Comparison of Optimal Control Techniques for Learning-based RRT

Deepak Paramkusam

Master of Science Thesis

Delft University of Technology

Delft Center for Systems and Control

Comparison of Optimal Control Techniques for Learning-based RRT

Master of Science Thesis

For the degree of Master of Science in Mechanical Engineering at Delft University of Technology

Deepak Paramkusam

February 16, 2018

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Copyright © Delft Center for Systems and Control (DCSC)
All rights reserved.

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Comparison of Optimal Control Techniques for Learning-based RRT

by

Deepak Paramkusam

in partial fulfillment of the requirements for the degree of

Master of Science Mechanical Engineering

Dated: February 16, 2018

Supervisor(s): ______________________
Prof.Dr.Ir. Martijn Wisse

____
Ir. Mukunda Bharatheesha

Reader(s): ______________________
Dr. Sergio Grammatico

____
Ir. Wouter J. Wolfslag

Abstract

Kinodynamic motion planning for a robot involves generating a trajectory from a given robot state to goal state while satisfying kinematic and dynamic constraints. Rapidly-exploring Random Trees (RRT) is a sampling-based algorithm that has been widely adopted for this [1]. However, RRT is not fast enough to enable its use in industrial applications. Recently, supervised learning has been used to pre-learn time consuming steps of RRT which resulted in improvement in planning times [2]. The supervised learning models require cost and control input of the system as training data which are generated using optimal control.

The training data can be obtained either by indirect optimal control or direct optimal control techniques. In this thesis, both the techniques are each used to generate cost and control inputs for a two-link manipulator using random initial-final state pairs. Then each dataset is used to train a model and the datasets are compared based on certain training metrics. K-nearest neighbours regression and multi-layer perceptron neural network are the supervised learning models used in this thesis. It is observed that both the datasets result in similar convergence of the models, but indirect optimal control approach allows upto 24-fold faster data generation and upto 3-fold reduction in dimensionality of training data compared to the direct optimal approach.

Real-world robots have torque limits based on actuator configuration. The torque limits are modelled as control constraints in both the optimal control techniques and the effect of this restriction on data generation and supervised learning is studied in this thesis. Direct optimal control is found to be better for data generation in this case due to the ease of applying control bounds as inequality constraints on the function approximations. Indirect optimal control is very tedious as active constraints should be known a priori to determine the switching points. An alternate method is explored instead where samples are generated similar to the unconstrained case but samples violating the constraints are removed. Poor control input learning is observed in both approaches and the models struggled to extrapolate. It is hypothesised that this is due to inability of the constrained data to fully capture the system dynamics. However, good cost prediction is achieved using neural networks.

Table of Contents

Acknowledgements ix

1 Introduction 1

1-1 Planning spaces 1
1-2 Kinodynamic motion planning 2
1-3 Sampling-based kinodynamic planning 3
1-4 Learning-based RRT 4
1-4-1 Optimal control 5
1-5 Thesis contributions 5
1-6 Thesis layout 6

2 Rapidly-exploring random trees 7

2-1 RRT algorithm 7
2-2 Variations of RRT 11
2-3 Learning-based RRT 12
2-4 Summary 13

3 Supervised learning for RRT 15

3-1 Supervised learning 15
3-1-1 Types of supervised learning algorithms 16
3-2 K-nearest neighbours 17
3-3 Artificial neural networks 19
3-3-1 Feed-forward neural networks 20
3-4 Summary 21

4 Data generation using optimal control **23**
4-1 Kinodynamic planning as optimal control problem 23
4-2 Indirect optimal control 24
4-2-1 Pontryagin principle 24
4-2-2 Pontryagin principle with constraints 25
4-2-3 Disadvantages of indirect optimal control 26
4-3 Direct optimal control 26
4-3-1 Discretization 26
4-3-2 Single-shooting 27
4-3-3 Multiple-shooting 28
4-3-4 Direct collocation 29
4-3-5 Constrained direct optimal control 29
4-4 Data generation for learning-based RRT 30
4-5 Summary 30

5 Experimental setup **31**
5-1 Test system 31
5-2 Generation of equations of motion 32
5-3 Generation of training data 35
5-3-1 Implementation of direct optimal control 36
5-3-2 Implementation of indirect optimal control 37
5-4 Implementation of supervised learning 38
5-4-1 Data pre-processing 39
5-4-2 Implementation of KNN and feed-forwards neural network 39
5-5 Implementation of learning-based RRT 40
5-6 Summary 40

6 Results and analysis **41**
6-1 Comparison metrics 41
6-2 Comparison of direct and indirect optimal control 42
6-2-1 Learning unconstrained data with KNN 48
6-2-2 Learning unconstrained data with feed-forward neural network 51
6-3 Effect of input constraints 54
6-3-1 Learning input constrained data with KNN 58
6-3-2 Learning input constrained data with feed-forward neural network 61
6-4 Performance in learning-based RRT 64
6-5 Summary 64

7 Conclusions **65**

A `urdf2eom` **69**

B Thesis source code **71**

Bibliography **73**

List of Figures

1-1 Example of RRT and solution obtained from it [1] 4

2-1 Illustration of χ_{free}. The rectangle is the state space χ, the circle the reachable space $(\chi - \chi_{jl})$ and χ_{obs} the obstacles. 8

2-2 Illustration of Voronoi diagram in 2-dimensions [16]. Each dot is a seed. Isolated seeds have larger Voronoi regions. 8

2-3 Failure of Euclidean metric [26]. p_2 is easier to reach than p_1 9

2-4 Tree expansion in RRT [1] 10

2-5 Tree expansion in bi-directional RRT 11

2-6 RRT*:x_{new} is connected to all nodes within the circle 12

2-7 RRT*:Suboptimal connections are removed 12

2-8 Illustration of learning-based RRT [2] 13

3-1 Illustration of KNN grouping with K = 5. Blue dot is the query input, red dots are training samples 17

3-2 Illustration of cross-validation error using multiple values of k. The k corresponding to the minima is chosen. 18

3-3 Human neuron [3] 19

3-4 Perceptron 19

3-5 Illustration of feed-forward neural network [4]. Each circle is a perceptron. 20

4-1 Illustration of single shooting method [5]. Dotted line represents the N-part discretized input u parametrised by a_k . Normal line represent the state x generated using system equations. 27

4-2 Illustration of multiple-shooting method [5]. Dotted line represents the discretised input u and the normal line the state x. s_i shows point of each shooting. Note the discontinuities at each interval. 28

5-1 Test system - 2-link manipulator [6]. θ_1, θ_2 are angular positions of the two joints. 32

5-2 Illustration of articulated bodies [7]. B_j are the links and A_j represent the articulated bodies. 34
5-3 Illustration of trajectory change due to u^* switching to U_+ under input constraint 38

6-1 Gaps in phase plot of x_i : q_{2i} vs qd_{2i} 43
6-2 Gaps in phase plot of x_i : q_{2i} vs qd_{2i} 44
6-3 4-D phase plots with direct approach (multiple shooting) without input constraints. Discolouration can be seen in a) which represents incompleteness in x_i. 45
6-4 4-D phase plots with indirect approach (Pontryagin principle) without input constraints. Uniform distribution of samples is observed. 46
6-5 Sample x_i vs x_f phase plot for direct data without input constraints. Other plots have similar coverage. 47
6-6 x_i vs x_f phase plot for indirect data without input constraints. Only plots with irregularities are shown. Other plots have uniform coverage. 47
6-7 KNN prediction profiles of indirect optimal control dataset (without constraints) 49
6-8 KNN prediction profiles of direct optimal control dataset (without constraints) . 50
6-9 Neural network prediction profiles of indirect optimal control dataset (without constraints) 52
6-10 Neural network prediction profiles of direct optimal control dataset (without constraints) 53
6-11 qd_{2i} vs q_2i plot. Non-uniformity can be clearly seen here. 54
6-12 4-D phase plots with direct approach (multiple shooting) with input constraints. The sparseness in the middle in a) can be clearly seen. 55
6-13 4-D phase plots with indirect approach (Pontryagin principle) with input constraints. U-shaped gaps can be observed at the bottom of each plot. 56
6-14 x_i vs x_f phase plot for direct data with input constraints. Uniform coverage is observed on all the plots. 57
6-15 x_i vs x_f phase plot for indirect data with input constraints. 57
6-16 Prediction profiles of indirect optimal control dataset (with input constraints) . . 59
6-17 Prediction profiles of direct optimal control dataset (with input constraints) . . . 60
6-18 Neural network prediction profiles of indirect optimal control dataset (with constraints) 62
6-19 Neural network prediction profiles of direct optimal control dataset (with constraints) 63

List of Tables

6-1 Data generation time without input constraints 43
6-2 Cleaning of direct data (unconstrained) 44
6-3 Cleaning of indirect data (unconstrained) 44
6-4 Optimal k for KNN 48
6-5 Average cross-validation error for KNN 48
6-6 Cross validation error for indirect dataset with neural network 51
6-7 Cross validation error for direct dataset with neural network 51
6-8 Data generation time with input constraints 54
6-9 Cleaning of direct data 58
6-10 Cleaning of indirect data 58
6-11 Optimal k for KNN 58
6-12 Average cross-validation error for KNN (input constrained) 58
6-13 Cross validation error for constrained indirect dataset with neural network 61
6-14 Cross validation error for constrained direct dataset with neural network 61
6-15 Run times with different algorithms 64

Acknowledgements

I would like to thank my supervisors Dr.Ir. Martijn Wisse and Ir. Mukunda Bharatheesha for their assistance during the writing of this thesis. This thesis could never have been completed without their support. Their advice was instrumental in developing my interest in motion planning and I learnt a lot under their guidance.

I would also like to thank my friends for their help and encouragement over the course of the thesis. I am especially grateful to my family, whose unwavering support helped me stay strong during the tough times.

Delft, University of Technology
February 16, 2018

Deepak Paramkusam

"I have not failed. I've just found 10,000 ways that won't work."
— *Thomas A. Edison*

Chapter 1

Introduction

Motion planning is one of the core components of robotics. Determining a path is a major challenge for robots due to its computational complexity. Motion planning problem is informally defined as generating a 'plan' for a given robot, so as to reach the goal state from the current state. The plan governs where a robot should move and how it should move from the current state. It can be defined using waypoints in a known environment or as a sequence of inputs to robot actuators [1]. Robots also require velocity and acceleration information while executing the motion. These are usually obtained from the robot dynamic equations. Generating a motion plan accurately and quickly is one of the open problems in robotics [8].

This chapter will give a brief overview of motion planning, specifically kinodynamic motion planning. Standard algorithms used for kinodynamic planning and techniques to improve their performance are also introduced. Then the contributions of this thesis are laid out based on the overview.

1-1 Planning spaces

A motion plan can be generated in various spaces based on how the robot is represented in the environment. A *space* is defined as a set of parameters that is used to represent a robot configuration or state [9]. A sequence of these parameters is also used to represent the trajectory Γ of the robot. Different spaces used for motion planning are:

1. *Cartesian space*
   Cartesian Euclidean space is one of the standard spaces used for planning. The x, y and z coordinates represent the position of the robot end effector (or centre of mass in case of mobile robots) and the orientation is represented in terms of direction cosines (1-1). The trajectory to be traversed by the end effector is described using these coordinates.

$$\begin{aligned} p &= (p_x, p_y, p_z) \\ \theta &= (\alpha, \beta, \gamma) \end{aligned} \tag{1-1}$$

2. *Configuration space*
   Configuration space (also known as joint space) uses generalized coordinates to represent the robot configuration. Number of generalized coordinates is equal to the number of degrees of freedom of the robot with each coordinate corresponding to the joint value (1-2). Configuration space is generally more convenient than Cartesian space as the robot dynamic equations are also formulated in configuration space. Coordinates can be transformed from configuration space to Cartesian space using the robot forward kinematic equations.

$$q = (q_1, q_2, q_3 ....q_n) \tag{1-2}$$

3. *State space*
   State space represents the complete dynamical state of the robot, not just the robot configuration. Dynamical systems are fully defined by their position and velocity [10]. So the robot state consists of position-velocity pairs (either Cartesian or configuration) of each degree of freedom (1-3).

$$x = (q, \dot{q}) \tag{1-3}$$

1-2 Kinodynamic motion planning

Kinodynamic motion planning aims to generate a robot trajectory Γ subject to simultaneous kinematic and dynamic constraints [1]. The term kinodynamic comes from the the combination *kinematic + dynamic*. Kinematic constraints include configuration limitations such as the joint limits and obstacles. Dynamic constraints include dynamic laws and bounds on velocity and acceleration. Planning is performed in the state space. Kinodynamic planning is formally defined below as in [1].

Let χ be the state-space of the robot. Let the state-space consist of the configuration q of the robot and its derivative $\dot{q}$. Consider an initial state $x_i \in \chi$ and final state $x_f \in \chi$. Then the kinodynamic planning problem involves finding a continuous time-optimal trajectory Γ such that

$$\Gamma : [0, T] \to \chi \mid_{\Gamma(0)=x_i \text{ and } \Gamma(T)=x_f} \tag{1-4}$$

In terms of the control input,

$$u : [0, T] \to U \tag{1-5}$$

Mathematically, kinodynamic planning is a standard two-point boundary value problem subject to constraints. Traditional control strategies were initially used to solve this problem. O'Dunlaing [11] provided the exact solution to the kinodynamic problem in one-dimension using bang-bang control and quadratic pursuer functions. It is observed that the algorithm is of polynomial-time complexity. Time complexity of an algorithm gives an estimate of the time taken to run the algorithm as a function of the input. Canny et.al [12] adapted the technique demonstrated by O'Dunlaing to get the exact solution to the kinodynamic problem in a plane and showed that this solution can be generated in exponential time. For the kinodynamic problem in 3-dimensions, no exact solution is known. Canny and Reif [13] proved

that the problem is non-deterministic polynomial-time hard (NP-hard) in 3-dimensions i.e has a time-complexity of at least polynomial order or higher.

To overcome this high time complexity, approximate solutions started being explored. Early approaches include using potential field techniques [14],[15] and dynamic programming [16],[17]. Donald and Xavier [18] and Papadimitriou [19] discretised the state space into a grid and then use graph-search techniques (like breadth-first search) to get an approximate solution. These algorithms have polynomial-time complexity for two-dimensional and three-dimensional systems.

1-3 Sampling-based kinodynamic planning

Sampling-based algorithms were introduced in late 1990s to obtain an approximate solution to the problem. Sampling-based approaches represent the state space as a graph and attempt to find a path from initial state to final state using different randomized sampling techniques. Sampling-based planners work on the principle of *probabilistic completeness*, which means that if a solution exists, the random planner will eventually find it [20]. But it is to be noted that optimality of the solution is not guaranteed. Sampling-based algorithms are judged based on their convergence rate and average speed of convergence (as traditional time-complexity analysis does not make sense for random algorithms that may not even converge). Nevertheless, it was observed that sampling-based algorithms are on an average at least an order of magnitude faster than the polynomial-time analytic algorithms [21]. Two major sampling-based algorithms for kinodynamic planning are *Probablistic Roadmap* and *Rapidly-exploring Random Trees*.

Probabilistic roadmap Probabilistic roadmap (PRM) was proposed by Amato and Wu [22] and Kavaraki et.al [23] in 1996. Probabilistic roadmap algorithm consists of two steps - roadmap construction and query. In the construction step, certain number of random states across the state space are sampled and a graph is generated connecting these states (if possible). This is a computationally expensive pre-processing step but needs to be performed only once. In the query step, initial and final states are input and Dijkstra's algorithm is used to obtain the trajectory between them from the graph. Probabilistic roadmap is useful in case of multiple queries.

Rapidly-exploring random trees Rapidly-exploring random trees (RRT) is a single-query algorithm presented by Lavalle and Kuffner [1] in 2000. Given an initial state in a graph, RRT attempts to incrementally construct a tree towards the goal state . Each intermediate state is randomly sampled and connected to the tree if the connection satisfies the kinodynamic constraints. The tree can be directed towards the final state by sampling states closer to the final state. Once the tree reaches the final state, a path from initial to final state is formed along the tree (Fig. 1-1).

Thus, RRT mainly differs from PRM in the following ways :

1. RRT performs the computationally expensive constraint checking only for each sampled state while PRM performs it over the entire state space (during the construction step).

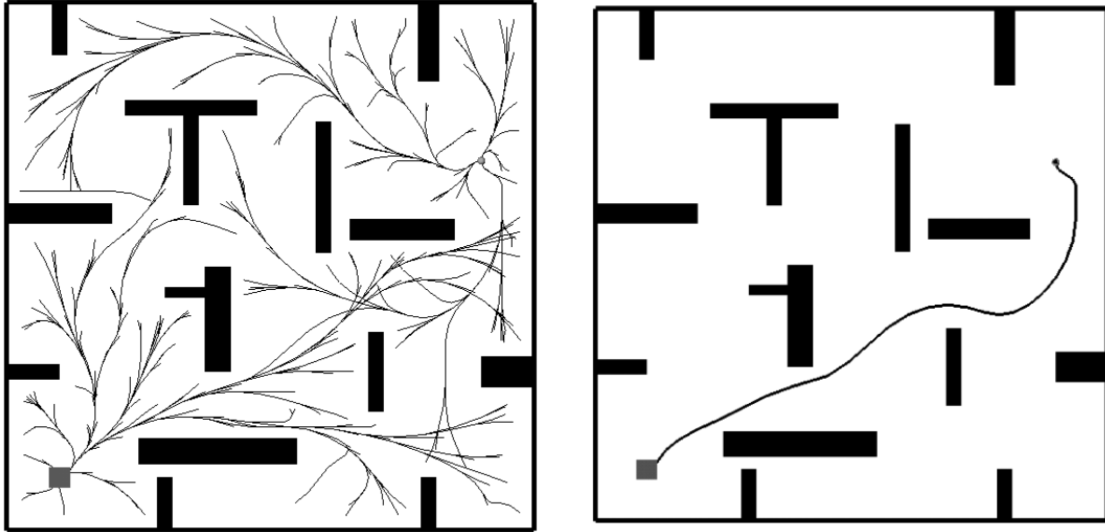


Figure 1-1: Example of RRT and solution obtained from it [1]

2. PRM needs to re-run the construction step if there is any change in the environment while RRT only needs to modify its constraint checking [23].

Even though RRT helped reduce the planning time, even lower planning times are desired for industrial applications. Noteworthy improvements to RRT include efficient state sampling [24], [25], bi-directional RRT [1], [26] and 'pruning' of the tree at regular intervals [27]. But it is observed that RRT is inhibited at its core by two time-consuming steps:

- *Nearest-neighbour step* : Finding nearest state to sampled state in the current tree
- *Steering step* : Finding input to move towards the sampled node while satisfying dynamic constraints

Improving these two steps can significantly improve the planning time of RRT. RRT and its improvement is the focus of this thesis and its working will be discussed in detail in Chapter 2.

1-4 Learning-based RRT

In recent times, supervised learning has been proposed to improve the planning time of RRT. Supervised learning is a machine learning technique. It involves analysing large amounts of data (known as *training data*) to try and infer a function or a model represented by that data. Inferred models help generalise over unknown inputs. These 'learned' models are then stored as a simple mapping, which can be used to predict the output and are faster than the actual computation [28].

Two of the widely used supervised learning algorithms are *k-nearest neighbour regression* and *multilayer neural networks* [29]. K-nearest neighbour algorithm is a simple regression

technique that predicts the output of an input based on its nearest neighbours in the training data. Weights can be assigned to the neighbours to control the bias of the prediction. Multilayer neural network uses artificial neurons (called *perceptrons*) to try and imitate the human brain. These networks can be trained for supervised learning using a technique called back-propagation. It has been proven that a neural network can approximate any continuous function [30]. Detailed explanation of these techniques is provided in Chapter 3.

Supervised learning approach to learn the nearest-neighbour step of RRT was presented by Bharatheesha et.al [31] and Palmieri and Arras [32]. The function governing the cost of traversal between two states (referred to as *cost function*) is learned in their work. Locally weighted projection regression (LWPR) and basis function models were used for the learning. Planning time is reduced by two-orders of magnitude with these methods. In Wolfslag et.al [2], the use of supervised learning for the steering step is demonstrated alongside learning the cost function for an inverted pendulum. The parametrised control input was learned using k-nearest neighbour algorithm for the steering step. This resulted in ten-fold reduction of planning time.

1-4-1 Optimal control

In [2] and [31], training data for learning is obtained through optimal control techniques. Optimal control uses optimization techniques to find a suitable control strategy subject to defined constraints [33]. An optimality criterion defines the goal of the problem. The kinodynamic planning problem can be formulated as an optimal control problem, with the optimality criterion being minimization of time or energy or weighted combination of both. Kinodynamic constraints are quantified using inequalities. The criterion is mathematically represented as a cost function and an optimization algorithm is used to find a solution which minimizes the cost function.

Two standard optimal control approaches are *direct and indirect optimal control*. Indirect optimal control is an older and more analytic approach that is based on calculus of variations [34]. The calculus of variations give the optimality conditions which are minimized to give the required solution. But this approach tends to grow unstable for systems that run for long periods of time . Direct optimal control is a numerical approach that grew in popularity with the advent of computers. Direct optimal control discretizes the system equations and transforms the optimal control problem into a non-linear programming problem (NLP), which is easy to solve [35]. More information on optimal control theory is given in Chapter 4.

Despite the disadvantages of indirect optimal control, this approach is used in [2] to generate the data for supervised learning. The authors show that indirect optimal control allows easy data generation with fewer parameters.

1-5 Thesis contributions

In this thesis, an effort is made to further investigate the use of supervised learning in RRT and use of optimal control for training data generation. Therefore, the contributions of this thesis are :

1. *Comparison of direct and indirect optimal control for data generation*
   The use of indirect optimal control for training data generation is detailed in literature, but comparison with direct optimal control has not been performed so far. The comparison is carried out in this thesis and the differences between the approaches are quantified.

2. *Use of multilayer neural network for learning*
   Simple regression techniques like k-nearest neighbour have been used so far for learning -based RRT. Neural networks, which are faster and more accurate, are used in this thesis.

3. *Study the effect of input constraints on learning*
   Learning-based RRT is demonstrated in [2] and [31] for a simple system without input constraints. Optimal control becomes much harder with the addition of input constraints and the data to be learnt is also bounded. The effect of this is studied in this thesis.

1-6 Thesis layout

The thesis is structured as follows:

Chapter 2 details the working of RRT and learning-based RRT.

Chapter 3 includes information about k-nearest neighbour algorithm and multilayer neural networks and motivation for choosing these techniques.

Chapter 4 explains the basics of optimal control theory, direct and indirect optimal control and how they are used to generate data for supervised learning.

Chapter 5 details the experimental setup used in this thesis and the approach used.

Chapter 6 evaluates the results obtained from the experiments and discusses the results.

Chapter 7 presents the conclusions and provides recommendations for future work.

Chapter 2

Rapidly-exploring random trees

Rapidly exploring random trees (RRT) is an iterative sampling-based algorithm which can efficiently search high-dimensional spaces. It is a part of a family of single-query algorithms called rapidly exploring dense trees (RDTs). RRT was originally designed for configuration space planning by Lavalle. It was later extended to kinodynamic planning by Lavalle and Kuffner [1] to generate open-loop solutions to the problem. The algorithm is probabilistically complete and will converge to a solution as number of iterations increase. But the solution is not optimal. This chapter discusses the RRT algorithm in detail and the bottlenecks in the algorithm are highlighted. Then the recently proposed learning-based RRT is presented. Learning-based RRT is a variation of RRT that uses supervised learning to make RRT faster [2]. The working of learning-based RRT and how it overcomes the bottlenecks of RRT is explained.

2-1 RRT algorithm

In RRT, the state space is discretely represented as a graph i.e each node in the graph corresponds to a state of the system. RRT algorithm then generates a tree connecting these nodes. The tree starts at the node corresponding to the initial state x_{init} and the tree expansion ends when the node corresponding to the final state x_{goal} is added to the tree. The path along the tree from x_{init} to x_{goal} is the required solution. RRT mainly consists of the following steps [1],[36]:

- Sampling step
- Nearest-neighbour step
- Steering step
- Termination check

Each of these steps is elaborated below.

Sampling step A node x is sampled randomly from the search space χ_{free} in this step. The search space χ_{free} is a subset of the state space χ. Infeasible regions of the state space χ_{inf} are not sampled [1]. χ_{inf} includes regions with obstacles χ_{obs} and regions not reachable due to joint limits χ_{jl}. In case of dynamic obstacles, χ_{obs} is updated every iteration. Example of χ_{free} is given in Fig. 2-1.

$$\chi_{free} = \chi \setminus \chi_{inf} \qquad (2\text{-}1)$$
$$\chi_{inf} = \chi_{obs} \cup \chi_{jl}$$

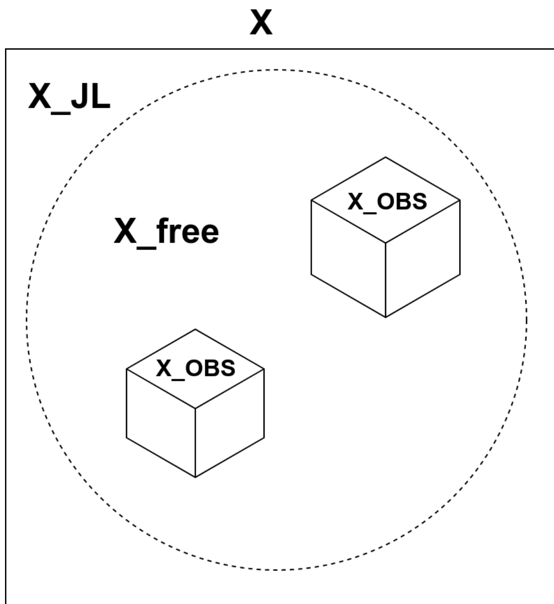


Figure 2-1: Illustration of χ_{free}. The rectangle is the state space χ, the circle the reachable space $(\chi - \chi_{jl})$ and χ_{obs} the obstacles.

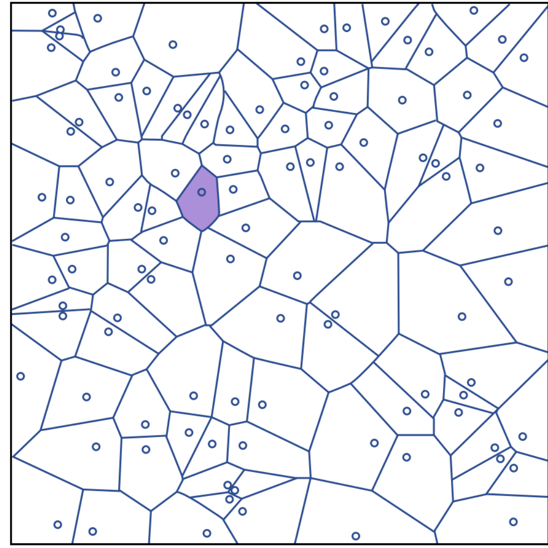


Figure 2-2: Illustration of Voronoi diagram in 2-dimensions [16]. Each dot is a seed. Isolated seeds have larger Voronoi regions.

x_{goal} is also sampled on a regular basis. This pushes the tree towards x_{goal} and is known as *goal bias.*

It is to be noted random sampling helps RRT explore faster than most of its peers such as PRM. This is due to *Voronoi bias.* Voronoi diagram of a set of nodes (called *seeds*) is a region of the graph around each seed. Every node in each region is closer to that region's seed than any other seed (Fig. 2-2). If each node of RRT is taken as a seed, the furthest vertices have the largest Voronoi regions (because there are no seeds in unexplored regions). When a new node is randomly sampled, it is more likely that it is sampled from larger Voronoi regions, thus pushing the tree into unexplored regions [37].

Nearest-neighbour step This step involves finding the nearest node x_{near} to the sampled node x in the current tree. The closeness of two nodes is determined using a *distance function.* Choice of the distance function has repercussions on the behaviour of the RRT. Ideal distance function is the optimal go-to cost between the states, but this computation is often

prohibitively time consuming. Heuristics are usually chosen instead. Euclidean distance (2-2) between the two states was originally chosen in [1] as the distance function.

$$D_{eucl} = \sqrt{\sum (q^i_{rand} - q^i)^2} \tag{2-2}$$

Glassman and Tedrake [38] and Spierenburg [39] show that the Euclidean metric (using either position or velocity) has no correlation to the motion in state space and has a negative effect on planning. This can be demonstrated using Fig. 2-3.

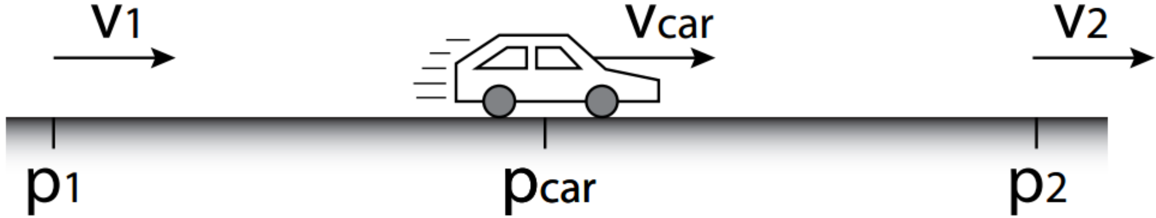


Figure 2-3: Failure of Euclidean metric [26]. p_2 is easier to reach than p_1

Let the euclidean distance from the car to the two points be equal (2-3). Therefore the euclidean distance metric gives same cost for both the points.

$$|p_1 - p_{car}| = |p_{car} - p_2| \tag{2-3}$$

But the cost to go from p_{car} to p_2 should be less than cost to p_1 as the car needs to reverse its velocity in the later case. Thus euclidean metric is incorrect in this case.

Many alternative heuristics take advantage of the system properties such as symmetry and dynamics. State energy, linearised traversal cost [40] and dynamics-based affine quadratic regulator (AQR) [38] are proven to be better heuristics.

Steering step Once x_{near} is found in the previous step, a viable input u to move towards x is applied for time period Δt. Δt can be a fixed value or random. u is chosen based on a *steering function*. Using this u and Δt, the system is then propagated (by integrating the system dynamic equations) from x_{near} to a new node x_{new}. Constraints like collision checking are also checked over the course of this simulation. If a valid x_{new} is found, it is added to the tree. This is demonstrated in Fig. 2-4.

RRT, in general, does not require a steering function. u can be chosen randomly from a set of viable control inputs and a solution can be found due to the algorithm's probabilistic completeness [1]. But a proper steering function can speed up the expansion process. Ideally, the steering function should return a control input which causes the system to eventually go from x_{near} to x. This is not feasible in practice as it is the kinodynamic problem itself. Alternate approaches include attempting multiple inputs and choosing the best [41], PID-based estimator [39] and polynomial interpolation between states [42]. But it should be noted that if a steering function is used, it increases the computation time of this step.

As previously mentioned, kinodynamic constraints are checked during the propagation step (also known as *local planning*). Numerical integration technique such as Runge-Kutta approach is generally used. Each point obtained during the propagation from x_{near} to x_{new}

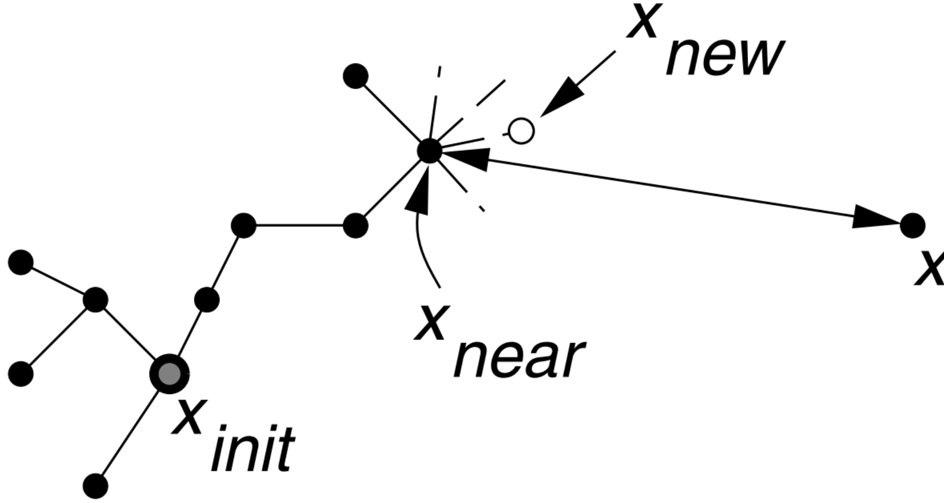


Figure 2-4: Tree expansion in RRT [1]

should be in χ_{free} to satisfy the kinematic constraints. Dynamic laws are automatically satisfied as they are used in the integration. The input u should satisfy the actuator limits if any. So any steering function used should be bounded accordingly.

Termination check If x_{new} node is x_{goal} (or within the tolerance limit of x_{goal}), the path from $x_{initial}$ to x_{goal} along the tree is a solution to the kinodynamic problem and the algorithm stops. If the iteration limit is reached, the algorithm has failed and no solution is found. Else the the algorithm goes back to the sampling step and iterates over the entire process.

The pseudo code of these steps to generate trajectory τ are given in Algorithm 1 as in [1]. $SAMPLE$ and $NEAREST$ functions perform steps 1 and 2. NEW_STATE function returns true if the connection to x_{new} is possible using u. $TERMINATE$ function ends the algorithm.

Algorithm 1 RRT

1: **procedure** RRT(x_{init})
2: $\tau.init(x_{init})$
3: **for** $i = 1....N$ **do**
4: $x_{rand} \leftarrow SAMPLE$
5: $x_{near} \leftarrow NEAREST(x, \tau)$
6: **if** $NEW_STATE(x, x_near, x_{new}, u)$ **then**
7: $\tau.add_vertex(x_{new})$
8: $\tau.add_edge(x_{near}, x_{new}, u)$
9: **if** $x_{new} = x_{goal}$ **then**
10: $TERMINATE$
11: **end if**
12: **end if**
13: **end for**
14: **end procedure**

Nearest neighbour step and steering step are the main impediments to RRT's planning time. Even though a single distance metric computation is negligible, the distance to every node in the tree needs to be checked every iteration. When the tree gets very large, this can add up and delay planning [39]. If a steering function is used, the steering step also causes delays. Ideal steering input is obtained using optimal control which is computationally expensive as it is an iterative process. These bottlenecks resulted in attempts to modify and improve RRT. The major variations of RRT are discussed in the next sections.

2-2 Variations of RRT

Two notable variations of RRT - bi-directional RRT and RRT* - are briefly discussed in this section. These variations improve RRT by taking modifying the tree structure. The former aims to improve planning time while the later aims to generate optimal solutions.

Bi-directional RRT Bi-directional RRT is same as vanilla RRT except that two trees are expanded - one from x_{init} and one from x_{goal}. Each tree is expanded alternately in each iteration from each x_{new} towards the closest node on the other tree x_{target} (Fig. 2-5). The algorithm is terminated when a node common to both the trees is found. Bi-directional RRT was shown to improve planning times relative to regular RRT [43]. But bi-directional RRT sometimes has discontinuities at the common node which is not desirable.

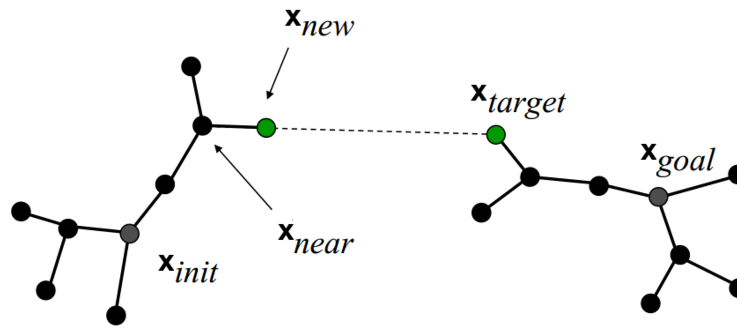


Figure 2-5: Tree expansion in bi-directional RRT

RRT* RRT* modifies RRT to assure asymptotic optimality. Regular RRT does not give optimal solutions as it returns the first solution it finds. RRT* obtains optimal solutions by "rewiring" the tree every iteration. This ensures that only the optimal connections are present in the tree. Every time x_{new} is found, instead of connecting it to x_{near}, x_{new} is connected to all nodes within a a certain range [44]. Costs (distance function value) are then propagated from the base to all the nodes and high-cost redundant connections are removed (Fig. 2-7 and Fig. 2-6). Other steps are same as regular RRT. It is to be noted that RRT* is slightly slower than regular RRT due to the additional rewiring step. However, on average, the planning time is similar to that of regular RRT [16].

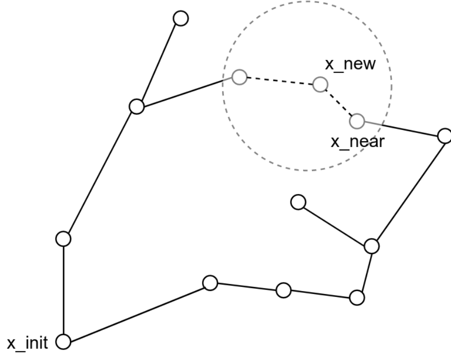


Figure 2-6: RRT*:x_{new} is connected to all nodes within the circle

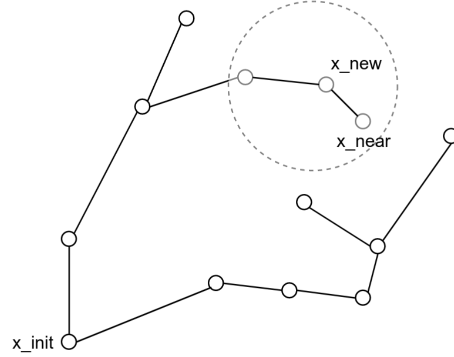


Figure 2-7: RRT*:Suboptimal connections are removed

2-3 Learning-based RRT

Learning-based RRT is a recently proposed supervised learning approach to improve the planning time of RRT. Unlike the innovations in the previous section, this approach aims to modify the core steps slowing RRT - distance function and steering function - with minimal loss of accuracy unlike heuristics.

In section 2-1, it was mentioned that the ideal distance metric is the optimal traversal cost between each state and the ideal steering function is the optimal input to move from x_{near} to x. The traversal cost and the optimal input can be generated using optimal control techniques, but it is very time consuming (NP-hard, as mentioned in Chapter 1). Learning-based RRT solves this problem by moving the optimal control part offline. Data generated by optimal control techniques is input to a supervised learning algorithm which learn these functions [2]. The learned models can directly replace the distance and steering functions in RRT, with no other changes to the algorithm. The learned models approximate the steering function and the distance function much faster than the optimal control approach, thus speeding up RRT's planning time.

Both the distance function and the steering function take in two states x_0, x_1 as the input. The distance function should return a scalar cost value j and the steering function a parametrized input vector u for each degree of freedom. Optimal control techniques therefore are used to generate datasets (x_0, x_1, j) and (x_0, x_1, u). The learning algorithm then generates models f_j and f_u which can predict j and u by learning the datasets. f_j and f_u can then replace the $NEAREST$ and NEW_STATE functions in Algorithm 1. This process is depicted in Fig. 2-8.

In the literature, k-nearest neighbour regression has been used so far to learn the steering function. LWPR and linear basis functions have been used for distance function alongside k-nearest neighbour regression [26].Both the optimal control approaches - direct and indirect optimal control- have been used for the data generation. Authors of [2] further recommend the use of indirect optimal control citing its lower data generation time and fewer parameters.

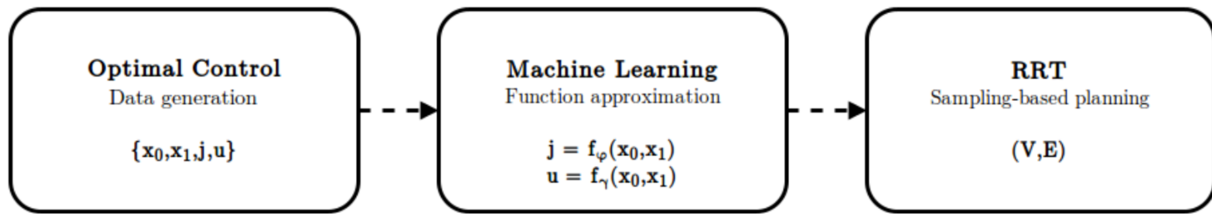


Figure 2-8: Illustration of learning-based RRT [2]

Being a very recent approach, properties of learning-based RRT is not known in detail. This thesis attempts to fill this gap by further investigating the learning and data generation parts of learning-based RRT.

2-4 Summary

This chapter details the rapidly-exploring random tree (RRT) algorithm, a common kinodynamic planning algorithm and its main steps. Two major variations of RRT are also introduced briefly. It is seen how the nearest neighbour search and steering function are the bottlenecks in the RRT algorithm. Learning-based RRT, a new variant of RRT, attempts to overcome these bottlenecks using a supervised learning approach. The learning requires large amounts of data which is obtained using optimal control techniques. The next chapter introduces some supervised learning algorithms used in learning-based RRT.

Chapter 3

Supervised learning for RRT

The previous chapter introduced the use of supervised learning to improve RRT planning time. This learning-based RRT has shown significant planning time reduction over the regular RRT. This chapter provides an introduction to supervised learning and how it works. Common supervised learning techniques are briefly discussed and the two supervised learning algorithms used in this thesis, k-nearest neighbour and artificial neural networks, are explained in detail.

3-1 Supervised learning

Supervised learning is a machine learning technique that approximates a function using training data. The learned function can then be used to predict outputs for previously unseen inputs. Learned function is mathematically represented as in 3-1.

$$g : X \rightarrow Y \;\; \forall \;\; A_n = ([x_1, y_1]....[x_n, y_n]) \in (X, Y)^n \tag{3-1}$$

where set X represents the input space, Y represents the output space, g is the learned function and A_n is the n - sample training set. Each sample in the training set consists of an input-output pair (x_i, y_i) and assumed to be generated with an unknown but fixed joint probability distribution function $P(x, y)$ [45]. The relation between X and Y encoded by $P(x, y)$ is approximated by g. g is searched in a space with all possible functions known as *hypothesis space*. The learning problem is known as a *classification* problem if the output is discrete and a *regression* problem if the output is continuous.

Supervised learning is an iterative process where g is tuned continuously. Therefore it becomes important to evaluate the how well g fits the actual output. *Loss* L is defined to measure this error in supervised learning. It measures the difference between the predicted output $\hat{y} = g(x)$ and actual output y [46]. The choice of the loss function varies depending on the type of supervised learning problem. Common loss functions are squared loss, absolute loss

and log-cosh loss (3-2).

$$\begin{aligned} L_{sq.loss} &= [g(x) - y]^2 \\ L_{abs.loss} &= |g(x) - y| \\ L_{log-cosh} &= log(cosh(g(x) - y)) \end{aligned} \tag{3-2}$$

Expectation of loss over an entire data set is termed as *risk* β (3-3).

$$\begin{aligned} \beta(g) &= \mathbb{E}[L(g(x), y)] \\ &= \int L(g(x), y)\ dP(x, y) \end{aligned} \tag{3-3}$$

Therefore, supervised learning basically involves finding a function g that minimizes β [47]. This is termed as *risk minimization*. Risk minimization consists of two parts - finding an appropriate g to minimize β (known as empirical risk minimization) and avoiding over-fitting (known as structural risk minimization).

An important point to note is that a learning algorithm is evaluated on a test data set (known as *validation set*). This is because the algorithm can have no error with training set yet perform poorly on samples not in the training set. Another common method to evaluate performance is k-fold cross validation. This involves splitting the training set in *k* equal parts (*folds*) and then one fold is used as validation set while other *k - 1* folds are used for training. This is repeated *k* time for each fold. The average performance of the k-folds is taken to give a single performance estimate [47]. The advantage of k-fold cross validation is that each fold is used for training and validation, avoiding any inherent bias.

3-1-1 Types of supervised learning algorithms

Supervised learning algorithms are broadly classified into *parametric* and *instance-based* algorithms depending on how they work. In parametric learning algorithms, a general form of the function g is assumed and its coefficients are modified iteratively such that β is minimized. Parametric algorithms have constant space complexity (space complexity is a measure of how much digital memory an algorithm requires to function) irrespective of the size of the training set due to the fixed number of tunable parameters [48]. This also makes parametric algorithms faster than non parametric algorithms. Training data is not required once the training is completed. But this approach is constrained by the selected form of g. Poor learning can occur if the assumption is incorrect. Examples of parametric learning algorithms are logistic regression, neural networks and naive Bayes [48].

Instance-based algorithms or non-parametric algorithms correlate each query with instances seen in the training set. They have don't parameters and make no assumptions about g (in fact, they don't explicitly formulate a g), which allows a more general learning. These algorithms don't have usually have a training phase, but instead use the training data whenever required to make predictions [49]. Because of this, these algorithms are also called *lazy learning* algorithms. However, requirement of training data at all times causes these algorithms have higher space complexity and the processing time compared to parametric algorithms. Examples of non-parametric algorithms are k-nearest neighbours, decision trees and support vector machines (SVMs).

Two supervised learning algorithms for learning-based RRT are discussed next in detail : k-nearest neighbours and artificial neural networks. Since the distance metric and the control input are continuous functions in learning-based RRT, regression is focussed upon in both these techniques.

3-2 K-nearest neighbours

K-nearest neighbours (KNN) is a simple non-parametric learning algorithm. The algorithm expects similar inputs to have similar outputs [50]. KNN algorithms consists of two steps-

- Whenever an input is given to the KNN algorithm, it first finds the k nearest samples to the input in the training set (Fig. 3-1). The nearest neighbours are found using a distance metric.
- The predicted output is the combination of the outputs of the nearest samples found in the previous step in case of regression. Commonly used combination is mean of the samples. In case of classification, the predicted output is the class of majority of the neighbours.

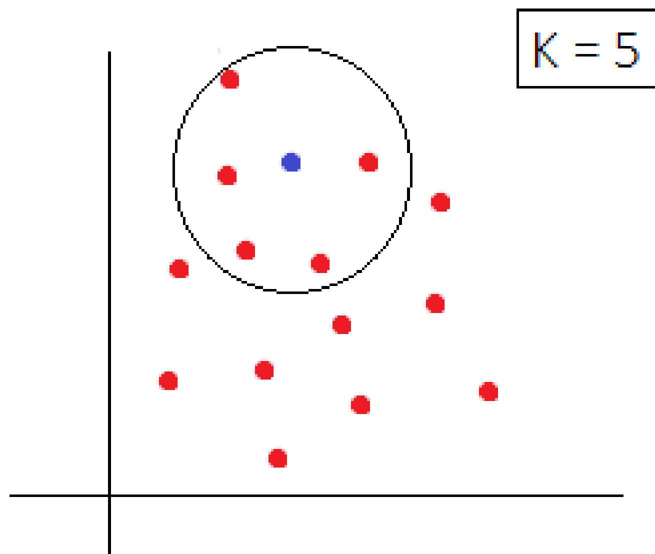


Figure 3-1: Illustration of KNN grouping with K = 5. Blue dot is the query input, red dots are training samples

KNN algorithm is influenced by the following factors -

1. *Distance meteric*
   Distance metric refers to the function used to find the nearest neighbours in the training set. Common metrics to find the distance between two samples p and q are Euclidean distance, Minkowski distance and Mahalanobis distance (3-4). All of them have similar

performance, but Minkowski distance is faster to compute for high dimensional datasets [51].

$$\begin{aligned} D_{eucl} &= \sqrt{\sum (p_i - q_i)^2} \\ D_{mink} &= \sum |p_i - q_i| \\ D_{maha} &= \sqrt{\sum \frac{(p_i - q_i)^2}{s_i^2}} \end{aligned} \tag{3-4}$$

where s_i is the standard deviation of p_i and q_i.

2. *Number of neighbours K*
   Choice of k makes a big difference in accuracy of the algorithm. If k is too low, over-fitting occurs and the algorithm becomes sensitive to noise. If it is high, the computation becomes expensive and under-fitting might occur. Standard method to select a proper k is cross validation using multiple values of k. Cross-validation error is least for optimal value of k (Fig. 3-2).

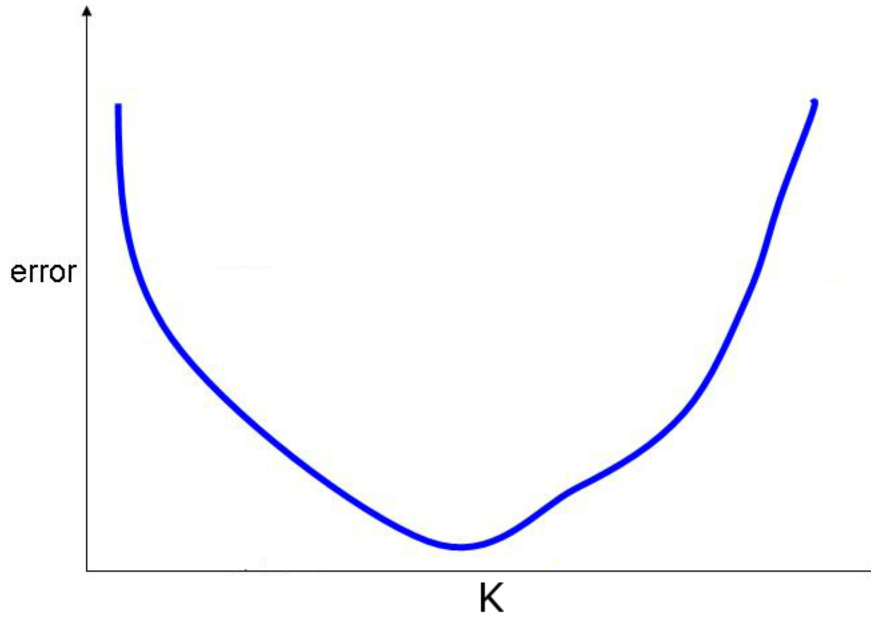


Figure 3-2: Illustration of cross-validation error using multiple values of k. The k corresponding to the minima is chosen.

3. *Combination function for prediction*
   When used for regression, the second step of KNN involves combining the outputs of the nearest neighbours to give the predicted output. The average of the outputs is generally taken (3-5).

$$y_{pred} = \frac{\sum y_i}{k} \tag{3-5}$$

   Other possible functions for combination include distance-weighted mean and local linear regression [52]. Computation time is affected by the function chosen.

KNN inherits the advantages and disadvantages of non-parametric learning algorithms. It is a very simple algorithm. It is not constrained by any function form and there is no learning stage. But it needs the training data for each prediction. KNN also gets computationally expensive as the training data size gets large, due to the large number of distances to calculate. KNN also suffers from "curse of dimensionality". Number of samples in a given volume of space decreases exponentially with number of dimensions i.e samples become sparse in space. That means as the the dimensions of the training set increases, number of nearest neighbours decreases[52]. This is problematic for KNN as its learning depends on the neighbours.

KNN is used in this thesis for learning-based RRT. It has been chosen to serve as a baseline and to allow comparison with previous research [2] which also uses KNN.

3-3 Artificial neural networks

Artificial neural network (ANN) is a popular parametric learning algorithm. ANN works by imitating the working of the human brain. The human brain consists of millions of interconnected neurons, a specialized cell that processes data using electric and chemical signals (Fig. 3-3). ANN uses an artificial neuron known as *perceptron* (Fig. 3-4). Multiple perceptrons are interconnected to form a network which is used for learning.

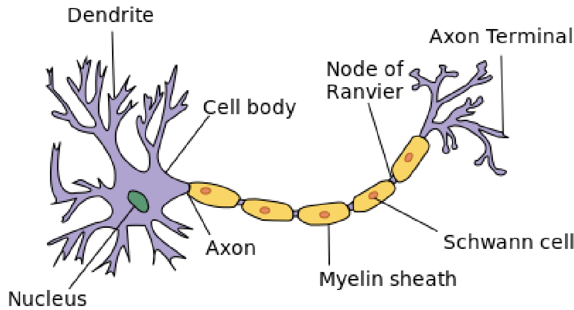


Figure 3-3: Human neuron [3]

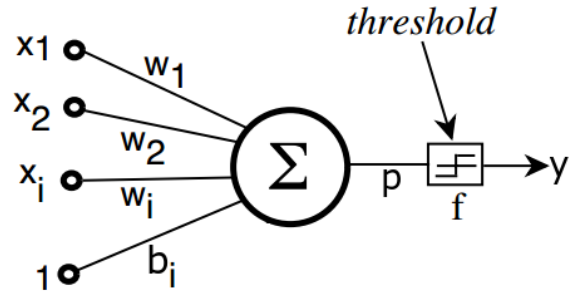


Figure 3-4: Perceptron

A perceptron has finite n number of inputs x_i each with its corresponding weight w_i [53]. Weighted sum of the inputs is then taken as in 3-6. Biases b_i are also added if an offset is required (3-4).

$$p = \sum_{i=i}^{n} x_i w_i + b_i \tag{3-6}$$

The sum p is then passed through a threshold function f. This threshold function determines the contribution of the perceptron i.e. if the perceptron activates or not. The output y is then represented as in 3-7

$$y = f(p) = f(\sum_{i=i}^{n} x_i w_i + b_i) \tag{3-7}$$

Commonly used threshold functions are sigmoid, hyperbolic tangent and rectified linear unit (ReLu) (3-8).

$$
\begin{aligned}
f_{sigmoid}(x) &= \frac{1}{1+e^{-x}} \\
f_{tanh}(x) &= \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \\
f_{relu}(x) &= \max(0,x)
\end{aligned}
\tag{3-8}
$$

Classification problems use a *softmax* threshold function in the output layer [54]. Softmax function outputs probabilities of belonging to various classes. Class with maximum probability is chosen as the output.

Perceptrons are then arranged in a network. Based on the structure of the network, the neural networks are of various types such as feed-forward neural network, recurrent neural network, deep neural network, radial basis function networks etc.

3-3-1 Feed-forward neural networks

Feed-forward neural network is the most popular network architecture [54]. Feed-forward network consists of perceptrons arranged in layers. The layers are of three types - *input layer, hidden layer* and *output layer*. Number of perceptrons in the input layer and the output layer are fixed and are equal to the dimensionality of the input and the output respectively. The hidden layers are in between the input and the output layers. The number of hidden layers and the number of perceptrons in each vary depending on the learning problem. In feed-forward network, each perceptron in a layer is connected to every perceptron in the next layer. Feed-forward network is illustrated in Fig. 3-5.

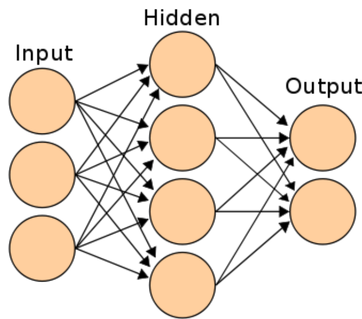


Figure 3-5: Illustration of feed-forward neural network [4]. Each circle is a perceptron.

Feed-forward networks are trained by tuning the weights (w_i) of each perceptron in a network using stochastic gradient descent algorithm [53]. This is done using a technique called *back-propagation* [53]. In backpropagation, for each training sample, random weights are initially assigned to each perceptron and the corresponding output across the network is calculated

using 3-7 for every perceptron. The error E between this generated output and the actual output is then calculated using a loss function.

The gradient of the error $\frac{\delta E}{\delta w_i}$ with respect to each weight w_i is then calculated by propagating the error backwards. The weights are then updated by subtracting a fraction α (known as the *learning rate*) of the cumulative gradient from the current weights (3-9). This is done iteratively till the loss is within acceptable limits. Once the network converges, the final weights are stored and the network is “trained”. New outputs can be predicted very quickly by using the perceptron equation.

$$w_i = w_i - \alpha \sum_{i=i}^{n} \frac{\delta E}{\delta w_i} \tag{3-9}$$

Variants of stochastic gradient descent like adaptive moment estimation (ADAM) and Adagrad [55] have been proposed in recent times. These algorithms allow varying learning rate α to help the network converge faster.

Being a parametric learning algorithm, feed-forward neural networks are heavily influenced by the structure of the network. In case of regression, universal approximation theorem for neural networks proves that any continuous function can be approximated using a single hidden layer [4]. Number of perceptrons in the hidden layer is also a critical parameter. High number of perceptrons cause over-fitting while low number may not model the data correctly. Proper number of hidden layers and number of perceptrons in each layer is generally chosen after multiple trials with different topologies.

Feed-forwards neural network is used for learning-based RRT to take advantage of its fast prediction compared to KNN. Effect of different topologies and their effect is also studied. The implementation is discussed in Chapter 5.

3-4 Summary

Supervised learning is introduced in this chapter and its general properties. Working of two learning algorithms - k-nearest neighbours and feed-forward neural networks - are discussed and motivations for choosing them for learning-based RRT are provided. These algorithms will be used for approximating the distance metric and the control input. The generation of data required for training these algorithms will be explored in the next chapter.

Chapter 4

Data generation using optimal control

Learning-based RRT uses supervised learning methods discussed in the previous chapter to model the distance function and steering function of RRT. Data required to train these models is generated using optimal control principles. Optimal control is a method of finding a suitable control policy for a given system so that a given performance criterion is achieved. The criterion is generally defined in terms of minimizing or maximizing a certain system parameter. It should be noted that optimal control is computationally expensive and time-consuming, which restricts its direct use in RRT [33].

In this chapter, the motion planning problem is formulated as an optimal control problem and various approaches to generate the distance function and the steering function values using optimal control are discussed.

4-1 Kinodynamic planning as optimal control problem

Consider a general dynamical system defined as a function f as in 4-1

$$\dot{x} = f(x, u, t) \tag{4-1}$$

where x is the state of the system, u the control input to the system and t is the time.

Optimal control problem aims to find the optimal control input u^* which will take the system from initial state x_i to final state x_f while optimizing the performance index (also known as cost functional) J. J is generally of the form as in 4-2 [34]. Note that this is an open-loop control system.

$$J = S(x(t_f), t_f) + \int_{t_0}^{t_f} V(x, u, t)dt \tag{4-2}$$

S is called the Meyer term and minimizes the terminal cost of the system (4-3). F is a positive semi-definite matrix.

$$S = x^T(t_f)Fx(t_f) \tag{4-3}$$

V is called the Lagrange term. It minimizes the energy of the system and the tracking error (4-4). Q and R are time varying matrices. They are positive-semi definite and positive definite respectively.

$$V = x^T(t)Qx(t) + u^T(t)Ru(t) \tag{4-4}$$

Constraints on the state or the control input are defined in terms of inequalities. For instance, if an actuator is limited to ± 400 Nm, it represented as in 4-5.

$$-400 \leq u \leq 400 \tag{4-5}$$

Recall that sampling-based kinodynamic planning problem involves finding a trajectory between two states of a robot subject to given constraints. When optimal control is applied to this problem, the robot's equations of motion form the dynamical system equations (Eq.4-1). Suppose an appropriate cost functional is chosen, such as minimization of system energy. Then the given initial and final states of the robot become the boundary conditions and the solution to this problem is the required input to traverse between the two states. It can be immediately seen how optimal control helps with the distance function and steering function generation. The cost functional represents the distance metric between the given initial and final states. The solution to the optimal control problem, the optimal control $u*$, is the required steering function value.

Two standard approaches to solving the optimal control problem are referred to as *direct* and *indirect* optimal control. *Indirect optimal control* uses the control Hamiltonian to get the first-order optimality conditions. The problem is first optimized analytically and then the solution is discretized. *Direct optimal control* approximates the state and control using function approximators and reformulates the problem as a non-linear programming problem. Here, the problem is first discretized, then solved. Both these approaches are discussed in the following sections.

4-2 Indirect optimal control

Indirect optimal control uses a method commonly known as *Pontryagin principle.* The principle states that the control Hamiltonian of the given system must be minimum at optimal control [34]. The principle makes use of calculus of variations and Lagrange multipliers to transform the problem into a two-point boundary value problem.

4-2-1 Pontryagin principle

Consider the dynamical system 4-1 with a general cost functional 4-2. Let the boundary conditions be 4-6

$$x(t_0) = x_0 \text{ and } x(t_f) = x_f \tag{4-6}$$

Then the steps to find the optimal control for the system using Pontryagin principle is as follows [29]:

1. Formulate the control Hamiltonian (also known as the Pontryagin H function)

$$\begin{aligned} H &= V(x,u,t) + \lambda^T \dot{x} \\ &= V + \lambda^T f \end{aligned} \tag{4-7}$$

   Here λ represents the Lagrange multipliers (also known as *co-state*) of the system.

2. Minimize H with respect to u to get u^*

$$\begin{aligned} \left(\frac{\delta H}{\delta u}\right)_* &= 0 \\ u^* &= \phi(x^*, \lambda^*, t) \end{aligned} \tag{4-8}$$

3. Substitute u^* in 4-7 to obtain optimal Hamiltonian H^*

$$H^* = H(x^*, u^*, \lambda^*, t) \tag{4-9}$$

4. Generate differential equations for x^* and λ^* from H^* and solve them using the boundary conditions 4-6

$$\dot{x^*} = \left(\frac{\delta H}{\delta \lambda}\right)_* \tag{4-10}$$

$$\dot{\lambda^*} = -\left(\frac{\delta H}{\delta x}\right)_* \tag{4-11}$$

5. Substitute solutions obtained in step 4 into u^* to get the optimal control

4-2-2 Pontryagin principle with constraints

The Pontryagin principle becomes much more complex in presence of state and control constraints. Consider the case of control constraints such as 4-12:

$$U_- \leq u \leq U_+ \tag{4-12}$$

where U_- and U_+ represent lower and upper bounds on control input. With this control constraint, equation 4-8 is no longer valid because it might not satisfy the constraint. The optimal u^* should then satisfy 4-13:

$$H(x^*, u^*, \lambda^*, t) \leq H(X^*, u, \lambda^*, t) \quad \forall\ u \in [U_-, U_+] \tag{4-13}$$

This equation is known as *Pontryagin's minimum principle* and is a necessary condition (but not sufficient) for optimality [34]. To satisfy equation 4-13, the control u needs to examined at each area the constraint is violated, resulting in a piece-wise function of u. The function complexity increases exponentially with state dimensionality due to the large number of possible conflicting regions. This is one of the drawbacks of indirect optimal control.

Consider next the case of state constraints as in 4-14, :

$$x_- \leq x \leq x_+ \tag{4-14}$$

where x_- and x_+ are bounds on the state. State constraints are transformed into equality constraints using penalty functions or slack variables [34]. Then the constrained problem is transformed into unconstrained problem and solved using the standard Pontryagin method.

$$\begin{aligned} \text{Constraint} &: \; g(x,t) \geq 0 \\ \text{Penalty func.} &: \; g(x,t)h(g) = 0 \\ \text{Slack var.} &: \; g(x,t) + \frac{1}{2}\alpha^2(t) = 0 \end{aligned}$$

where $h(g)$ is a penalty function and α is a slack variable

4-2-3 Disadvantages of indirect optimal control

Pontryagin principle provides an elegant method to solve the optimal control problem, but it suffers from the following disadvantages:

- The approach requires solving the differential equations which are strongly non-linear and unstable [34].
- Constraints result in state-dependent switches (as mentioned under control constraints)
- Requires explicit expression for $u*$.

4-3 Direct optimal control

Direct optimal control is a numerical optimal control approach that was developed over the last 40 years to overcome the issues with indirect optimal control and take advantage of processing power of computers. In direct optimal control, the control and the states are discretized to form a non-linear programming problem (NLP). NLP is easier to solve than two-point boundary value problems using efficient iterative algorithms like sequential-quadratic programming (SQP)[56]. Direct optimal control is more stable than indirect optimal over long time steps.

4-3-1 Discretization

Control input and states are discretized to represent them as lower order splines or polynomials. Standard procedure is to split the time interval into k subintervals and fit a polynomial in each interval. Continuity and derivatives at each interval are forced using equality constraints. Lagrange polynomial L (4-15) is commonly used for the fitting [57].

$$L(x) = \sum_{j=0}^{k} y_j l_j(x) \quad \text{for k datapoints } (x_i, y_i) \tag{4-15}$$

$$l_j(x) = \prod_{m=0}^{k} \frac{x - x_m}{x_j - x_m}, \quad m \neq j$$

Coefficients of the polynomials a_k are tuned during the optimization process. Given the initial conditions and the polynomial coefficients, the control and state can be uniquely determined. Solving the equations is not necessary at every step [54],[55].

The control input is often discretized as a piece-wise constant for simplicity, but polynomial function are also commonly used [5]. Let the control input be discretized as in 4-16.

$$u(t) = u_k(a_k, t) \tag{4-16}$$

where a_k is polynomial coefficients at kth subinterval and t is the time. But based on how the states are discretized, direct optimal control is of three main variants:

1. Single-shooting (direct sequential approach)
2. Multiple-shooting
3. Direct collocation (direct simultaneous approach)

The next subsections will briefly discuss each of the approaches.

4-3-2 Single-shooting

In the single-shooting approach, only the control input is discretized (4-16). Then the differential equations are integrated using the control input to obtain the final state $\hat{x}_f$ (Fig. 4-1) [55],[56]. Continuity is enforced at subintervals using equality constraints. Adaptive ODE solver is commonly used for this process [58].

The error ϵ between the simulated final state $\hat{x}_f$ and actual final state x_f is calculated and ϵ is minimized iteratively using SQP. Cost function can evaluated after each iteration using the state and the control input.

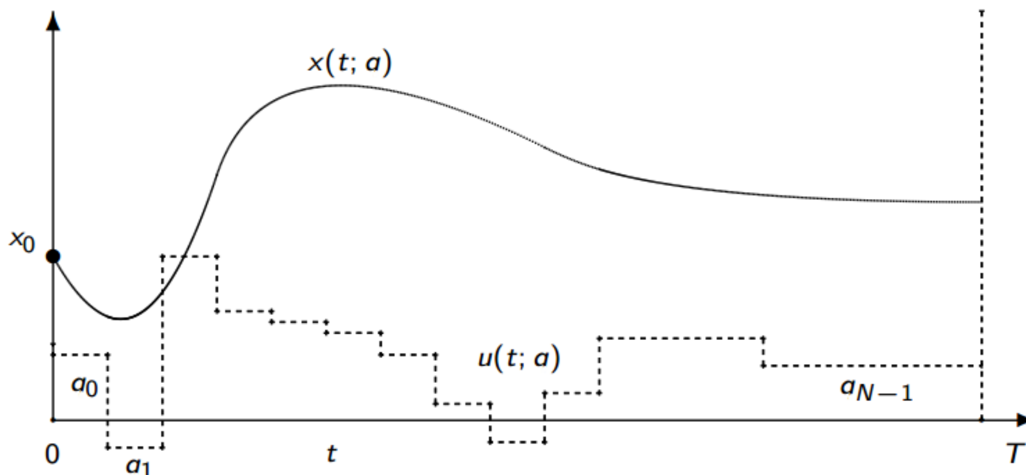


Figure 4-1: Illustration of single shooting method [5]. Dotted line represents the N-part discretized input u parametrised by a_k . Normal line represent the state x generated using system equations.

The process is as follows:

1. Given initial state x_0 and final state x_f, guess control coefficients a_k
2. Integrate the differential equations till $T = t_f$ and estimate the error ϵ at boundaries

$$\epsilon = \hat{x}_f - x_f$$

3. Adjust a_k using SQP to minimze ϵ. Repeat until $\epsilon \approx 0$

The single shooting approach is advantageous because it has lower dimensionality compared to the other two approaches. But the intermediate ODE solution (step 2) for x can depend non-linearly with a_k, which can cause the initial-value problem to become unstable [58]. Small change in initial guess can produce large differences at boundaries.

4-3-3 Multiple-shooting

Multiple-shooting method also discretises only the control input. However, the states are propagated within every subinterval and not over the entire time period. Also, continuity between intervals is not enforced (Fig. 4-2). The error ϵ_k at each interval is minimized iteratively using SQP instead of only at the boundary point.

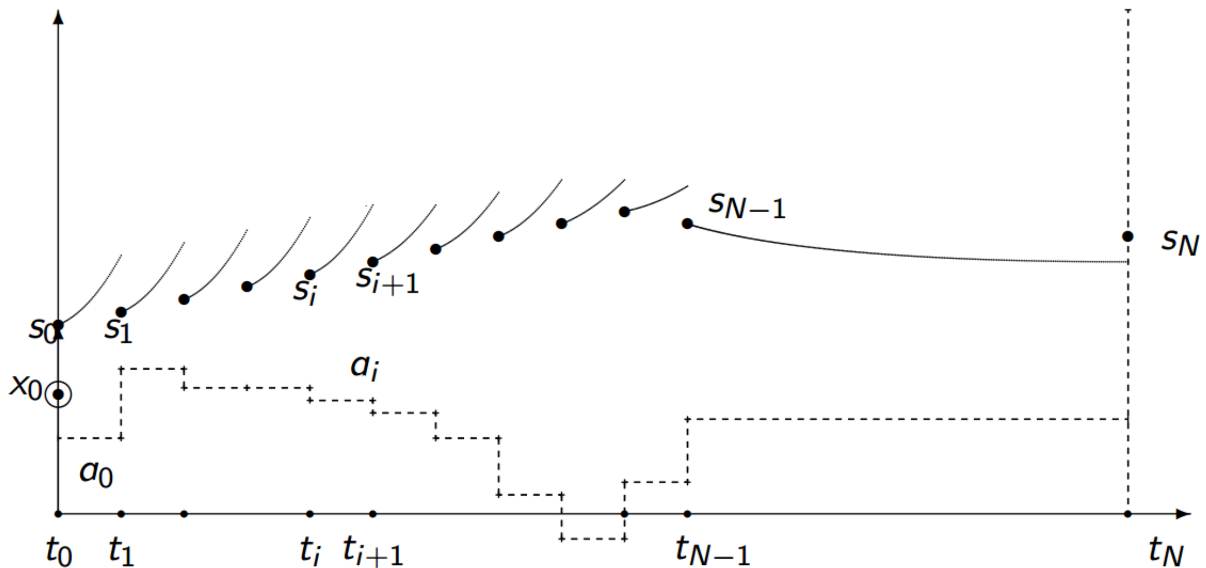


Figure 4-2: Illustration of multiple-shooting method [5]. Dotted line represents the discretised input u and the normal line the state x. s_i shows point of each shooting. Note the discontinuities at each interval.

The method works as follows-

1. Given initial state x_0 and final state x_f, guess initial states at each interval s_k and control a_k at each interval

2. Integrate the differential equations over each interval to get x_k and estimate error at each interval ϵ_k
$$\epsilon_k = x_k - s_k$$
3. Adjust a_k using SQP to minimize each ϵ_k until each $\epsilon_k \approx 0$

Unlike single-shooting, since the integration occurs at each interval, non-linearities are not propagated and it can handle unstable systems well [5]. Its structure allows it to be parallelized easily. But multiple-shooting has much higher dimensionality compared to single-shooting.

4-3-4 Direct collocation

Direct collocation is a bit different from the previous two approaches. It does not involve integration of the differential equations but instead tries to match the derivative of the approximating function with the state derivative at every interval.

In direct collocation method, control input, the state as well as the cost functional are discretized [55,57]. The control input is discretized as in other approaches and the state is approximated in each interval using polynomials. The midpoint of each interval is termed as *collocation point* [59]. Next, state polynomials and their derivatives are equated to the state and state derivative in their respective intervals. This gives the coefficients of the state polynomials over each interval as a function of interval length h, state at each interval x_k and control u_k [59]. The state and state derivative at each collocation point can be calculated using this. The error at each collocation point is calculated using 4-17, which is also known as defect constraint [59]:
$$\epsilon_c = \dot{x_c} - f(x_c, u_c) \tag{4-17}$$
The cost functional is discretized as J_k using numerical integration approaches such as the trapezoidal method [60]. Then a_k (coefficients of u_k) and b_k (coefficients of x_k) are iteratively tuned using SQP to minimize J_k with $\epsilon_c = 0$ as an equality constraint at each interval.

Collocation method is very large dimensionality approach due to additional constraints from state and cost discretization. But it handles unstable systems very well and solves for all collocation points simultaneously unlike single and multiple shooting.

4-3-5 Constrained direct optimal control

Input constraints are easier to handle in direct optimal control compared to indirect optimal control. Input constraints become additional constraints to the non-linear programming problem. If polynomial approximations are used, k inequality constraints are added to the problem. They can be handled using penalty functions or barrier functions [58]. If the input is discretized as a piece-wise constant function, the constraint simply becomes
$$U_- \leq a_k \leq U_+$$
State constraints can be similarly handled in direct collocation method by simply enforcing the constraints at collocation points. In single and multiple shooting this is not possible as the state is not discretized but instead simulated using control input. In this case, the constraints need to be reformulated as integral constraints or discretized as interior point constraints [59].

4-4 Data generation for learning-based RRT

The previous two sections illustrate various methods for solving the optimal control problem. As discussed in section 4-1, solving the kinodynamic problem using optimal control gives us the required distance metric and steering function for learning-based RRT.

Problems with indirect optimal (section 4-2) and the robustness of direct optimal control make direct optimal approach an obvious choice for data generation . But indirect optimal control has two features that make its use viable [2]. Firstly, indirect optimal control uses fewer parameters compared to direct optimal control. This is important in learning as dimensionality of parameters can affect learning (Chapter 3). Lower parameter dimensionality also means faster and efficient training. Another feature of indirect optimal control is that data can be generated just by sampling initial state x_i, initial co-state λ_i and final time t_f. No optimization is required. This is because every combination of x_i, λ_i and t_f returns a viable final state x_f, thus effectively sampling over the entire search space [2]. Quantifying the influence of these features and how they affect learning-based RRT is one of this thesis' contributions. Both indirect optimal control and direct optimal control methods are used to generate data for a test case and influences of each are studied.

It is also seen in section 4-2-2 and 4-3-5 that optimal control gets more complex with the addition of input and state constraints. Input constraints are very common in most systems due to actuator limits. The influence of input constraints on data generation and learning are also investigated using the test case in this thesis. State constraints are not considered. These implementations and experiments are discussed in next chapter.

4-5 Summary

This chapter details the use of optimal control theory for generating data for learning-based RRT. Kinodynamic planning problem is formulated as an optimal control control and solution to the optimal control returns the steering and distance metric. Two approaches to solving the optimal control problem - direct and indirect - are introduced and each approach is explained briefly. In the next chapter, implementation of these approaches for learning-based RRT will be discussed.

Chapter 5

Experimental setup

The previous chapters detailed the theory behind learning-based RRT and how it is used to make state space RRT faster. Data generation for learning using optimal control theory is also explained. This chapter describes the implementation of these concepts on a simple dynamical system. This test system is then used to investigate the properties of learning-based RRT and achieve the contributions of this thesis mentioned in chapter 1, namely

- Comparison of direct and indirect optimal for training set generation
- Study the effect of input constraints on learning

The experimental approach followed is as below-

1. Generation of equations of motion of test system
2. Unconstrained and constrained training data generation using optimal control. ACADO optimal control software [61] is used for direct optimal control whereas indirect optimal control is coded in MATLAB
3. Implementing K-nearest neighbours and feed-forward neural networks for supervised learning using Sci-kit python library.

5-1 Test system

Earlier work on learning- based RRT ([2], [26]) used an inverted pendulum system for demonstration of the algorithm. In this thesis, a slightly more complex dynamical system, an open-chain *2-link manipulator* with both the links in the same plane (Fig. 5-1) is chosen as the test system.

A 2-link manipulator is a non-linear system. It consists of 2 uniform rectangular links of masses m_1 and m_2 and lengths l_1 and l_2. They are connected to each other by a revolute

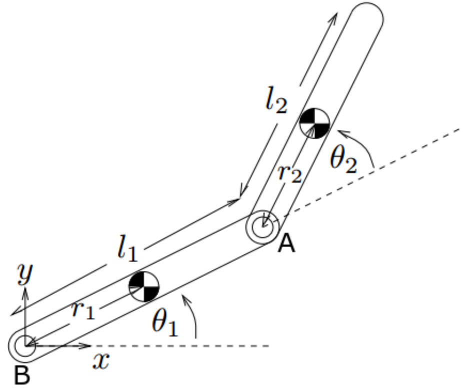


Figure 5-1: Test system - 2-link manipulator [6]. θ_1, θ_2 are angular positions of the two joints.

joint at point A. The other end of the first link is connected to ground by another revolute joint B. Other end of the second link is free. Both the joints are assumed to rotate about an axis perpendicular to axes $x - y$ and are attached to actuators which control the joint torques T_1 and T_2. Thus the system has 2 degrees of freedom. The manipulator is assumed to be in gravity-free environment.

In this thesis, the links are assumed to have negligible thickness with centers of mass at the centre of each link. Other properties are assumed as in 5-1.

$$\begin{aligned} m_1 = m_2 &= 1\ kg \\ l_1 = l_2 &= 1\ m \\ r_1 = r_2 &= 0.5\ m \end{aligned} \tag{5-1}$$

where r_i is the distance to center of mass of each link from one end.

5-2 Generation of equations of motion

Motion of a system is described in terms of its position, velocity and acceleration with respect to a chosen reference frame. Inertial properties of a system are its mass, moment of inertia, surface area, coefficients of friction etc. Equations of motion are Newton's laws applied a dynamic system. Equations of motion of a system are second order ordinary differential equations that relate the system's motion to the forces acting on it in terms of its inertial proprieties [6].

Equations of motion are known as *forward dynamics* equations if they describe the system motion using the forces acting on it. The opposite i.e solving for forces using the motion, is termed as *inverse dynamics* equations [62]. Forward dynamics equations are required for learning-based RRT as they form the system equations in the optimal control formulation (4-1). They are also used to simulate the system in the steering step of RRT.

Forward dynamics equations of motion for open-chain multi-link manipulators as in the test case can be found using three different methods. All the methods generate equivalent equations but in different forms.

Newton-Euler method In Newton-Euler method, equations for translational acceleration (*Newton equations*) and equations for rotational acceleration (*Euler equations*) are formulated for each link in the system (5-2). Constraint equations between each link are also formulated. Constraints equations are equations that relate the common points between two connected bodies [62]. All these combined equations are linear with respect to generalized coordinates q_i and therefore can be solved using standard algorithms like Cramer's rule to obtain the equations of motion.

$$F_i = m_i \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} a_i \qquad (5\text{-}2)$$
$$T_i = I_i \ddot{\theta} + \dot{\theta}_i \times I_i \dot{\theta}_i$$

where F_i is the force acting on the centre of mass of ith link, T_i is torque about its centre of mass, m_i is its mass, I_i is the inertia matrix, a_i is the acceleration of its centre of mass, $\ddot{\theta}$ is its angular acceleration and $\dot{\theta}$ is its angular velocity.

Despite the simplicity of Newton-Euler method, formulating the equations becomes tedious in complex systems as number of equations become very high. For n links, $6n$ Newton-Euler equations are obtained with 3 constraint equations per joint. This method has a polynomial time complexity and is thus computationally expensive. Therefore it is mainly used for simple systems with upto 10 links [62].

Lagrangian method Lagrangian method uses Lagrangian of a system and generalized coordinates to generate the equations of motion. The Lagrangian L of a system is the difference between its kinetic and potential energy. Equations of motion are then derived from L using 5-3 [6].

$$\frac{d}{dt}\left(\frac{\delta L}{\delta q_i}\right) - \frac{L}{q_i} = F_i \qquad (5\text{-}3)$$

where L is the Lagrangian, q_i are the generalized coordinates and F_i is the force on ith generalized coordinate. Number of equations obtained using Lagrangian method is equal to number of degrees of freedom. Lagrangian method is an elegant way to generate equations of motion but requires an explicit expression of the Lagrangian which might not always be feasible. Formulating the Lagrangian is also tedious for complex systems.

Featherstone's articulated body algorithm Featherstone's articulated body algorithm is an efficient numerical algorithm to solve forward dynamics equations for open-chain multi-body systems. The algorithm is structurally recursive and has a linear time-complexity [7]. The algorithm uses *spatial coordinates*, which are coupled translational and rotational components of a body (5-4).

$$\text{Spatial velocity } \hat{v} = \begin{bmatrix} \dot{\theta} \\ v \end{bmatrix} \quad \text{Spatial acceleration } \hat{a} = \begin{bmatrix} \ddot{\theta} \\ a \end{bmatrix} \qquad (5\text{-}4)$$
$$\text{Spatial force } \hat{f} = \begin{bmatrix} F \\ T \end{bmatrix} \quad \text{Spatial inertia } \hat{I} = \begin{bmatrix} 0 & M \\ I & 0 \end{bmatrix}$$

Articulated body algorithm divides an n-link manipulator into n subchains called *articulated bodies*. Each subchain consists of j to n links in isolation from the base link where $1 \leq j \leq n$ (Fig. 5-2). Link j is called the *handle* of the jth subchain [7].

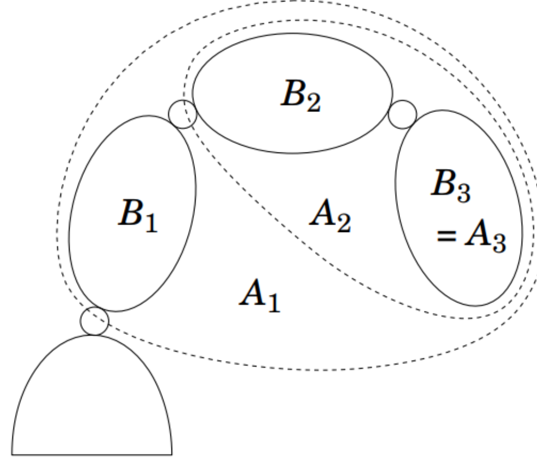


Figure 5-2: Illustration of articulated bodies [7]. B_j are the links and A_j represent the articulated bodies.

The algorithm considers each articulated body one after another and relates spatial acceleration of each handle to the spatial force applied at the joint [63]. The relation is given by equation 5-5.

$$\hat{f}_j = \hat{I}_j^A \hat{a}_j + \hat{Z}_j^A \tag{5-5}$$

where $\hat{f}_j$ is the spatial force on jth link, $\hat{I}_j^A$ is its articulated inertia, $\hat{a}_j$ is spatial acceleration and $\hat{Z}_j^A$ is the articulated static force of jth link. Articulated inertia $\hat{I}_j^A$ is the inertia of the jth articulated body and articulated static force $\hat{Z}_j^A$ is the force needed at the inboard joint of the jth articulated body to keep it from accelerating [63].

$\hat{I}_j^A$ depends only on spatial inertia $\hat{I}_j$ of the jth link and articulated inertia $\hat{I}_{j+1}^A$ of $(j+1)$th subchain. Also, $\hat{Z}_j^A$ is depends only upon $\hat{Z}_{j+1}^A$(articulated static force of $(j+1)$th link), $\hat{I}_{j+1}^A$ and $\hat{f}_{j+1}$ (force acting upon $(j+1)$th link). Therefore $\hat{I}_j^A$ and $\hat{Z}_j^A$ can be recursively computed for all articulated bodies starting from the nth articulated body [7].

Articulated body algorithm thus comprises of 3 steps-

- *First step*: Spatial velocities $\hat{v}_j$, isolated static forces $\hat{Z}_j$ and isolated spatial inertia $\hat{I}_j$ of all links are calculated
- *Second step*: Articulated inertia $\hat{I}_j^A$ and articulated static force $\hat{Z}_j^A$ on each link is calculated recursively starting from the nth articulated body using Eq.5-5.
- *Third step*: Spatial acceleration of each link and joint accelerations are computed using $\hat{f}_j$.

Despite the complexity of articulated body algorithm compared to the other two methods, it is very fast and suitable for use in dynamics software [7].

In learning-based RRT, an explicit expression of equations of motion in terms of generalized coordinates is required as it is used in indirect optimal control and RRT. Therefore Lagrangian method is generally used for the generation. However, as previously mentioned, formulating the Lagrangian becomes cumbersome for complex systems. Articulated body algorithm was designed for complex systems [7], but it is a numerical method. To solve this problem, articulated body algorithm was adapted to generate explicit symbolic equations of motion over the course of this thesis [1] [64]. The adaptation was written in MATLAB using its symbolic toolbox.

Using this implementation of articulated body algorithm, equations of motion for the test case are obtained (with the values in 5-1 substituted) as in 5-6. This was verified using the Lagrangian method. It can be observed that the equations are highly non-linear.

$$\ddot{\theta}_1 = \frac{-(48T_1 - 48T_2 + 24\dot{\theta}_1\dot{\theta}_1 sin(\theta_2) + 24\dot{\theta}_2\dot{\theta}_2 sin(\theta_2) + 18\dot{\theta}_1\dot{\theta}_1 sin(2\theta_2) - 72T_2 cos(\theta_2) + 48\dot{\theta}_1\dot{\theta}_2 sin(\theta_2))}{36cos(\theta_2)cos(q\theta_2) - 64} \tag{5-6}$$

$$\ddot{\theta}_2 = \frac{48T_1 - 240T_2 + 120\dot{\theta}_1\dot{\theta}_1 sin(q2) + 24\dot{\theta}_2\dot{\theta}_2 sin(\theta_2) + 36\dot{\theta}_1\dot{\theta}_1 sin(2\theta_2) + 18\dot{\theta}_2\dot{\theta}_2 sin(2\theta_2) + 72T_1 cos(\theta_2) - 144T_2 cos(\theta_2) + 48\dot{\theta}_1\dot{\theta}_2 sin(\theta_2) + 36\dot{\theta}_1\dot{\theta}_2 sin(2\theta_2)}{18cos(2\theta_2) - 46}$$

where T_1, T_2 are the control inputs, θ_1, θ_2 are angular positions of the two joints, $\dot{\theta}_1, \dot{\theta}_2$ are joint angular velocities and $\ddot{\theta}_1, \ddot{\theta}_2$ are the joint angular accelerations.

5-3 Generation of training data

The above generated equations of motion can now be used to generate training data for learning-based RRT using optimal control principles as discussed in previous chapter. The optimal control problem is now formulated for kinodynamic planning of the test case (Section 5-1) as described in Chapter 4. It is observed from the equations of motion that the system has two control inputs T_1, T_2 and its state x is defined by 2 generalized coordinates and their derivatives (5-7).

$$\begin{aligned} u &= [T_1, T_2] \\ x &= [\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2] \end{aligned} \tag{5-7}$$

Initial and goal states x_i and x_f are chosen randomly from pre-determined range of angular position and angular velocity for every sample. The ranges are taken as in 5-8.

$$\begin{aligned} \theta_1, \theta_2 &\in [0, 2\pi] \text{ radians} \\ \dot{\theta}_1, \dot{\theta}_2 &\in [-30, 30] \text{ rad/s} \end{aligned} \tag{5-8}$$

[1] Code available at https://github.com/DeepakParamkusam/urdf2eom. See Appendix A

The system equation for optimal control can then be derived from 5-7 and 5-6 as in 5-9.

$$\dot{x} = \begin{bmatrix} \dot{\theta_1} \\ \ddot{\theta_1} \\ \dot{\theta_2} \\ \ddot{\theta_2} \end{bmatrix} \tag{5-9}$$

This forms the constraint equations. Next, a cost functional J needs to be chosen for the optimal control problem. Minimization of system energy is a commonly chosen performance index for dynamic systems and the same is chosen in this thesis[26]. This is formulated using the Lagrange term as

$$J = \int_{t_0}^{t_f} u^T(t)Ru(t)dt$$

where R is positive definite matrix chosen to be 1. Then J becomes (5-10).

$$J = \int_{t_0}^{t_f} T_1^2 + T_2^2 dt \tag{5-10}$$

In the constrained case, the actuators attached to the joints are assumed to have torque limits equal to ± 400 Nm.

$$\begin{aligned} -400 \leq T_1 \leq 400 \\ -400 \leq T_2 \leq 400 \end{aligned} \tag{5-11}$$

This completes the formulation of kinodynamic planning as optimal control problem for the test case. Next the implementation of indirect and direct optimal control methods to the test problem with and without input constraints are discussed.

5-3-1 Implementation of direct optimal control

Of the methods detailed in Section. 4-3, multiple shooting is used for direct optimal control in this thesis. Multiple shooting is chosen over single shooting because non-linearities are not propagated in multiple shooting and thus errors are localized. Although direct collocation is more robust, it tends to be less accurate than multiple shooting due to its lower order.

Multiple shooting is implemented in this thesis using *Automatic Control and Dynamic Optimization (ACADO) toolkit* software. ACADO toolkit is an optimal control and optimization software written in C++ at KU Leuven [61]. It contains frameworks for wide range of direct optimal control algorithms such as single and multiple shooting, model predictive control and multi-objective optimization. ACADO was chosen over other common optimal control packages like IPOPT, PROPOT and MUCSOD [64],[65] due to the following reasons-

- ACADO is user-friendly and supports formulation of the optimal control problem using symbolic expressions [61]. This allows direct substitution of Eq. 5-6 into the source code without any pre-processing.

- ACADO is self contained. It has NLP solver, Runge-Kutta integrator, function approximators and major optimal control algorithms already implemented in the framework. No external tools are necessary.
- ACADO is open-source. It is free for use by academics and can be modified as desired.

Optimal control problem formulated previously is coded in ACADO (see Appendix B). The input is discretized as a piece-wise constant function over 20 intervals over 0.5s time period and fourth order Runge-Kutta integrator is used for simulation with 10000 integrator steps. Other parameters for multiple shooting are chosen are given in 5-12. x_i and x_f are randomly sampled from the chosen range (5-8). Cost of each solution and control input are returned by ACADO as solutions to optimal control problem. In constrained case, the input constraints are added to ACADO using simple inequalities. No other changes to the code are required. 100 thousand samples are generated for the unconstrained case and the constrained case.

$$
\begin{aligned}
t_f &= 0.5s \qquad (5\text{-}12)\\
\text{Tolerance} &= 10^{-4}\\
\text{No. of samples} &= 100000 \qquad (5\text{-}13)
\end{aligned}
$$

5-3-2 Implementation of indirect optimal control

Indirect optimal control approach follows the Pontryagin's principle described in Section. 4-2 and is programmed in MATLAB and Python. However, Runge-Kutta integration is coded in Python (Appendix B).

The expressions for $\dot{x}^*$ and the corresponding co-states $\dot{\lambda}_x^*$ are generated in MATLAB in the same way as Eq. 4-6.

$$
\begin{aligned}
\dot{x}^* &= \frac{d}{dt}[\theta_1^*, \theta_2^*, \dot{\theta}_1^*, \dot{\theta}_2^*] \qquad (5\text{-}14)\\
\dot{\lambda}_x^* &= \frac{d}{dt}[\lambda_{\theta_1}^*, \lambda_{\theta_2}^*, \lambda_{\dot{\theta}_1}^*, \lambda_{\dot{\theta}_2}^*]\\
u^* &= \phi(x^*, \lambda^*, t)
\end{aligned}
$$

Here, the advantage of indirect optimal control for data generation demonstrated in [2] comes into play. Instead of solving the above boundary value problem, authors of [2] assert that every combination of intial state x_i, initial co-state λ_i and final time t_f results in an optimal trajectory to corresponding x_f by integrating Eq. 5-14. Therefore, if t_f is fixed (it is chosen to be 0.5s), only x_i and λ_i are required to generate training data. This approach also shows that u^* need not be generated for learning. Instead, the co-states themselves can be learnt as they are the ones being used to generate x_f [2]. Learning co-states also reduces the number of parameters to be learnt to 4 (number of co-states). Low size of training data, as seen in Chapter 3, helps supervised learning. Cost (distance metric) can also written in terms of the co-state and generated at the same time. This makes data generation much faster compared to the direct optimal approach as the boundary value problem need not be solved. A dataset of 100 thousand samples are generated using this approach too.

Input constrained indirect optimal control is infeasible for learning-based RRT because of two reasons-

- u^* switches to the input bounds at certain points in the trajectory due to the constraints as mentioned in Chapter 3. It is hard to find all the switching points for a non-linear system [29].
- When u^* switches, the co-states change as well at the switching point (Fig. 5-3). That mean two sets of λ^* exist for certain trajectories. When applied to learning-based RRT, both the sets of co-states need to be learnt along with the switching points and states at switching points. Also, trajectories where this happens should also be predicted.

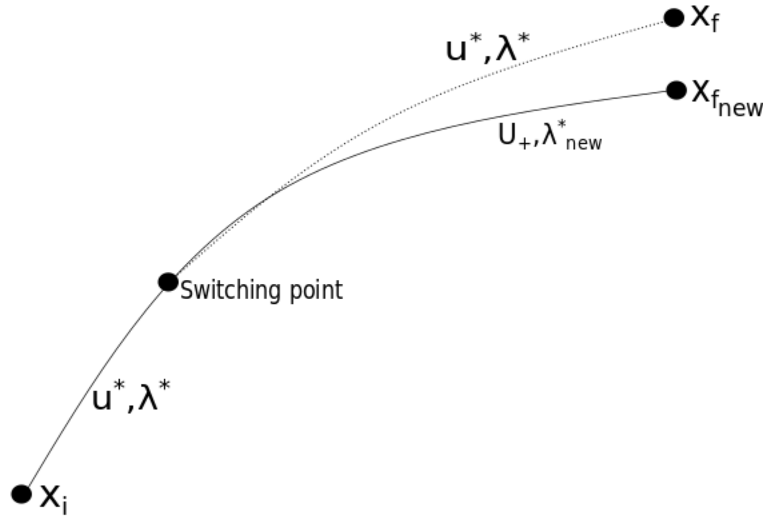


Figure 5-3: Illustration of trajectory change due to u^* switching to U_+ under input constraint

An alternate approach for constrained indirect optimal control is attempted in this thesis. u^* is formulated as in Eq. 4-8 assuming there are no input constraints. Once a random x_i and and its co-state λ_i are chosen, u^* is calculated during every step of the integration alongside x_f. If u^* is observed to exceed the input constraints at any point, that sample is discarded. Only samples which always satisfy the input constraints are stored. Thus constrained data is geenrated using indirect optimal control.

5-4 Implementation of supervised learning

Cost generated during the optimal control process is the distance metric and is a scalar continuous quantity. Therefore, 100000 sample datasets for cost and control input generated in the previous section are used for training learning algorithms. Separate models are generated for each. The inputs to the learning algorithms are the initial and final states x_i and x_f. Steering input is an array and has different sizes in direct and indirect optimal control. Direct optimal control returns a 20 element array for each steering input (as input is discretized into 20 time intervals) while indirect optimal control returns a 4 element array (equal to number of co-states). Each element of the array is continuous as well. Since all outputs are continuous, the learning becomes a regression problem.

K-nearest neighbours (KNN) and feed-forward neural networks described in Chapter 3 are the two algorithms are used for learning-based RRT in this thesis. They are implemented using

scikit-learn, a popular Python machine learning library [66]. Scikit-learn includes efficient implementations of many supervised learning algorithms. Scikit-learn is user-friendly and distributed under BSD license. The code for the implementations are provided in Appendix B.

5-4-1 Data pre-processing

Learning algorithms are sensitive to biases inherent in the data and to large data variations [43]. Therefore, before training the networks, the data needs to be pre-processed. First, the 100000 samples are divided into 2 datasets in the ratio 9:1. The bigger 90000 sample data set is used for training (using k-fold cross validation) while the smaller 100000 sample set is used for testing the learning.

Common pre-processing techniques are standardization and scaling. Standardization normalises the data and removes biases. Standardization is performed using 5-15 where σ and μ are the data set's mean and standard deviation respectively.

$$x_{i,std} = \frac{x_i - \sigma}{\mu} \tag{5-15}$$

Scaling transforms the data to a different range (usually to [0,1]). Scaling is performed using 5-16. $min(x)$ and $max(x)$ are the minimum and maximum values of the data set. It is recommended to scale data for neural networks for faster convergence [44].

$$x_{i,scaled} = \frac{x_i - min(x)}{max(x) - min(x)} \tag{5-16}$$

Dataset cleaning is another pre-processing step. When multiple samples with similar inputs have very different outputs, learning algorithms tend to take the mean of the outputs during prediction. But taking the mean can result in incorrect predictions and is undesirable [2]. This is more prevalent in non-parametric algorithms like KNN than in parametric algorithms. Therefore, only one of the similar samples is kept and the others are discarded. This is known as dataset cleaning. Similar samples are identified using euclidean metric in this thesis and only the sample with least cost is kept.

5-4-2 Implementation of KNN and feed-forwards neural network

Scikit-learn provides a very efficient implementation of KNN and the same is used in this thesis. Euclidean distance is used for finding the nearest neighbours with uniform weights for all neighbours. Number of nearest neighbours taken under consideration (the value of k) is chosen by computing errors for a range of values of k. k-range is chosen from the range 2 to 30.

Feed-forward neural network is also implemented using scikit-learn. Since the test case' state has 4 elements, 4 input perceptrons are used. Similarly, 40 output perceptrons are used for the output layer for direct optimal control data and 4 output perceptrons for indirect optimal control data. Single hidden layer is used and number of perceptrons in it are varied.

Rectified linear unit (ReLu) threshold function (Eq. 3-8) is used for all perceptrons with square error as the loss function. ADAM solver is used for back-propagation due to its better performance compared to stochastic gradient descent (Sec. 3-3-1). Other parameters chosen for feed-forward network are below-

$$\begin{aligned} \text{Batch size} &= 200 \\ \text{Epochs} &= 100 \\ \text{Tolerance} &= 10^{-4} \\ \text{Learning rate } \alpha &= 10^{-3} \end{aligned} \tag{5-17}$$

Batch size is the number of samples that are used per iteration for training the network. Epochs are the number of times entire data set is used for training. For a 90000 sample training set, there are $\frac{90000}{200} = 450$ iterations per epoch. Training stops if the tolerance is reached or the epoch limit is reached.

5-5 Implementation of learning-based RRT

Learning-based RRT detailed in Chapter 2 was implemented completely in Python by authors of [2]. The same code is reused in this thesis with modifications as required for the chosen test system and the feed-forward neural network (Appendix B). The major parameters chosen of learning-based RRT are as follows-

$$\begin{aligned} \text{Max. nodes } &= 10000 \\ \text{Goal tolerance } &= 0.5 \\ \text{Max. state error} &= 0.5 \\ \text{Goal bias } &= 20\% \end{aligned}$$

5-6 Summary

This chapter details various ways to generate data and learn it for learning-based RRT. First equations of motion are generated using articulated body algorithm and they are used to generate training data using multiple shooting and Pontryagin's principle for unconstrained and constrained cases. The use of this data to train feed-forward neural network and KNN is also detailed. Comparison of the data generation and its effect on learning is analysed in the next chapter.

Chapter 6

Results and analysis

The previous chapter detailed the experiments performed in this thesis to investigate the influence of indirect and direct optimal control on the learning-based RRT. The results of those experiments are analysed in this chapter. Different metrics used for comparison of the optimal control approaches are first introduced and then the results are analysed based on the metrics. Constrained learning-based RRT is also compared using the same metrics.

6-1 Comparison metrics

The optimal control data and its corresponding learning is compared using *data generation time*, *quality metrics* and *learning metrics*. These metrics are evaluated for both indirect and direct optimal control training data using KNN and feed-forward neural network as the learning algorithms.

Data generation time Data generation time in this thesis refers to the time taken to generate the training data using indirect and direct optimal control approaches. Generation time is an important metric as it is the most time-consuming off-line step in learning-based RRT and entirely depends on the optimal control approaches. Sample generation time is average of the data generation time and gives a quantifiable estimate of the runtime of each optimal control approach per sample. Lower sample generation time is desired as it allows faster implementation of learning-based RRT.

Quality metrics Quality of data is a measure of how well the given data correctly represents the system it is trying to model [67]. Poor quality data can result in improper learning. Quality metrics used in this thesis are data completeness and data uniqueness. These metrics are explained in detail below -

1. *Data completeness*
   Data completeness is a a measure of how well the data covers the state space. Data

isolated to a certain part of the state space may not fully capture the system dynamics and can result in inaccurate or biased predictions [67]. Data completeness is evaluated by plotting the *phase plot* of the input. Phase plot of a dynamic system represents each state as a point. Gaps in these phase plots will indicate incompleteness of data as they represent states not present in the training set.

2. *Data uniqueness*
   Data uniqueness is measure of distinctiveness of each sample in the training data. As mentioned in the previous chapter, similar or close inputs with different outputs in the training data can confuse the learning algorithm. Data cleaning is used to remove the similar inputs and also used as a measure of uniqueness of data in this thesis [67]. Lower the number of similar data, better the uniqueness of the data. Euclidean distance is used to find similar samples and sample with least cost is kept among them. It should be noted that data cleaning should not affect the completeness of the data and euclidean distance should be chosen accordingly.

Learning metrics Learning metrics are commonly used for measuring the performance of learning algorithms [68]. But they are used in this thesis to compare the different training datasets with the learning algorithm kept constant. This helps evaluate which dataset is more easily learned. Learning metrics consist of average cross validation error and prediction profile.

1. *Average cross validation error*
   k-fold cross validation was introduced in chapter 3 to evaluate a learning algorithm. The same method is used to evaluate mean squared error of training in each fold. Lower values signify better learning. Average error over all the k folds gives the average cross validation error. Number of folds is chosen to be 10 in this thesis.

2. *Prediction profile*
   Prediction profile is another measure of the learning algorithm's performance. Accuracy of the predictions made from the trained models can be observed in the prediction profiles. Predictions are made using the test dataset previously kept aside. These test predictions are plotted against the actual values for both cost and control input. The $x = y$ line represents predictions which are same as actual values. Therefore, closer the points are to the $x = y$ line, better the learning.

6-2 Comparison of direct and indirect optimal control

First, the comparison is performed for the unconstrained input case. As discussed in the previous chapter, two datasets are generated for the test case - one using the multiple shooting method and one using Pontryagin's principle. 100000 sample dataset was generated using indirect approach and 500000 sample dataset using the direct approach. This is because control input generated using direct optimal approach has larger parameter size (20 per control input) compared to indirect approach (4 co-states). More the number of parameters to learn, larger the required dataset. These datasets are then used to train k-nearest neighbours (KNN) and feed-forward neural network. The trained models are then used in learning-based RRT. Comparison of these datasets is now performed based on the previously detailed metrics.

Data generation time The time taken to generate each data set is found to be as in Table. 6-1. It is observed that Pontryagin approach is more than 100 times faster than multiple shooting approach. This is mainly because multiple shooting (and all direct optimal control approaches) is an iterative method. The iterative process increases the data generation time. Modified Pontryagin approach used in this thesis (Section 5-3-2) also reduces generation time as it uses co-states to generate final state and cost in a single integration step.

Table 6-1: Data generation time without input constraints

Optimal control approach	Generation time for 100k samples (hours)	Average sample generation time (s)
Pontryagin (Indirect)	0.62 (1 thread)	0.022
Multiple shooting (Direct)	66.4 (8 threads)	2.39

Data completeness For the test case, it is not possible to directly visualize the phase plot since the input dimension is 8. Therefore, the phase space is split into multiple lower dimension plots. x_i and x_f in the input are mapped using 4-dimensional scatter plot with the 4th dimension represented using a color gradient. For x_i vs x_f, all the combinations of their elements are plotted using 2-dimensional scatter plots.

In the direct optimal control approach, both initial state x_i and final state x_f are randomly and uniformly sampled from a predefined state space (5-8).

$$\theta_1, \theta_2 \in [0, 2\pi] \text{ radians}$$
$$\dot{\theta}_1, \dot{\theta}_2 \in [-30, 30] \text{ rad/s}$$

However, 4-dimensional phase plot of x_i was observed to have some discolouration implying gaps in the phase plot while x_f had uniform coverage (Fig.6-3). The gaps in x_i can be seen more clearly in Fig.6-1 and Fig.6-2. x_i vs x_f was observed to have uniform coverage. One of x_i vs x_f plots is given in Fig.6-5. Gaps in x_i could be caused due to non-convergence of samples from those regions. Only successful optimizations which lead from chosen x_i to x_f are included in training set.

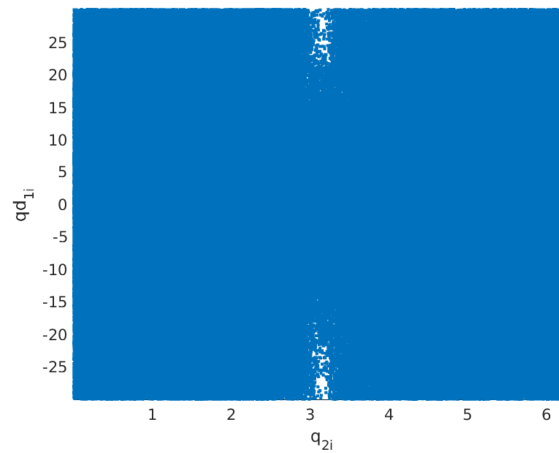


Figure 6-1: Gaps in phase plot of x_i : q_{2i} vs qd_{2i}

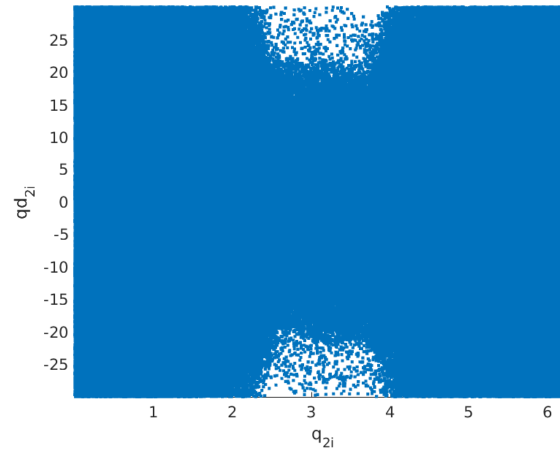


Figure 6-2: Gaps in phase plot of x_i : q_{2i} vs qd_{2i}

In indirect optimal control approach, x_i randomly sampled as in the direct approach and its phase plot is therefore observed to be uniformly distributed (Fig.6-4a). x_f is generated using x_i and co-state λ. Its phase plot is observed to also cover the entire state space and even slightly exceed it (Fig.6-4b). However, their x_i vs x_f phase plots shows insufficient coverage (Fig.6-6) due to correlation between x_i and x_f.

From the 4-dimensional phase plots of direct and indirect datasets, it is observed that both datasets are complete. Direct dataset has gaps in x_i while indirect dataset has gaps in x_i vs x_f. This incompleteness can affect the learning using these datasets.

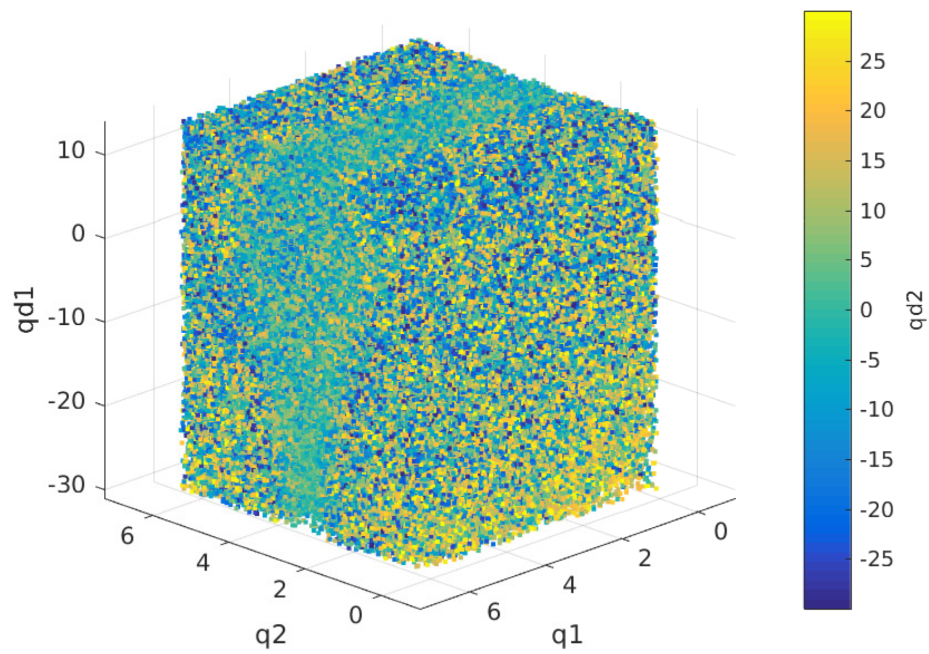
Data uniqueness Data cleaning (Sec. 5-4-1) was performed on each data set using Euclidean distances 3 and 4. Cleaning with Euclidean distance more than 4 was observed to result in loss of data. The number of similar samples in each case is shown in Table.6-9 and Table.6-10. Percentage of similar samples is observed to be higher in indirect approach compared to the direct approach.

Table 6-2: Cleaning of direct data (unconstrained)

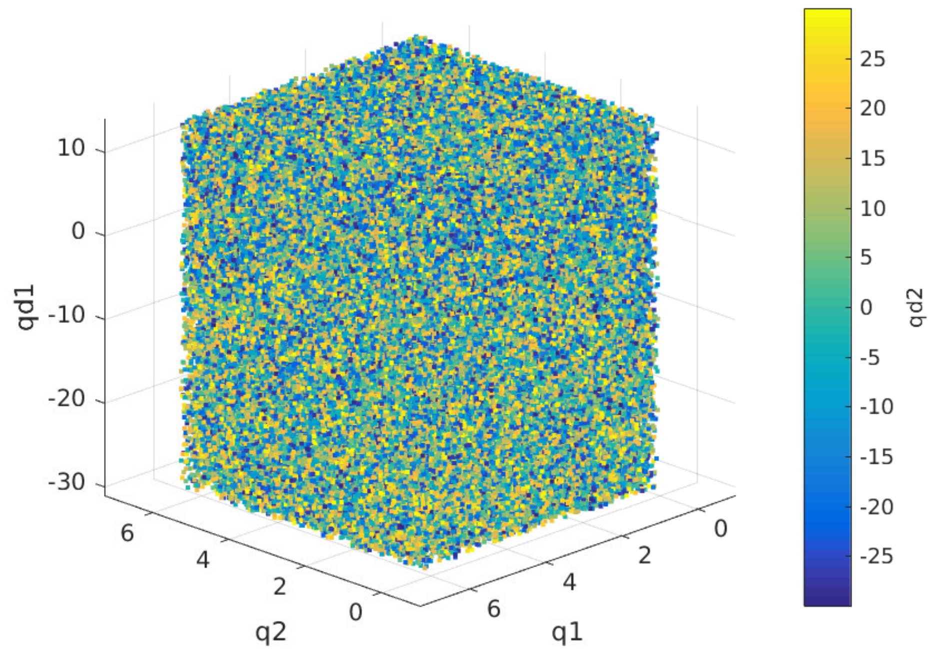
Chosen Euclidean dist.	% of similar samples
3	9
4	28

Table 6-3: Cleaning of indirect data (unconstrained)

Chosen Euclidean dist.	% of similar samples
3	16
4	31

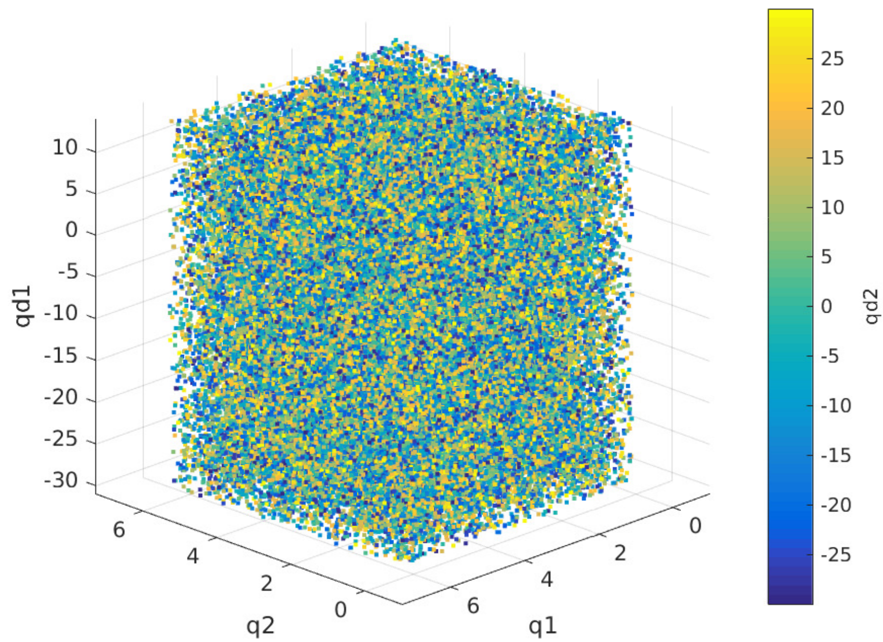


a) x_i (Direct approach)

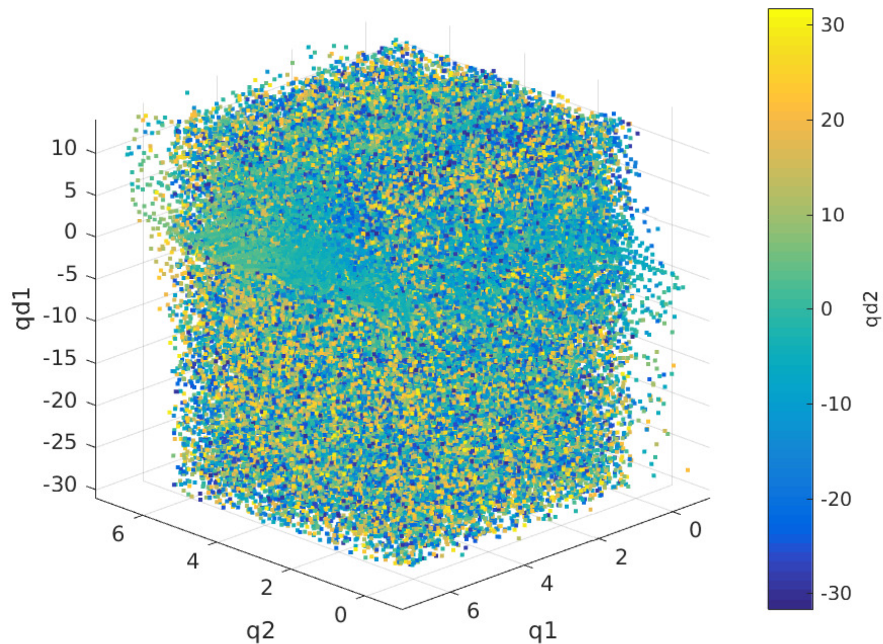


b) x_f (Direct approach)

Figure 6-3: 4-D phase plots with direct approach (multiple shooting) without input constraints. Discolouration can be seen in a) which represents incompleteness in x_i.



a) x_i (Indirect approach)



b) x_f (Indirect approach)

Figure 6-4: 4-D phase plots with indirect approach (Pontryagin principle) without input constraints. Uniform distribution of samples is observed.

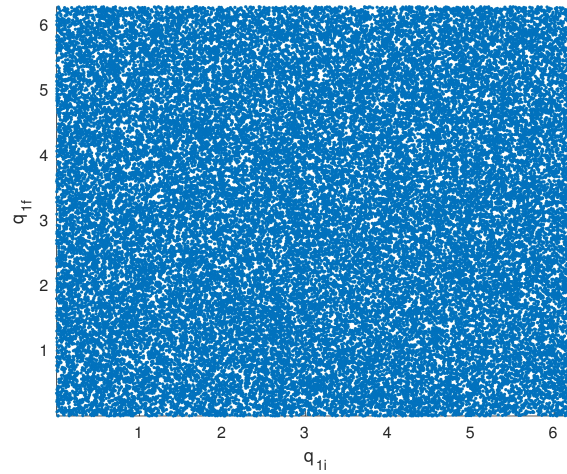


Figure 6-5: Sample x_i vs x_f phase plot for direct data without input constraints. Other plots have similar coverage.

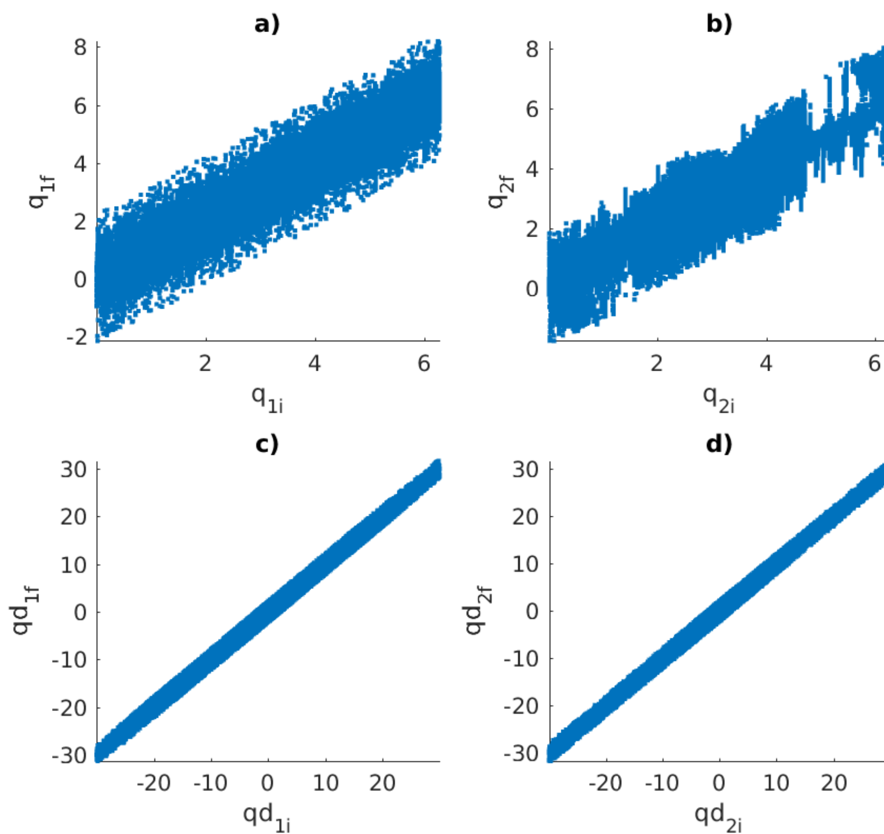


Figure 6-6: x_i vs x_f phase plot for indirect data without input constraints. Only plots with irregularities are shown. Other plots have uniform coverage.

6-2-1 Learning unconstrained data with KNN

After cleaning, the datasets are pre-processed to facilitate learning. The datasets are first standardized. Initial tests showed that the large cost range in direct dataset was affecting learning. Therefore the cost is bound to 60000 in direct dataset to mitigate this. Then the two datasets are used to train two models using k-nearest neighbour algorithm. One model is for predicting the steering input and another for predicting the distance metric. Ideal number of neighbours (value of k) for the KNN algorithms was determined using cross-validation error for different values of k (as specified in Section.3-2). $k = 13$ and 2 was found to be best to model distance metric for direct dataset and indirect dataset respectively. For steering control, $k = 7$ was found to be better for indirect dataset while $k = 18$ performed better for direct dataset (Table.6-4). This difference could be due to the higher dimensionality of data generated using direct optimal control approach.

Table 6-4: Optimal k for KNN

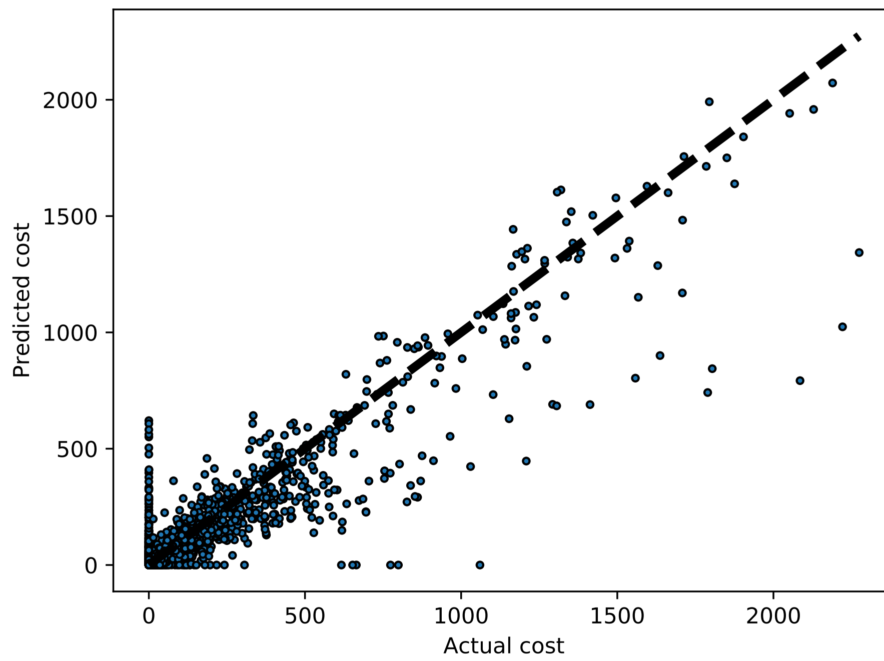
Optimal control approach	Optimal k	
	Cost	Control
Pontryagin (Indirect)	2	7
Multiple shooting (Direct)	13	18

Average cross-validation error Average cross-validation error over 10 folds is given in Table. 6-5. The error is scaled to allow comparison between the two datasets. It is observed that the error values are high, especially for control input. This suggests that the KNN algorithm was not able to learn very well from either dataset. However, if the errors are compared to each other, it is seen that the dataset obtained using indirect optimal control shows lower validation error compared to the direct optimal control one.

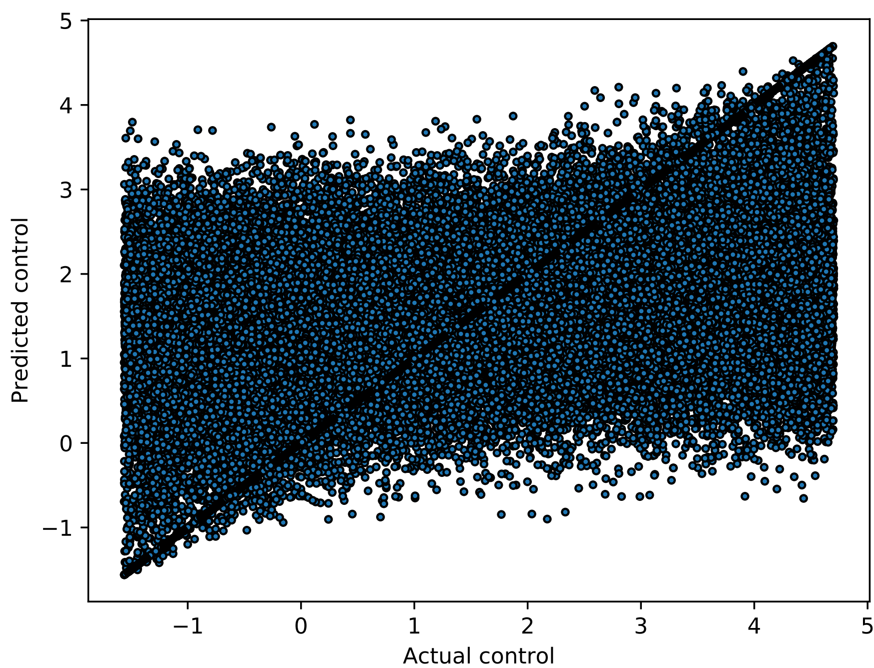
Table 6-5: Average cross-validation error for KNN

	Average MSE (scaled)	
Optimal control approach	Cost	Control input
Multiple-shooting (Direct)	0.891	0.986
Pontryagin (Indirect)	0.194	0.8213

Prediction profile Cost and control input predictions using the indirect dataset are given in Fig.6-7 and the same for direct dataset in Fig.6-8. As expected from the high cross-validation error, the control input predictions are poor and inaccurate for both indirect and direct datasets with wide spread about the $x = y$ line. This implies that KNN is not able to correctly model the control input. Cost predictions are a little better with lower error. Cost predictions from the indirect dataset are much better than the predictions from direct dataset with low spread across the $x = y$ line. Cost predictions from indirect dataset are observed to have lower error. Lower range of cost in the indirect dataset could be the influencing factor.

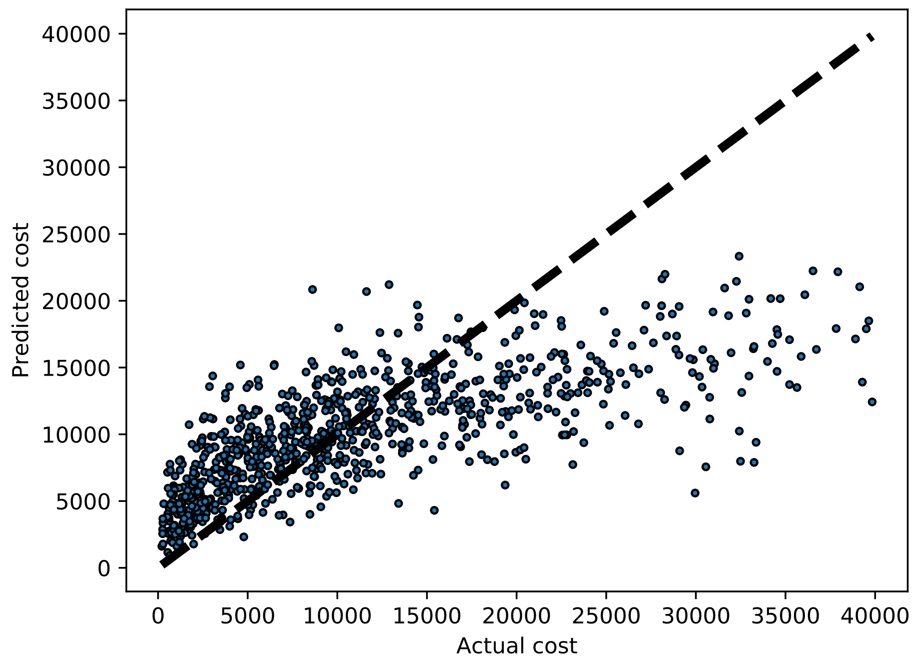


a) Distance metric prediction profile

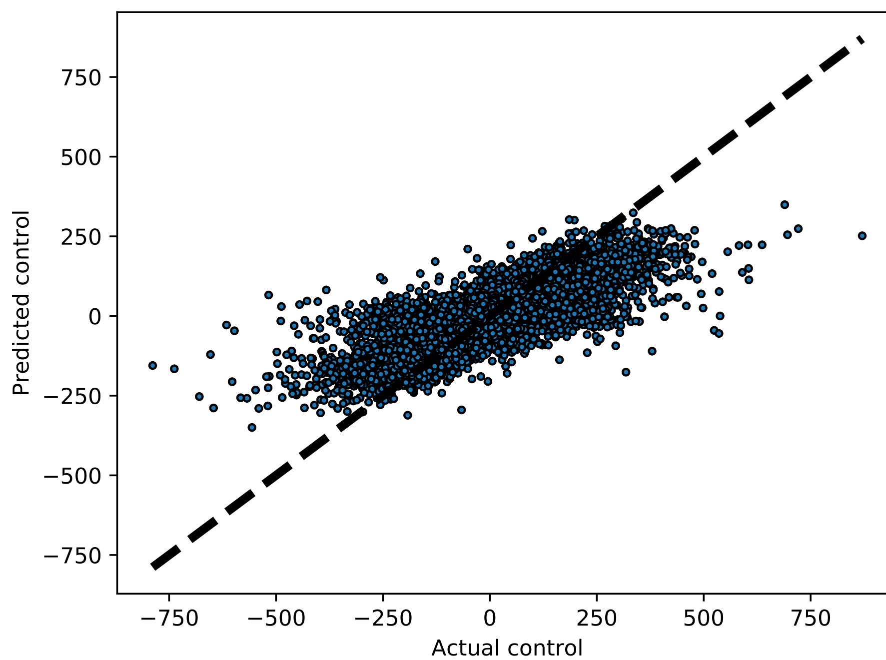


b) Co-state prediction profile (Steering input)

Figure 6-7: KNN prediction profiles of indirect optimal control dataset (without constraints)



a) Distance metric prediction profile



b) Steering input prediction profile

Figure 6-8: KNN prediction profiles of direct optimal control dataset (without constraints)

6-2-2 Learning unconstrained data with feed-forward neural network

The datasets are next used to train feed-forward neural networks. Again two models are trained for each dataset; one for cost and other for control input. Number of hidden layers and number of neurons in each layer are the tunable parameters here. There is no method to find the correct number of hidden layers or number of neurons in each layer [50]. The optimum values are therefore found using trial and error.

Average cross-validation error It is observed that 2 hidden layers give good convergence. Cross-validation error for indirect dataset with different network structures is given in Table.6-6 and the same for direct dataset is given in Table.6-7. Number of elements in array (a,b) represents number of hidden layers and each element gives number of neurons in that layer.

Table 6-6: Cross validation error for indirect dataset with neural network

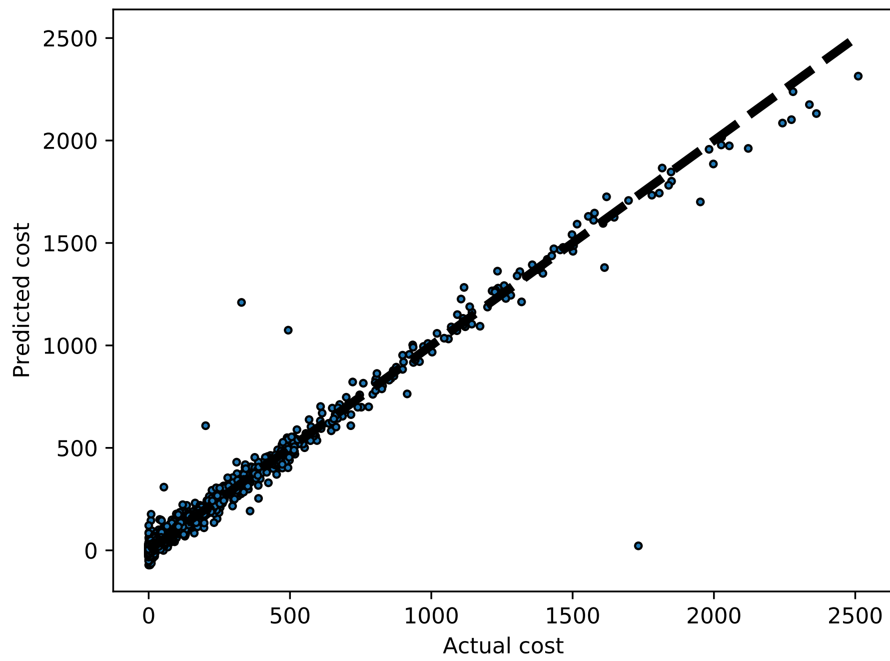
Cost		Control input	
Neurons/layer	MSE (scaled)	Neurons/layer	MSE (scaled)
(75,20)	0.05893	(100,20)	0.7554
(75,50)	0.02836	(100,50)	0.7432
(75,75)	0.0353	(100,100)	0.7207

Table 6-7: Cross validation error for direct dataset with neural network

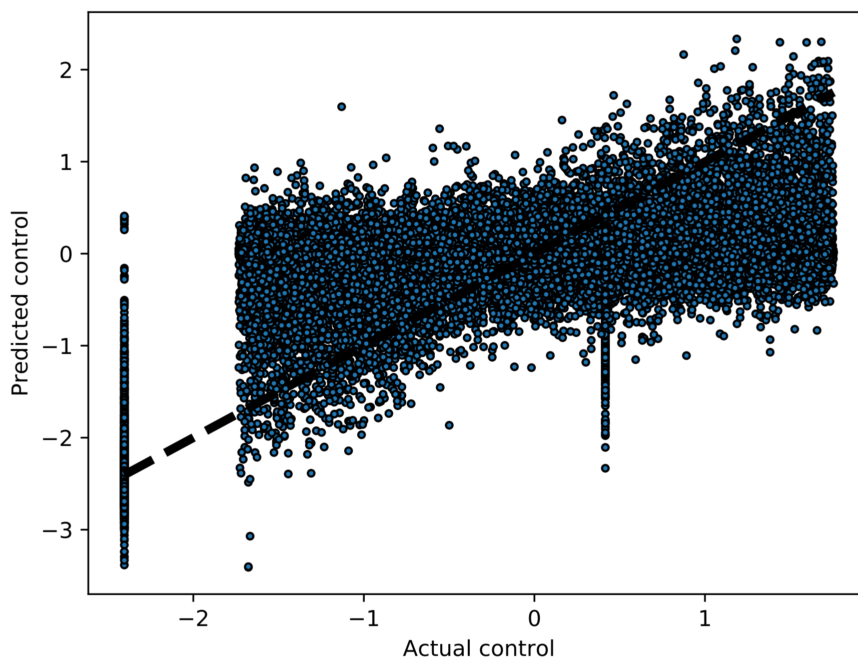
Cost		Control input	
Neurons/layer	MSE (scaled)	Neurons/layer	MSE (scaled)
(100,20)	0.3743	(100,20)	0.3813
(100,60)	0.3845	(100,50)	0.3754
(100,70)	0.399	(100,80)	0.3761

The control input errors are high with both the datasets. Therefore poor predictions are expected similar to KNN. But surprisingly, direct dataset shows lower error for control input compared to indirect dataset. Also, it is noticed that mean squared error of the cost is very low with the indirect dataset. This implies very good learning of cost. Cost in the direct dataset has higher range and variance compared to the indirect dataset alongside lower number of samples. This could explain the difficulty of learning the cost using direct dataset.

Prediction profile The prediction profiles using feed-forward neural networks for the two unconstrained datasets are shown in Fig.6-9 and Fig.6-10. The cost model displays very good prediction in the indirect case as expected from the low mean square error. The input predictions are not good with wide spread about the $x = y$ line. Thus neural network also failed to correctly learn the control input for the chosen test case. With the direct dataset, the high number of control parameters could be making the learning difficult.

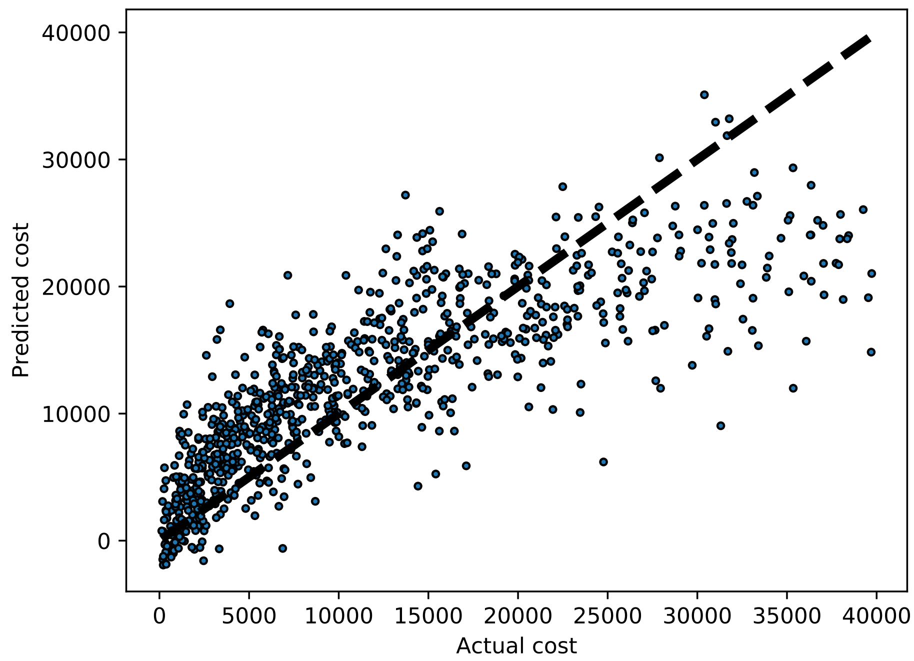


a) Cost prediction profile

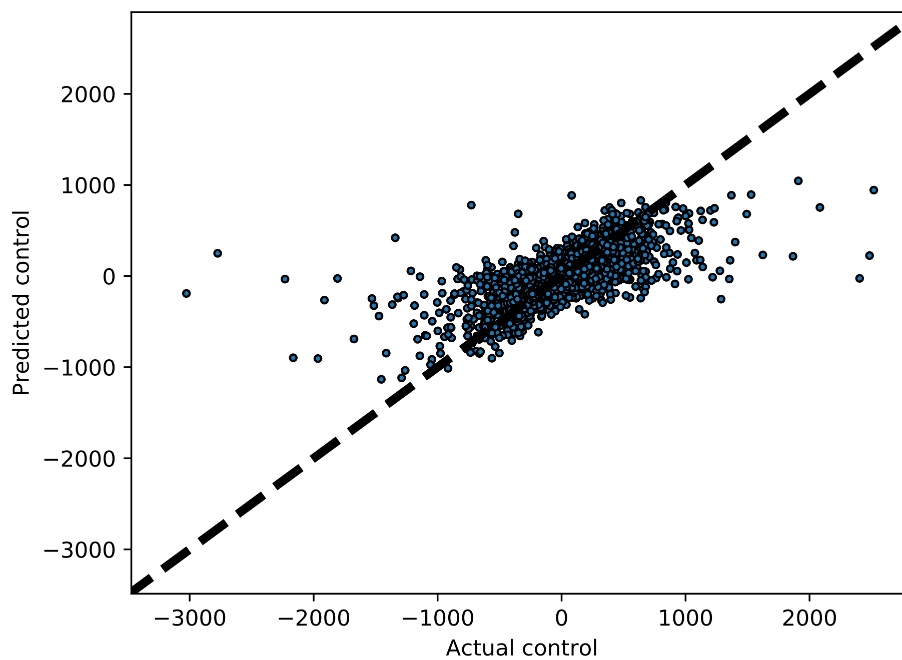


b) Control input prediction profile

Figure 6-9: Neural network prediction profiles of indirect optimal control dataset (without constraints)



a) Cost prediction profile



b) Control input prediction profile

Figure 6-10: Neural network prediction profiles of direct optimal control dataset (without constraints)

6-3 Effect of input constraints

The effect of input constraints on the datasets and learning is investigated in this section. Datasets are generated the same way as before except input constraints are applied now (5-11). 10000 sample direct dataset is used as the 100000 sample dataset was discovered to be faulty. The metrics are again evaluated to see the effect of the input constraints.

$$-400 \leq T_1 \leq 400$$
$$-400 \leq T_2 \leq 400$$

Data generation time Application of input constraints was not observed to induce any significant change in data generation time compared to unconstrained case (Table. 6-8). Indirect method still vastly out-paces direct approach. This is because input constraints checking in both the approaches is a simple boolean operation which does not cause any additional overhead.

Table 6-8: Data generation time with input constraints

Optimal control approach	Generation time for 100k samples (hours)	Average sample generation time (s)
Pontryagin (Indirect)	0.64 (1 thread)	0.023
Multiple shooting (Direct)	66.7 (8 threads)	2.41

Data completeness Phase plot of x_i and x_f using direct optimal control approach with input constraints in shown in Fig.6-12. Non-uniform colouration of the phase plot can be noticed in Fig.6-12a. This mean that there is a gap in that region. This can be more clearly seen in Fig.6-11. No gaps are observed in x_i vs x_f plots (sample graph in Fig.6-14)

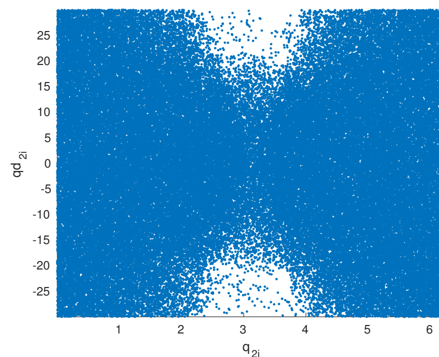
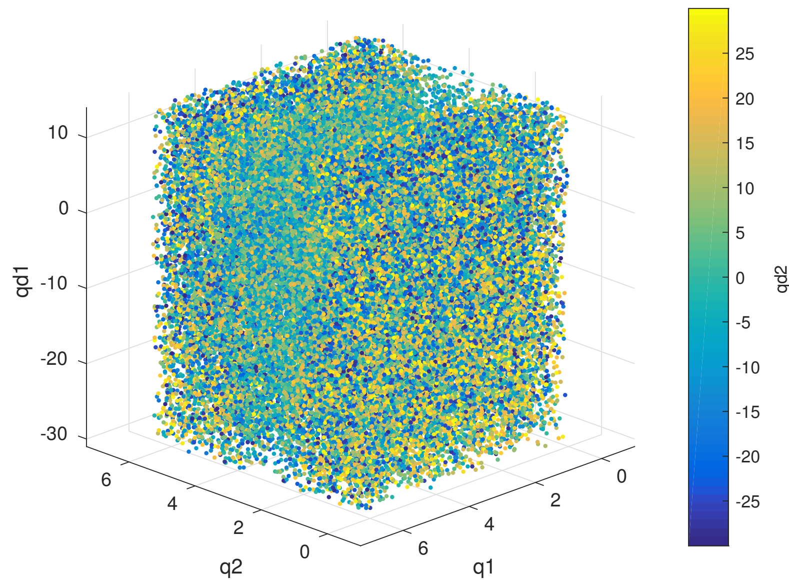


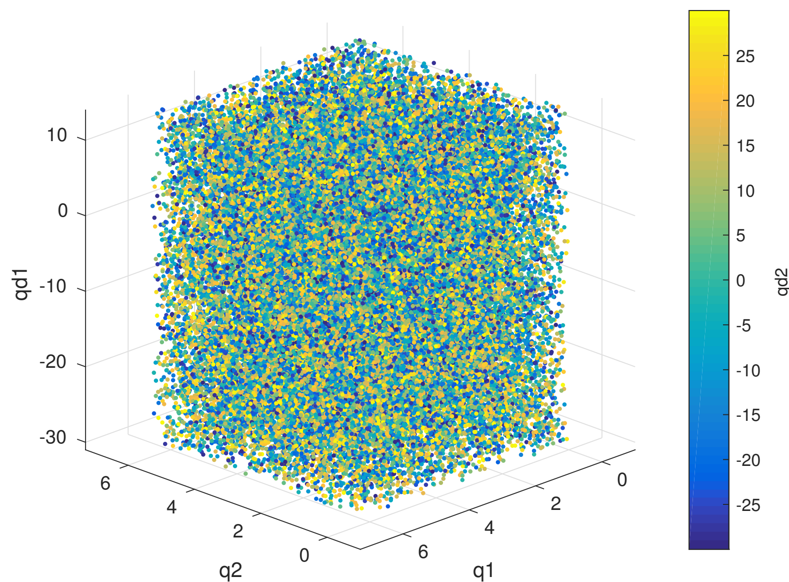
Figure 6-11: qd_{2i} vs q_2i plot. Non-uniformity can be clearly seen here.

In the case of indirect optimal control approach with constraints, very obvious gaps can be seen at bottom of the phase plots of x_i and x_f (Fig.6-13). Other gaps can also be seen in x_i vs x_f phase plot alongside gaps due to correlation (Fig.6-15g, h).

Reduction in state space coverage in the constrained case is hypothesized to be due to the input constraints. The reachability of some regions of the state space is reduced as they violate the constraints. This makes the data incomplete.

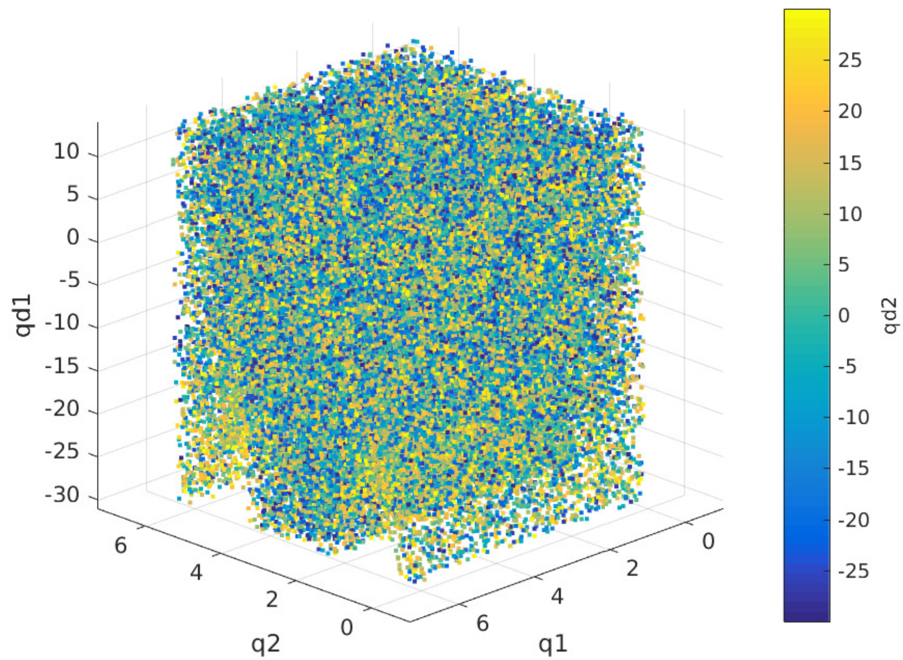


a) x_i (Direct approach)

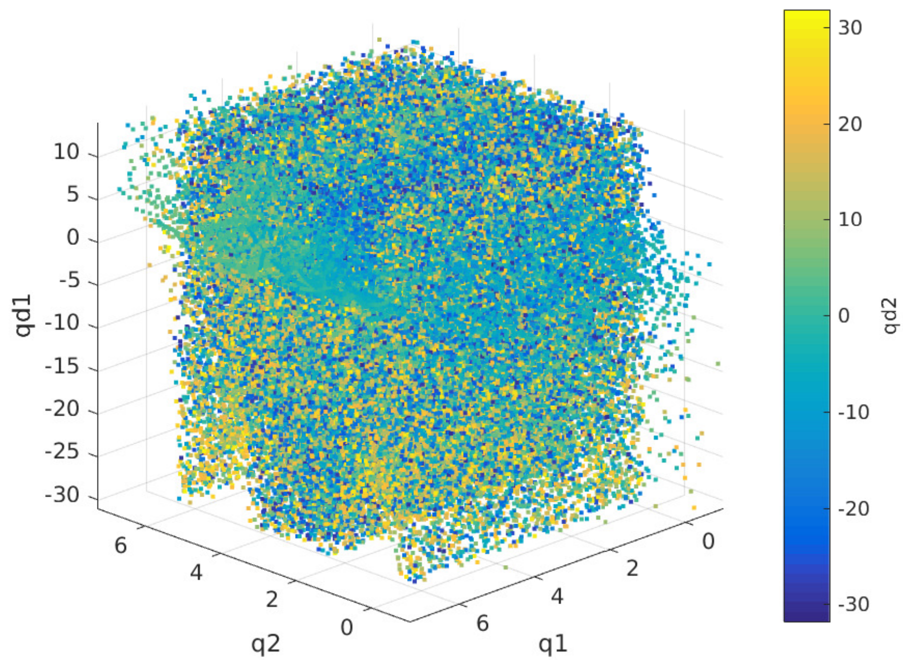


b) x_f (Direct approach)

Figure 6-12: 4-D phase plots with direct approach (multiple shooting) with input constraints. The sparseness in the middle in a) can be clearly seen.



a) x_i (Indirect approach)



b) x_f (Indirect approach)

Figure 6-13: 4-D phase plots with indirect approach (Pontryagin principle) with input constraints. U-shaped gaps can be observed at the bottom of each plot.

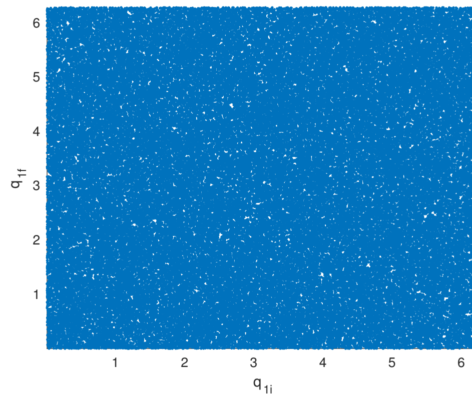


Figure 6-14: x_i vs x_f phase plot for direct data with input constraints. Uniform coverage is observed on all the plots.

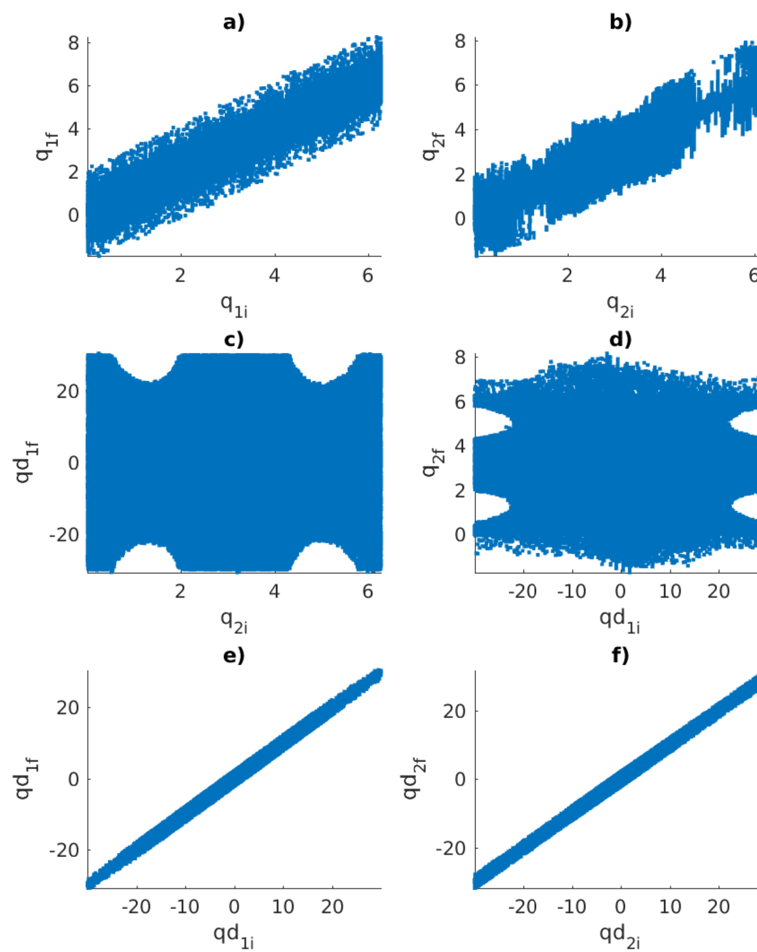


Figure 6-15: x_i vs x_f phase plot for indirect data with input constraints.

Data uniqueness Data cleaning was performed on each data set again using euclidean distances 3 and 4. The number of similar samples in each case is shown in Table.6-9 and Table.6-10. Higher percentage of similar samples are noticed in the indirect constrained dataset as before. Also, it is observed that these constrained datasets have higher percentage of similar samples than the unconstrained datasets. Input constraints could be a possible reason for this. Input constraints force data generation from a restricted state space which causes samples to be closer to each other (compared to an unconstrained case).

Table 6-9: Cleaning of direct data

Chosen Euclidean dist.	% of similar samples
3	10
4	27

Table 6-10: Cleaning of indirect data

Chosen Euclidean dist.	% of similar samples
3	20
4	39

6-3-1 Learning input constrained data with KNN

Same as before, the datasets are standardized, and direct dataset is bounded at 40000 before being used to train two models using KNN. Values of k for the input constrained datasets were found as in Table.6-11.

Table 6-11: Optimal k for KNN

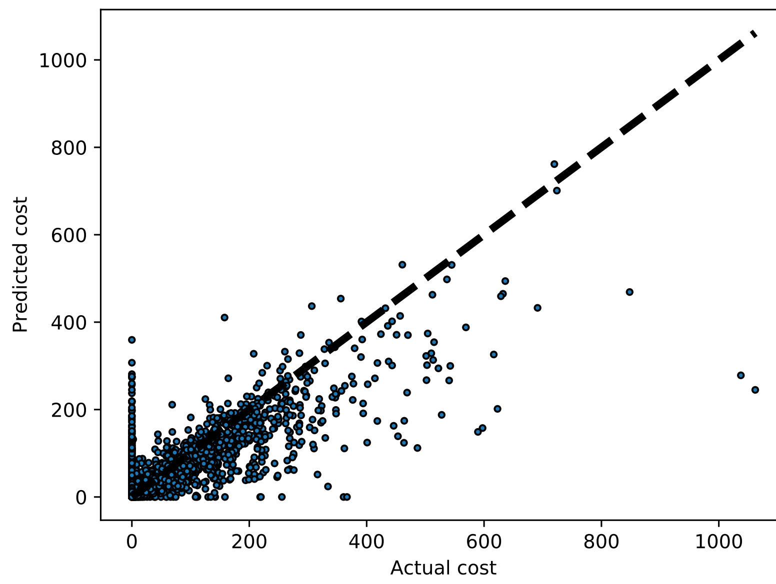
Optimal control approach	Optimal k	
	Cost	Control
Pontryagin (Indirect)	3	6
Multiple shooting (Direct)	4	22

Average cross-validation error Cross-validation errors with KNN for the constrained datasets are given in Table.6-12. The overall error values is found to have reduced compared to the unconstrained data. But the error for control input learning is still high. Even the cost error is not as good as that observed in neural network previously.

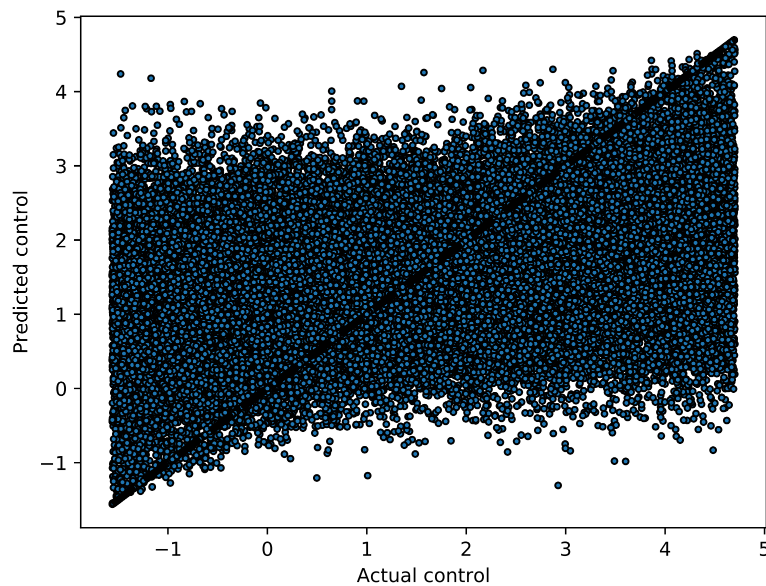
Table 6-12: Average cross-validation error for KNN (input constrained)

	Average MSE (scaled)	
Optimal control approach	Cost	Control input
Multiple-shooting (Direct)	0.299	0.804
Pontryagin (Indirect)	0.365	0.749

Prediction profile KNN prediction profiles with constrained indirect and direct datasets are given in Fig.6-16 and Fig.6-17. Control input predictions are inaccurate even with these datasets. Cost predictions are better with the indirect dataset with low spread, but they are very poor with direct dataset.

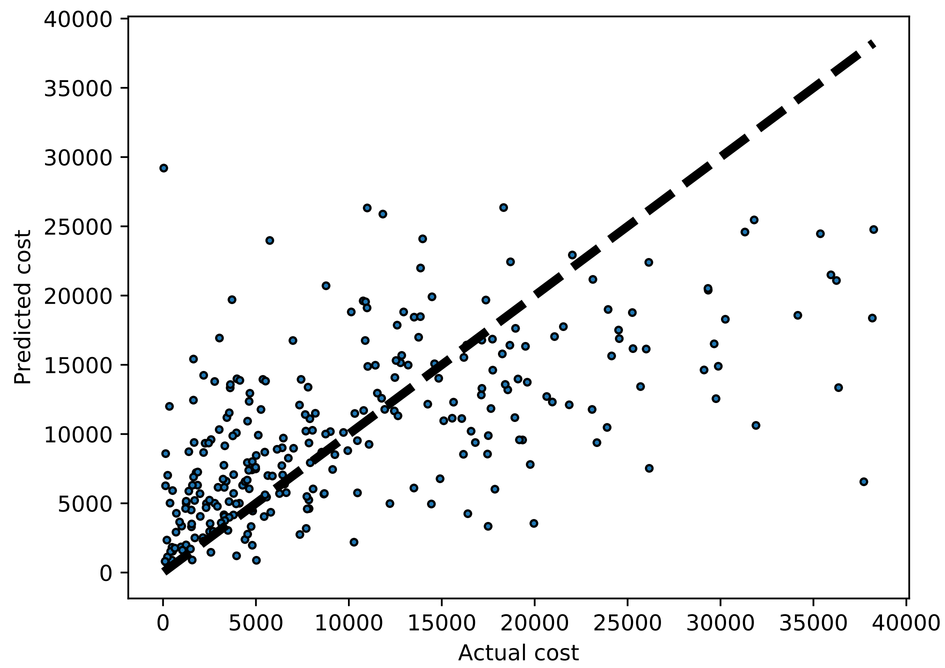


a) Distance metric prediction profile

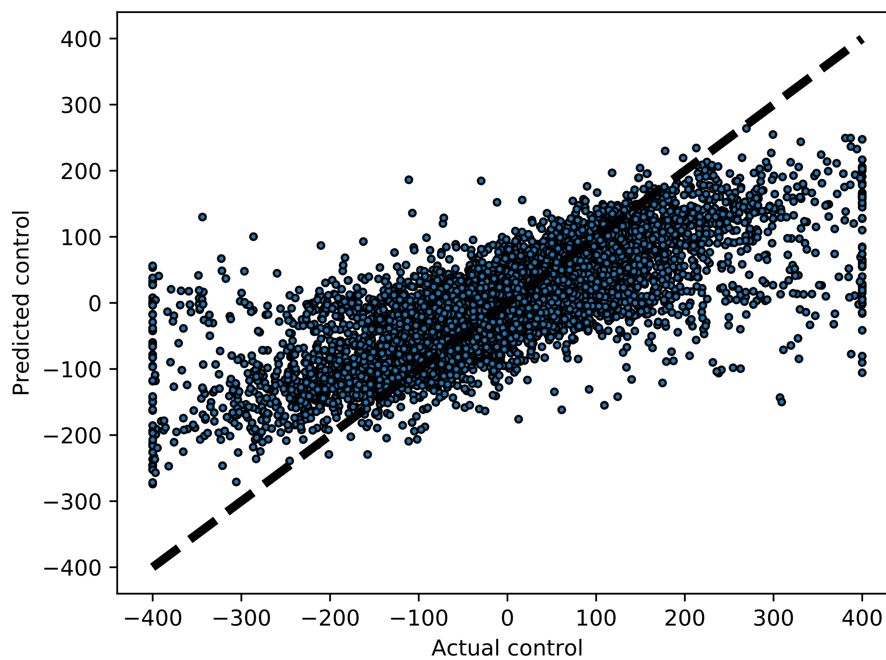


b) Co-state prediction profile (Steering input)

Figure 6-16: Prediction profiles of indirect optimal control dataset (with input constraints)



a) Distance metric prediction profile



b) Steering input prediction profile

Figure 6-17: Prediction profiles of direct optimal control dataset (with input constraints)

An important point to note is that the predicted control inputs with the constrained direct dataset do not violate the input constraints i.e all the predictions are between ±400Nm. This is because KNN takes the average of the neighbours and since none of them exceed the constraints, the predictions also do not exceed the constraints. This is not so straight forward however in the indirect dataset as the control input is not directly predicted but generated from the co-states and the initial state. Therefore some co-state predictions can result in control input that violate the input constraints depending on the initial state.

6-3-2 Learning input constrained data with feed-forward neural network

The constrained datasets are next used to train feed-forward neural network of different structures.

Average cross-validation error Mean squared errors for different network structure with the constrained datasets are given in Table.6-13 and Table.6-14. 2-hidden layers were observed to be better for constrained indirect dataset too. Similar to unconstrained case, high control input error was obtained with both datasets and low cost error was obtained with indirect dataset.

Table 6-13: Cross validation error for constrained indirect dataset with neural network

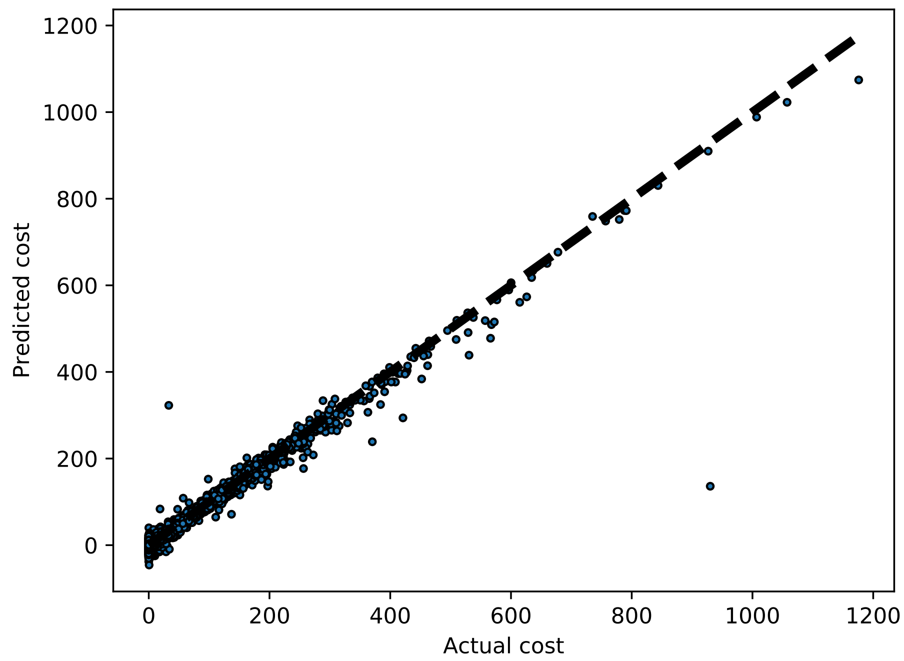
Cost		Control input	
Neurons/layer	MSE (scaled)	Neurons/layer	MSE (scaled)
(60,20)	0.0621	(80,10)	0.7548
(60,30)	0.0134	(80,40)	0.7433
(60,50)	0.022	(80,50)	0.7146

Table 6-14: Cross validation error for constrained direct dataset with neural network

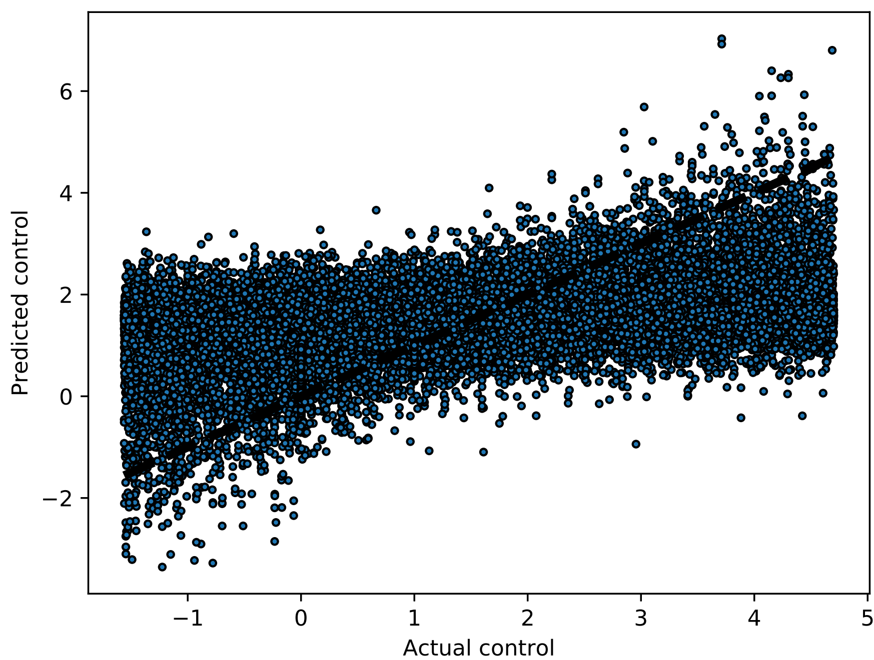
Cost		Control input	
Neurons/layer	MSE (scaled)	Neurons/layer	MSE (scaled)
(100,60)	0.5341	(100,40)	0.3791
(100,100)	0.5020	(100,50)	0.3711
(100,80)	0.5106	(100,100)	0.3142

Prediction profile The prediction profiles for neural network with input constrained datasets are in Fig.6-19 and Fig.6-19. Cost predictions very close to $x = y$ line are obtained using indirect dataset. Cost predictions using direct dataset also show low spread. Control predictions with direct dataset, though still poor, is observed to be better than predictions from previous methods.

Unlike the KNN algorithm, the predictions from direct dataset are observed to violate the input constraints i.e give prediction outside ±400Nm. This is because feed-forward neural network is a parametric learning algorithm which learns a function using the constrained dataset. But the learned function itself is not constrained and thus can give predictions which violate the input constraints.

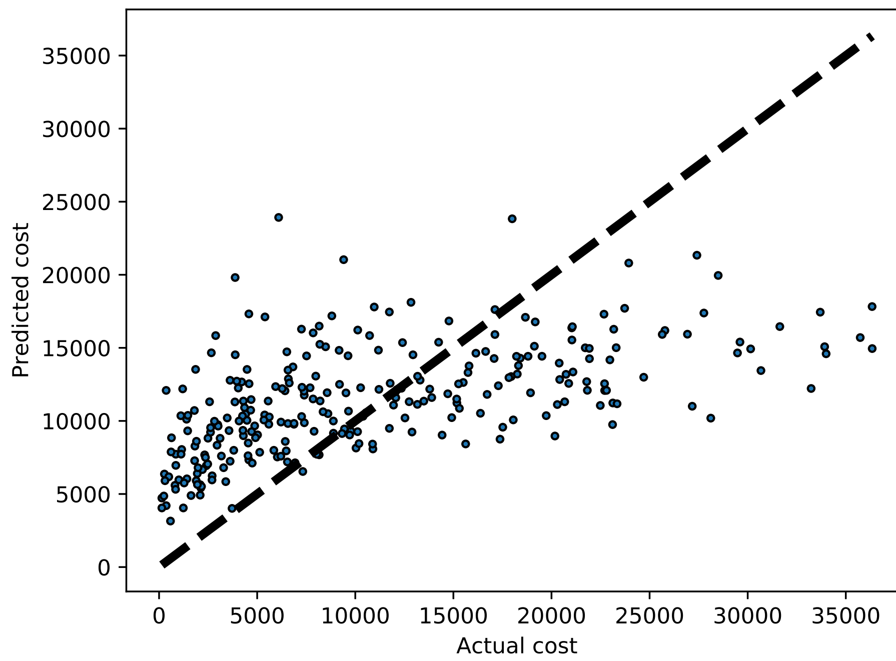


a) Cost prediction profile

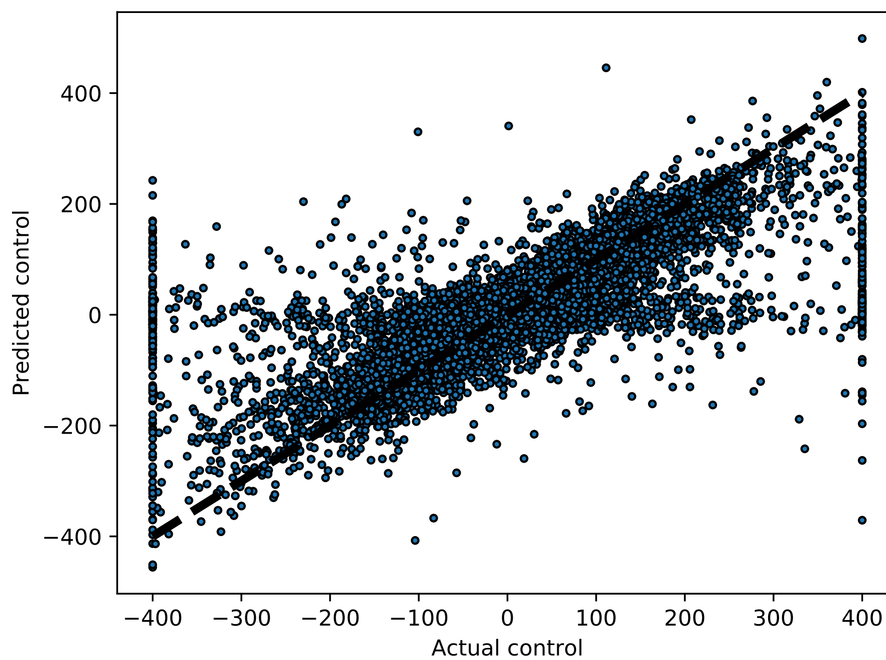


b) Control input prediction profile

Figure 6-18: Neural network prediction profiles of indirect optimal control dataset (with constraints)



a) Cost prediction profile



b) Control input prediction profile

Figure 6-19: Neural network prediction profiles of direct optimal control dataset (with constraints)

6-4 Performance in learning-based RRT

The models trained in the previous sections are next used in learning-based RRT. Both unconstrained and input constrained KNN and neural network models were incorporated into the learning-based RRT code and attempts were made to generate a motion plan for an initial-final state pair (6-1) 100 times. The convergence rate can be inferred by this. As mentioned in the previous chapter, the node limit was fixed at 10000 nodes.

$$\begin{aligned} x_i &= [0, 0, 0, 0] \\ x_f &= [0, 2\pi, 0, 0] \end{aligned} \tag{6-1}$$

However, convergence of learning-based RRT was not observed in any of the 100 attempts with both KNN and multi-layer neural network. Different initial-final state pairs were tested to no avail. Lack of convergence is attributed to poor cost and control input predictions. Inaccurate control input prediction transforms learning-based RRT to regular RRT (with random control inputs). But inaccurate cost predictions is problematic as learning-based RRT can no longer correctly find the nearest node on the tree. This prevents expansion towards the final state.

Run time, which is the time taken to converge or to run out of nodes, of KNN-based RRT and neural network-based RRT was compared during the above attempts. Run time is affected only by the learning algorithm used and not by the optimal control approaches. Primary observation was that the run time, though lower than regular RRT, is much higher than that expected from learning-based RRT due to poor model predictions. Feed-forward neural network-based RRT however has faster run time compared to KKN-based RRT. This is because KNN finds the nearest neighbours every-time the cost or the control input model is called. This is computationally expensive (Chapter 3) and calling feed-forward neural network is much faster.

Table 6-15: Run times with different algorithms

	Avg. run time per 1000 nodes
KNN-based RRT	265.8 seconds
ff-NN-based RRT	170.1 seconds

6-5 Summary

The results of the experiments detailed in previous chapter are analysed in this chapter. Indirect optimal approach is observed to be much faster than direct optimal control. Data completeness was explored and datasets obtained using indirect optimal control approach were found to be less data complete. Poor learning of control input was demonstrated by both KNN and neural networks with both indirect and direct datasets. But cost function was accurately learned by feed-forward neural network using indirect datasets. With regards to input constrained datasets, no significant difference was observed in the learning. However, only KNN trained with direct datasets was able to predict control input without violating the input constraints. Learning-based RRT implemented here failed due to inaccurate predictions by the learned models.

Chapter 7

Conclusions

This thesis started out with introducing learning-based RRT and how it is faster than regular RRT. The thesis aimed to determine the best optimal control approach to generate training data for learning-based RRT. The effect of input constraints on the data generation and learning was another point of interest. The previous two chapters described the various experiments conducted using a test case to achieve these goals. The inferences and conclusions based on the results obtained from these experiments are discussed next.

Comparison of optimal control approaches The comparison of the optimal control approaches is made using the training datasets generated using multiple-shooting method and Pontryagin's principle. The first difference between the two approaches was observed in the data generation time. Multiple-shooting (direct optimal approach) proved to be almost 100 times slower than Pontryagin's principle (indirect optimal approach) due to its iterative nature. Looking at the datasets themselves, it was noticed that both datasets were incomplete. Incompleteness in direct approach is because of the failure to converge in some regions during the optimization process. Indirect dataset is generated by randomly sampling initial state and co-state and getting the final state from integration of equations of motion. Since the integration time is fixed at 0.5 seconds, the final states obtained are close to the initial states. When used for training KNN and feed-forward neural network algorithms for control input, both the datasets were observed to perform badly with high mean squared error and inaccurate predictions. Cost learning was slightly better with indirect dataset. Due to inaccurate predictions, learning-based RRT failed as well. Possible reasons for poor learning are:

- Insufficient training data could be one reason for poor learning. 100000 sample dataset was generated with indirect approach and 500000 sample dataset with direct approach for learning in this thesis. It is possible that larger datasets are required for proper learning of cost and control input. However, it should be noted that direct optimal control is very time consuming ($\approx$ 2.4s per sample) and larger datasets take days to generate.

- Poor control learning with direct dataset is suspected to be due to the high number of parameters to be learned for control input (40 parameters). Large variation of costs in direct dataset could be the reason for poor learning of cost. Cost was bounded to help with learning but it did not prove to be sufficient.

Effect of input constraints The direct and indirect optimal control approaches are used to generate training datasets again, this time with input constraints. No significant change in data generation time was observed with application of input constraints. This is because input constraints checking poses no additional overhead. However, input constraints were observed to cause a decrease in data completeness and uniqueness in both direct and indirect datasets. This is because input constraints force sampling from certain regions of state space which causes the samples to be sparse in some regions. Therefore, as expected, learning of control input using KNN and feed-forward neural networks with these datasets was very poor similar to the unconstrained datasets. But the cost model trained using feed-neural network and indirect dataset was observed to have low error and accurate prediction. When dealing with an input constrained system, it is desired that the predicted control input also satisfy the constraints. KNN trained using direct dataset was observed to satisfy this criteria. This is because KNN takes the average of the nearest neighbours and since all neighbours satisfy the input constraints, the prediction also satisfies the input constraint. However, this does not happen with indirect dataset or feed-forward neural network. Indirect dataset predict co-states and not control, therefore control cannot be constrained directly. Also neural network generates a parametric function using the training data which is not constrained.

The major conclusions determined from this thesis are:

1. Neither optimal control approach proved to be better for this test case (2-link manipulator) as both approaches resulted in inaccurate trained models. However, indirect optimal control approach was observed to be faster and direct optimal control approach more data complete of the two.
2. Input constraints reduce the feasible state space and reduce the completeness of the dataset. This can affect learning.
3. If constrained control predictions are required, a non-parametric learning algorithm should be used to directly predict the control inputs. That means direct optimal control approach is better in this case.

Future work The following are a few viable research directions to improve and extend the contributions of this thesis:

- The optimal control approaches used a fixed time interval of 0.5 seconds to generate the samples. Samples with variable time steps can tried to increase the completeness of the datasets and obtain better learning.
- Use of classification instead of regression for training can be investigated to constrain the control input within the bounds even with parametric learning algorithms.

- Only the effect of input constraints was investigated in this thesis. Application of state constraints would be the next logical step to investigate.
- It is suspected that high dimensionality of control input in direct datasets was the cause of poor learning with that dataset. Dimensionality reduction using techniques such principal component analysis (PCA) could help overcome this issue.

Appendix A

urdf2eom

`urdf2eom` is a software developed over the course of this thesis to generate symbolic equations of motion using Roy Featherstone's rigid body algorithms for serial link manipulators.

The robot is described using Unified Robot Description Format (URDF), which is a standard XML convention for describing robot models. `urdf2eom` parses the URDF file and generates equations of motion in terms of configuration coordinates $(q, \dot{q}, \ddot{q})$ and torque τ. The code generates forward dynamic equations using Articulated body algorithm and inverse dynamic equations using Iterative Newton-Euler algorithm [12]. However, the code currently only supports fixed-base serial link manipulators with revolute and continuous joints. The code and instructions on how to use it can be found at :

https://github.com/DeepakParamkusam/urdf2eom

Appendix B

Thesis source code

The implementations of the experiments performed in this thesis can be found in the thesis repository below:

https://github.com/DeepakParamkusam/learning-based-RRT

The implementations were written in MATLAB, C++ and Python and have the following dependencies :

- MATLAB Symbolic toolbox
- Automatic Control and Dynamic Optimization (ACADO) Toolkit [7]
- numpy Python library
- scikit-learn Python library [66]

The code is divided into different folders in the repository as follows:

2link_direct : This folder contains the implementation of direct optimal control for 2-link manipulator. The code is written in C++ and uses the ACADO Toolkit. ACADO toolkit returns solutions in multiple files. They can be consolidated using the scripts in this folder.

2link_indirect : Implementation of indirect optimal for 2-link manipulator can be found in this folder. The indirect optimal control equations are generated in MATLAB(`gen_eom.m`) while the data generation is performed in Python.

training_data : Data generated in this thesis is stored here. This folder also contains the Python code used for cleaning the data.

2link_NN : The folder contains the code for training the KNN and ANN with the data. scikit-learn library is used for the training. Trained model are stored in `trained_models` folder

2link_rrt : This folder contains the implementations of learning-based rrt.

Bibliography

[1] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[2] W. Wolfslag, M. Bharatheesha, T. Moerland, and M. Wisse, "Rrt-colearn: towards kinodynamic planning without numerical trajectory optimization," *arXiv preprint arXiv:1710.10122*, 2017.

[3] "Neuron." https://en.wikipedia.org/Neuron. Accessed: 6-10-2017.

[4] "Artificial neural network." https://en.wikipedia.org/Artificialneuralnetwork. Accessed: 6-10-2017.

[5] M. Diehl, "Numerical optimal control." http://www.syscop.de/files/2014ss/noc-summer-school/OptimalControl.pdf, 2014.

[6] R. M. Murray, Z. Li, S. S. Sastry, and S. S. Sastry, *A mathematical introduction to robotic manipulation*. CRC press, 1994.

[7] R. Featherstone, *Rigid body dynamics algorithms*. Springer, 2014.

[8] H. M. Choset, *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[9] G. Lopez, "Slides for control methods for robotics course," 2016.

[10] D. Jeltsema, "Slides for modelling and non-linear system theory," 2016.

[11] C. Ó'Dúnlaing, "Motion planning with inertial constraints," *Algorithmica*, vol. 2, no. 1, pp. 431–475, 1987.

[12] J. Canny, A. Rege, and J. Reif, "An exact algorithm for kinodynamic planning in the plane," *Discrete & Computational Geometry*, vol. 6, no. 3, pp. 461–484, 1991.

[13] J. Canny and J. Reif, "New lower bound techniques for robot motion planning problems," in *Foundations of Computer Science, 1987., 28th Annual Symposium on*, pp. 49–60, IEEE, 1987.

[14] J. Barraquand, B. Langlois, and J.-C. Latombe, "Numerical potential field techniques for robot path planning," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 2, pp. 224–241, 1992.

[15] S. S. Ge and Y. J. Cui, "New potential functions for mobile robot path planning," *IEEE Transactions on robotics and automation*, vol. 16, no. 5, pp. 615–620, 2000.

[16] G. Sahar and J. M. Hollerbach, "Planning of minimum-time trajectories for robot arms," *The International journal of robotics research*, vol. 5, no. 3, pp. 90–100, 1986.

[17] K. Shin and N. McKay, "A dynamic programming approach to trajectory planning of robotic manipulators," *IEEE Transactions on Automatic Control*, vol. 31, no. 6, pp. 491–500, 1986.

[18] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *Journal of the ACM (JACM)*, vol. 40, no. 5, pp. 1048–1066, 1993.

[19] C. H. Papadimitriou, "An algorithm for shortest-path motion in three dimensions," *Information Processing Letters*, vol. 20, no. 5, pp. 259–263, 1985.

[20] E. Frazzoli, "Slides for principles of autonomy and decision making." https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-410-principles-of-autonomy-and-decision-making-fall-2010/lecture-notes/MIT16_410F10_lec15.pdf, 2010.

[21] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[22] N. M. Amato and Y. Wu, "A randomized roadmap method for path and manipulation planning," in *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, vol. 1, pp. 113–120, IEEE, 1996.

[23] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[24] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pp. 2997–3004, IEEE, 2014.

[25] A. Yershova, L. Jaillet, T. Siméon, and S. M. LaValle, "Dynamic-domain rrts: Efficient exploration by controlling the sampling domain," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pp. 3856–3861, IEEE, 2005.

[26] M. Jordan and A. Perez, "Optimal bidirectional rapidly-exploring random trees," 2013.

[27] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with rrts," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pp. 1243–1248, IEEE, 2006.

[28] R. Babuska, "Lecture notes for knowledge based control systems," 2010.

[29] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of machine learning.* MIT press, 2012.

[30] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 2, no. 4, pp. 303–314, 1989.

[31] M. Bharatheesha, W. Caarls, W. J. Wolfslag, and M. Wisse, "Distance metric approximation for state-space rrts using supervised learning," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pp. 252–257, IEEE, 2014.

[32] L. Palmieri and K. O. Arras, "Distance metric learning for rrt-based motion planning with constant-time inference," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pp. 637–643, IEEE, 2015.

[33] D. E. Kirk, *Optimal control theory: an introduction.* Courier Corporation, 2012.

[34] D. S. Naidu, *Optimal control systems.* CRC press, 2002.

[35] P. DraĬğg, K. Styczeń, M. Kwiatkowska, and A. Szczurek, "A review on the direct and indirect methods for solving optimal control problems with differential-algebraic constraints," in *Recent Advances in Computational Optimization*, pp. 91–105, Springer, 2016.

[36] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[37] S. R. Lindemann and S. M. LaValle, "Steps toward derandomizing rrts," in *Robot Motion and Control*, pp. 287–300, Springer, 2006.

[38] E. Glassman and R. Tedrake, "A quadratic regulator-based heuristic for rapidly exploring state space," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pp. 5021–5028, IEEE, 2010.

[39] W. Spierenburg, "Motion planning in the state space," Master's thesis, TU Delft, 2016.

[40] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems.," in *ICINCO (1)*, pp. 222–229, 2004.

[41] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 1478–1483, IEEE, 2011.

[42] Q.-C. Pham, "A general, fast, and robust implementation of the time-optimal path parameterization algorithm," *IEEE Transactions on Robotics*, vol. 30, no. 6, pp. 1533–1540, 2014.

[43] M. Jordan and A. Perez, "Optimal bidirectional rapidly-exploring random trees," 2013.

[44] M. Likhachev, G. J. Gordon, and S. Thrun, "Ara*: Anytime a* with provable bounds on sub-optimality," in *Advances in Neural Information Processing Systems*, pp. 767–774, 2004.

[45] E. Alpaydin, *Introduction to machine learning*. MIT press, 2014.

[46] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[47] N. M. Nasrabadi, "Pattern recognition and machine learning," *Journal of electronic imaging*, vol. 16, no. 4, p. 049901, 2007.

[48] N. R. Draper and H. Smith, *Applied regression analysis*. John Wiley & Sons, 2014.

[49] P. Cunningham, M. Cord, and S. J. Delany, *Supervised Learning*, pp. 21–49. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[50] A. Moore, "Instance-based learning." http://www.ccs.neu.edu/home/rjw/csg220/lectures/instance-based-1.pdf, 2005.

[51] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, "On the surprising behavior of distance metrics in high dimensional spaces," in *ICDT*, vol. 1, pp. 420–434, Springer, 2001.

[52] J. Bejar, "K-nearest neighbours." http://www.lsi.upc.edu/~bejar/apren/docum/trans/03d-algind-knn-eng.pdf, 2012.

[53] C. M. Bishop, *Neural networks for pattern recognition*. Oxford university press, 1995.

[54] K. L. Priddy and P. E. Keller, *Artificial neural networks: an introduction*, vol. 68. SPIE press, 2005.

[55] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[56] A. V. Rao, "A survey of numerical methods for optimal control," *Advances in the Astronautical Sciences*, vol. 135, no. 1, pp. 497–528, 2009.

[57] B. Chachuat, "Direct solution methods." http://la.epfl.ch/files/content/sites/la/files/shared/import/migration/IC_32/Slides19-21.pdf, 2009.

[58] A. Barclay, P. E. Gill, and J. B. Rosen, "Sqp methods and their application to numerical optimal control," *Report NA*, pp. 97–3, 1997.

[59] P. DraÌğg, K. Styczeń, M. Kwiatkowska, and A. Szczurek, "A review on the direct and indirect methods for solving optimal control problems with differential-algebraic constraints," in *Recent Advances in Computational Optimization*, pp. 91–105, Springer, 2016.

[60] V. Yadav, "Direct collocation for optimal control." https://mec560sbu.github.io/2016/09/30/direct_collocation/. Accessed: 6-10-2017.

[61] B. Houska, H. Ferreau, and M. Diehl, "ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization," *Optimal Control Applications and Methods*, vol. 32, no. 3, pp. 298–312, 2011.

[62] J. J. Craig, *Introduction to robotics: mechanics and control*, vol. 3. Pearson Prentice Hall Upper Saddle River, 2005.

[63] B. V. Mirtich, *Impulse based Dynamic Simulation of Rigid Body Systems*. PhD thesis, University of California at Berkeley, 1996.

[64] D. Paramkusam, “urdf2eom.” https://github.com/DeepakParamkusam/urdf2eom, 2017.

[65] B. Houska, H. Ferreau, M. Vukov, and R. Quirynen, “ACADO Toolkit User’s Manual.” http://www.acadotoolkit.org, 2009–2013.

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

[67] L. L. Pipino, Y. W. Lee, and R. Y. Wang, “Data quality assessment,” *Commun. ACM*, vol. 45, pp. 211–218, Apr. 2002.

[68] P. Domingos, “A few useful things to know about machine learning,” *Communications of the ACM*, vol. 55, no. 10, pp. 78–87, 2012.