatile enough to model heteroscedastic sources of uncertainty for pushing based on an ensemble of MDNs, as first introduced in Chapter 3.

When planning an action sequence the robot can take account of regions of high uncertainty. This is important because actions in uncertain regions of the state space can lead to unrecoverable states. We model these as incurring a cost that rises with the uncertainty predicted by the forward model. But, what push planner can we use? The choice is complicated by the fact that the overall cost function is typically not a differentiable function of the actions. In this case, a path integral formulation [Kappen, 2005; Theodorou et al., 2010; Williams et al., 2015] works well. We also utilise re-planning each step to account for model inaccuracies. Thus, our planner is a model predictive path integral (MPPI) controller that performs uncertainty averse pushing.

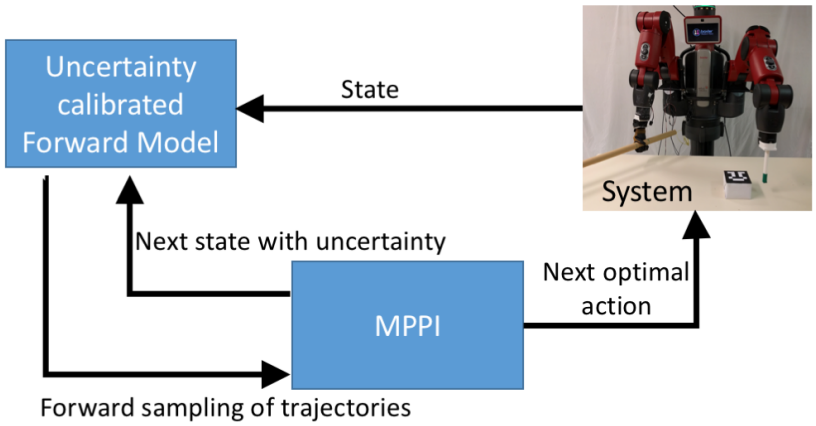


Fig. 7.1 Picture shows a high-level diagram of the approach. The forward model is trained and acquired from collected pushing data, and is able to provide uncertainty estimates for in its predictions. The system comprises the robot, the box and the environment.

The technical contributions of this work are[1]: (i) applying ensembles of MDNs and Gaussian Processes to learn uncertainty aware forward models of push manipulation; (ii) using a learnt forward model with model predictive path integral (MPPI) control to push an object to a given goal pose with a many step plan; and (iii) two algorithms for uncertainty averse push planning.

This chapter is organised as follows. Section 7.1 gives a brief background on the basic representations underpinning our approach. Section 7.2 explains the problem formulation and algorithms. Section 7.3 details the experimental study. Section 7.4 presents a discussion of the obtained results. Finally, Section 7.5 concludes and discusses future work.

## 7.1 Preliminaries

In this work, we define a trajectory as a sequence of states $\mathbf{x}_t$, actions $\mathbf{u}_t$ and associated predictive uncertainty $\hat{\sigma}_{t,n}$ at time step $t$. Thus, let $\mathbf{s}_{t,n}$ be a convenient tuple, such that $\mathbf{s}_{t,n} = (\mathbf{x}_{t,n}, \mathbf{u}_{t,n}, \hat{\sigma}_{t,n})$. Then, using $n$ to index a trajectory and $i$ to index a discrete timestep,

[1] Since in this published work both Michael J. Mathew and I are co-first authors. In this thesis we make clear the division of work. I have individually contributed to (i); both Michael J. Mathew and I co-contributed and developed (ii); and, finally, contribution (iii) has been solely developed by Michael J. Mathew.

the optimal control problem is cast by defining the following cost function for a trajectory $n$ starting from timestep $i$ until $T$, i.e. $\tau_{i,n} = [\mathbf{s}_{i,n}, \mathbf{s}_{i+1,n}, \mathbf{s}_{i+2,n}, ..., \mathbf{s}_{T,n}]$:

$$S_i = S(\tau_{i,n}) = \phi_T(\mathbf{x}_T) + \sum_{t=i}^{T-1} r_t(\mathbf{x}_t, \mathbf{u}_t, \hat{\sigma}_t) \tag{7.1}$$

where, $r_t(\cdot,\cdot,\cdot)$ is the immediate cost function for every timestep and $\phi_T(\cdot)$ is the final cost function at the last time step $T$. Our goal is to find a policy defined as a sequence of actions $\mathbf{u}_t$ that minimises the above cost function in Eq. 7.1.

The value function at a state $\mathbf{x}_i$ is defined as the minimum expected cumulative cost considering the agent starts $\mathbf{x}_i$ and proceeds optimally from that state to the goal. This can be written as

$$V(\mathbf{x}_i) = \min_{\mathbf{u}_{i:T}} E[S_i] \tag{7.2}$$

At any state $\mathbf{x}_i$ we seek to find a set of control commands or actions $\mathbf{u} = (\mathbf{u}_i, ...\mathbf{u}_T)$ that minimises the expected cumulative cost from that state. Note that Eq. 7.2 represents the expectation over all possible paths (i.e. trajectories). Hence, the trajectory index $n$ is dropped for convenience.

## 7.2 Approach

Our proposed uncertainty averse push approach is composed of two parts. First, an uncertainty calibrated forward model is learnt using data collected via self-supervision from a variety of pushes on a given object. Second, using the learnt forward model, we formalise our approach as a stochastic optimal control problem that is solved via Model Predictive Path Integral control (MPPI). In addition, we also describe an alternative path integral based approach to find a low cost trajectory that can be followed by using the MPPI controller.

### 7.2.1 Forward models with predictive uncertainty

In Chapter 3 we discussed a couple of approaches to capture uncertainty in predictions. For instance, we have seen that GPs provide an effective and theoretically sound tool to incorporate meta-uncertainty predictions [Rasmussen, 2006]. However, standard GPs present a few drawbacks due to their inability to scale to high dimensional spaces or large data sets, and *homoscedasticity* assumption when modelling aleatoric uncertainty. Therefore, to overcome these limitations we utilise an ensemble of mixture density networks (E-MDN) as described in Chapter 3.3.2.

We have also seen in Chapter 3.1.4 that arbitrary densities can be learnt via gradient descent with Mixture Density Networks (MDNs) [Bishop, 1994]. In such models, if we choose the mixture components to be Gaussian, the network outputs the parameters of a Gaussian mixture model conditioned on a suitable choice of input vector $\mathbf{h} \in \mathbb{R}^n$. Thus, modelling arbitrary multi-modal densities as defined in equation 7.3.

$$p_\theta(\mathbf{x}_{t+1}|\mathbf{h};\theta) = \sum_{k=1}^{K} \pi_k(\mathbf{h};\theta)\mathcal{N}(\mathbf{x}_{t+1}|\mu_\mathbf{k}(\mathbf{h};\theta), \sigma_k^2(\mathbf{h};\theta)), \tag{7.3}$$

where, $\pi_k(\mathbf{h};\theta)$, $\mu_k(\mathbf{h};\theta)$ and $\sigma_k(\mathbf{h};\theta)$ are the network outputs which form the parameters of the mixture. In order to make sure the network outputs valid parameters for the mixture, Bishop [1994] suggests using a softmax layer to represent $\pi_k$, such that $\sum_{k=1}^{K} \pi_k(\mathbf{h};\theta) = 1$, whereas an exponential layer is able to ensure that $\sigma_k$ is positive definite, and finally $\mu_k$ can be defined a linear combination of hyperbolic tangent activation functions. Comprehensive practical considerations when implementing MDNs are discussed by Bishop [1994] and Graves [2013].

Concretely, for learning a forward model for pushing, we define the concatenated state-action vector as $\mathbf{h} = [\mathbf{x}_t, \mathbf{u}_t] \in \mathbb{R}^6$. Where, $\mathbf{x}_t \in \mathbb{R}^3$ is the state at time step $t$, composed of the position and orientation of the box on the plane, i.e. $\mathbf{x}_t = [x_t, y_t, \theta_t]$. The vector $\mathbf{u}_t$ is the action taken at time $t$, and is encoded as a direction vector $[px_t, py_t]$, and a single real

value $a_t$ normalised between $[0,1]$ that indicates the contact location on the box edge, i.e. $\mathbf{u}_t = [px_t, py_t, a_t]$ (all quantities are given in the object frame). Furthermore, as has been demonstrated by [Lakshminarayanan et al., 2017] and introduced in Chapter 3.3.2, one can form an ensemble of MDN models so as to obtain uncertainty estimates for the forward model's predictions. Thus, if an ensemble is composed of $M$ members, the final model can be written as:

$$p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{M}\sum_{m=1}^{M} p_{\theta_m}(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t; \theta_m) \tag{7.4}$$

This model allows us to compute the statistics of interest from the ensemble, which are the mean prediction and the variance. When when trained accordingly, these quantities can reflect the predictive uncertainty of the model [Lakshminarayanan et al., 2017], and are given by:

$$\hat{\mu} = \frac{1}{M}\sum_{m=1}^{M} \mu_{\theta_m} \tag{7.5}$$

$$\hat{\sigma}^2 = \frac{1}{M}\sum_{m=1}^{M} (\sigma_{\theta_m}^2 + \mu_{\theta_m}^2) - \hat{\mu}^2 \tag{7.6}$$

Using E-MDNs as forward models provides several advantages. Such models scale well with large training datasets, time complexity of predictions are no longer a function of the size of training data (as is the case with GPs). Even more importantly, as intuitively demonstrated in Chapter 3.3.2, E-MDNs are able to model both meta-uncertainty and aleatoric uncertainty (including heteroscedasticity). Therefore, the predictive uncertainty we refer throughout this paper represents both inherent uncertainty in dynamics, and meta uncertainty due to lack of data, as given by Eq. 7.6.

### 7.2.2 Uncertainty averse model predictive path integral control

Having defined our choice of uncertainty calibrated forward model, we now need to define a way to use this model so as to find the right sequence of push actions to move the object to a desired goal. In order to accomplish this task, one would naturally expect that is easier to exploit the already known part of the state-action space rather than exploring unknown regions (parts of the forward model). This suggests that moving through more certain regions of the state-action space may be safer than venturing to regions not experienced during training of the forward model. Not only that, but also exploring actions that yield more certain outcomes when considering input-dependent uncertainty (heteroscedasticity). We utilise the path integral based approach to model predictive control in [Williams et al., 2015].

The path integral formulation for solving stochastic optimal control problems permits one to find policy updates by calculating expectations over trajectory roll outs [Theodorou et al., 2010]. This is an alternative to directly solving the non-linear HJB equation via backward integration, and allows one to find the command updates that minimise the cost-to-go in Eq 7.7 as a weighted average over $N$ forward sampled trajectories. This choice of control strategy is motivated in the context of pushing two reasons. First, it is able to cope with non-linear dynamics. Second, it supports arbitrary state-dependent costs and is solvable using sampling based approaches [Kappen, 2005] (see also Chapter 4.4).

$$S(\tau_{i,n}) = \phi(\mathbf{x}_T) + \sum_{t=i}^{T-1} r(\mathbf{x}_t, \mathbf{u}_t, \hat{\sigma}_t). \tag{7.7}$$

Having defined the cost-to-go in Eq 7.7, one wants to find the optimal action sequence as $\overset{*}{\mathbf{u}} = \arg\min_{\mathbf{u}} E[S_i]$. Note that, since using the path integral formulation we are able to define arbitrary state-dependent costs, in this work the cost is also a function of the predictive uncertainty $\hat{\sigma}_t$, in addition to the state $\mathbf{x}_t \in \mathbb{R}^n$ and controls $\mathbf{u}_t \in \mathbb{R}^m$. The importance of each $n^{th}$ sample is given by a weight, defined as the exponential of the cost-to-go $S(\tau_{i,n})$, which is given by:

$$w_{i,n} = \frac{\exp(-\frac{1}{\lambda}S(\tau_{i,n}))}{\sum_{l=1}^{N} exp(-\frac{1}{\lambda}S(\tau_{i,l}))}, \tag{7.8}$$

where $\lambda$ can be seen as a temperature parameter for the softmax distribution in Eq 7.8. It affects the control update given by Eq 7.9.

$$\Delta \mathbf{u}_i = \sum_{n=0}^{N} w_{i,n} \delta \mathbf{u}_{i,n}(\eta). \tag{7.9}$$

Thus, the Eq. 7.9 above defines control command updates as an expectation, or weighted average, over sampled control disturbances $\delta \mathbf{u}_{i,n}$ with weights given by $w_{i,n}$. Here, control disturbances are given in a similar fashion as in [Williams et al., 2015]. However, we introduce an exploration decay parameter $\eta$, i.e.

$$\delta \mathbf{u}_{i,n}(\eta) = \eta \frac{1}{\sqrt{\rho}} \frac{\varepsilon}{\sqrt{\Delta t}}, \tag{7.10}$$

where $\Delta t$ is the time step magnitude, with $\varepsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, which has same dimensionality as the control actions $\mathbf{u_t} \in \mathbb{R}^m$. The parameter $\rho$ can be seen as a constant responsible for controlling the magnitude or level of exploration of the sampled disturbances $\delta \mathbf{u}_{i,n}$, whereas a suitable exploration decay schedule for $\eta$ helps to ensure local convergence even when the number $N$ of sampled trajectories is small. In all experiments presented in this paper $\eta$ is always initialised as $\eta = 1.0$ and geometrically decays over a chosen number of decay steps $L$ as detailed in Algorithm 14.

The advantage of utilising a model predictive approach is that it allows the incorporation of feedback into the system. It proceeds as follows. Stochastic trajectory rollouts using our learnt forward model are generated starting at the current sensed state $i$. Each generated trajectory contains $T$ time steps, and represents a look-ahead window of fixed $T$. The sampled rollouts allows us to update our fixed window of size $T$ of control commands, such that $\mathbf{u}_i \leftarrow \mathbf{u}_i + \Delta \mathbf{u}_i$, as given by Eq. 7.9. Then, the first control command $\mathbf{u}_0$ of the $T$ steps is executed on the robot. After this first push, the new state of the object is fed back into the

optimiser and the process is repeated till task convergence. This procedure is outlined by Alg. 14.

In this work, the immediate cost function used in our formulation is

$$r(\mathbf{x}_i, \mathbf{u}_i, \hat{\sigma}_i) = \gamma * \hat{\sigma}_i + (\mathbf{x}_i - \mathbf{x}_{goal})^T Q(\mathbf{x}_i - \mathbf{x}_{goal}) + \mathbf{u}_i^T R \mathbf{u}_i, \tag{7.11}$$

and the final cost is given by

$$\phi(\mathbf{x}_T) = (\mathbf{x}_T - \mathbf{x}_{goal})^T Q(\mathbf{x}_T - \mathbf{x}_{goal}) \tag{7.12}$$

where $Q$ and $R$ are positive-definite $3x3$ gain matrices, and $\gamma$ is a positive constant. These constants define relative costs for minimising uncertainty, reaching the desired goal, and minimising magnitude of control commands.

By adding the $\hat{\sigma}$ term in equation 7.11, the samples that pass through an uncertain region are penalized more and hence would contribute less to the control update in equation 7.9. In the remainder of this chapter, the state of the object to be pushed is defined by its position and orientation on the plane under the quasi-static assumption, subject to only planar motion, i.e. $\mathbf{x}_i = [x_i, y_i, \theta_i]$.

**Algorithm 14** The modified MPPI algorithm with exploration decay, a modification of the the original algorithm proposed by [Williams et al., 2015] that uses an uncertainty calibrated forward model (Eq 7.5 and 7.6)

**Given**:
N: Number of samples;
T: Number of timesteps;
L: Number of decay steps;
$(\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_T)$: Initial action sequence;
$\eta_{init}$: Initial $\eta$, for exploration decay, Eq 7.10;
$\Delta t, \mathbf{x}_{init}, \hat{\mu}(\cdot,\cdot), \hat{\sigma}(\cdot,\cdot)$: System sampling dynamics;
$\phi, r, \mathbf{R}, \mathbf{Q}, \lambda, \gamma$: Cost parameters;
$\mathbf{u}_{init}$: Value to initialise new controls to;

**while** *task not completed* **do**
  $\eta = \eta_{init}$
  **for** $l = 0$ **to** $L$ **do**
    **for** $n = 1$ **to** $N$ **do**
      $\mathbf{x_0} = \mathbf{x}_{init}$
      $\hat{\sigma_0} = \hat{\sigma}(\mathbf{x}_0, \mathbf{0})$ (given by Eq 7.6)
      **for** $i = 1$ **to** $T-1$ **do**
        $\mathbf{x}_{i+1} = \hat{\mu}(\mathbf{x}_i, \mathbf{u}_i + \delta\mathbf{u}_{i,n}(\eta))$ (given by Eq 7.5)
        $\hat{\sigma_{i+1}} = \hat{\sigma}(\mathbf{x}_i, \mathbf{u}_i + \delta\mathbf{u}_{i,n}(\eta))$ (given by Eq 7.6)
        $S(\tau_{i+1,n}) = S(\tau_{i,n}) + r(\mathbf{x_i}, \mathbf{u_i}, \hat{\sigma}_i)$
      **end for**
      $S(\tau_{T,n}) = \phi(\mathbf{x_T})$
    **end for**
    **for** $i = 1$ **to** $T$ **do**
      $\mathbf{u}_i = \mathbf{u}_i + \Delta\mathbf{u}_i$ (with $\Delta\mathbf{u}_i$ given by Eq 7.9)
    **end for**
    $\eta = 0.99\eta$ (exploration decay)
  **end for**
  send_control_command($\mathbf{u}_0$)
  **for** $i = 1$ **to** $T-1$ **do**
    $\mathbf{u}_i = \mathbf{u}_{i+1}$
  **end for**
  $\mathbf{u}_{T-1} = \mathbf{u}_{init}$
  Update current state
  Check task completion
**end while**

### 7.2.3 An alternative approach to uncertainty averse planning

The performance of MPPI for uncertainty averse approach described in the previous section may depend on the cost function defined. The challenge lies in finding a right tradeoff choosing different constant values for $\mathbf{Q}$ and $\gamma$ in Eq. 7.11.

Instead of directly finding optimal control commands as a push controller, an alternative approach would be to decouple the process of finding a trajectory that reaches the goal, and only then optimise this trajectory so as to minimise uncertainty. It works as follows. First, a simple path is defined from start to desired goal configuration (e.g. an straight line, or a path from a previous moment). Second, using this given initial path, the learnt forward model can be used to find a low uncertainty variation of this path. Finally, repeating this process for various iterations yields a low-uncertainty trajectory that can be then given to the MPPI controller to be followed. The decoupling of the uncertainty cost and the final goal cost gives a significant improvement in performance. Hence, we also propose a path integral based approach to find an optimal trajectory in the state-action space that can then be followed by the MPPI. This alternative approach draws inspiration from the STOMP planner Kalakrishnan et al. [2011]. However, our formulation uses a different cost function and improvement equations. Apart from finding a low cost trajectory, kinematic constraints are also imposed on the optimisation problem to find paths within the workspace of the system. The problem of finding an optimised trajectory with low uncertainty can be formulated as:

$$\begin{gathered} \text{Minimise: } S(\tau_{i,n}) = \alpha \sum_{i=0}^{T} \hat{\sigma}_i \\ \text{Subject to state constraints: } \mathbf{x}_{min} \leq \mathbf{x} \leq \mathbf{x}_{max} \end{gathered} \tag{7.13}$$

where, $\sigma_i$ is the uncertainty predicted for state $\mathbf{x}$ at time $i$, and $\alpha$ is a positive constant. This approach requires that the uncertainty for each considered state is estimated beforehand, thus constructing an uncertainty heat-map. Here we perform this heatmap computation using via Monte Carlo sampling. Thus, by iterating over a subset of states (e.g. states on a grid) and

random actions for each state, the average uncertainty is estimated using the learnt forward model with Eq. 7.6 for each state. With this uncertainty heat-map, Alg. 15 starts with a initial trajectory leading from start to goal. For instance, this initial trajectory could be simple linear interpolation from start to goal. If the total number of steps in this interpolation is $T$, then the optimisation is performed for states from 1 to $T-1$. This ensures the path continues to reach the goal from the start. The initial trajectory is improved iteratively by usinq Eq. 7.8 for the cost defined in Eq. 7.13. The iterative update is given by:

$$\Delta \mathbf{x}_i = \sum_{n=0}^{N} w_{i,n} \delta \mathbf{x}_{i,n} \tag{7.14}$$

**Algorithm 15** Alg. 15 optimises a trajectory with respect to forward model predictive uncertainty. The output of the algorithm is a trajectory with low uncertainty cost which can be tracked by the MPPI push controller defined in Alg. 14.

**Given**:
N: Number of samples;
T: Number of time steps;
$f(\mathbf{x}, \mathbf{u})$: Learnt forward dynamics;
$\varepsilon$: Threshold of cost change;
$\tau$: $(\mathbf{x}_0, \mathbf{x}_2, ..., \mathbf{x}_{T-1})$;
$\alpha$: positive constant multiplier to cost;
**while** $S \geq \varepsilon$ **do**
  **for** $n = 0$ **to** $N$ **do**
    **for** $i = 1$ **to** $T-1$ **do**
      $\delta \mathbf{x}_{i,n} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
      $\mathbf{x}_{i,n} = \mathbf{x}_i + \delta \mathbf{x}_{i,n}$
      Enforce_state_constraints($\mathbf{x}_{n,i}$)
      $S(\tau_{i+1,n}) = S(\tau_{i,n}) + \frac{\alpha}{T} \hat{\sigma}_i$
    **end for**
  **end for**
  **for** $i = 1$ **to** $T-1$ **do**
    $\mathbf{x}_i = \mathbf{x}_i + \Delta \mathbf{x}_i$ (with $\Delta \mathbf{x}_i$ given by Eq 7.14)
    Enforce_state_constraints($x_i$)
  **end for**
**end while**

## 7.3 Experiments

The experiments 1 and 2 that are outlined below validate the fundamental ability to push objects to a goal location using the E-MDN as the choice of learnt forward model on a real robot (Baxter), initially without considering uncertainty in the cost function. Subsequently, experiments 3 and 4 demonstrate in simulation that with the same approach we are able to generate uncertainty averse trajectories in order to push an object to a desired goal location. In the experiments conducted in the real robot, we have found that uncertainty averse pushes achieved in simulation were sometimes difficult to reproduce on the Baxter platform, this

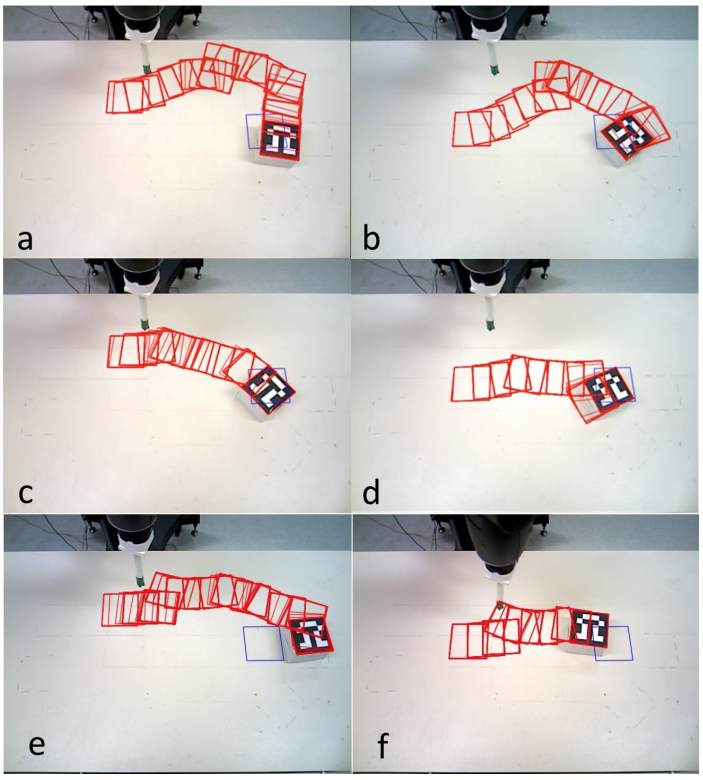


Fig. 7.2 Push sequences from two positions with MPPI. The first row shows the results for Box2D (a and b), the second row has the outcomes for E-MDN (c and d) and the final third row shows the results for GP (e and f). The red frames depict the starting and intermediate locations, while the blue frame shows the goal location. In the figures (a) and (c) the goal has the same orientation as but different position from the initial box pose, whereas in (b) and (d) the goal is orientated 90 degrees counter-clockwise with respect to the initial pose of the box. In our experiments, the MPPI strategy using E-MDN as a forward model reached the goal with fewer pushes.

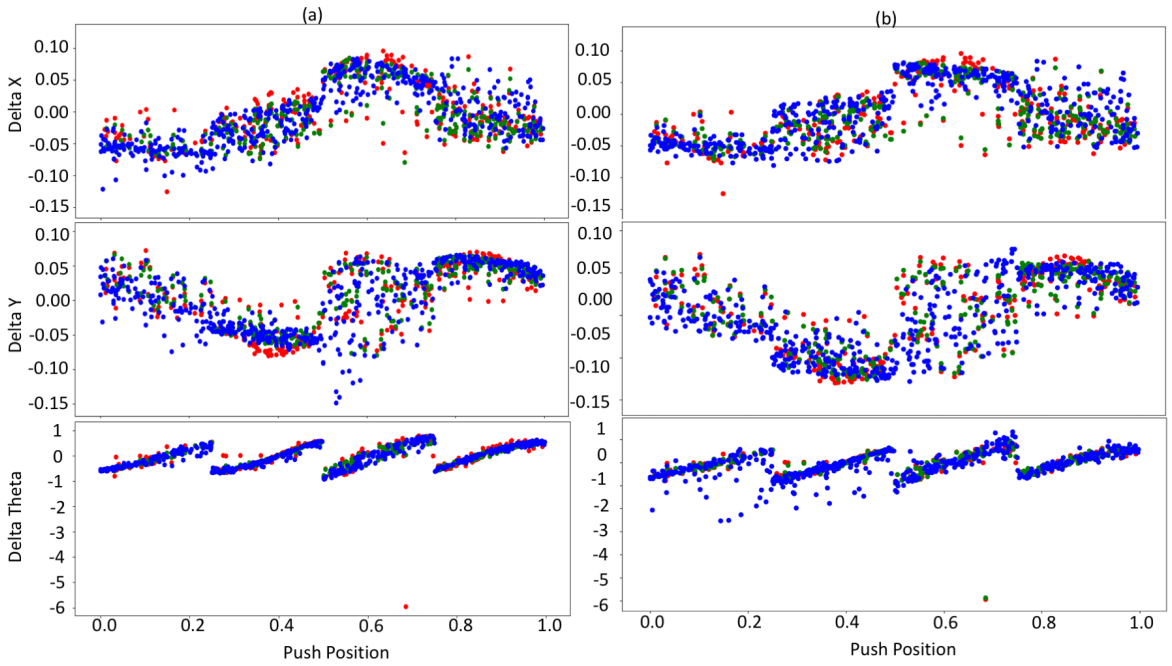


Fig. 7.3 (a) Predictions of the E-MDN. (b) Predictions of the GP. The top row shows predicted displacement in the X direction. The middle shows the displacements in the Y direction. Theta shows the change in orientation. The dots in red depict the true delta, whilist the dots in green are the predictions on the training data set. Finally, the blue dots portray the prediction on a test set created from a distribution with the same mean and covariance as the original training data set.

was due to kinematic limitations of the robot. For instance, the inverse kinematic solver occasionally failed to find solutions for planned pushes. Therefore, we demonstrate general pushing on the real robot, and the extra experiments on uncertainty averse pushing (which requires using a much larger workspace) is demonstrated in simulation only. We proceed to describe the experiments in the real robot. Subsequently, we will proceed to report the simulation experiments which demonstrate uncertainty averse pushing capabilities.[2]

### 7.3.1 Experiment 1

In this experiment we trained the E-MDN model on a set of 326 pushes, gathered from the Baxter robot [3]. The E-MDN utilised had $M = 10$ members in the ensemble, each member being an MDN with 3 hidden layers with 20 neurons per layer. The number of mixtures in each MDN was chosen to be $K = 1$ for simplicity. The model was trained for 3000 epochs with stochastic gradient descent using the Adam [Diederik and Jimmy, 2014] optimiser. We utilised batch size 5 and the learning rate was 0.001. Following the training protocol described by [Lakshminarayanan et al., 2017], we used 0.005 as the adversarial coefficient for generating adversarial examples for training. Fig. 7.3(a) and (b) shows the predictions given by the trained E-MDN model and the GP model respectively.

To show that uncertainty rises when the amount of available data is limited we also gathered data from randomised pushes in Box-2D [Catto, 2011]. Then we lesioned the data for various parts of the state space, and trained the E-MDN model. The uncertainty should be higher in the lesioned parts of space, and this is what we see in Fig. 7.4, in which the average uncertainty for a given state was obtained via marginalisation over the action space at the given state (average uncertainty for a given box state $\mathbf{x}$ was calculated using Monte-Carlo action samples).

### 7.3.2 Experiment 2

We performed experiments with the real robot, using the MPPI planner to plan with the learnt models from Experiment 1, and also with a physics simulator suitable for planar pushing called Box-2D [Catto, 2011]. We use this to investigate whether the push planning framework can be combined with a variety of forward models to achieve many-step push manipulations. Some push sequences are visualised in Fig. 7.2. These show that both learning methods

[2] See video summarising the proposed approach and experiments: https://youtu.be/LjYruxwxkPM

[3] see data collection setup, in which the robot applies random pushes to the object and periodically restarts the box to an initial location: https://youtu.be/pRDvkDkCSTQ

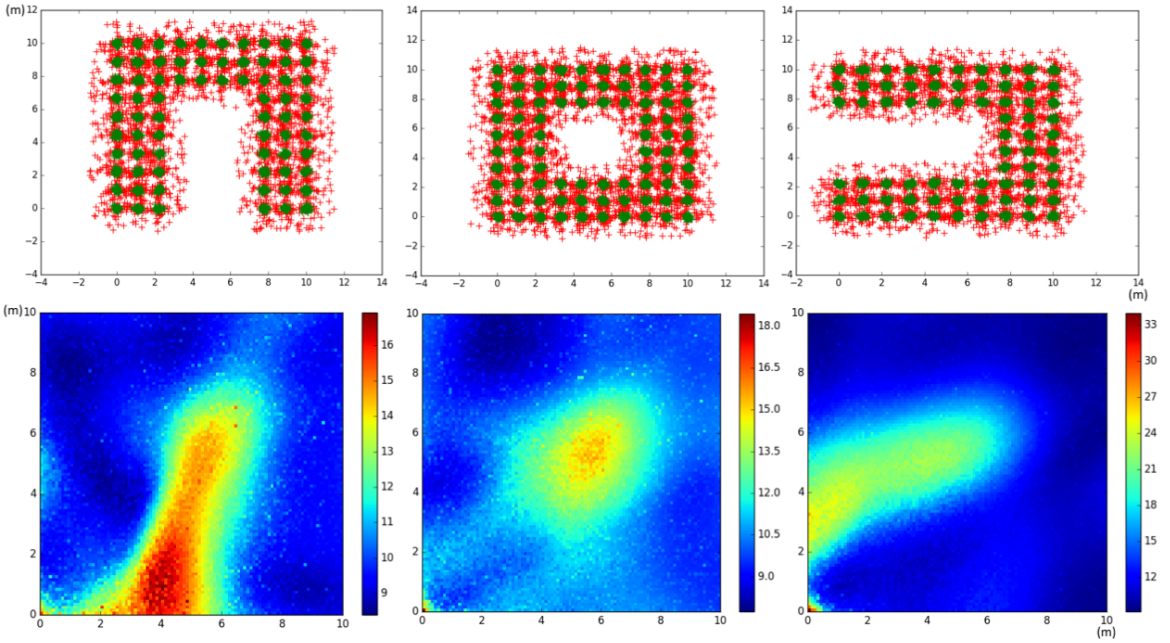

Fig. 7.4 The figure shows acquired simulated data from randomly applied pushes in Box2D. The dataset is subsequently lesioned in different ways (left). The corresponding heat maps depicting the forward model predictive uncertainty in different regions of the state space are also shown (right). Starting locations are represented as green crosses, positions after pushes are depicted as red crosses.

terminate with positions close to, but not at, the goal, in terms of both orientation and position. Table 7.1 shows that both substantially reduce the cost in paired trials, and that there is no clear difference in performance between the different predictors underpinning MPPI. The cost parameters for the MPPI were chosen to be $\mathbf{Q} = diag(1.5, 1.5, 0.01)$, $\gamma = 0$, $\mathbf{R} = \mathbf{0}$. The optimisation horizon was set to $T = 2$ and the number of sampled trajectory rollouts was chosen to be $N = 150$, $L = 20$, with $h = 1$, $\rho = 1.0$, and $\Delta t = 0.05$.

### 7.3.3 Experiment 3

Utilising Alg. 14, together with the cost function defined by Equation 7.11 set to penalise uncertainty for 150 pushes, and having the uncertainty penalty switched off there after. The aim of this experiment was to show in simulation that, with the right trade-off between the goal and the uncertainty gains, a box can be pushed to a desired location while avoiding

| Treatment | Starting pose | Initial cost | Final cost | Steps |
|---|---|---|---|---|
| MPPI-Box2D | Pose 1 | 0.800 | 0.112 | 13 |
| MPPI-E-MDN | Pose 1 | 0.795 | 0.057 | 8 |
| MPPI-GP | Pose 1 | 0.764 | 0.255 | 12 |
| MPPI-Box2D | Pose 2 | 0.766 | 0.097 | 12 |
| MPPI-E-MDN | Pose 2 | 0.768 | 0.079 | 8 |
| MPPI-GP | Pose 2 | 0.729 | 0.072 | 9 |

Table 7.1 The cost is a weighted average of the position error (metres) and orientation error (radians).

regions with high uncertainty. The E-MDN was trained with 261 pushes collected from simulation. The parameters of the model were $M = 10$, in which each MDN member of the ensemble had a single hidden layer with 25 units, and the number of mixtures was set to $K = 1$. The MPPI parameters were set to $\mathbf{Q} = diag(0.5, 0.5, 0.5)$, $h = 2.0$, $\rho = 2.0$, $T = 2$, $\Delta t = 0.05$, $N = 10$, $L = 0$ (exploration decay not utilised) and the uncertainty penalty was set to $\gamma = 115$ for 150 pushes, and then $\gamma = 0$ afterwards. The results for this first experiment are shown by Figures 7.5 (a) and (b), in which two distinct trajectories are obtained as a result of either penalising or not penalising for model uncertainty.

### 7.3.4 Experiment 4:

Finally, this experiment makes use of Alg. 15, which performs optimisation to find a low uncertainty cost trajectory first. This low uncertainty cost trajectory is then followed by Alg. 14 push controller, but this time, we do not need to penalise uncertainty in its running cost, since the trajectory has already been optimised for that. The results for this experiment are shown in Fig 7.6.

## 7.4 Results

The results obtained in experiment 1 first depicted in Fig. 7.3 showed that both E-MDN (Fig. 7.3(a)) and GP (Fig. 7.3(a)(b)) are able to model similar predictive distributions of

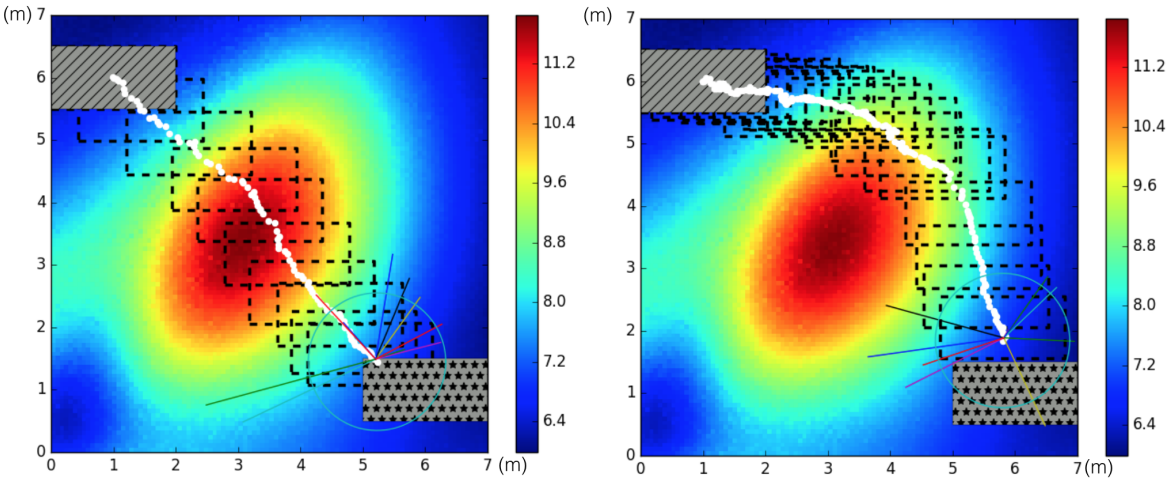


Fig. 7.5 Uncertainty averse pushing (Alg. 14). Start and goal locations are respectively depicted by the top and bottom grey rectangles. The black dashed boxes show sub-sampled box locations along the trajectory, which is shown in white. In simulation the box is pushed at its center of rotation (CoR), thus it will not rotate. (a) With no uncertainty penalty, the box tends to follow a straight line towards the goal. (b) When penalizing uncertainty, the box avoids it. The heuristic cost was defined such that uncertainty acts as a penalty for 150 pushes, and becomes zero afterwards (i.e. $\gamma = 0$). In this experiment we set $\gamma = 115$, $\mathbf{Q} = diag(0.5, 0.5, 0.5)$, $h = 2.0$, $\rho = 2.0$, $T = 2$, $\Delta t = 0.05$, $N = 10, L = 0$. The multi-coloured lines are forward sampled trajectories from the current box state. Finally, the circle radius depicts the current uncertainty in dynamics.

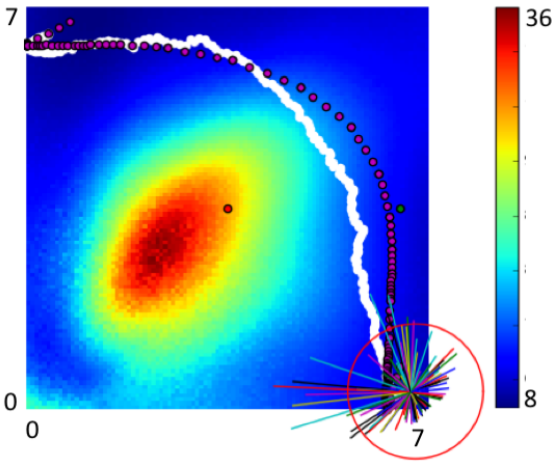


Fig. 7.6 Uncertainty averse pushing (Alg. 15). The red dots are the way-points to be followed and are generated by Alg. 15. Alg. 14 then attempts to follow these, producing the actual trajectory (white line). Forward sampled trajectories from the current box state are shown as multi-coloured lines, and the circle radius represents current region uncertainty.

motion displacements when applying pushes from different sides of the object. Experiment 1 also demonstrated that the E-MDN forward model learnt in simulation with missing data

is able to successfully capture regions with high predictive uncertainty due to lack of data (meta-uncertainty).

In experiment 2, demonstrated the ability of the proposed approach to push a test object to a desired desired final configurations (Fig. 7.2). The learnt E-MDN, GP and Box2D physics simulator forward model were compared in terms of final cost performance and number of pushes when pushing an object from an initial to a final location as depicted. All methods consistently reduced the cost, taking the object as close as possible to the desired configuration. The MPPI push controller in combination with E-MDN forward model was able to reach the goal with less pushes than the other models, as well as with a lower cost. In this sense, the GP showed inferior performance than the E-MDN, this may be due to the fact that E-MDN heteroscedastic modelling capabilities, allowing it to be more robust. Similar effects were seen in the toy examples discussed Chapter 3.3.2, in which the standard GP showed worst RMSE error when fitting a heteroscedastic data. Pushing data seem to possess the same effects, hence, poorer performance of GP-based MPPI. Curiously, these results are also supported by [Bauzá and Rodriguez, 2017], who demonstrated that a non-standard Variational Heteroscedastic Gaussian Process (VHGP) yields better accuracy as a push prediction model than a GP. However, we argue that a disavantage of a VHGP model is that it does not scale well with large amounts of data, whereas an E-MDN does, and is equally powerful for modelling more complicated aleatoric effects of the data such as heteroscedasticity, as well as meta-uncertainty. Furthermore, we proposed not only a model, but also a complete approach for pushing.

In experiment 3, we showed that the MPPI controller is also able to successfully push an object from an initial configuration to a final configuration while avoiding regions with high predictive uncertainty. This capability may be useful for various reasons. For instance, once entering regions with high predictive uncertainty the object may enter unrecoverable states, and have an undefined behaviour since the knowledge of its dynamics in terms of the

learnt forward model may be no longer valid. Thus, the ability to avoid uncertainty while manipulating objects is an important feature to achieve robust manipulation. We showed that the MPPI controller is a suitable choice of control strategy that is able to exploit predictive model in an unique fashion by treating high levels of uncertainty as imaginary obstacles. A tradeoff was also demonstrated: without uncertainty penalty the point mass particle tends to follow a straight line towards target, whereas increasing the uncertainty penalty makes the agent choose pushes that deflect the point mass particle from regions of high predictive uncertainty.

In the final experiment 4, demonstrated the application of the alternative approach described by Alg. 15. The alternative trajectory optimisation method decouples the problem by iteratively improving a given nominal trajectory given a pre-computed uncertainty heat-map generated using the learnt forward model. We found that this approach achieves good results, while requiring less tuning of the cost function in Alg. 14, since the trajectory is already pre-optimised by Alg. 15.

## 7.5 Conclusion

A push planning approach that uses a learnt forward model is presented. The proposed pushing approach is also capable of taking into account the reliability of the learnt model in terms of predictive uncertainty. Initially we showed how a learnt forward model can be used by a real robot to push the object to a target location using the MPPI approach. Later, we showed the modification to this basic MPPI to accommodate predictive uncertainty in two ways. In the first way the predictive uncertainty is directly inserted into the MPPI cost function so as to optimise a next control command in a model predictive fashion. This approach achieved robust pushing avoid regions of high predictive uncertainty in simulation. In the second formulation, a trajectory that is uncertainty averse is pre-computed using a path-integral update (Alg. 15) and only afterwards the MPPI (Alg. 14) is used to follow it.

We have shown that both algorithms exhibit the desired uncertainty-averse behaviour subject to tuning. In addition we have created the data gathering framework on a Baxter robot, and shown that E-MDNs and GPs produce very similar estimates of model uncertainty for real data. In the real-robot experiments E-MDNs were able to achieve goal targets with less pushes, we conjecture E-MDNs were better suitable as forward models since they are able to capture more complex statistical properties of the data, such as heteroscedasticity. This seem to be aligned with results shown in related work, such as in [Bauzá and Rodriguez, 2017]. Experiments showed that Alg. 14 works on the real robot, and that either one or both learning methods outpeformed a physics simulator, when used as the forward model for push planning. As future work we aim to investigate the application of such uncertainty-calibrated forward models and MPPI strategy so as to guide the execution of other types of manipulation actions, such as grasping. For instance, the generative models learnt in Chapter 5 could arguably be modelled using an E-MDN based contact model, but this time, the learnt model would not only predict and generate finger link locations for grasping, but also associated uncertainty estimates from the robot acquired grasping experience. Exploiting uncertainty could also allow the development of novel incrementally learnt generative grasp synthesis approaches using self-training via active learning, thus harnessing predictive uncertainty for improving models for grasp synthesis.

# Chapter 8

# Discussion

In this thesis, we addressed the problem of devising algorithms and strategies for robustifying object manipulation skills acquired from demonstration and derived from learnt physical models in non-prehensile tasks such as pushing. In Section 8.1, this chapter summarises the main contributions of the thesis. A summary of this thesis is given by Section 8.2. Finally, ongoing and future work are discussed in Section 8.3.

## 8.1 Conclusions

In this thesis we have demonstrated that

- Parametric models for generative grasp synthesis offer significant time performance gains over non-parametric methods.
- The choice of perceptual strategy does influence underlying manipulation task performance. As such they active perceptual strategies can be devised so as to improve manipulation in terms of execution safety and efficiency in information acquisition.
- Generative grasp synthesis can be used as a gateway to active perception approaches. They are useful in providing cues, and allowing the definition of task-oriented and

information theoretic utility functions to aid a given grasping task. Ultimately active perception can be used to reduce the number of views required for grasp synthesis and to maximise safety in grasp execution.

- Not only reasoning about future perceptual utility gains is useful, but also taking into account uncertainty in the execution of manipulation actions is important. Exploiting predictive uncertainty from learnt forward models provides a fundamental framework for robust manipulation.

- Robot manipulation skills are the result of a tight collaboration between predicting future perceptual outcomes, and predicting future manipulation action outcomes. The mechanisms that allow these capabilities to be implemented in real robot platforms are based on generative and uncertainty-calibrated predictive models, as well as strategies for active perception.

In short, we have made the following contributions in this thesis:

- A framework for generative grasp synthesis with applications for real-time grasp synthesis suitable for multi-fingered robot hands.
- A sensorisation method for under-actuated hands, such as the Pisa/IIT SoftHand.
- An active vision approach for view selection that makes use of generative grasp synthesis methods to perform perceptual predictions in order to leverage grasp performance, taking into account grasp execution safety and contact information.
- An approach to model and exploit predictive uncertainty from learnt physics applied to push manipulation.

Throughout the development of this thesis, we therefore investigated a manipulator system as a whole. We have seen that prediction of perceptual and manipulation action outcomes

form the basis for the development of robust manipulation skills. Perceptual predictions relied upon generative models for grasp synthesis, and non-prehensile skills made use of uncertainty-calibrated learnt physical models. We depict in Fig. 8.1 the different components of a robot manipulator that have been studied and which this thesis has contributed to. In Chapter 5 we developed an understanding and improvements to generative grasp synthesis using parametric models. In turn, in Chapter 6 this family of generative grasp synthesis models were shown to be suitable so as to allow active perception to take place via reasoning over perceptual predictions. Finally, in in Chapter 7, we go beyond grasping and reasoning about predictive uncertainty in action outcomes themselves and devise an approach to exploit learnt predictive dynamic models for pushing.

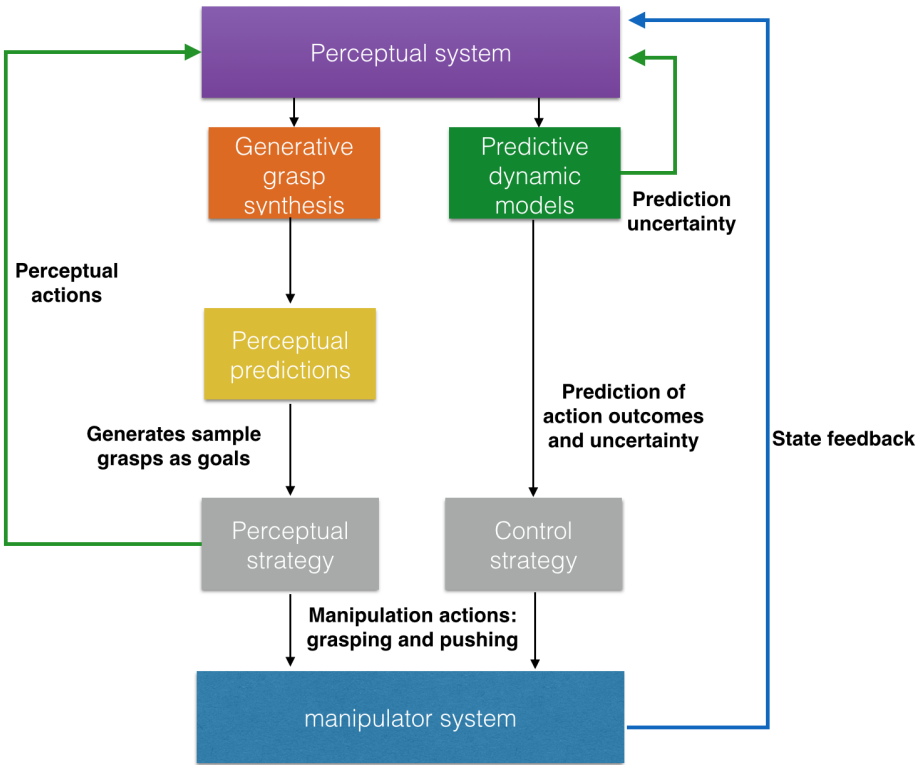


Fig. 8.1 Modules in a robotic manipulation pipeline to which this thesis have made contributions to. Generative grasp synthesis approaches (orange) (Chapter 5) form the basis for perceptual predictions (yellow) which can be used for devising active perceptual strategies (grey) based on active vision (Chapter 6). Finally, uncertainty-calibrated learnt predictive models (green) allowed us to devise strategies to exploit the knowledge of uncertainty in manipulation action outcomes and devise uncertainty-averse pushing approaches (Chapter 7).

## 8.2 Summary

**Chapter 1**

is the introduction of this thesis. It provided motivation and an overview of the main contributions of this thesis. In addition, a description of the working scenarios for grasping and pushing were given, describing the hardware utilised in the later chapters of this thesis.

**Chapter 2**

a literature review chapter that introduced the state-of-the-art and fundamentals of robotic manipulation, including prehensile and non-prehensile manipulation skills instantiated in this thesis as grasping and pushing. The problem of grasp synthesis has been introduced, as well as the two main strands for analytic and data-driven grasp synthesis. In particular, it presented a distinction between discriminative and generative grasp synthesis, and identified key features of generative approaches that were adopted and developed further in later chapters of this thesis. It also introduced the problem of pushing, the difficulties faced by analytic approaches, and recent work on data-driven methods for learning forward models for push manipulation. The problem of modelling uncertainty was also discussed. Furthermore, it also introduced the fundamentals of active perception for manipulation which enabled us to formulate perceptual strategies to aid manipulation tasks.

**Chapter 3**

introduced the basic techniques for approximate probability densities, as well as discriminative approaches utilised as predictive models in later chapters. The predictive models introduced captured uncertainty in their prediction.

**Chapter 4**

a background chapter that introduced the basic notions underlying the design, control and motion synthesis for robot manipulators.

**Chapter 5**

describes the approach and underlying algorithms for learning parametric multimodal distributions as generative models for grasp synthesis. The introduced parametric generative grasp synthesis approach. We showed that the proposed approach presented benefits in terms of computational time. We also introduced a novel online grasp synthesis algorithm, which offered benefits when dealing with dynamically changing point clouds, such as moving objects. A validation of this approach was given on a real Boris robot platform. A sensorisation method for under-actuated hands such as the Pisa/IIT SoftHand was also described, allow us to apply the discussed methods of grasp synthesis to this type of robotic hand. Furthermore, the discussed approaches were also made openly available as framework for generative grasp synthesis.

**Chapter 6**

we showed that generative approaches for grasp synthesis, such as the one described in previous chapter, can be used to define active perceptual strategies to better assist in grasp tasks. We described an approach to tackle the problem of improving robot grasp performance using active vision. An approach consisting of two parts was proposed. The first is a task oriented, contacted-based view selection that allows the robot to explore good quality grasp contact points. The second part another that permits the robot predict geometric information gain given grasp hypothesis and probabilistic representation of the workspace, thus allowing the robotic agent to investigate its workspace to make sure candidate grasp trajectories will not lead to collisions with unseen parts of the object to be grasped. It also allowed the robot to seek for more geometric information when not enough contacts are available for grasp synthesis. We

showed that the proposed approach yields better grasp success rate when compared to a random view selection strategy, while using fewer camera views for grasp generation.

**Chapter 7**

goes beyond predicting perceptual utility gains, and reason about uncertainty in manipulation action execution. We described an approach for learning uncertainty-calibrated forward models for non-prehensile manipulation instantiated in the context of pushing. These models were useful when combined with stochastic optimal control techniques. In particular, we showed that a model-predictive control approach can be readily used as a control method to achieve robust pushing.

## 8.3 Future work

**Improvements and further evaluation of grasp synthesis framework** we would like to improve and perform further performance evaluations on the approaches implemented as part of the generative grasp synthesis framework developed in Chapter 5. We conjecture that a future contribution involving a comprehensive comparison between various grasp synthesis algorithms would provide insightful results. We envisage a framework as such could be seen as a data-driven version of another related, albeit analytic based framework, such as the GraspIt framework [Miller and Allen, 2000].

**Active learning and uncertainty-calibrated contact models** the generative models learnt in Chapter 5 could arguably benefit from the developments of Chapter 7, such that an E-MDN based contact model could also be considered. Hence, the final learnt model would not only generate finger link locations for grasping, but also predict associated uncertainty estimates from the robot acquired grasping experience. Exploiting uncertainty could also allow the development of novel incrementally learnt generative grasp

synthesis approaches using active learning, thus harnessing predictive uncertainty for improving models for grasp synthesis.

**Predicting object motion behaviour during grasp execution** further future work steaming from Chapter 7 could use learnt forward models for predicting object motion, and thus harness this information so as to execute grasps that yield minimum uncertainty in object motion prediction. In turn, given a predicted object motion, one could be able to reason about reactive motor actions during grasp execution to counter-act unstable behaviour (assuming objects should remain stable and caged while grasping, so as to prevent slippage).

**Real-time active perception and grasping** another interesting avenue would be to explore the benefits of real-time perceptual strategies by making use of the real-time grasp synthesis approach variation described in Chapter 5. This would allow us to combine the notions developed in Chapter 6 as a continuous perceptual problem, in which a continuous motion of the depth sensor can be optimised, as opposed to discrete view selection choices.

**Conditional hand configuration models** using dynamic motion primitives as described in 4, or alternatively, probabilistic models for motion synthesis as surveyed by [Calinon, 2016], one could utilise such task parametrised motion models to redefine a hand configuration model in Eq. 5.14. These models are also based on parametric mixtures and would therefore define a complete fully parametric grasp synthesis method in combination with motion synthesis capabilities that may take task requirements into account, or be conditioned on task-related features.

# References


Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283.

Agrawal, P., Nair, A., Abbeel, P., Malik, J., and Levine, S. (2016). Learning to poke by poking: Experiential learning of intuitive physics. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 5092–5100, USA. Curran Associates Inc.

Aloimonos, J., Weiss, I., and Bandyopadhyay, A. (1988). Active vision. *International Journal of Computer Vision*, 1(4):333–356.

Amato, N. M. and Wu, Y. (1996). A randomized roadmap method for path and manipulation planning. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 1, pages 113–120 vol.1.

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

Atanasov, N., Sankaran, B., Le Ny, J., Pappas, G., and Daniilidis, K. (2014). Nonmyopic View Planning for Active Object Classification and Pose Estimation. *IEEE Trans. on Robotics*, 30(5):1078–1090.

Atance, C. M. and O'Neill, D. K. (2001). Episodic future thinking. *Trends in Cognitive Sciences*, 5:533–539.

Azzalini, A. and Bowman, A. W. (1990). A look at some data on the old faithful geyser. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 39(3):357–365.

Bajcsy, R. (1988). Active perception. In *Proc IEEE, 76:996–1005*.

Balasubramanian, R., Xu, L., Brook, P. D., Smith, J. R., and Matsuoka, Y. (2012). Physical human interactive guidance: Identifying grasping principles from human-planned grasps. *IEEE Transactions on Robotics*, 28(4):899–910.

Ballard, D. H. (1991). Animate vision. *Artif. Intell.*, 48(1):57–86.

Bauzá, M. and Rodriguez, A. M. (2017). A probabilistic data-driven model for planar pushing. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3008–3015.

Behrens, M. (2013). Robotic manipulation by pushing at a single point with constant velocity : modeling and techniques.

Bekiroglu, Y., Huebner, K., and Kragic, D. (2011). Integrating grasp planning with online stability assessment using tactile sensing. In *2011 IEEE International Conference on Robotics and Automation*, pages 4750–4755.

Belter, D., Kopicki, M., Zurek, S., and Wyatt, J. (2014). Kinematically optimised predictions of object motion. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4422–4427.

Ben Amor, H., Kroemer, O., Hillenbrand, U., Neumann, G., and Peters, J. (2012). Generalization of human grasping for multi-fingered robot hands. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Bicchi, A. and Kumar, V. (2000). Robotic grasping and contact: a review. In *IEEE International Conference on Robotics and Automation*.

Bishop, C. M. (1994). Mixture density networks. Technical report.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Bishop, C. M. and Nasrabadi, N. M. (2007). Pattern recognition and machine learning. *J. Electronic Imaging*, 16:049901.

Bohg, J., Johnson-Roberson, M., León, B., Felip, J., Gratal, X., Bergstrom, N., Kragic, D., and Morales, A. (2011). Mind the gap-robotic grasping under incomplete observation. In *IEEE International Conference on Robotics and Automation*, pages 686–693. IEEE.

Bohg, J., Morales, A., Asfour, T., and Kragic, D. (2014). Data-driven grasp synthesis x2014;a survey. *IEEE Transactions on Robotics*, 30(2):289–309.

Borst, C., Fischer, M., and Hirzinger, G. (2004). Grasp planning: how to choose a suitable task wrench space. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, volume 1, pages 319–325 Vol.1.

Boyce, W. E., DiPrima, R. C., and Meade, D. B. (2017). *Elementary Differential Equations and Boundary Value Problems, Loose-Leaf Print Companion*. John Wiley & Sons.

Brochu, E., Cora, V. M., and de Freitas, N. (2010). A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *CoRR*, abs/1012.2599.

Calinon, S. (2016). A tutorial on task-parameterized movement learning and retrieval. *Intelligent Service Robotics*, 9(1):1–29.

Catalano, M., Grioli, G., Farnioli, E., Serio, A., Piazza, C., and Bicchi, A. (2014). Adaptive synergies for the design and control of the pisa/iit softhand. *The International Journal of Robotics Research*, 33(5):768–782.

Catto, E. (2011). Box2d: A 2d physics engine for games.

Chen, S., Li, Y., and Kwok, N. M. (2011). Active vision in robotic systems: A survey of recent developments. *I. J. Robotic Res.*, 30:1343–1377.

Cosgun, A., Hermans, T., Emeli, V., and Stilman, M. (2011). Push planning for object placement on cluttered table surfaces. In *Proc. of the IEEE, IROS- 2011*.

Coumans, E. and Bai, Y. (2019). Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org.

Curtis, N. and Xiao, J. (2008). Efficient and effective grasping of novel objects through learning and adapting a knowledge base. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2252–2257. IEEE.

Da Silva, B., Konidaris, G., and Barto, A. (2012). Learning parameterized skills. *arXiv preprint arXiv:1206.6398*.

Deisenroth, M. and Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *ICML-2011*.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38.

Detry, R. (2010). *Learning of multi-dimensional, multi-modal features for robotic grasping*. PhD thesis, PhD thesis, University of Liege, 2010. Supervisor: Justus Piater.

Detry, R., Kraft, D., Kroemer, O., Bodenhagen, L., Peters, J., Krüger, N., and Piater, J. (2011). Learning grasp affordance densities. *Paladyn. Journal of Behavioral Robotics*, 2(1):1–17.

Detry, R. and Piater, J. (2013). Unsupervised learning of predictive parts for cross-object grasp transfer. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Diederik, P. K. and Jimmy, B. (2014). Adam: A method for stochastic optimization. *CoRR-2014*.

Dogar, M. and Srinivasa, S. (2011). A framework for push-grasping in clutter. *Robotics: Science and systems VII*.

Ferrari, C. and Canny, J. (1992). Planning optimal grasps. In *International Conference on Robotics and Automation*, pages 2290–2295.

Findlay, J. M. and Gilchrist, I. D. (2003). Active vision: The psychology of looking and seeing.

Finn, C., Goodfellow, I. J., and Levine, S. (2016). Unsupervised learning for physical interaction through video prediction. *CoRR-2016*.

Fisher, R. A. (1953). Dispersion on a sphere. In *Proc. Roy. Soc. London Ser. A.*, volume 217, pages 295–305. Royal Society.

Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *ICML-2016*.

Gilmore, R. (2012). *Lie groups, Lie algebras, and some of their applications*. Courier Corporation.

Goins, A. K., Carpenter, R., Wong, W. K., and Balasubramanian, R. (2014). Evaluating the efficacy of grasp metrics for utilization in a gaussian process-based grasp predictor. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3353–3360.

Goyal, S., Ruina, A., and Papadopoulos, J. (1989). Limit surface and moment function descriptions of planar sliding. *Proceedings, 1989 International Conference on Robotics and Automation*, pages 794–799 vol.2.

Graves, A. (2013). Generating sequences with recurrent neural networks. *CoRR-2013*.

Gualtieri, M., ten Pas, A., Saenko, K., and Platt, R. (2016). High precision grasp pose detection in dense clutter. *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 598–605.

Hajek, B. (2015). *Random processes for engineers*. Cambridge university press.

Hartenberg, R. and Danavit, J. (1964). *Kinematic synthesis of linkages*. New York: McGraw-Hill.

Hartenberg, R. S. and Denavit, J. (1955). A kinematic notation for lower pair mechanisms based on matrices. *Journal of applied mechanics*, 77(2):215–221.

Hinton, G. E. (1999). Products of experts. In *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, volume 1, pages 1–6 vol.1.

Hjelm, M., Detry, R., Ek, C. H., and Kragic, D. (2014). Representations for cross-task, cross-object grasp transfer. In *IEEE International Conference on Robotics and Automation*.

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W. (2013). Octomap: an efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206.

Howe, R. D. and Cutkosky, M. R. (1996). Practical force-motion models for sliding manipulation. *The International Journal of Robotics Research*, 15(6):557–572.

Jia, Y.-B. and Erdmann, M. (1999). Pose and motion from contact. *The International Journal of Robotics Research*, 18(5):466–487.

Jiang, Y., Moseson, S., and Saxena, A. (2011). Efficient grasping from rgbd images: Learning using a new rectangle representation. *2011 IEEE International Conference on Robotics and Automation*, pages 3304–3311.

Johns, E., Leutenegger, S., and Davison, A. J. (2016). Deep learning a grasp function for grasping under gripper pose uncertainty. *CoRR*, abs/1608.02239.

Kalakrishnan, M., Chitta, S., Theodorou, E., Pastor, P., and Schaal, S. (2011). Stomp: Stochastic trajectory optimization for motion planning. In *Proc. of the IEEE, ICRA-2011*.

Kappen, H. J. (2005). Path integrals and symmetry breaking for optimal control theory. *Journal of statistical mechanics: theory and experiment*.

Kappen, H. J. (2011). Optimal control theory and the linear bellman equation.

Kavraki, L. and Latombe, J. . (1994). Randomized preprocessing of configuration space for path planning: articulated robots. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)*, volume 3, pages 1764–1771 vol.3.

Kendall, A. and Gal, Y. (2017). What uncertainties do we need in bayesian deep learning for computer vision? In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 5574–5584. Curran Associates, Inc.

Kim, J., Iwamoto, K., Kuffner, J. J., Ota, Y., and Pollard, N. S. (2013). Physically based grasp quality evaluation under pose uncertainty. *IEEE Transactions on Robotics*, 29(6):1424–1439.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kirk, D. E. (2012). *Optimal control theory: an introduction*. Courier Corporation.

Kirkpatrick, D., Mishra, B., and Yap, C.-K. (1992). Quantitative steinitz's theorems with applications to multifingered grasping. *Discrete Comput. Geom.*, 7(3):295–318.

Kiureghian, A. D. and Ditlevsen, O. (2009). Aleatory or epistemic? does it matter? *Structural Safety*, 31(2):105 – 112. Risk Acceptance and Risk Communication.

Kopicki, M., Detry, R., Adjigble, M., Stolkin, R., Leonardis, A., and Wyatt, J. L. (2015). One-shot learning and generation of dexterous grasps for novel objects. *The International Journal of Robotics Research*. first published on September 18, 2015.

Kopicki, M., Detry, R., Schmidt, F., Borst, C., Stolkin, R., and Wyatt, J. L. (2014). Learning dextrous grasps that generalise to novel objects by combining hand and contact models. In *IEEE International Conference on Robotics and Automation*, pages 5358–5365. IEEE.

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

Kopicki, M., Zurek, S., Stolkin, R., Mörwald, T., and Wyatt, J. (2011). Learning to predict how rigid objects behave under simple manipulation. In *2011 IEEE International Conference on Robotics and Automation*, pages 5722–5729.

Koubaa, A. (2016). *Robot Operating System (ROS): The Complete Reference (Volume 1)*. Springer Publishing Company, Incorporated, 1st edition.

Krainin, M., Curless, B., and Fox, D. (2011). Autonomous generation of complete 3d object models using next best view manipulation planning. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5031–5037. IEEE.

Kumra, S. and Kanan, C. (2017). Robotic grasp detection using deep convolutional neural networks. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 769–776.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pages 6402–6413.

Lavalle, S. M., Kuffner, J. J., and Jr. (2000). Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, pages 293–308.

Le, Q. V., Kamm, D., Kara, A. F., and Ng, A. Y. (2010). Learning to grasp objects with multiple contact points. In *2010 IEEE International Conference on Robotics and Automation*, pages 5062–5069.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

Lee., J. M. (2002). *Introduction to Smooth Manifolds*. Springer.

Lee, S. H. and Cutkosky, M. R. (1991). Fixture planning with friction. *Journal of Engineering for Industry*, 113(3):320–327.

Lenz, I., Lee, H., and Saxena, A. (2013). Deep learning for detecting robotic grasps. *CoRR*, abs/1301.3592.

Levine, S., Pastor, P., Krizhevsky, A., and Quillen, D. (2016). Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection. *arXiv*.

Liu, H., Wu, K., Meusel, P., Seitz, N., Hirzinger, G., Jin, M. H., Liu, Y. W., Fan, S. W., Lan, T., and Chen, Z. P. (2008). Multisensory five-finger dexterous hand: The dlr/hit hand ii. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3692–3697.

Liu, Y.-H. (2000). Computing n-finger form-closure grasps on polygonal objects. *The International Journal of Robotics Research*, 19(2):149–158.

Lynch, K. M., Maekawa, H., and Tanie, K. (1992). Manipulation and active sensing by pushing using tactile feedback. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1:416–421.

Lynch, K. M. and Park, F. C. (2017). *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, New York, NY, USA, 1st edition.

Mahler, J., Liang, J., Niyaz, S., Laskey, M., Doan, R., Liu, X., Ojea, J. A., and Goldberg, K. (2017). Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *CoRR*, abs/1703.09312.

Mahler, J., Pokorny, F. T., Hou, B., Roderick, M., Laskey, M., Aubry, M., Kohlhoff, K., Kröger, T., Kuffner, J. J., and Goldberg, K. Y. (2016). Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards. *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1957–1964.

Mason, M. T. (1982). *Manipulator grasping and pushing operations*. PhD thesis, MIT.

Miall, R. C. and Wolpert, D. M. (1996). Forward models for physiological motor control. *Neural networks : the official journal of the International Neural Network Society*, 9 8:1265–1279.

Miller, A. T. and Allen, P. K. (2000). Graspit!: A versatile simulator for grasp analysis. In *in Proc. of the ASME Dynamic Systems and Control Division*. Citeseer.

Mishra, B. (1995). Grasp metrics: Optimality and complexity. Technical report, New York, NY, USA.

Montesano, L., Lopes, M., Bernardino, A., and Santos-Victor, J. (2008). Learning object affordances: From sensory–motor coordination to imitation. *Trans. Rob.*, 24(1):15–26.

Moravec, H. and Elfes, A. (1985). High resolution maps from wide angle sonar. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, volume 2, pages 116–121.

Morrison, D., Corke, P., and Leitner, J. (2018). Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach. *CoRR*, abs/1804.05172.

Mostafa, F. (2011). Applications of neural networks in market risk.

Murray, R. M. (2017). *A mathematical introduction to robotic manipulation*. CRC press.

Nunez-Varela, J., Ravindran, B., and Wyatt, J. L. (2012). Where do i look now? gaze allocation during visually guided manipulation. In *IEEE ICRA*. IEEE.

Ortega, R. and Spong, M. W. (1988). Adaptive motion control of rigid robots: a tutorial. In *Proceedings of the 27th IEEE Conference on Decision and Control*, pages 1575–1584 vol.2.

Overmars, M. H. (1992). A random approach to motion planning. Technical report.

Peshkin, M. A. and Sanderson, A. C. (1988). The motion of a pushed, sliding workpiece. *IEEE Journal on Robotics and Automation*, 4(6):569–598.

Pinto, L. and Gupta, A. (2016). Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3406–3413.

Pinto, L. and Gupta, A. (2017). Learning to push by grasping: Using multiple tasks for effective learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2161–2168.

Pollard, N. S. (1996). Synthesizing grasps from generalized prototypes. In *Intl. Conference on Robotics and Automation*, pages 2124–2130.

Popov, V. L. (2010). *Coulomb's Law of Friction*, pages 133–154. Springer Berlin Heidelberg, Berlin, Heidelberg.

Rasmussen, E, C. (2006). *Gaussian processes for machine learning*. Citeseer.

Redmon, J. and Angelova, A. (2015). Real-time grasp detection using convolutional neural networks. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1316–1322.

Reif, J. H. (1979). Complexity of the mover's problem and generalizations. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 421–427.

Rietzler, A., Detry, R., Kopicki, M., Wyatt, J. L., and Piater, J. (2013). Inertially-safe grasping of novel objects. In *Cognitive Robotics Systems: Replicating Human Actions and Activities (Workshop at IROS 2013)*.

Roa, M. A. and Suárez, R. (2015). Grasp quality measures: Review and performance. *Auton. Robots*, 38(1):65–88.

Saggio, G., Riillo, F., Sbernini, L., and Quitadamo, L. R. (2015). Resistive flex sensors: a survey. *Smart Materials and Structures*, 25(1):013001.

Sahbani, A., El-Khoury, S., and Bidaud, P. (2012). An overview of 3d object grasp synthesis algorithms. *Robot. Auton. Syst.*, 60(3):326–336.

Santaera, G., Luberto, E., Serio, A., Gabiccini, M., and Bicchi, A. (2015). Low-cost, fast and accurate reconstruction of robotic and human postures via imu measurements. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2728–2735.

Saxena, A., Driemeyer, J., and Ng, A. Y. (2008a). Robotic Grasping of Novel Objects using Vision. *International Journal of Robotics Research*, 27(2):157.

Saxena, A., Wong, L., and Ng, A. (2008b). Learning grasp strategies with partial shape information. In *Proceedings of AAAI*, pages 1491–1494. AAAI.

Schaal, S. (2006). *Dynamic Movement Primitives -A Framework for Motor Control in Humans and Humanoid Robotics*, pages 261–280. Springer Tokyo, Tokyo.

Scott, D. W. and Sain, S. R. (2004). Multi-dimensional density estimation.

Shimoga, K. B. (1996). Robot grasp synthesis algorithms: A survey. *The International Journal of Robotics Research*, 15(3):230.

Siciliano, B. and Khatib, O., editors (2008). *Springer Handbook of Robotics*. Springer.

Siciliano, B., Sciavicco, L., Villani, L., and Oriolo, G. (2010). *Robotics: modelling, planning and control*. Springer Science & Business Media.

Sommer, N., Li, M., and Billard, A. (2014). Bimanual compliant tactile exploration for grasping unknown objects. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6400–6407. IEEE.

Song, D., Ek, C. H., Huebner, K., and Kragic, D. (2011). Multivariate discretization for bayesian network structure learning in robot grasping. In *2011 IEEE International Conference on Robotics and Automation*, pages 1944–1950.

Stark, M., Lies, P., Zillich, M., Wyatt, J., and Schiele, B. (2008). Functional object class detection based on learned affordance cues. In *Computer Vision Systems*, pages 435–444. Springer.

Stengel, R. F. (2012). *Optimal control and estimation*. Courier Corporation.

Sudderth, E. B. (2006). *Graphical models for visual object recognition and tracking*. PhD thesis, MIT, Cambridge, MA.

ten Pas, A. and Platt, R. (2015). Using geometry to detect grasp poses in 3d point clouds. In *ISRR*.

Theodorou, E., Buchli, J., and Schaal, S. (2010). A generalized path integral control approach to reinforcement learning. *JMLR-2010*.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press.

Todorov, E. (2005). Stochastic optimal control and estimation methods adapted to the noise characteristics of the sensorimotor system. *Neural computation*.

Ude, A., Nemec, B., Petrić, T., and Morimoto, J. (2014). Orientation in cartesian space dynamic movement primitives. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2997–3004.

Williams, G., Aldrich, A., and Theodorou, E. (2015). Model predictive path integral control using covariance variable importance sampling. *arXiv preprint arXiv:1509.01149*.

Williams, N. W., Penrose, J. M. T., Caddy, C. M., Barnes, E., Hose, D. R., and Harley, P. (2000). A goniometric glove for clinical hand assessment: Construction, calibration and validation. *Journal of Hand Surgery*, 25(2):200–207.

Wolpert, D. M. and Flanagan, J. R. (2001). Motor prediction. *Current Biology*, 11:R729–R732.

Yi, W. and Ballard, D. (2009). Recognizing behaviour in hand-eye coordination patterns. *International Journal of Humanoid Robotics*, 06(03):337–359.

Yu, K.-T., Bauzá, M., Fazeli, N., and Rodriguez, A. (2016). More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing. *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 30–37.

Zhou, J., Bagnell, J. A., and Mason, M. T. (2017). A fast stochastic contact model for planar pushing and grasping: Theory and experimental validation. In *RSS-2017*.

Zito, C., Kopicki, M. S., Stolkin, R., Borst, C., Schmidt, F., Roa, M., and Wyatt, J. L. (2013). Sequential trajectory re-planning with tactile information gain for dexterous grasping under object-pose uncertainty. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 4013–4020. IEEE.

# Appendix A

# Groups and rigid body transformations

## A.1 Groups

A group is a mathematical structure that is composed of a set of elements $G$ together with an operation $\boxdot$ that obeys three basic axioms (see [Gilmore, 2012; Lee., 2002] for more details):

**Closure**: If $a$ and $b$ are members of the group, then the application of $\boxdot$ yields $c = a \boxdot b$ that is also a member of the group.

**Identity**: The group $G$ has an identity element $I$ such that $I \boxdot a = a \boxdot I = a$, where $a, I \in G$.

**Inverse**: Each member $a$ of the group $G$ must have have a unique corresponding inverse $a^{-1}$ such that $a \boxdot a^{-1} = a^{-1} \boxdot a = I$.

### A.1.1 *GL(n)*

Consider for example the so called *general linear group* $GL(n)$. It consists of all non-singular square matrices with dimensions $n \times n$ together with the operation of matrix multiplication. It can be easily shown that this is indeed a group. First, multiplying two square matrices always yield another square matrix with the same dimensions, i.e. it is closed under multiplication. Second, the identity element exists and is denoted by a $n \times n$ matrix $\mathbf{I}$ whose main diagonal

elements are set to one, and all other elements are zero. Finally, by construction we have assumed that all matrices are non-singular, which means that these matrices always have unique inverse, thus satisfying the last group axiom.

Throughout this thesis we consider two groups of interest with respect to the multiplication operation. The *special orthogonal group* in three dimensions $SO(3)$ - which is a subgroup of $GL(n)$ - and the *special Euclidean group* $SE(3)$, a subgroup of $GL(n+1)$. The $SO(3)$ manifests itself as the group of rotations, whereas $SE(3)$ represents the group of rigid body transformations (a rotation followed by a translation).

### A.1.2 *SO(3)*

The special orthogonal group in three dimensions is defined as the set of all $3 \times 3$ matrices $\mathbf{R}$ with unit determinant, i.e. $det(\mathbf{R}) = 1$, that preserve the inner product. In other words,

$$SO(3) = \{\mathbf{R} \in GL(3) | \mathbf{R}^{\mathrm{T}}\mathbf{R} = I \wedge det(\mathbf{R}) = 1\}. \tag{A.1}$$

This group represents set of rotations in three dimensional space $\mathbb{R}^3$ (Euclidean space). A body is able to rotate around the x, y and z axis. These respective *intrinsic rotations* can be given by three different matrices:

A *roll* is given by a counterclockwise rotation of $\alpha$ around the x-axis and is given by:

$$\mathbf{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \tag{A.2}$$

A *pitch* is given by a counterclockwise rotation of $\beta$ around the y-axis, and is given by

$$\mathbf{R}_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \tag{A.3}$$

Finally, a *yaw* is given by a counterclockwise rotation around the z-axis and is given by the following matrix

$$\mathbf{R}_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.4}$$

### A.1.2.1 Euler angles

It follows that by combining the roll-pitch-yaw intrinsic rotations above we are able to use the minimum number of three parameters necessary to represent a full orientation of a body. Using Euler angles, a rotation can be represented as $\boldsymbol{\phi} = [\alpha, \beta, \gamma] \in \mathbb{R}^3$ such that:

$$\mathbf{R}(\boldsymbol{\phi}) = \mathbf{R}_z(\gamma)\mathbf{R}_y(\beta)\mathbf{R}_x(\alpha) \tag{A.5}$$

$$\mathbf{R}(\boldsymbol{\phi}) = \begin{bmatrix} \cos\beta\cos\gamma & -\cos\alpha\sin\gamma + \sin\alpha\sin\beta\cos\gamma & \sin\alpha\sin\gamma + \cos\alpha\sin\beta\cos\gamma \\ \cos\beta\sin\gamma & \cos\alpha\cos\gamma + \sin\alpha\sin\beta\sin\gamma & -\sin\alpha\cos\gamma + \cos\alpha\sin\beta\sin\gamma \\ -\sin\beta & \sin\alpha\cos\beta & \cos\alpha\cos\beta \end{bmatrix} \tag{A.6}$$

where, considering the elements of $\mathbf{R}$ as $r_{ij}$, and for $\beta$ in the interval $(-\pi/2, \pi/2)$, the inverse solution is given by [Siciliano et al., 2010]:

$$\begin{aligned} \gamma &= \arctan 2(r_{21}, r_{11}) \\ \beta &= \arctan 2(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}) \\ \alpha &= \arctan 2(r_{32}, r_{33}) \end{aligned} \tag{A.7}$$

And for $\beta \in (\pi/2, 3\pi/2)$:

$$\begin{aligned}\gamma &= \arctan 2(-r_{21}, -r_{11}) \\ \beta &= \arctan 2(-r_{31}, -\sqrt{r_{32}^2 + r_{33}^2}) \\ \alpha &= \arctan 2(-r_{32}, -r_{33})\end{aligned} \tag{A.8}$$

These solutions are however degenerate at $\cos(\beta) = 0$, in which case a singularity occurs such that it becomes only possible to determine the sum or different beteween $\gamma$ and $\alpha$.

Hence, although Euler angles provide a minimal representation for the orientation of a body, such singularity problems mean we are no longer able to recover $\boldsymbol{\phi}$ back from $R(\boldsymbol{\phi})$ for certain configurations.

#### A.1.2.2 Unit quaternions

An alternative representation for rotations is to use *unit quaternions*. Although it does not constitutes a minimal representation, since quaternions are defined in $q \in \mathbb{R}^4$, this choice of representation does not suffer from singularity problems, and also offers the benefit of the quaternion algebra when composing rotations without requiring to convert quaternions to rotation matrices and vice-versa. A unit quaternion can be seen as a rotation of $\theta$ around an unit axis $\boldsymbol{e} = [e_x, e_y, e_z] \in \mathbb{R}^3$. This can be written as a unit quaternion as follows:

$$\mathbf{q} = \exp\{\frac{\theta}{2}(e_x\mathbf{i} + e_y\mathbf{j} + e_z\mathbf{k})\} = \cos\frac{\theta}{2} + (e_x\mathbf{i} + e_y\mathbf{j} + e_z\mathbf{k})\sin\frac{\theta}{2} \tag{A.9}$$

where, we can conveniently write $\mathbf{q}$ in its vector form as $\mathbf{q} = [q_x, q_y, q_z, q_w]$. We may also distinguish between its real and vector part by denoting $\mathbf{q} = \{\mathbf{q_v}, q_w\}$, where $\mathbf{q_v} = [q_x, q_y, q_z]$. Note that expression in Eq. A.9 is an extension to the Euler's formula for four-dimensional complex space. One may also see $\boldsymbol{\omega} = \theta\boldsymbol{e}$ as an angular velocity vector. It represents a rotation of $\boldsymbol{\omega}$ angular units per unit of time in space. Thus, Eq. A.9 is often referred to as the *exponential map* of $\boldsymbol{\omega}$, which can be written more explicitly as

$$\mathbf{q} = \exp(\boldsymbol{\omega}) = \begin{cases} [\sin(\|\boldsymbol{\omega}\|)\boldsymbol{\omega}, \cos(\|\boldsymbol{\omega}\|)] & \boldsymbol{\omega} \neq \mathbf{0} \\ [0,0,0,1] & otherwise \end{cases} \tag{A.10}$$

where, $\theta = \|\boldsymbol{\omega}\|$ and $\boldsymbol{\omega} = \theta\mathbf{e} \in \mathbb{R}^3$.

Conversely, the *logarithmic map* of a unit quaternion $\mathbf{q}$ is given by

$$\boldsymbol{\omega} = \log(\mathbf{q}) = \begin{cases} \arccos(q_w)\frac{\mathbf{q}_v}{\|\mathbf{q}_v\|} & \mathbf{q}_v \neq \mathbf{0} \\ [0,0,0] & otherwise \end{cases} \tag{A.11}$$

Multiplying two quaternions $\mathbf{q}_a = [\mathbf{q}_v^a, q_w^a]$ and $\mathbf{q}_b = [\mathbf{q}_v^b, q_w^b]$ is given by

$$\mathbf{q}_a * \mathbf{q}_b = [q_w^a q_w^b - \mathbf{q}_v^a \cdot \mathbf{q}_v^b,\ q_w^a \mathbf{q}_v^b + q_w^b \mathbf{q}_v^a + \mathbf{q}_v^a \times \mathbf{q}_v^b] \tag{A.12}$$

Composing two rotations is simple. For instance, if $\mathbf{q}$ is a rotation of 30 degrees around an axis $\boldsymbol{e}$ (clockwise when we look at the same direction as $\boldsymbol{e}$), then the composite $\mathbf{q} * \mathbf{q}$ is a rotation of 60 degrees around the same axis, where $*$ denotes quaternion multiplication.

In addition, the inverse rotation $\mathbf{q}^{-1}$ is given by the conjugate quaternion $\bar{\mathbf{q}} = [-\mathbf{q}_v, \mathbf{q}]$, which is obtained by simply negating its vector part.

The rotation matrix corresponding to a unit quaternion $\mathbf{q} = [q_x, q_y, q_z, q_w]$ is given by

$$\mathbf{R}_\mathbf{q} = \begin{bmatrix} 1-2q_y^2-2q_z^2 & 2(q_x q_y - q_z q_w) & 2(q_x q_z + q_y q_w) \\ 2(q_x q_y + q_z q_w) & 1-2q_x^2-2q_z^2 & 2(q_y q_z - q_x q_w) \\ 2(q_x q_z - q_y q_w) & 2(q_y q_z + q_x q_w) & 1-2q_x^2-2q_y^2 \end{bmatrix}. \tag{A.13}$$

### A.1.3 *SE(3)*

We would like to not only represent orientation or rotation transformations, but also the position or translation of a rigid body in space. The $SE(3)$ group represents the set of all rigid body transformations. This group is the Cartesian product between the $SO(3)$ and $\mathbb{R}^3$. Note that, even though translations clearly form an Euclidean space since $\mathbf{t} \in \mathbb{R}^3$, the $SE(3)$ forms

a non-Euclidean manifold, because its rotational component, the $SO(3)$, is not an Euclidean space.

This group is often represented using homogeneous transformation matrices of the following form:

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \text{ where } T \text{ is a } 4 \times 4 \text{ matrix} \tag{A.14}$$

Thus, the $SE(3)$ can be classified as a subgroup of $GL(n+1)$.

## A.2 Representations of pose

A rigid body transformation can be used to encode the position and orientation of an object with respect to some reference frame. This rigid body transformation shall be referred to as the *pose* of the particular object in consideration. In the previous section we have seen that we might represent these members of $SE(3)$ as $4 \times 4$ homogeneous transformation matrices. In matrix form, we may denote the pose of an object $o$ with respect to a base reference frame $b$ by writing $\mathbf{T}_o^b$. This is equivalent to say that, if a point $\mathbf{p}_o = [x_o, y_o, z_o, 1]^T$ is given relative to the object $o$, then we are able to know the position of such point with respect the base frame $b$ by

$$\mathbf{p}_b = \mathbf{T}_o^b \mathbf{p}_o \tag{A.15}$$

However, the representation for $\mathbf{T}_o^b$ requires 16 floating-point values to be stored in a computer. It would be desirable to represent members of $SE(3)$ more concisely, but still have the equivalent power of the operations provided by matrix algebra. This can be done by using the aforementioned unit quaternion representation for the rotational part of a member of the $SE(3)$.

Therefore, we will represent a pose as a vector $\mathbf{x} \in \mathbb{R}^7$, where the first three coordinates correspond to the position $\mathbf{t}$ and the last four will correspond to the coordinates of a unit quaternion $\mathbf{q}$, such that

$$\mathbf{x} = [\mathbf{t}, \mathbf{q}] = [t_x, t_y, t_z, q_x, q_y, q_z, q_w], \tag{A.16}$$

with identity element

$$\mathbf{id} = [0, 0, 0, 0, 0, 0, 1]. \tag{A.17}$$

Finally, observe that we can describe Eq. A.16 in matrix form as

$$\mathbf{T} = \begin{bmatrix} \mathbf{R_q} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}. \tag{A.18}$$