TÉCNICO LISBOA

Safety Based Machine Learning for Mechanical Systems

Luís Maria Aires Barros Falcão de Melo

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor(s): Prof. Rodrigo Martins de Matos Ventura

Examination Committee

Chairperson: Prof. António Manuel Raminhos Cordeiro Grilo
Supervisor: Prof. Rodrigo Martins de Matos Ventura
Member of the Committee: Prof. Francisco António Chaves Saraiva de Melo

November 2022

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

To my family and friends

Acknowledgments

I would like to start with a special thanks to my family that provided for my education and gave me all the necessary support throughout the years.

Next, I would like to thank my supervisor Prof. Rodrigo Ventura for his guidance and the time spent during our weekly meetings.

To all my friends, specially the ones that embarked on this journey alongside me, with a special thanks to António Bessa, that accompanied me through my Master degree and thesis, Gonçalo Teixeira for all his advices and Ricardo Cardoso for his counsel and patience.

Finally, last but certainly not least, I would like to thank my girlfriend, Leonor, who put up with me on my lowest moments and was always there for me to lift me up and encourage me to do better.

Resumo

O uso de robô autónomos para manufaturação tem vindo a ser uma área emergente na robótica. Quando um sistema autónomo está no processo de aprendizagem do seu modelo dinâmico ou o ambiente envolvente, é critico que a segurança do sistema seja assegurada, que pode ir desde evitar colisões com obstáculos até à prevenção de violações das restrições impostas pelo sistema. O uso de uma abordagem de caixa negra (*black-box*) para modelar as dinâmicas do sistema geralmente requer uma grande quantidade de dados e esta abordagem não generaliza bem. Pode-se evitar isto utilizando uma abordagem de caixa cinzenta (*grey-box*) que combina métodos de aprendizagem profunda com o conhecimento a priori da física.

Nesta tese, foi desenvolvido um método de controlo com aprendizagem em segurança que combina MPC e uma rede Lagrangiana profunda. A rede lagrangiana aprende o modelo dinâmico do sistema utilizando os conhecimentos físicos a priori. O MPC controla o sistema e garante a sua segurança impondo restrições lineares e não lineares, tendo em conta a incerteza associada com o modelo dinâmico enquanto o aprende. Os resultados experimentais foram obtidos utilizando um braço robótico simulado com 2 graus de liberdade a executar uma tarefa de rastreamento. Utilizando o método desenvolvido, o robô aprende o modelo do sistema online de maneira segura, enquanto cumpre o seu objetivo de seguir uma trajetória.

Keywords: Aprendizagem em segurança, Aprendizagem automática com antecedentes físicos, Model predictive control

Abstract

The use of autonomous robots for manufacturing is an emerging area in robotics. When these autonomous systems are learning their dynamical models or environments, it is critical that safety is ensured, which can go from avoiding collisions with obstacles to preventing violations of the system constraints. Additionally, using a black-box approach to model the system dynamics usually requires a large amount of data and do not generalize well. This is avoided by combining physics priors with deep learning methods in a grey-box approach.

In this work, it was developed a safety based learning control method that combines MPC and a deep Lagrangian network. The deep Lagrangian network learns the dynamical model of the system using physics priors. The MPC controls the system and ensures its safety by imposing linear or nonlinear constraints and accounting for the uncertainty of the dynamical model while learning. The experimental results were obtained with a simulated 2-DOF robotic arm executing a tracking task. By using this method, the robot is able to safely learn the model of the system online while also achieving the objective of tracking a trajectory.

Keywords: Safety based learning, Machine learning with physics priors, Model predictive control

Contents

Declaration . . . iii
Acknowledgments . . . vii
Resumo . . . ix
Abstract . . . xi
List of Tables . . . xv
List of Figures . . . xvii
List of Abreviations . . . xix

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Objectives . . . 4
1.3 Contributions . . . 4
1.4 Thesis Outline . . . 4

2 Background **6**
2.1 Safety Based Methods . . . 6
2.1.1 Gaussian Process . . . 6
2.1.2 Nonlinear Model Predictive Control . . . 7
2.1.3 Lyapunov Stability . . . 11
2.1.4 Hamilton-Jacobi Reachability . . . 12
2.2 Deep Learning . . . 13
2.2.1 Artificial Neuron Model . . . 13
2.2.2 Activation function . . . 14
2.2.3 Artificial Neural Networks Architecture . . . 15
2.2.4 Loss function . . . 16
2.2.5 Regularization . . . 16
2.2.6 Network training . . . 18
2.3 Mechanical System Dynamics . . . 20
2.3.1 Lagrangian Mechanics . . . 21
2.3.2 Lagrangian Dynamics for Mechanical Systems . . . 22
2.4 Related Work . . . 23

2.4.1 Safety based learning in robotics . . . 23

3 Deep Lagrangian Networks and Learning Based Control Model 25

3.1 Incorporating Lagrangian Mechanics into Deep Learning . . . 26

3.1.1 Symmetry and Positive Definiteness of the Inertia Matrix . . . 28

3.1.2 Deriving and Computing the derivatives . . . 28

3.2 Learning Based Control Model . . . 29

4 Results 31

4.1 Problem Description . . . 31

4.2 2-Degree of Freedom Arm Dynamics . . . 32

4.3 Model Predictive Controller of 2-Degree of Freedom Arm . . . 34

4.3.1 Model Predictive Control Problem . . . 34

4.4 Neural Network training, Trajectory Generation and MPC Parameters . . . 36

4.5 Training Validation . . . 36

4.6 Online training . . . 38

5 Conclusions 42

5.1 Future Work . . . 42

Bibliography 43

List of Tables

2.1 Number of nodes of a scenario tree for multi-stage NMPC for different lengths of the prediction horizon (N_p) and different number of uncertainties, assuming three branches per uncertainty [27]. 10
2.2 Common non-linear activation functions 14

4.1 Results obtained for the online training. 38

List of Figures

1.1 AERCam free flyer 2
1.2 SPHERES free flyer 3

2.1 ”Panel (a) shows four samples drawn from the prior distribution. Panel (b) shows the situation after two datapoints have been observed. The mean prediction is shown as the solid line and four samples from the posterior are shown as dashed lines. In both plots the shaded region denotes twice the standard deviation at each input value x” [24]. 7
2.2 MPC Scheme [25] 8
2.3 Scenario tree for the multi-stage approach. x_k^j represents the state in stage k and position j, u_k^j the control inputs and $d_k^{r(j)}$ is the uncertainty parameter, with the realization of the uncertainty r being a function of j. 9
2.4 Scenario tree for the multi-stage approach with a robust horizon. 11
2.5 Comparison between the biological neuron and the artificial neuron. 13
2.6 Neural Network architecture - Multilayer Perceptron with 3 hidden layers 15
2.7 Multiplayer perceptron approximating 4 different functions. The data points are show as blue dots, the resulting network functions are shown in red curves and the outputs of the hidden layers are displayed in by the dashed curves[43]. 16
2.8 Comparison between a neural network model that underfits the data (left), is a good fit (middle), or is overfitting (right) 17
2.9 Dropout Neural Net Model. (a) A standard neural net with 2 hidden layers. (b) An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped [44] 18
2.10 Difference between a convex loss function (left) and a nonconvex loss function (right) . . 18
2.11 Computation graph for y = f(x1, x2) = ln(x1) + x1x2 - sin(x2) [45]. 19

3.1 The bias-variance tradeoff as a function of model expressiveness [69]. 26
3.2 The computational graph of the Deep Lagrangian Network (DeLaN). Shown in blue and green is the neural network with the three separate heads computing g(q), ld (q), lo(q). The orange boxes correspond to the reshaping operations and the derivatives contained in the Euler-Lagrange equation. For training the gradients are backpropagated through all vertices highlighted in orange [49]. 27

3.3 (a) Computational graph of the Lagrangian layer. The green boxes represent the learnable parameters. The upper computational sub-graph corresponds to the standard network layer while the lower sub-graph is the extension of the Lagrangian layer to simultaneously compute $\frac{\partial h_i}{\partial h_{i-1}}$. (b) Computational graph of the chained Lagrangian layer to compute L, $\frac{dL}{dt}$ and $\frac{\partial L}{\partial q_i}$ using a single feed-forward pass [49]. 29
3.4 Full learning based control model architecture 30

4.1 Simulation environment. The blue and orange lines represent links 1 and 2 of the arm. The obstacle is represented by the blue circle. 31
4.2 2-degree of freedom robot arm 32
4.3 Neural network predicted torque (red) and test torque (black). (b), (c) and (d) represent the torque decomposed in inertial, Coriolis and centrifugal and gravitational matrices, respectively 37
4.4 The first graphic represents the error between the desired angles and the actual angle of the arm in joint 1 (blue line) and joint 2 (orange line). The two middle graphics show the desired angles (orange line) and the actual angle of the arm (blue line). The fourth graphic shows the input torque to joint 1 (blue line) and joint 2 (orange line) 37
4.5 Comparison between the torque applied to the system by the controller (black line) and the predicted torque by the neural network model (red line). 38
4.6 Error between the applied torque and predicted torque. 39
4.7 Comparison between the torque applied to the system by the controller (black line) and the predicted torque by the neural network model (red line) for each time the model was updated. 39
4.8 Error between the applied torque and predicted torque for each time the model was updated. 40
4.9 The first graphic represents the error between the desired angles and the actual angle of the arm in joint 1 (blue line) and joint 2 (orange line). The two middle graphics show the desired angles (orange line) and the actual angle of the arm (blue line). The fourth graphic shows the input torque to joint 1 (blue line) and joint 2 (orange line) 41
4.10 Distance between joint 1 (blue line) and joint 2 (orange line) to the obstacle. The arm remains safe if the blue and orange line remain above 0 41

List of Abreviations

DeLaN	Deep Lagrangian Networks
DOF	Degree of Freedom
GP	Gaussian Process
ISS	International Space Station
MLP	Multi Layer Perceptron
MPC	Model Predictive Control
MSE	Mean Squared Error
NN	Neural Network
NMPC	Nonlinear Model Predictive Control
ODE	Ordinary Differential Equation
ROA	Region of Atraction

Chapter 1

Introduction

This chapter presents the motivation for this work, the objective of this thesis as well as its contributions, and in the end, the outline of the thesis.

1.1 Motivation

The use and research of autonomous free-flying robots is emerging in recent years as the interest in space robotics and machine learning methods is increasing. These robots have multiple potential uses in space manufacturing including the construction of large structures, such as space telescopes too large to be launched into orbit by current launch technology, habitats composed of multiple modules, satellite servicing, and satellite deorbiting by attaching propulsive elements.

Free-flying space robots are composed by a manipulator attached to a floating base in which the manipulator contains a series of links, joints and one or more end-effectors [1]. The potential for free-flyers to perform challenging tasks in space such as in-orbit assembly, on-orbit servicing and refuelling, and debris removal has been acknowledged since the late eighties [2, 3].

Early research on free-flying space robots focused mainly on kinematics and dynamics. At the time, the free-floating base featured in this type of robots posed new modelling challenges when compared with fixed and mobile-based robots. In 1987, there were a couple of articles that addressed the kinematics modelling issues of a free-floating base. The Generalized Jacobian Matrix which provides a mapping between the joint and end-effector motions of a free-flying manipulator system was introduced in [2]. A different approach was introduced in [3] called the Virtual Manipulator approach, in this approach, all the motions in the system are referred with respect to the system center of mass that is fixed in space due to the nonexistence of external forces acting on the system.

In the nineties, the attention of the research turned to the control or mitigation of the free-flyer base only using the joint actuators of the manipulator. In 1991, [4] showed that the constraints imposed by the free-flying robots are nonholonomic using the Frobenius theorem, this means that the constraint cannot be expressed in terms of the position variables alone and must include the time derivative of one or more of those variables [5]. There was another problem that emerged in the nineties: how to deal

with dynamic singularities? These dynamic singularities are dependent on the path chosen for the joint variables because of the nonholonomic nature of the constraints. [6] demonstrated how to find regions of the end-effector workspace that are free from singularities (regardless of the path taken) and a path dependent region. By the end of the century, [7] presented the Reaction Null-space Control strategy that provided an effective way for path tracking the end-effector such that no disturbances are induced on the base of the spacecraft.

The kinematics and dynamics of the free-flyers were successfully validated, first in ground experiments [8], and later in flight experiments by the Japanese Space Agency in the ETS-VII mission [9], although the first free-flyer space operation was in nineteen ninety seven with the AERCam Sprint [10].

Figure 1.1: AERCam free flyer

Up to this time, both the ETS-VII experiments and the AERCam Sprint heavily relied on teleoperation and it was only in the next decade that the attention of the research community turned to application of free-flyers in other challenges and autonomy.

One of the applications of free-flying space robots is active debris removal, in which the spacecraft is tasked with the capturing of debris in space like a nonoperational satellite. This application rose new concerns: impedance control and momentum control. Impedance control is needed in order to avoid the creation of more debris in the moment of contact when capturing an object. In [9, 11], it was proposed a mass-damper-spring system model to control the motion of the end-effector and [12] later claimed to have developed a more computationally efficient method that took into consideration uncertainties in the physical properties of the target object, ensuring robust control of the end-effector. Momentum control addresses the problem of capturing a tumbling object by absorbing its angular momentum [13].

In 2007, the Orbital Express Program [14] launched and was tasked to demonstrate technologies such as satellite capture, autonomous rendezvous, on-orbit refuelling and autonomous fly around visual inspection of satellite. At the same time, the MIT developed a new brand of free-flyers to operate in the International Space Station (ISS) called SPHERES 1.2. These marked many new demonstrations in

terms of free-flyer technology: first time a spacecraft flew around another autonomously through the use of its cameras and without the sharing of navigation information between them; first time that a battery was transferred to another spacecraft autonomously using a manipulator arm.

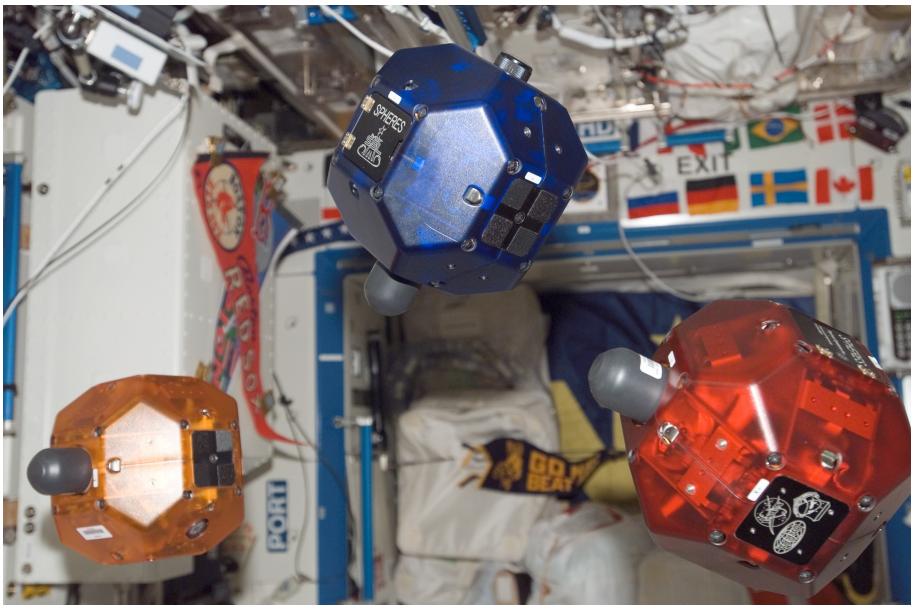


Figure 1.2: SPHERES free flyer

In the next decade, research brought forward topics like computational efficiency and modern control techniques like model-predictive control. The research done over the last years can be divided into three different categories: motion planning, control and modelling. Motion planning research focused for example on generating feasible trajectories for autonomous grasping free-tumbling debris [15]; development of a convex-programming-based guidance algorithm to capture a tumbling object on-orbit using a manipulator [16]. Research in control aimed at tracking previously generated reference trajectories using a nonlinear model predictive controller [17]. Finally, research in modelling tried to improve the knowledge in terms of model parameters and dynamics. [18, 19] estimate the inertial parameters of the combined robot and rigid body system by optimizing an exciting trajectory. The rigid body assumption is dropped which extended the class of objects that can be manipulated by the free-flyer. In [20, 21], in order the capture the dynamics of non-rigid bodies, it was used equivalent mechanical models like the spring-mass or pendulum.

The latest free-flyer operating in the ISS is the Astrobee [22] which is used for the most recent experiments involving free-flyers. The work done in [23] was tested in the Astrobee and delivered safe trajectories for in-orbit assembly combining model learning using information gain and model predictive control.

The main challenge approached in this thesis is safety, which goes from preventing actions that could harm the system to preventing actions that could harm the environment itself, e.g. collision avoidance with obstacles and other agents, poor handling of dangerous materials. To analyse safety and to accurately control the robot motion, a model of the system is usually needed. However, in a mobile manipulation scenario, dynamically models of the system are typically not fully known a priori. This mo-

tivates the use of machine learning methods to learn the system despite its uncertainties and control the robot safely, while trying to optimize the actions to reach its goal, with the best performance possible. This means that there is a compromise between safety and performance and a balance between the two is needed.

Finally, although the motivation for this thesis started with the autonomous free-flyers, the work done ended up being more general and can be applied, not only to free-flyers, but to any mechanical system.

1.2 Objectives

The objective of this thesis is to develop a learning based controller that provides safety guarantees to the system. The dynamical system proposed to test this controller is a basic 2 degree of freedom robotic arm, instead of a more complex free-flyer model. The robotic arm needs to learn its own dynamics and perform a designated task while guaranteeing safety, e.g. the robot cannot collide with obstacles or violate system constraints.

1.3 Contributions

The main contributions of this thesis are:

- Implementation of a new learning based control method that combines a Deep Lagrangian network with a Multi-Stage Robust Model Predictive Controller in order to safely learn the system dynamics.
- Elaboration of a method to model the system uncertainty with respect to the neural network online training performance.
- Simulation results for a robotic arm with 2 degrees of freedom showing that the robot can perform a task safely while learning the system dynamics

1.4 Thesis Outline

This thesis will follow the subsequent outline:

- Chapter 2: Theoretical Background and detailed explanations on concepts related with machine learning and safety based learning methods as well as Lagrangian mechanics. The chapter also provides the state of the art and related work on learning based methods with safety guarantees.
- Chapter 3: Presents the deep neural network that combines Lagrangian Mechanics and Deep learning methods. The full architecture of the learning based controller is also provided.
- Chapter 4: Presents the problem description and the dynamics and controller that were used. Provides the results obtained in both offline and online training and subsequent discussion and analysis.

- Chapter 5: Wraps the work done and provides a brief summary and conclusion of this thesis as well as sugestions for future work.

Chapter 2

Background

This chapter explains the key topics used throughout the work. It starts by reviewing some of the methods used to ensure safety while learning. Later, it is given an extended background on deep learning as well as the basics of mechanical systems using a Lagrangian. The chapter is finalized with the state of the art on safety based learning.

2.1 Safety Based Methods

2.1.1 Gaussian Process

Fundamentals

A Gaussian process (GP), which is well described in literature, namely in [24], is a generalization of the Gaussian probability distribution. The multivariate Gaussian distribution output is a vector where each entry corresponds to a Gaussian variable. To understand Gaussian processes, one must first think of a function as if it was a vector where every entry of x in the vector has the value of $f(x)$. Based on this, it can assumed that, the multivariate distribution returns a function instead of a vector or scalar.

The Gaussian process definition taken from [24] is as follows:

"Definition A Gaussian process *is a collection of random variables, any Gaussian process finite number of which have a joint Gaussian distribution."*

A Gaussian process is completely specified by its mean function and covariance function. The mean function $m(\mathbf{x})$ and the covariance function mean function $k(\mathbf{x}, \mathbf{x}')$ of a real process $f(\mathbf{x})$ are defined as:

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]\,, \tag{2.1}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))]\,, \tag{2.2}$$

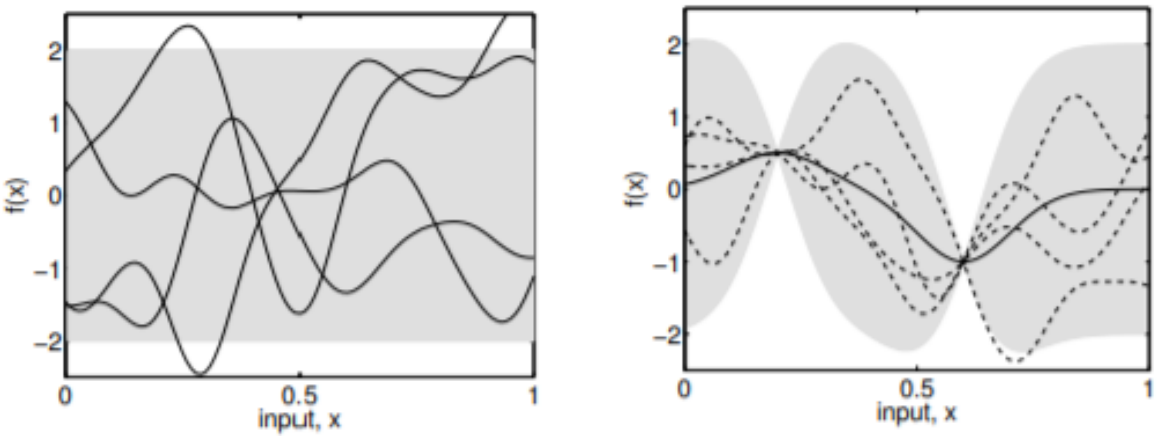
and the GP as

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')). \tag{2.3}$$

Prior and posterior distributions

Consider a simple one dimensional regression problem, where an input x maps to an output $f(x)$. Figure 2.1 shows a number of sample functions taken from the prior and posterior distribution. The prior distribution represents the samples of functions expected to observe without seeing any data. The shaded region denotes twice the pointwise standard deviation. The posterior distribution represents the samples of functions that pass through data points acquired through observations (it is possible to choose also functions that pass near these points). In figure 2.1 (b), after the two data points are added, the uncertainty around these points decreases, which is shown by the reduction of the shaded area near them.

When the system is initialized and no data has been gathered, there is an initial prior distribution of the model. However, when the system starts collecting data the prior distribution is updated into a posterior distribution, and when more data is collected it will then be updated again.



(a) Prior (b) Posterior

Figure 2.1: "Panel (a) shows four samples drawn from the prior distribution. Panel (b) shows the situation after two datapoints have been observed. The mean prediction is shown as the solid line and four samples from the posterior are shown as dashed lines. In both plots the shaded region denotes twice the standard deviation at each input value x" [24].

2.1.2 Nonlinear Model Predictive Control

Nonlinear Model Predictive Control (NMPC) is an optimization based method used in nonlinear systems for feedback control. [25].

The main applications of NMPC are tracking and stabilization problems. In a controlled process with states defined as $x(n)$ being each state measured at a discrete time instant n = 0, 1, 2..., a tracking control's objective is to determine the control input $u(n)$ in which $x(n)$ follows a given reference $x^{ref}(n)$ as well as possible. A stabilization problem is in essence a tracking problem but x^{ref} is a constant and does not change for every single instant.

One particular characteristic of NMPC is its ability to explicitly take constraints into account. These

constraints can be either on the state or on the control, or both. For this reason, NMPC is a method with good performance when dealing with constrained problems and that is why its use in safety based problems makes sense.

The NMPC scheme is as follows: for every sampling instant n the goal is to optimizes the predicted future behavior of the system limited over time, called horizon $k = 0, ..., N-1$ of length $N \geq 2$. The first element of the resulting optimal control sequence is then used as a feedback control value for the next sampling interval.

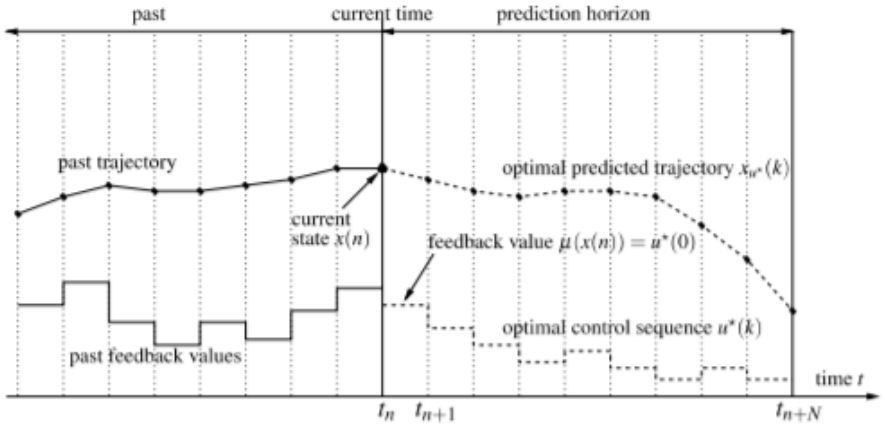


Figure 2.2: MPC Scheme [25]

The Algorithm

To track the robot in a given trajectory an algorithm for a time varying reference has to be considered. In [25], a number of algorithms are presented for different problems. In this report, the focus will be on the algorithm in section 3.3 of the book [25] where x^{ref} varies with time and represents a trajectory of the system.

First, one needs to define a cost function of the system, the cost function is 0 when the current state $x(n)$ is the same as $x^{ref}(n)$ independent to the fact that a control value was used in order to do so. When the state $x(n)$ does not coincide with $x^{ref}(n)$(current state is not in the predicted trajectory) the cost is positive.

$$\begin{aligned} &C(n, x^{ref}(n), u^{ref}(n)) = 0 \text{ for all } k \in \mathbb{N}_0, \\ &C(n, x(n), u(n)) > 0 \text{ for all } n \in \mathbb{N}_0 \text{ with } x \neq x^{ref}(n). \end{aligned} \tag{2.4}$$

The basic NMPC algorithm, taken from [25], is as follows:

Algorithm (Basic NMPC algorithm for time varying reference x^{ref}) At each sampling time:

1. Measure the state $x(n)$ of the system.

2. Set $x_0 = x(n)$, solve the optimal control problem

$$\begin{aligned}&\text{minimize} \quad J_N\big(n, x_0, u(\cdot)\big) := \sum_{k=0}^{N-1} \ell\big(n+k, x_u(k, x_0), u(k)\big)\\ &\text{with respect to} \quad u(\cdot) \in \mathbb{U}^N(x_0), \quad \text{subject to}\\ &x_u(0, x_0) = x_0, \qquad x_u(k+1, x_0) = f\big(x_u(k, x_0), u(k)\big)\end{aligned}$$

where J_N represents the sum of the cost function over an horizon, and $x_u(k, x_0)$ represents the prediction of the state for instant k.

3. Denote the obtained optimal control sequence by $u^*(\cdot) \in \mathbb{U}^N(x_0)$

4. Define the NMPC-feedback value as $u^*(0)$ and use this control value in the next sampling period.

Robust Multi-Stage Nonlinear Model Predictive Controller

Robust control is used to ensure that the control action satisfies the system constraints under the presence of uncertainty. The approach that was taken to achieve this was a multi-stage one [26, 27].

The idea behind the multi-stage approach is to consider various scenarios, where a scenario is defined by one possible realization of all uncertain parameters at every control instant within the horizon. The family of all the possible scenarios can be represented as a tree structure, named scenario tree. It is assumed that for the generation of the scenario tree, the minimum, maximum and nominal values of the uncertainties were taken.

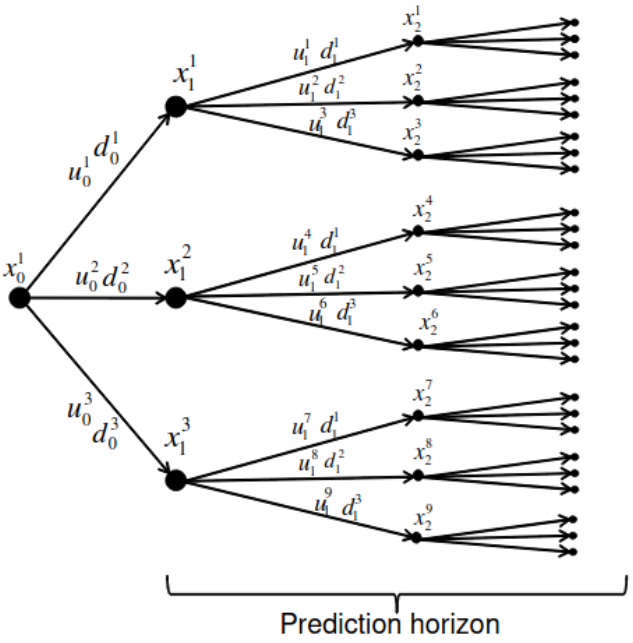


Figure 2.3: Scenario tree for the multi-stage approach. x_k^j represents the state in stage k and position j, u_k^j the control inputs and $d_k^{r(j)}$ is the uncertainty parameter, with the realization of the uncertainty r being a function of j.

A scenario corresponds to one path from the root node in the left side of the tree to a leaf node on the right. One example would be scenario S4 that follows the path: $x_0 \rightarrow x_1^2 \rightarrow x_2^4 \rightarrow x_3^4 \rightarrow x_4^4$. Starting with the root node, the MPC problem is solved while taking into account different values for the uncertainty parameters which lead to the different branches in the tree. The decisions u branching from the same node are identical because they were based on the same information, this means that u_k^j is identical for the same values of k. The system equation for the multi stage model is given by:

$$x_{k+1}^j = f(x_k^{p(j)}, u_k^j, z_k^{p(j)}, d_k^{r(j)}) \tag{2.5}$$

where the function p(j) refers to the parent state of x_k and the considered realization of the given uncertainty is given by r(j) via $d_k^{r(j)}$.

The main challenge of the multi-stage approach is the rapid growth in the size of the scenario tree with respect to the prediction horizon. This problem is commonly known as the curse of dimensionality and table 2.1 shows how the number of scenarios to be evaluated increases with respect to the predicting horizon and the number of uncertain parameters.

	# of uncertainties			
N_P	1	2	3	4
1	3	9	27	81
2	9	81	729	6,561
3	27	729	19,683	531,441
4	81	6,561	531,441	43,046,721
5	243	59,049	14,348,907	3,486,784,401
6	729	531,441	387,420,489	28,242,953,648

Table 2.1: Number of nodes of a scenario tree for multi-stage NMPC for different lengths of the prediction horizon (N_p) and different number of uncertainties, assuming three branches per uncertainty [27].

The strategy taken to deal with this problem is to assume that, after a certain point, the uncertainty remains constant and stops branching. This simplification is justified because modelling accurately in a point in time far into the future is not very critical as the control inputs are recomputed for every timestep. Figure 2.4 shows the scenario tree with the proposed simplification.

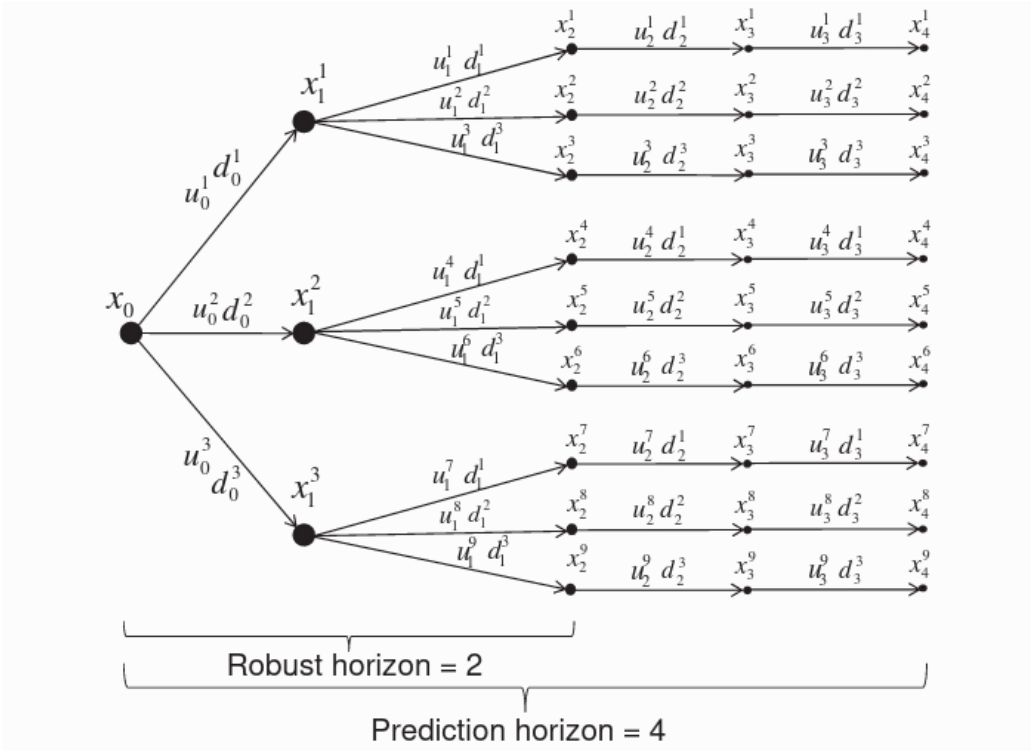


Figure 2.4: Scenario tree for the multi-stage approach with a robust horizon.

2.1.3 Lyapunov Stability

Equilibrium Points Stability and Region of Attraction

An equilibrium point can be characterized in terms of Lyapunov stability in the following manner [28]:

- Stable: if all solutions starting at nearby points stay nearby;
- Asymptotically stable: if all solutions starting at nearby points not only stay nearby, but also tend to the equilibrium point as time approaches infinity;
- Unstable: otherwise.

The region of attraction (ROA) is a safe subset of the state space in which a given controller renders an equilibrium point asymptotically stable [29]. The ROA of the closed-loop is of special interest for safety based control, as it represents the subset of the state space that is forward invariant, where any state trajectory starting within this subset stays within it forever and eventually converges to a goal state [30]. The computation of ROA for a fixed policy uses Lyapunov Functions [28].

Lyapunov functions are continuously differential functions $v : \mathcal{X} \to \mathbb{R}_{\geq 0}$ with $v(0) = 0$ and $v(\mathrm{x}) > 0$ for all $\mathrm{x} \in \mathcal{X} \setminus \{0\}$. Showing the stability of a system using Lyapunov functions is similar to the application of the gradient descent method on quasiconvex functions. If, given a policy π, applying the dynamics of the system on the state maps it to strictly smaller values on the Lyapunov function ("going downhill"), then, this means that the state will eventually converge to the equilibrium point at the origin [30].

2.1.4 Hamilton-Jacobi Reachability

Reachability analysis

Reachability analysis is extremely powerful for the formal verification of the safety of dynamical systems. ”In reachability analysis, one computes the reach-avoid set, defined as the set of states from which the system can be driven to a target set while satisfying time-varying state constraints at all times.” [31].

Consider the following example: A vehicle is constrained to drive in a straight line along a road with a traffic light that turns yellow as the vehicle approaches. The vehicle cannot be passing through the intersection with a red traffic light on. The options then are to either accelerate or slam on the brake. However, depending on the vehicle speed, distance to the intersection, and time, the traffic violation might be inevitable [32]. Reachability determines the scenarios, called states, from which the vehicle, in this case, can remain safe, and it also provides the action that should be taken. Reachability analysis usually needs a model of the system to operate and the accuracy of the results obtained increase as the model becomes more accurate. However. it has been seen in recent papers and studies that this approach can be used in systems with uncertain models [33], [32], [34].

Hamilton-Jacobi Reachability Towards Safety

Hamilton-Jacobi reachability analysis is the most general formulation, having already a large number of papers published specifically dealing with safety problems [33], [32], [34] and multiple numerical toolboxes already developed for future testing [35], [36], [37].

Safety in a control system can be translated into a constraint satisfaction problem as there are regions of the state space that the system has to avoid and it should only operate in regions that comply every constraint. Hamilton Jacobi Reachability ability to handle a wide variety of constraint-satisfaction problems, as well as, its capability to be applied to general controlled nonlinear systems with disturbances or adversarial behavior is what captivates the use of this method to obtain safety guarantees in control systems.

The main idea of this method is to define a functional that scores trajectories (and control signals) based on their performance in achieving a desired objective (reaching the target, staying within the constraint, etc.), and then to evaluate the states based on the score of the optimal trajectories [32]. There are two different ways of posing a safety problem when working with Hamilton-Jacobi Reachability analysis. In an optimal control problem, the system has only one set of inputs and the trajectories are optimized through its inputs. In a two-player zero-sum differential the ”two players” can be the system and the disturbances [33]. The system tries to maximize the outcome of the game by not breaking any constraints and keeping its best performance, while the disturbance tries to minimize this outcome and tries to drive the system into breaking the constraints. In the end, the two player game always ends up having a worst case analysis of the system.

The main challenge and limitation of Hamilton-Jacobi Reachability lies in its expensive computational cost, which scales exponentially with the number of dimensions in the system. For this reason, systems with more than 4 dimensions are usually unsolvable when using this method.

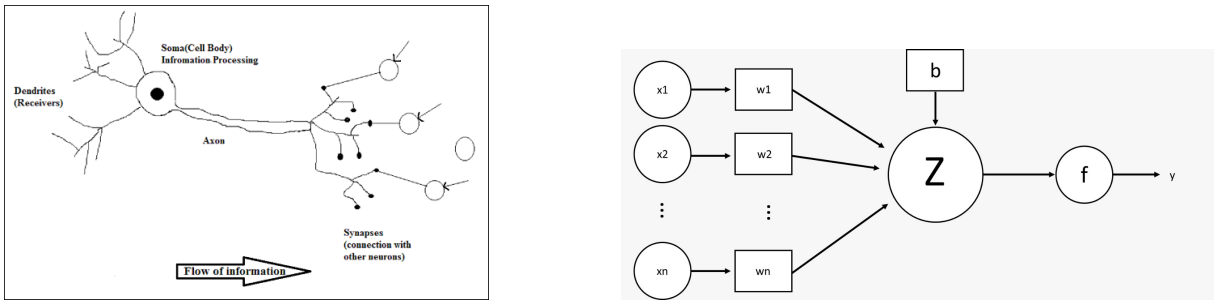
2.2 Deep Learning

2.2.1 Artificial Neuron Model

The biological neuron is a nerve cell that processes information and communicates with other neurons via specialized connections called synapses. The neuron is composed by:

- **Dendrites**: Tree-like branches, responsible for receiving the information from other neurons it is connected to;
- **Soma**: Cell body of the neuron, responsible for processing the information received from dendrites.
- **Axon**: Filament extruding from the soma, responsible for sending the information.
- **Synapses**: Connection between the axon and other neuron dendrites.

Based on the description of each component, one can make the analogy between the biological neuron and the artificial neuron in which the dendrites and axon correspond to the input and output of each node (soma).



(a) Biological neuron

(b) Perceptron

Figure 2.5: Comparison between the biological neuron and the artificial neuron.

Let $x_i \in \mathbb{R}^{n_x}$ be the set of input to the neuron, where n_x is the input size, and $w_i \in \mathbb{R}^{n_x}$ be the set of weights. Lets consider also that the input feature $x_0 = 1$ and $w_0 = b$, where b represents the bias term. The net input, z, can be calculated as a weighted sum of each input and the bias:

$$z = w_1 x_1 + w_2 x_2 + ... + w_n x_n + b \tag{2.6}$$

The output of the perceptron can then be calculated by applying the activation function on the net input:

$$y = f(z) \tag{2.7}$$

where f is the activation function, which will more elaborated in the following subsection.

2.2.2 Activation function

Activation functions are used in artificial intelligence in order to help the network learn complex patterns in the data. The activation function takes the input of the neuron and transforms it into an output restricted to a limit which is then passed to other neurons. The main reasons for using non-linear activation functions is to keep the output of each neuron from exceeding a certain limit, which in the case of deep neural networks would lead to computational issues, and to add non-linearity into a neural network so it can learn patterns with a greater complexity.

Some of the desirable features of an activation function are:

- **Vanishing gradient problem**: Neural networks are trained using the gradient descent method when the backward propagation step (described in section 2.) is applied. For this reason, it is crucial that the activation function does not shift the gradient values to zero causing every layer to be unable learn;
- **Zero-Centered**: Output of the activation function should be symmetrical at zero so that the gradients do not shift to a particular direction.
- **Computational Expense**: Activation functions are applied after every layer and need to be calculated millions of times in deep networks. Hence, they should be computationally inexpensive to calculate.
- **Differentiable**: As mentioned, neural networks are trained using the gradient descent process, hence the layers in the model need to differentiable or at least differentiable in parts. This is a necessary requirement for a function to work as activation function layer.

In the table below are listed some of the most common activation functions:

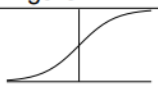
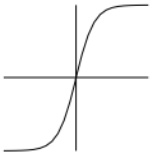
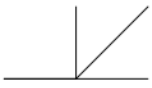
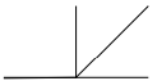
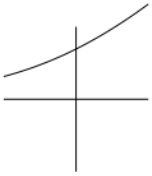
Name	Function	Derivative	Figure	Range
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	$g'(x) = g(x)(1-g(x))$		(0,1)
tanh	$\sigma(x) = \frac{e^{x}-e^{-x}}{e^{z}+e^{-z}}$	$g'(x) = 1-g(x)^2$		(-1, 1)
ReLU	$g(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0. \end{cases}$	$g'(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } 1 \geq 0. \end{cases}$		[0, ∞)
Leaky ReLU	$g(x) = \begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0. \end{cases}$	$g'(x) = \begin{cases} 0.01 & \text{if } x < 0 \\ x & \text{if } 1 \geq 0. \end{cases}$		[0, ∞)
Softplus	$g(x) = \ln(1+e^{x})$	$g'(x) = \frac{1}{1+e^{-x}}$		(0, ∞)

Table 2.2: Common non-linear activation functions

2.2.3 Artificial Neural Networks Architecture

The general architecture of a feed-forward networks consists of a series of multiple neurons stacked together forming multiple layers where each neuron is connected to all the neurons of the adjacent layer, as shown in figure 2.6.

The layers are characterized into three different types:

- Input layer: Single layer that feeds the data into the neural network. The number of neurons is determined by the number of input features.
- Hidden layer: Intermediate layer that does all the computations and extract the features from the data. Depending on the complexity of the problem, the number of hidden layers vary.
- Output layer: Single layer that outputs the prediction of the model. The number of neurons is determined by the number of outputs the problem.

This Neural Networks architecture is forward in nature because the information enters the model through the input layer, flows through the series of hidden layers, and finally through the output layer without ever looping back to previous layers.

The depth of a neural network is defined by the number of layers it contains while the width is determined by the number of neurons that form each layer.

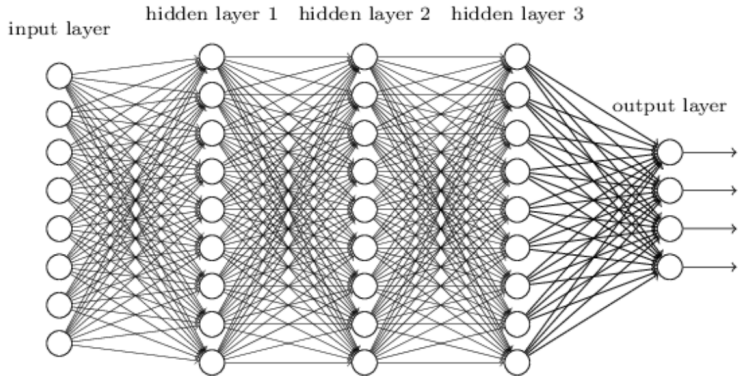


Figure 2.6: Neural Network architecture - Multilayer Perceptron with 3 hidden layers

Feed-forward networks have been widely studied for their general approximation properties [38–42]. For this reason, neural networks are considered to be universal approximators. This means that a network can represent any continuous function provided that there are sufficient hidden units and hidden layers [43]. The main challenge then resides in training the network, where the suitable parameter values need to be found given a set of training data and there is also the issue of overfitting which will be discussed in the next subsection.

Figure 2.7 shows how a neural network with two layers is able to approximate multiple functions and how the hidden layers work together in order to achieve the final prediction.

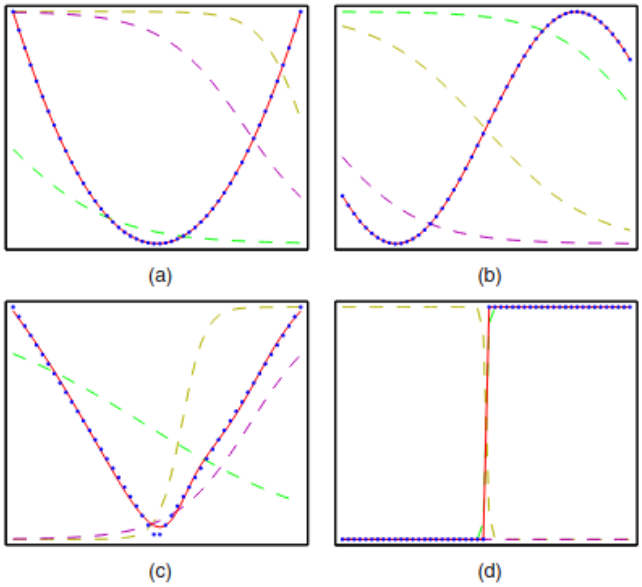


Figure 2.7: Multiplayer perceptron approximating 4 different functions. The data points are show as blue dots, the resulting network functions are shown in red curves and the outputs of the hidden layers are displayed in by the dashed curves[43].

2.2.4 Loss function

A deep learning problem is usually seen as an optimization problem where an objective or loss function is minimized or maximized in search for the parameters that provide the best solution. Neural networks seek to minimize the error or loss.

The loss function or cost function is a measure of how far the predicted result is from the true value. In a neural network, the loss is a function of the parameters of the network such as the weights and biases.

One of the most common loss functions is the mean squared error (MSE), regularly used on linear regression problems:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2, \tag{2.8}$$

where y_i is the the true value and $\hat{y}_i$ is the predicted value of the function in position i, and n is the number of datapoints.

In a neural network, one can define the loss function using the MSE as follows similarly but, y_i is the the true value and $\hat{y}_i$ is the predicted value of the output of the neural network on the example i, and n is the number of training examples.

2.2.5 Regularization

Overfitting is the phenomenon in which a neural network has a high performance on the training data, but fails to perform well on a different set of data, from the same problem domain. This usually happens

because the neural network learns the noise as an underlying concept of the data and it means that the model as high variance and low bias. Bias is the difference between the average prediction of the model and the true value of the function or parameter while variance is the variability of model prediction for a given data point or a value which indicates the data spread.

A neural network with high complexity is usually more susceptible to overfitting. The model is able to learn patterns in the data that were just caused by some random fluctuation or error.

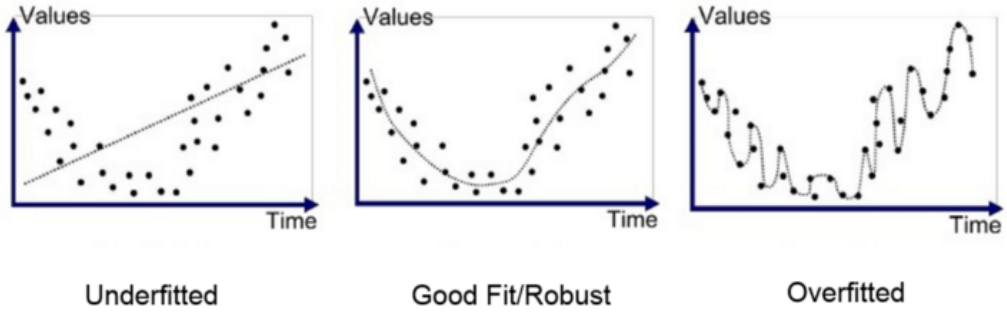


Figure 2.8: Comparison between a neural network model[2] that underfits the data (left), is a good fit (middle), or is overfitting (right)

The set of methods used to prevent and overcome the problem of overfitting are called regularization. Regularization prevents overfitting by adding extra information to the model. One example is the addition of a penalty term to the cost function in order to discourage the coefficients from reaching larger values, penalizing the model complexity. It is important that the regularization method used finds the best balance between bias and variance as there is always a trade-off between the two.

Ridge Regression

Ridge regression or L2 regularization is the most common regularization technique. Starting with the regularization term defined:

$$\Omega(W) = \frac{\lambda}{2}||W||_2^2 \tag{2.9}$$

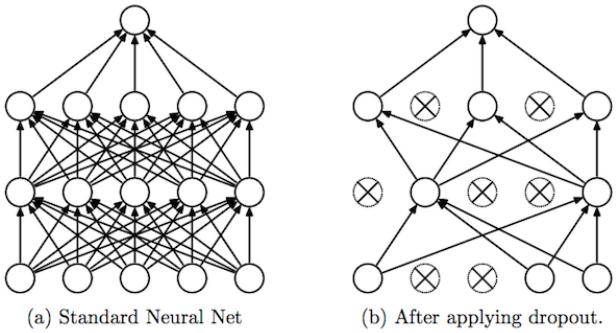
in which W is the weight matrix of the network and λ the regularization parameter, adding Ω to the loss function of the neural network shrinks the weight values towards zero. By penalizing large weights, the variance of the model is reduced preventing overfitting.

Dropout

Dropout was first introduce in 2014 [44]. In dropout regularization, there is a probability for each neuron to remain unused during the training of the model, this happens during the forward propagation and weight update step. This results in a neural network that is not overly dependent on some neuron

[2]Image taken from website: https://towardsdatascience.com/overfitting-vs-underfitting-a-conceptual-explanation-d94ee20ca7f9

and forces the network to learn a more balanced representation avoiding overfitting. In figure 2.9 is represented an example of the dropout regularization with probability of 50%



(a) Standard Neural Net (b) After applying dropout.

Figure 2.9: Dropout Neural Net Model. (a) A standard neural net with 2 hidden layers. (b) An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped [44]

2.2.6 Network training

In the previous sections, it was presented the architecture of neural networks and some of the methods and concepts that are used in order to approximate nonlinear functions. However, there is still one important component to discuss, the training. Training the network basically means adjusting the parameters of the network until the output is close enough to the expected.

Artificial neural network training is the problem of minimizing a large-scale nonconvex cost function. In a linear regression problem, the loss function is convex which ensures a global minimum, that means that by minimizing the loss function it will eventually reach the global minimum. However, in order to model arbitrary functions, a neural network needs to be nonlinear making the loss function nonconvex, with multiple local minimums. For this reason, the minimum of the function is almost impossible to find in a single step only. This problem is not limited to machine learning but is indeed a very general problem in mathematical optimization. The most common approaches to resolve this issue rely on the use of the gradient descent method.

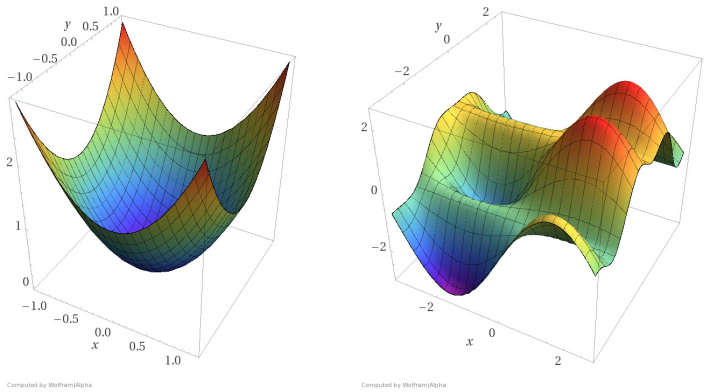


Figure 2.10: Difference between a convex loss function (left) and a nonconvex loss function (right)

Automatic Differentiation

In deep learning, it is required to compute and evaluate derivatives in order to minimize the loss function. Besides automatic differentiation, there are three methods used in computer programs to compute the derivatives of a function:

- Manual differentiation: The derivatives are manually computed and then coded;
- Numerical differentiation: The derivatives are numerically approximation using finite differences;
- Symbolic differentiation: The derivatives are calculated using expression manipulation in computer algebra, essentially an automated version of the Manual differentiation.

Manual differentiation can be too tedious and prone to error and, like the name suggests, is not automated. Numerical differentiation is easy to implement, but suffers from issues like its inaccuracy due to truncation and rounding errors [45], and the fact that it scales poorly for gradients, which makes it useless for machine learning problems with a large number of parameters. Symbolic differentiation addresses all of these problems but raises others, such as, the highly complex and cryptic expressions that suffer from ”expression swell” [45]. In addition, manual and symbolic differentiation require the function to be expressed in closed form, limiting the ability to use control flow mechanism like conditionals, loops and recursions [45].

Automatic differentiation uses symbolic differentiation rules to produce numerical values of derivatives [45], and for this reason, it is considered to partly numerical and partly symbolic. Until recently, automatic differentiation had been underused in the machine learning community, but with the emergence of deep learning, there are many projects [46] that are turning automatic differentiation into a mainstream concept. This is because automatic differentiation allows for accurate derivative computations, is computationally efficient, and requires minimal code changes in order to be used [45].

When a computer program uses automatic differentiation, it computes the derivatives by decomposing the program into its primitive operations, e.g. binary arithmetic operations, transcendental functions like the exponential, the logarithm, and the trigonometric functions, etc, with know derivatives. The derivative of the overall decomposition is obtained by combining the derivative of each constituent operation using the chain rule. Figure 2.11 shows a computation graph for an example function,

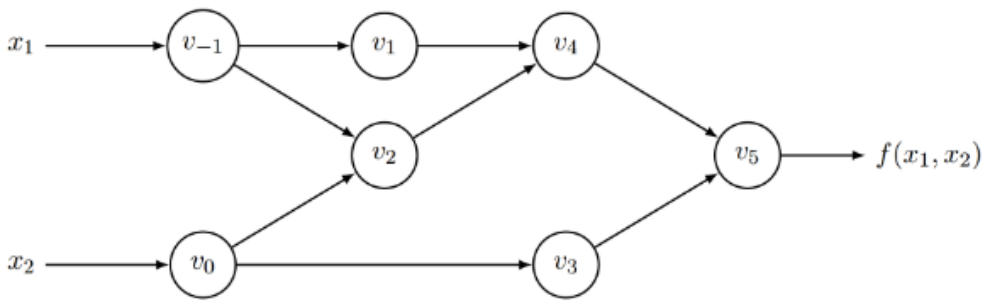


Figure 2.11: Computation graph for y = f(x1, x2) = ln(x1) + x1x2 - sin(x2) [45].

The two modes of automatic differentiation, forward mode (left to right) and reverse mode (right to left) which correspond to the direction in which the chain rule is evaluated. Starting with forward mode as it is conceptually simpler. Forward auto differentiation involves augmenting each intermediate variable during evaluation of a function with its derivative, so instead of individual floating point values flowing through a function, one can think of these as tuples of individual intermediate values also called primals paired with their derivatives also know as tangents. Forward mode has constant memory requirements and computation complexity depends on the number of inputs [45].

In reverse mode, rather than propagating derivatives forward they will be propagated backwards from the output. This is carried out in a two part process. First with the forward pass through the function evaluating intermediate variables as before but instead of simultaneously computing derivatives, the dependencies of the expression tree is instead store in memory. After the completion of the forward pass, the partial derivatives of the output with respect to the intermediate variables quantities, known as adjoints, are computed as:

$$\bar{v}_i = \frac{\partial f}{\partial v_i} \tag{2.10}$$

which represents the sensitivity of a considered output f with respect to changes in v_i. Reverse mode corresponds to a more general back-propagation method.

Gradient Descent Method and Back-propagation

In a general way, gradient descent starts with an initial guess of the parameters and from there, tries to determine the direction in which the loss function steeps downwards the most, and steps into that direction. This process is repeated until the the point achieved is as close to the minimum as possible. To be able to figure out the direction in which the loss function values decrease the most, it is necessary to calculate the gradient of the function with respect to all the parameters. A gradient is a multidimensional generalization of a derivative; it is a vector that contains all the partial derivatives of the function with respect to each variable.

The back-propagation algorithm was first used for optimizing neural networks in [47]. The introduction of the back-propagation in neural networks revolutionized the machine learning world in the eighties and enabled the huge success that neural networks have today.

The back-propagation algorithm is an efficient and accurate method for computing the gradients of complex models, that can have an extensive number of parameters,such as neural networks. When training the model, the network first starts by applying the forward pass in which the loss is computed. Then, the backward pass is applied in order to compute the gradients of the learnable parameters.

2.3 Mechanical System Dynamics

Newtonian mechanics are based on the very well known vectorial equation F = ma, being F the applied force and m and a the mass and acceleration of the system, that relates the external forces

and moment vectors to the the time rate of change of the translational and angular momenta. This approach is called a direct approach due to the direct relationship between the imposed force and the system motion [48]. Although Newtonian mechanics can be applied to all mechanical systems, they lack efficiency when the Cartesian coordinates (x,y,z) do not give the simplest description of a system. One example is the pendulum which can be analyzed more easily in relation to the swing angle instead of the Cartesian coordinates.

A different approach, named Lagrangian mechanics, used the principles of work and energy, regularly called indirect or analytical approach. The Lagrangian mechanics starts the system analysis with the Lagrangian, a scalar-like function. Then, by applying D'Alembert's Principle of Virtual Work or Hamilton's Principle of Least Action, a constraint is placed on the Lagrangian which finally yields the equations of motion for the system [48].

Describing the equations of motion for mechanical systems through the use of Lagrangian mechanics has been extensively studied in literature. This thesis will follow the demonstrations and notation used in [48] which does a good job explaining this subject.

2.3.1 Lagrangian Mechanics

Let **q** represent the vector of generalized coordinates that completely define the configuration of a mechanical system structure relative to a reference configuration, $T(q, \dot{q}, t)$ the kinetic energy of the system and $V(q, t)$ the potential energy of the system. The Lagrangian, given in (2.11), is a function of generalized coordinates **q** that completely describe the dynamics of a given system.

$$\mathcal{L}(q, \dot{q}, t) = T - V \tag{2.11}$$

The Hamilton principle of least action states that the true trajectory of q(t) between known starting and ending points $q(t_0)$ and $q(t_1)$ will minimize or maximize the action I [38, 17], defined as:

$$I = \int_{t_0}^{t_1} \mathcal{L}(q, \dot{q}, t) dt \tag{2.12}$$

The true system motion always minimizes the action I, hence the name least action principle, even though there are an infinite number of trajectories that connect the $q(t_0)$ and $q(t_1)$.

The Hamilton principle is enforced by setting the first variation of the action to zero:

$$\delta I = \delta \int_{t_0}^{t_1} \mathcal{L}(q, \dot{q}, t) dt = 0 \tag{2.13}$$

Computing the variation of the action using integration by parts satisfying the initial and final conditions $\delta q(t_0) = \delta q(t_1) = 0$ leads to:

$$\begin{aligned}\delta I = \delta \int_{t_0}^{t_1} \mathcal{L}(q, \dot{q}, t) dt &= \int_{t_0}^{t_1} \left[\frac{\partial \mathcal{L}}{\partial q} \cdot \delta q + \frac{\partial \mathcal{L}}{\partial \dot{q}} \cdot \delta \dot{q} \right] dt \\ &= \int_{t_0}^{t_1} \left[\frac{\partial \mathcal{L}}{\partial q} + \frac{d}{dt} \left(\frac{\partial \mathcal{L}}{\partial \dot{q}} \right) \right] \delta q dt + \left[\frac{\partial \mathcal{L}}{\partial \dot{q}} \cdot \delta \dot{q} \right]_{t_0}^{t_1} \\ &= \int_{t_0}^{t_1} \left[\frac{\partial \mathcal{L}}{\partial q} + \frac{d}{dt} \left(\frac{\partial \mathcal{L}}{\partial \dot{q}} \right) \right] dt,\end{aligned} \tag{2.14}$$

Combining equations (2.13) and (2.14) one obtains the Euler-Lagrangian equation:

$$\frac{d}{dt} \left(\frac{\partial \mathcal{L}}{\partial \dot{q}} \right) - \frac{\partial \mathcal{L}}{\partial q} = 0 \tag{2.15}$$

Accounting for Nonconservative Forces

External forces in Lagrangian forces can vary from a control actuator to dissipation or friction. The Euler-Lagrangian equation derived in (2.15) assumed that the system was not subject to external forces and only to generalized forces which derived from a scalar potential. To take these additional generalized forces Q into account, the action I presented in (2.12) needs to be modified in order to incorporate the work on the the system due to the generalized forces W_Q.

$$I = \int_{t_0}^{t_1} \left[\mathcal{L}(q, \dot{q}, t) + W_Q \right] dt. \tag{2.16}$$

Applying the calculus of variation to now yields the following Euler-Lagrange equation:

$$\frac{d}{dt} \left(\frac{\partial \mathcal{L}}{\partial \dot{q}} \right) - \frac{\partial \mathcal{L}}{\partial q} = Q. \tag{2.17}$$

A rigorous derivation of (2.17) using D'Alembert's Principle of Virtual Work is given in [48].

2.3.2 Lagrangian Dynamics for Mechanical Systems

A mechanical system is composed by a set of generalized coordinates $\mathbf{q} \in \mathbb{R}^n$, their correspondent rate of changes $\dot{\mathbf{q}} \in \mathbb{R}^n$, called generalized velocities, and a control input **U** that defines the external forces applied to the system. The Lagrangian of a Mechanical system is given by (2.11) where the Kinetic energy T is:

$$T(q, \dot{q}) = \frac{1}{2} \dot{q}^T H(q) \dot{q}. \tag{2.18}$$

where H(q) is the symmetric and positive definite generalized inertia matrix, the positive definiteness ensures that all non-zero velocities lead to positive kinetic energy [49]. Replacing the kinetic energy in (2.11) with (2.18) results in:

$$\mathcal{L}(q, \dot{q}, t) = \frac{1}{2} \dot{q}^T H(q) \dot{q} - V(q). \tag{2.19}$$

This Lagrangian is not unique and every $\mathcal{L}$ which yields the correct equations of motion is valid.

Applying the calculus of variation results in the Euler-Lagrangian equation in (2.17) and by replacing L with (2.19) and then dV/dq = g(q) yields the second order ordinary differential equation (ODE) described by:

$$H(q)\ddot{q} + \dot{H}(q)\dot{q} - \frac{1}{2}\left(\frac{\partial}{\partial q}(\dot{q}^T H(q)\dot{q})\right)^T + g(q) = U. \tag{2.20}$$

The equation in (2.20) can also be rewritten as:

$$H(q)\ddot{q} + C(q,\dot{q}) + g(q) = U. \tag{2.21}$$

where g(q) is the gravitational potential vector and $C(q,\dot{q})$ describes the forces generated by the Centripetal and Coriolis forces.

2.4 Related Work

2.4.1 Safety based learning in robotics

The research and interest in safety based learning methods in control and artificial intelligence has been rapidly increasing over the last decade, [50] provides a complete review of the recent advance made in machine learning towards safety under uncertainties. The research done in this report revolves around the methods developed more recently, even though this problem has been around since the turn of the century, [51] uses Lyapunov-based reinforced learning to allow the learning agent to switch between pre-computed ”base-level” controllers with preferable safety and performance properties. Although this provided solid theoretical guarantees, the agent's behavior was extremely constrained.

Model Predictive Control

Model predictive control has established itself as the primary control method for the systematic handling of system constraints, [52] reviews and summarizes recent research on learning based MPC and addresses its use for safe learning. MPC in combination with Gaussian process models are proposed in: [53], where a safe MPC scheme is introduced (SafeMPC) that guarantees the existence of return trajectories to safe regions of the state space at any time and with high probability; [54] for modelling and controlling of unmanned quadrotors; [55] to safely increase the performance of autonomous miniature race cars; and [56] where, based on generated uncertain models, the controller computes linear and angular velocities in real time while the robot does not violate safety constraint, increasing its performance as the model uncertainty is reduced with experience.

Hamilton-Jacobi Reachability Analysis

Starting with Hamilton-Jacobi Reachability, the reader is first directed to [31] where an overview of basic Hamilton Jacobi-Reachability theory is given as well as instructions for using the most recent nu-

merical tools for computing reachable sets. Through the use of these tools, there has been already multiple differential games and optimal control problems solved with Reachability analysis. Some examples include aircraft auto-landing [57], automated aerial refueling [58], multiplayer reach-avoid games[59] and real time safe motion planning [60]. Most of the papers researched in this area studied the application of this method in uncertain systems, in [32], [34] and [33] Gaussian process is paired with Reachability methods in order to achieve safety under uncertainties. In [33] a framework is proposed using Hamilton Jacobi reachability methods where the system starts out conservative and, as more data is acquired, it becomes less conservative while guaranteeing safety. The proposed framework is then successfully demonstrated experimentally on a quadrotor vehicle. There has also been some advances in order to solve the high computational cost of Hamilton-Jacobi Reachability analysis for systems with an increased number of dimensions. [61], [62] try to solve this challenging problem by decomposing the computation of the reachable sets into smaller dimensional computations.

Lyapunov Stability

Lyapunov functions are used for stability certification of dynamical systems as well as estimating the Region of Attraction which is the reason why its use in safety based problems has been prominent. In [30], [63], [64], [65], multiple Lyapunov approaches to reinforced safe learning are proposed with the use of Gaussian process to deal with the system uncertainties or through the use of Constrained Markov decision problems algorithms. [66] proposes the development of a neural network Lyapunov function and a training algorithm that adapts it to the shape of the largest safe region in the state space, only with knowledge of the inputs and outputs of the dynamics. The issue of model uncertainty in safety critical control can also be addressed with the use of barrier functions. In [67] it is proposed the use of Control Barrier Functions as well as Control Lyapunov Functions to develop a method used in a bipedal robot. In [68] the use of Control Lyapunov functions is also used and compared with a robust MPC.

Chapter 3

Deep Lagrangian Networks and Learning Based Control Model

This chapter will present the neural network architecture that was used to learn the dynamics of the mechanical system and will also describe the full architecture of the learning based controller.

The problem presented in this thesis is to learn the dynamical model of a mechanical system and control it, in order to safely perform a certain task. To successfully engineer a controller for a robotic domain, one needs to rely on accurate models of the system that is going to be controlled. There are two main approaches to obtain these models:

- **White-box model**: This approach makes assumptions about the system like its kinematic structure, inertia properties and forces acting on it, heavily relying on prior knowledge of physics models to make predictions. This approach usually produces models with high bias, which can be defined as error between the average model prediction and the ground truth;

- **Black-box model**: This approach treats the system's equations of motions as if they were any other function and tries to use machine learning techniques to fit the data, optimizing the parameters and minimizing a loss. This approach usually produces models with high variance, which can be defined as the variability of model prediction for a given data point, and require a large amount of training data.

The approach that was taken in this thesis would be one called a grey-box approach. This approach uses the physical models of the system for which these models are accurate, and learns from data the parts of the system dynamics that are difficult to model. This way, the model can find a better bias-variance tradeoff.

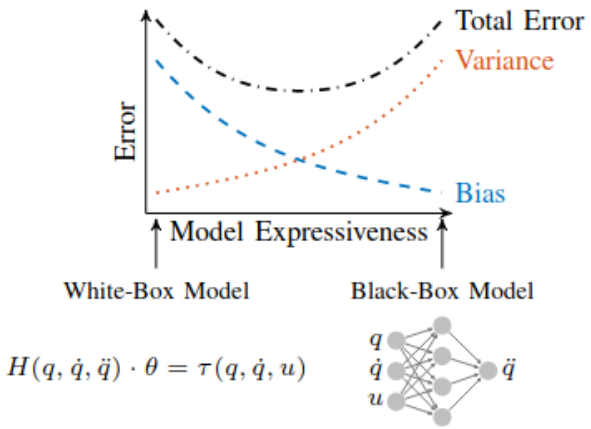


Figure 3.1: The bias-variance tradeoff as a function of model expressiveness [69].

3.1 Incorporating Lagrangian Mechanics into Deep Learning

The work in this thesis was inspired by the work done in [49] where the Lagrangian is incorporated in the learning problem. By doing this, instead of learning the model dynamics directly from data or trying to estimate H(q) and g(q) from the masses, lengths and moments of inertia of the system, this approach tries to learn the unknown functions g(q) and H(q), from the ODE in (2.21), using a neural network. Instead of learning the matrix H(q) directly, the neural network learns the lower triangular matrix L(q), which will be further discussed in 3.1.1. The matrices H(q) and g(q) are represented in the neural network as:

$$\hat{H}(q) = \hat{L}(q;\theta)\hat{L}(q;\theta)^T \quad and \quad \hat{g}(q) = \hat{g}(q;\psi) \tag{3.1}$$

where $\hat{}$ represents the the network approximations and θ and ψ represent the network learning parameters. The network parameters are learned by solving an optimization problem that consists in minimizing the violation of the Lagrangian mechanics physical law obtained in (2.21). The optimization problem is the following [49]:

$$(\theta^*, \psi^*) = \underset{\theta,\psi}{\operatorname{argmin}} \quad L\left(\hat{f}^{-1}(q, \dot{q}, \ddot{q}; \theta, \psi), \tau\right) \tag{3.2}$$

$$\begin{aligned} with \quad \hat{f}^{-1}(q, \dot{q}, \ddot{q}; \theta, \psi) = \hat{L}(q;\theta)\hat{L}(q;\theta)^T\ddot{q} + \frac{d}{dt}(\hat{L}(q;\theta)\hat{L}(q;\theta)^T)\dot{q} \\ - \frac{1}{2}\left(\frac{\partial}{\partial q}(\dot{q}^T\hat{L}(q;\theta)\hat{L}(q;\theta)^T\dot{q})\right)^T + \hat{g}(q;\psi) \end{aligned} \tag{3.3}$$

$$s.t. \quad 0 < x^T\hat{L}(q;\theta)\hat{L}(q;\theta)^T x \quad \forall x \in \mathbb{R}_0^n \tag{3.4}$$

where $\hat{f}^{-1}$ is the inverse model and L a differentiable loss function, figure 3.2 shows the computational graph of $\hat{f}^{-1}$.

It is also possible to obtain the forward model by solving equation (3.3) for $\ddot{q}$ described as:

$$\ddot{q} = f(q, \dot{q}, \tau; \theta, \psi) = \left(\hat{L}(q;\theta)\hat{L}(q;\theta)^T\right)^{-1}\left(\tau - \frac{d}{dt}(\hat{L}(q;\theta)\hat{L}(q;\theta)^T)\dot{q} + \frac{1}{2}\left(\frac{\partial}{\partial q}(\dot{q}^T\hat{L}(q;\theta)\hat{L}(q;\theta)^T\dot{q})\right)^T - \hat{g}(q;\psi)\right) \tag{3.5}$$

The formulation of the optimization problem also prove that the obtained parameters should generalize arbitrary velocities and accelerations as both functions $\hat{L}$ and $\hat{g}$ are not dependent of $\dot{q}$ and $\ddot{q}$. The formulation also guaranties the invertiblity of $\hat{L}\hat{L}^T$, due to the constraint (3.4), which will be further explained in the next later. Solving the optimization problem in (3.2) is neither direct or trivial for three reasons:

- Ensuring the uniqueness of the Lagrangian: This can be done by using a regularization technique called L2 regularization, which adds a penalty term on the network weights, in equation (3.2)

$$(\theta^*, \psi^*) = \underset{\theta,\psi}{\operatorname{argmin}} \quad L\left(\hat{f}^{-1}(q, \dot{q}, \ddot{q}; \theta, \psi), \tau\right) + \lambda\Omega(\theta, \psi) \tag{3.6}$$

- Non-violation of the constraint in (3.4): The kinetic energy is always positive for all velocity values which can only be ensured by the symmetry and positive definiteness of H.

- Deriving the derivatives in (3.3): The derivatives $\frac{d}{dt}(LL^T)$ and $\frac{\partial}{\partial q}(\dot{q}^T LL^T\dot{q})$ cannot be computed using automatic differentiation as time, t, is not an input of the network and most implementations of automatic differentiation do not allow the backpropagation of the gradient through the computed derivatives [49].

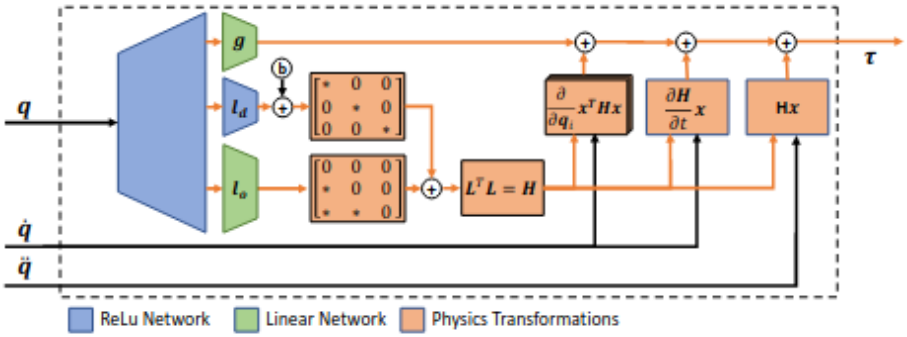


Figure 3.2: The computational graph of the Deep Lagrangian Network (DeLaN). Shown in blue and green is the neural network with the three separate heads computing g(q), ld (q), lo(q). The orange boxes correspond to the reshaping operations and the derivatives contained in the Euler-Lagrange equation. For training the gradients are backpropagated through all vertices highlighted in orange [49].

3.1.1 Symmetry and Positive Definiteness of the Inertia Matrix

The method used to ensure the symmetry and positive definiteness of the Inertia Matrix H was based in [49]. Forcing the symmetry and positive definiteness of H, not only ensures that H is invertible and the model can be used as a forward model in (3.5), but also that the kinetic energy always remains positive for all possible velocity values. The condition is represented by the contraint in (3.4)

The process to ensure this condition starts by representing the matrix H as a product of a lower triangular matrix L, ensuring the symmetry of H. This diagonal matrix is separated in two different matrix, the matrix l_d that represents the diagonal elements of L, and the matrix l_o that represents the off-diagonal elements of L. These two matrices have different heads in the neural network model and each head can have a different activation function in the output layer, as shown in 3.2. The positive definiteness and invertibility of the matrix H is ensured if its diagonal is positive, so if the activation function of the l_d is a non-negative activation function like ReLu or SoftPlus, this condition is ensured. The 3.2 also shows that l_o and g share the same linear activation function.

3.1.2 Deriving and Computing the derivatives

The derivatives $\frac{d}{dt}(LL^T)$ and $\frac{\partial}{\partial q}(\dot{q}^T LL^T \dot{q})$, present in the Centrifugal and Coriolis term of the equation, cannot be computed using automatic differentiation and they must be available in the forward pass for the computation of τ using the inverse model in (3.3). These derivatives also need to be computed analytically to enable the computation of the second order derivatives using automatic differentiation during the back-propagation step , essential to end-to-end training. For both derivatives, one starts by computing the derivative of L and then substitutes the reshape derivatives of the vectorized form l, as was done in [49]. The first derivative $\frac{d}{dt}(LL^T)$ yields:

$$\frac{d}{dt}H(q) = \frac{d}{dt}(LL^T) = L\frac{d}{dt}(L^T) + \frac{d}{dt}(L)L^T \tag{3.7}$$

where, with the application of the chain rule, $\frac{dL}{dt}$ can be substituted with the reshaped form of:

$$\frac{d}{dt}l = \frac{\partial l}{\partial q}\frac{\partial q}{\partial t} + \sum_{i=1}^{N}\frac{\partial l}{\partial W_i}\frac{\partial W_i}{\partial t} + \sum_{i=1}^{N}\frac{\partial l}{\partial b_i}\frac{\partial b_i}{\partial t} \tag{3.8}$$

where i is the i-th layer of the network that consists of and affine transformations and a non linearity g such has: $h_i = g_i(W_i^T h_{i-1} + b_i)$. The weights, W_i, and biases, b_i, are time-invariant which means that it is possible to simplify equation (3.8) with $\frac{dW_i}{dt} = 0$ and $\frac{db_i}{dt} = 0$:

$$\frac{d}{dt}l = \frac{\partial l}{\partial q}\dot{q} \tag{3.9}$$

The derivative $\frac{\partial l}{\partial q}$ can be computed by applying the chain rule recursively due to the structure of the network and the differentiability of the non-linearity g:

$$\frac{\partial l}{\partial q} = \frac{\partial l}{\partial h_{N-1}}\frac{\partial h_{N-1}}{\partial h_{N-2}} \quad ... \quad \frac{\partial h_1}{\partial q} \tag{3.10}$$

where $\frac{\partial h_i}{\partial h_{i-1}}$ is defined as:

$$\frac{\partial h_i}{\partial h_{i-1}} = diag\Big(g'(W_i^T h_{i-1} + b_i)W_i\Big) \tag{3.11}$$

where g' is the derivative of the non-linearity. The second derivative $\frac{\partial}{\partial q}(\dot{q}^T L L^T \dot{q})$ can be computed in the same way:

$$\frac{\partial}{\partial q_i}(\dot{q}^T L L^T \dot{q}) = \dot{q}^T \Big(\frac{\partial L}{\partial q_i} L^T + L \frac{\partial L^T}{\partial q_i}\Big)\dot{q} \tag{3.12}$$

where $\frac{\partial L}{\partial q_i}$ is computed with the demonstration shown in equation (3.10).

All the derivatives must be computed in a real-time control loop and their computational complexity must be minimal. To compute simultaneously l and $\frac{\partial l}{\partial q}$, it was used an extended standard layer represented in figure 3.3 where:

$$a_i = W_i h_{i-1} + b_i \quad h_i = g_i(a_i) \quad \frac{\partial h_i}{\partial h_{i-1}} = diag(g_i'(a_i))W_i \tag{3.13}$$

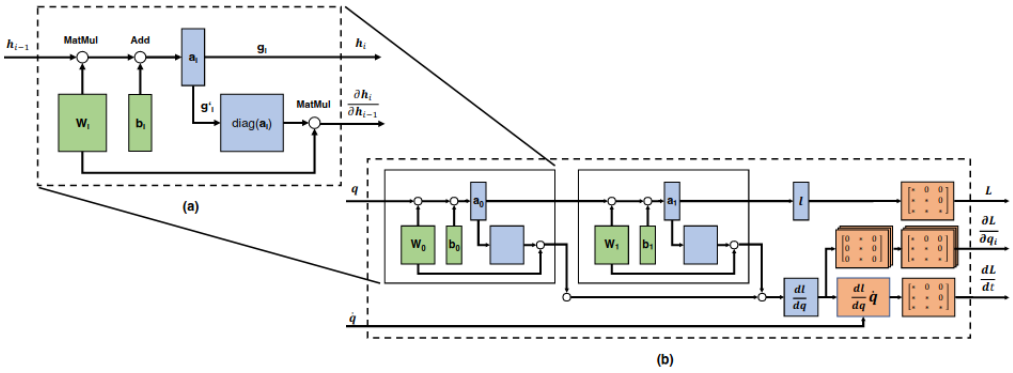


Figure 3.3: (a) Computational graph of the Lagrangian layer. The green boxes represent the learnable parameters. The upper computational sub-graph corresponds to the standard network layer while the lower sub-graph is the extension of the Lagrangian layer to simultaneously compute $\frac{\partial h_i}{\partial h_{i-1}}$. (b) Computational graph of the chained Lagrangian layer to compute L, $\frac{dL}{dt}$ and $\frac{\partial L}{\partial q_i}$ using a single feed-forward pass [49].

3.2 Learning Based Control Model

The full architecture of the learning based control model is composed by a Robust Multi-Stage Nonlinear Model Predictive Controller and a Deep Neural Network running in parallel. Unlike the work done in [49] where the final torque applied to the system was a combination of the network torque and a compensation torque determined by a PD controller, the MPC used in this thesis will receive the inertia, Coriolis and Centrifugal, and gravitational matrices from the forward pass of the neural network, as these are needed to compute the necessary forces to be applied to the system using its equations of motion.

Then, the MPC determines the optimal control values that minimizes the objective function based on the dynamical model that was learned by the network. While training, the network still needs to compute the predicted torque, just like in [49], and compare it to its true value in order to determine the loss and apply the gradient descent and backpropagation method. The full architecture of the learning based control model is shown in figure 3.4.

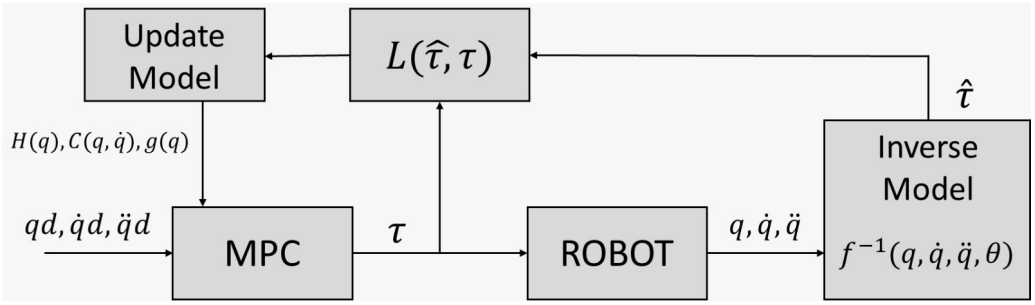


Figure 3.4: Full learning based control model architecture

When the system starts, the MPC does not have any knowledge about the dynamical model of the system and the matrices that define the model are randomly generated. After a few iterations, the neural network starts training with a batch of obtained data and, when the training is completed, the dynamical model in the MPC is updated. The process is repeated after a certain number of iterations and the dynamical model that governs the system is repeatedly updated.

In the beginning of this process, the MPC cannot guarantee that the input torque applied to the system does not force the robotic arm to violate any safety constraint. By applying an uncertainty term to the matrices determined by the neural network model, this problem can be avoided. The general idea is that, when the network loss is still too high and the dynamical model is still precarious, the associated uncertainty related to this model is also high. For a model already trained and with lower loss values, the uncertainty term can be lower, close to zero.

The uncertainty term described as d is directly related to the network error. More specifically, d is related to the error of the predicted matrices compared to their ground truth. During training, the network takes the maximum error of each predicted matrix $H(q), C(\dot{q}, q), g(q)$ and defines it as the upper and lower bound of the uncertainty of each matrix.

Chapter 4

Results

This chapter will present the results obtained in this thesis. First, the problem is described in order to understand how the system behaves and what results are expected. Then, after validating the training offline, the results obtained via online training are presented and explained in detail and were all obtained with the Monte Carlo technique with 20 runs. The full model was implemented in python and the MPC was implemented using the ”do mpc” library [26].

4.1 Problem Description

The system consists of a simulated 2-degree of freedom arm with two actuators in each joint. Besides the simulated arm, the simulated environment also contains a circle posing as an obstacle that the arm must not trespass. The robotic arm must follow a trajectory while safely learning the dynamics of the system. In figure 4.1 is displayed the simulated environment.

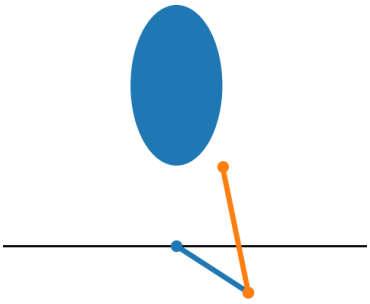


Figure 4.1: Simulation environment. The blue and orange lines represent links 1 and 2 of the arm. The obstacle is represented by the blue circle.

4.2 2-Degree of Freedom Arm Dynamics

The work in this thesis will focus on the learning and control of a 2-degree of freedom robot arm. The equations of motion of the robot arm can be calculated using the Lagrangian formulation of dynamics. The robot arm model is presented in figure 4.2 where:

- L_1, L_2 = Length of the first and second link;
- m_1, m_2 = Mass of the first and second link;
- θ_1, θ_2 = Joint angle of the first and second link;
- g = Gravitational acceleration.

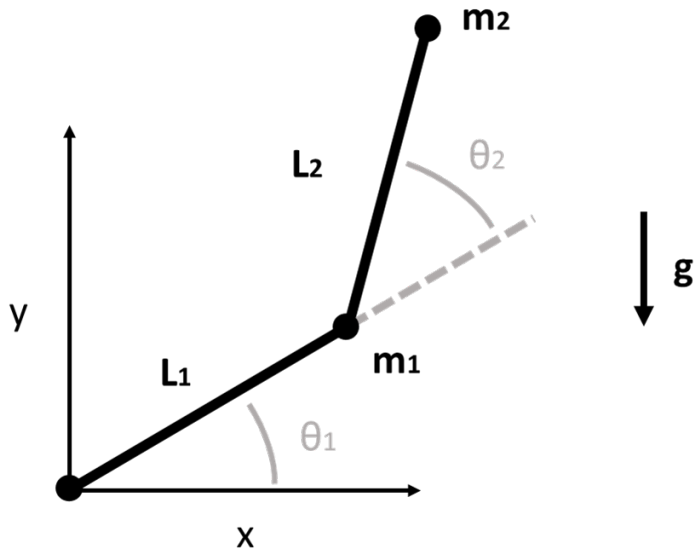


Figure 4.2: 2-degree of freedom robot arm

In order to calculate the kinetic, T, and potential, V, energy for the Lagrangian, one must first calculate the position of the two point masses:

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} L_1 cos(\theta_1) \\ L_1 sin(\theta_1) \end{bmatrix} \tag{4.1}$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} L_1 cos(\theta_1) + L_2 cos(\theta_1 + \theta 2) \\ L_1 sin(\theta_1) + L_2 sin(\theta_1 + \theta 2) \end{bmatrix} \tag{4.2}$$

The velocity of the two point masses can be calculated by taking the derivatives of the positions:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{y}_1 \end{bmatrix} = \begin{bmatrix} -L_1 sin(\theta_1) \\ L_1 cos(\theta_1) \end{bmatrix} \dot{\theta}_1 \tag{4.3}$$

$$\begin{bmatrix} \dot{x}_2 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} -L_1 sin(\theta_1) - L_2 sin(\theta_1 + \theta_2) & -L_2 sin(\theta_1 + \theta_2) \\ L_1 cos(\theta_1) + L_2 cos(\theta_1 + \theta_2) & L_2 cos(\theta_1 + \theta_2) \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} \tag{4.4}$$

With this information, being v_i and T_i the velocity and kinetic energy in point mass m_i, is now possible to determine the kinetic energy as:

$$T_i = \frac{1}{2} m_i v_i^2 = \frac{1}{2} m_i (\dot{x}_i + \dot{y}_i)^2 \tag{4.5}$$

By replacing the velocities of the point masses in 4.5 with the ones in 4.3 and 4.4, the kinetic energy of both point masses is then given by:

$$T_1 = \frac{1}{2} m_1 L_1^2 \dot{\theta}_1^2 \tag{4.6}$$

$$T_2 = \frac{1}{2} m_2 ((L_1^2 + 2L_1 L_2 cos(\theta_2) + L_2^2)\dot{\theta}_1^2 + 2(L_2^2 + L_1 L_2 cos(\theta_2))\dot{\theta}_1 \dot{\theta}_2 + L_2^2 \dot{\theta}_2) \tag{4.7}$$

$$T = T_1 + T_2 \tag{4.8}$$

The potential energy of the system is only dependent on the height of each mass, given by y_i, and is determined by:

$$V_1 = m_1 g y_1 = m_1 g L_1 sin(\theta_1) \tag{4.9}$$

$$V_2 = m_2 g y_2 = m_2 g (L_1 sin(\theta_1) + L_2 sin(\theta_1 + \theta_2)) \tag{4.10}$$

$$V = V_1 + V_2 \tag{4.11}$$

The Lagrangian of the system is given by the difference between the kinetic and potential energy as it was shown in 2.11. To obtain the following equations of motion of this system, one must solve the Euler-Lagrangian equation in 2.17 with the determined Lagrangian.

$$\begin{aligned} \tau_1 = {} & (m_1 L_1^2 + m_2 (L_1^2 + 2L_1 L_2 cos(\theta_2) + L_2^2))\ddot{\theta}_1 + m_2 (L_1 L_2 cos(\theta_2) + L_2^2)\ddot{\theta}_2 \\ & - m_2 L_1 L_2 sin(\theta_2)(2\dot{\theta}_1 \dot{\theta}_2 + \dot{\theta}_2^2) + (m_1 + m_2) L_1 g cos(\theta_1) + m_2 g L_2 cos(\theta_1 + \theta_2) \end{aligned} \tag{4.12}$$

$$\tau_2 = m_2 (L_1 L_2 cos(\theta_2) + L_2^2)\ddot{\theta}_1 + m_2 L_2^2 \ddot{\theta}_2 + m_2 L_1 L_2 \dot{\theta}_1^2 sin(\theta_2) + m_2 g L_2 cos(\theta_1 + \theta_2) \tag{4.13}$$

where τ_1 and τ_2 are the joint torques. The equations of motion can also be written in the form of:

$$\tau = H(\theta)\ddot{\theta} + C(\theta, \dot{\theta}) + g(\theta) \tag{4.14}$$

where:

- $H(\theta)$ = $\begin{bmatrix} (m_1 L_1^2 + m_2(L_1^2 + 2L_1 L_2 cos(\theta_2) + L_2^2)) & m_2(L_1 L_2 cos(\theta_2) + L_2^2) \\ m_2(L_1 L_2 cos(\theta_2) + L_2^2) & m_2 L_2^2 \end{bmatrix}$, is the inertia matrix;

- $C(\theta, \dot{\theta})$ = $\begin{bmatrix} -m_2 L_1 L_2 sin(\theta_2)(2\dot{\theta}_1 \dot{\theta}_2) \\ m_2 L_1 L_2 \dot{\theta}_1^2 sin(\theta_2) \end{bmatrix}$, represents the vector of Coriolis and centrifugal forces;

- $g(\theta)$ = $\begin{bmatrix} (m_1 + m_2) L_1 g cos(\theta_1) + m_2 g L_2 cos(\theta_1 + \theta_2) \\ m_2 g L_2 cos(\theta_1 + \theta_2) \end{bmatrix}$, is the gravitational potential vector.

4.3 Model Predictive Controller of 2-Degree of Freedom Arm

The control scheme proposed to control the robot arm is a model predictive controller. Model predictive control is an optimization control scheme that predicts the future behavior of the system over a finite time window, called horizon. The model predictive controller computes an optimal control input based on the systems model, the current state of the system and is also subject to system constraints. The described optimization problem is often called the model predictive control problem.

4.3.1 Model Predictive Control Problem

To define the control problem, one must first define the system's state representation, control inputs and algebraic states:

$$X = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} \quad \dot{\theta} = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} \tag{4.15}$$

$$U = \begin{bmatrix} \tau \end{bmatrix} \quad \tau = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} \tag{4.16}$$

$$Z = \begin{bmatrix} \ddot{\theta} \end{bmatrix} \quad \ddot{\theta} = \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} \tag{4.17}$$

where X are the states of the system, U the control inputs and Z the algebraic states. θ, $\dot{\theta}$ and $\ddot{\theta}$ represent the angles, angular velocity and angular acceleration of both joints and tau represents the torque applied in each joint.

The general optimization control problem is defined as:

$$\min_{\boldsymbol{X}_{0:N-1}, \boldsymbol{U}_{0:N-1}, \boldsymbol{Z}_{0:N-1}} m(x_{N+1}) + \sum_{k=0}^{N} l(x_k, z_k, u_k, p_k) \tag{4.18}$$

$$\text{subject to:} \quad x_0 = \hat{x}_0 \tag{4.19}$$

$$x_{k+1} = f(x_k, u_k, z_k, p_k) \tag{4.20}$$

$$g(x_k, u_k, z_k, p_k) \leq 0 \tag{4.21}$$

$$x_{lb} \leq x_k \leq x_{ub} \tag{4.22}$$

$$u_{lb} \leq u_k \leq u_{ub} \tag{4.23}$$

$$z_{lb} \leq z_k \leq z_{ub} \tag{4.24}$$

$$g_{\text{terminal}}(x_{N+1}) \leq 0 \tag{4.25}$$

where x_k, u_k, z_k and p_k are the states of the system, the control input, the algebraic states and the (uncertain) parameters at time step k, respectively. N is the prediction horizon and $\hat{x}_0$ is the current state estimate. The lesser bounds for the states, inputs and algebraic states are defined as x_{lb}, u_{lb} and z_{lb} while the respective upper bounds are defined as x_{ub}, u_{ub} and z_{ub}. The model of the system is defined by $f(\cdot)$. Terminal constraints and general nonlinear constraints can be defined as $g_{\text{terminal}}(\cdot)$ and $g(\cdot)$, which may also be soft constraints. The objective function consists of two elements, the mayer term $m(\cdot)$ is the cost of the terminal state and the lagrange term $l(\cdot)$ is the cost of each stage k.

For the robotic arm, the objective function had the same expression for the terminal state and for each stage. The objective function used could either try to minimize the error between the joint angles and the desired joint angles, θ and θ_d, or the error between the end-effector position and desired position, e and e_d, depending on the problem. The expressions for both functions are represented as:

$$l(\cdot) = m(\cdot) = (\theta - \theta_d)^2 \tag{4.26}$$

$$l(\cdot) = m(\cdot) = (e - e_d)^2 \tag{4.27}$$

The model of the robotic arm system, $f(\cdot)$, is defined by ODE obtained in 2.21 and the nonlinear constraint, $g(\cdot)$ is defined as the distance from the robotic arm to an obstacle, defined as a circular region

$$f(\theta, \dot{\theta}, \ddot{\theta}) = H(\theta)\ddot{\theta} + C(\theta, \dot{\theta}) + g(\theta) - \tau \tag{4.28}$$

$$g(\cdot) = (p_{\text{arm}} - p_{\text{obs}})^2 - r \tag{4.29}$$

where p_{arm} represents the joint positions of the robotic arm in the Cartesian space, p_{obs} represents the obstacle position in the Cartesian space and r is the radius of the obstacle.

The optimization control problem for the robotic arm is defined as:

$$\min_{\boldsymbol{X}_{0:N-1}, \boldsymbol{U}_{0:N-1}, \boldsymbol{Z}_{0:N-1}} m(x_{N+1}) + \sum_{k=0}^{N} l(x_k, z_k, u_k, p_k) \tag{4.30}$$

$$\text{subject to:} \quad x_0 = \hat{x}_0 \tag{4.31}$$

$$x_{k+1} = f(x_k, u_k, z_k, p_k) \tag{4.32}$$

$$g(x_k, u_k, z_k, p_k) \leq 0 \tag{4.33}$$

$$-\pi \leq \theta_k \leq \pi \tag{4.34}$$

$$-5 \leq u_k \leq 5 \tag{4.35}$$

$$\tag{4.36}$$

4.4 Neural Network training, Trajectory Generation and MPC Parameters

The neural network is a MLP with two hidden layers, each layer with 64 neurons. The weights were initialized with Xavier Uniform distribution and the biases to 0.0001. The mini batch size is 512 for the offline training and 300 for the online. The model was trained using the Adam optimizer and the learning rate was 0.005. Finally, the number of epochs was 1000.

The data set used for the trajectory generation was a modified version of the one used in [49]. The data set was created by [49] and contained all single stroke characters. The data set was modified in order to allow smooth transitions when the drawn character switched, this smoothness was motivated by the fact that the model had a much worse performance when the system was subjected to a high variance of torques and accelerations. The characters were spatially and temporally re-scaled in order to comply with the robotic arm kinematics and the joint references were computed using inverse kinematics [49]. It was verified that the neural network had a better performance when the controller tracked the desired joint angles, instead of the desired Cartesian positions.

The MPC that controlled the arm had a prediction horizon of 100 and a robust horizon of 2, the time step for the controller was 0.04.

4.5 Training Validation

To be able to validate that the model can be trained successfully and subsequently applied to the MPC, in this phase, the network was trained offline with data collected by the simulated arm. The arm was simulated using its well known equations of motion, derived in section 4.2, and the MPC used this same equations to define the model of the system. The network was trained with 2000 samples of training data and tested for 459 samples. The training loss of the neural network was 0.01141 and the the error of the predicted torque on the test set was 0.00675. In figure 4.3 the graphics for the predicted torque and real torque are displayed.

Figure 4.3 shows that the neural network can successfully learn the dynamical model of the system, which can be verified by the small error between the ground truth and DeLaN torque. Turning to the decomposed torque in (b), (c) and (d). The inertial matrix is more accurate for joint 1 than joint 2. The

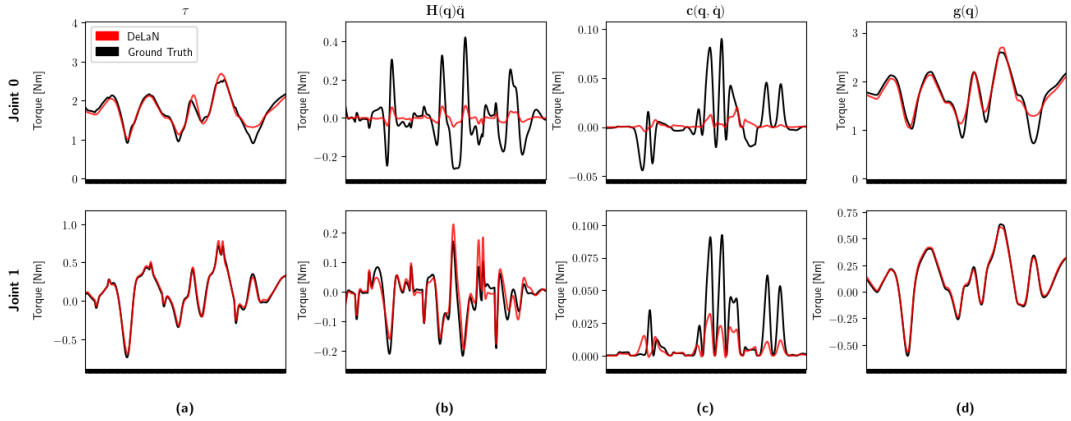


Figure 4.3: Neural network predicted torque (red) and test torque (black). (b), (c) and (d) represent the torque decomposed in inertial, Coriolis and centrifugal and gravitational matrices, respectively

Coriolis and centrifugal matrix have values so small that the error associated with them is negligible. The error associated with the gravitational matrix is minimal for both joints. Overall, the accuracy of the predicted torque is high because the gravitational matrix has a much higher weight in the torque value compared to the other matrices.

The physical dynamical model in the MPC was then replaced by the neural network model. Then, for every control loop iteration, the network receives the generalized positions and velocities of the system and outputs the estimated matrices $H(q), C(\dot{q}, q), g(q)$. The robotic arm is able to successfully track the desired positions, shown in figure 4.4, while also ensuring that the system remains safe at all times and does not violate any constraints.

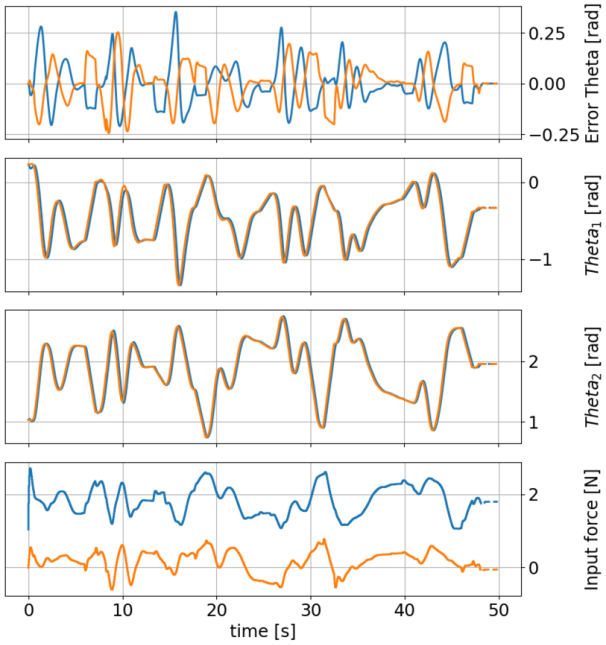


Figure 4.4: The first graphic represents the error between the desired angles and the actual angle of the arm in joint 1 (blue line) and joint 2 (orange line). The two middle graphics show the desired angles (orange line) and the actual angle of the arm (blue line). The fourth graphic shows the input torque to joint 1 (blue line) and joint 2 (orange line)

4.6 Online training

For the online training, the simulated arm collects data and the network is trained using batches of the same gathered data, simultaneously. The arm was again simulated using the physical model of the system. The MPC, on the other hand, starts by having and imperfect model of the system with a high level of uncertainty associated and changes its model by learning in real time.

The simulation had a duration of 1200 time steps and the system trains and updates its model every 300 steps for a total of 4 changes. Table 4.1 and figures 4.5 and 4.7 show the performance evolution of the neural network model, after each training. The table compares the applied torque and the respective predicted torque by the model forward pass using the mean squared error as well as the neural network training loss when it is updated.

Number of Steps	MSE(θ_1)	MSE(θ_2)	NN Loss
300	18.64	12.94	0.07459
600	0.01792	0.01605	0.05428
900	0.03439	0.01715	0.05337
1200	0.02990	0.01300	0.05076

Table 4.1: Results obtained for the online training.

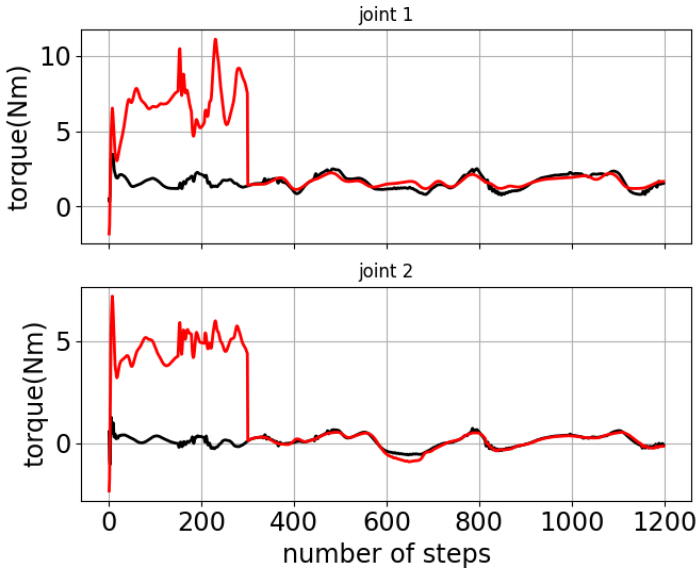


Figure 4.5: Comparison between the torque applied to the system by the controller (black line) and the predicted torque by the neural network model (red line).

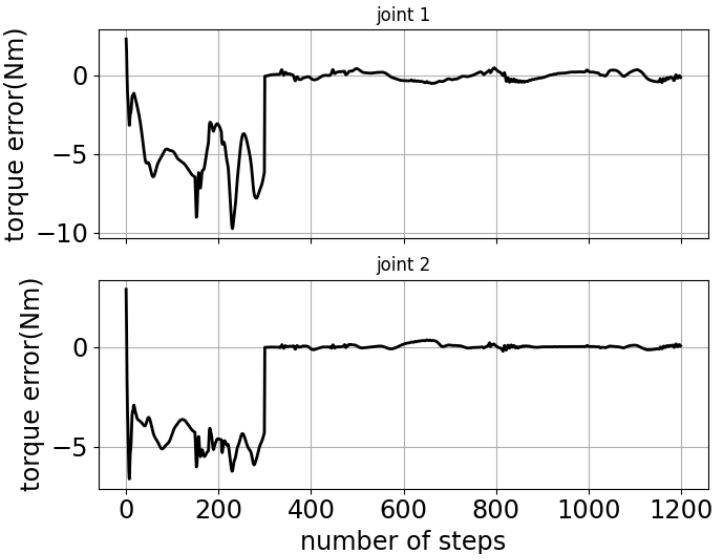
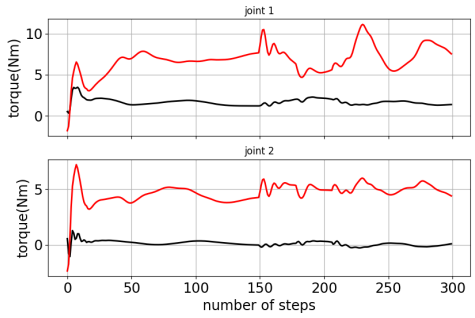
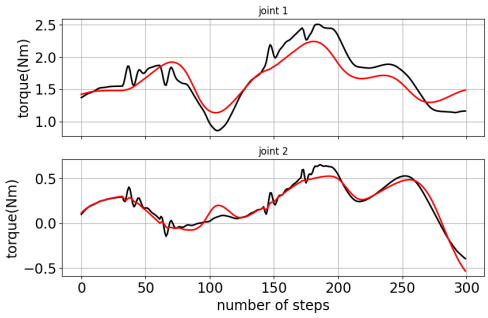


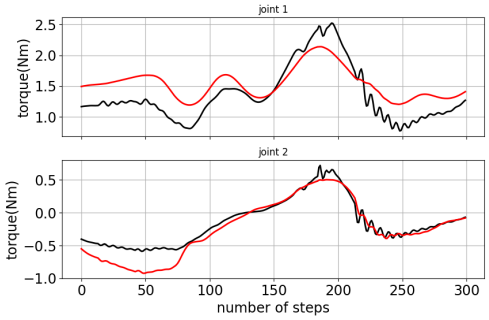
Figure 4.6: Error between the applied torque and predicted torque.



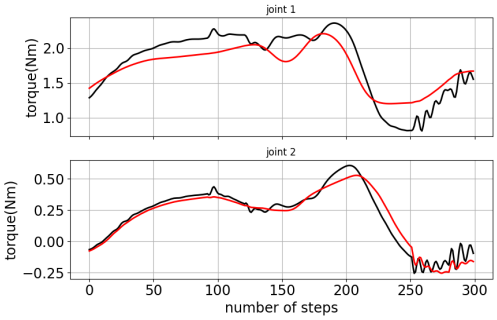
(a) Between step 0 and 300



(b) Between step 300 and 600

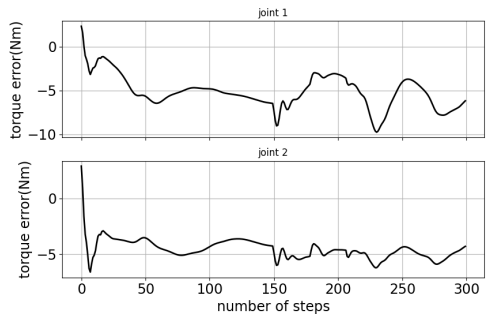


(c) Between step 600 and 900

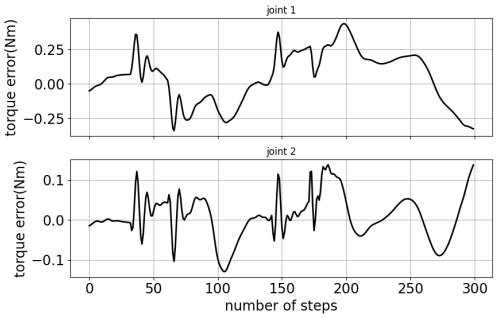


(d) Between step 900 and 1200

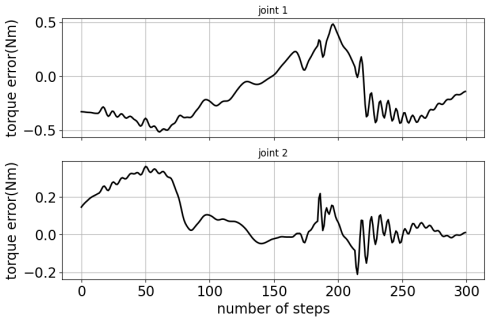
Figure 4.7: Comparison between the torque applied to the system by the controller (black line) and the predicted torque by the neural network model (red line) for each time the model was updated.



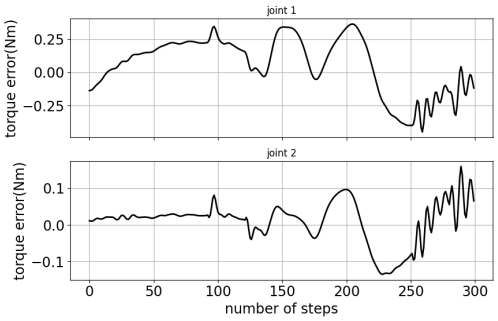
(a) Between step 0 and 300



(b) Between step 300 and 600



(c) Between step 600 and 900



(d) Between step 900 and 1200

Figure 4.8: Error between the applied torque and predicted torque for each time the model was updated.

The system starts with a high error for the model predictions, but after the first training batch (in step 300), the error drastically decreases. After the first model update, the model error remains constant which indicates that the model converges after the first training batch. The figure 4.7 shows that, despite the great improvement after the first batch of data is trained, the model still has some trouble modeling the parts where the torque has more variance and sharper changes.

The performance of the model when it is trained online is worse than the offline performance. This is mainly due to the fact that the online model trains with less data. In addition, the data for the offline training was obtained with a controller that used the correct dynamics of the system as a model instead of a predicted model of the system which may lead to more variance in the accelerations and torques.

Even though the online training does not have the same performance as the offline, the robotic arm still remains safe and is able to perform the task at hand despite having an imperfect model, specially in the beginning. This shows a high level of robustness, expected of the MPC. In figure 4.9 and 4.10 is possible to see that the system remains stable and is able to keep track of its objective.

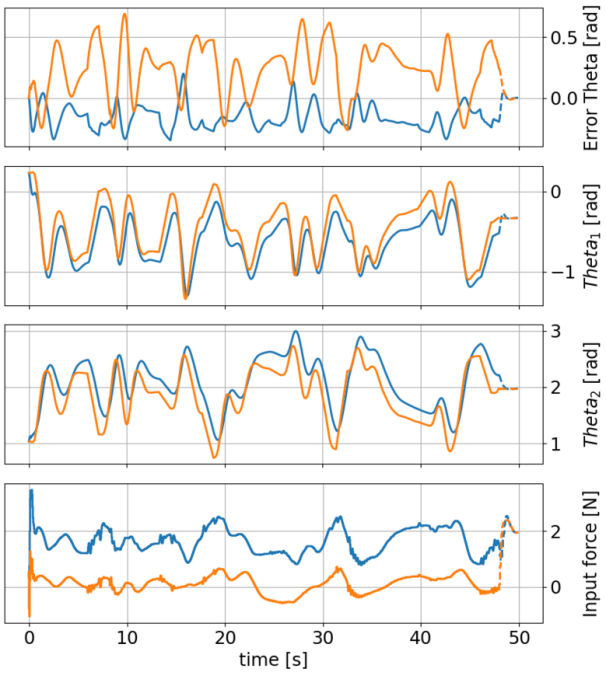


Figure 4.9: The first graphic represents the error between the desired angles and the actual angle of the arm in joint 1 (blue line) and joint 2 (orange line). The two middle graphics show the desired angles (orange line) and the actual angle of the arm (blue line). The fourth graphic shows the input torque to joint 1 (blue line) and joint 2 (orange line)

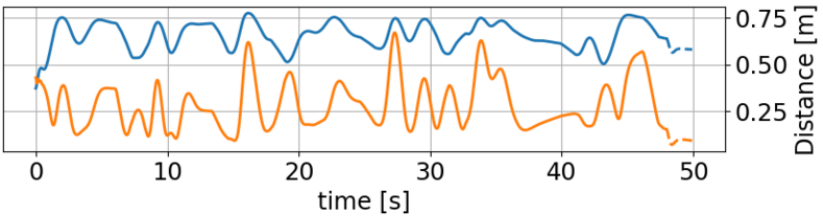


Figure 4.10: Distance between joint 1 (blue line) and joint 2 (orange line) to the obstacle. The arm remains safe if the blue and orange line remain above 0

Chapter 5

Conclusions

This thesis introduced a safety based learning control method that combines MPC with a deep Lagrangian network.

The deep Lagrangian network is a neural network that uses physics priors to learn the dynamical model of a system. This prior knowledge of the system is obtained by incorporating Lagrangian Mechanics into the deep learning model.

The MPC was extended into a multi-stage robust NMPC in order to account for the uncertainties in the system model while training. The new controller accounts for the uncertainties in the system by considering the various possible scenarios imposed by the uncertain parameters. The uncertainty parameter decreases with the network loss, this means that, the closer the controller model is to the real dynamical model of the system, the closer the uncertainty parameter is to zero. The use of the multi-stage robust NMPC ensures that the system does not violate any safety constraints because, even though the model used by the controller is imperfect, there is always an associated uncertainty parameter that depends on the model error.

The results obtained in this thesis showed the capabilities of the method to learn the dynamical model of a 2-degree of freedom robotic arm while performing a task safely. The results were first obtained offline to test the performance of the neural network model. The neural network was able to learn the model with a considerably low error and when the new learned model was applied to the controller, the robotic arm performed the requested tracking task without violating any safety constraints.

The results obtained online were promising, although the neural network model performance decreased slightly, which was expected. Still, the controller was able to remain safe and learned the dynamics of the system while performing its task, which was the main goal of this work.

5.1 Future Work

This thesis may open a number possible ideas that can be developed on top of the work that was done.

The work done in this thesis was highly motivated by the recent developments in the area of au-

tonomous free-flyer robots. In order to properly use this work on free flyers, the following work would be necessary

- **Modelling the elastic potential vector**: Due to the nonexistence of gravity in space, the gravitational potential vector, g(q), is a vector full of zeros. For this reason, it is useless for the neural network to model g(q). Instead, it is proposed that the network models the elastic potential vector to account for the flexible elements that space manipulators often have, mainly due to their low weight and the nonexistence of gravity.
- **Floating basis for the arm:** To faithfully simulate a free-flyer arm, the basis of the arm must not be static and supported but instead, it needs to be floating and reactive to the arm movements.

Bibliography

[1] S. A. A. Moosavian. Dynamics and control of free-flying robots in space: A survey. *IFAC Proceedings Volumes*, 37(8):621–626, 2004. ISSN 1474-6670. doi: https://doi.org/10.1016/S1474-6670(17)32047-5. IFAC/EURON Symposium on Intelligent Autonomous Vehicles, Lisbon, Portugal, 5-7 July 2004.

[2] Y. Umetani and K. Yoshida. Continuous path control of space manipulators mounted on omv. *Acta Astronautica*, 15(12):981–986, 1987.

[3] Z. Vafa and S. Dubowsky. On the dynamics of manipulators in space using the virtual manipulator approach. In *Proceedings. 1987 IEEE International Conference on Robotics and Automation*, volume 4, pages 579–585. IEEE, 1987.

[4] Y. Nakamura and R. Mukherjee. Nonholonomic path planning of space robots via a bidirectional approach. *IEEE Transactions on Robotics and Automation*, 7(4):500–514, 1991. doi: 10.1109/70.86080.

[5] B. Siciliano, O. Khatib, and T. Kröger. *Springer handbook of robotics*, volume 200. Springer, 2008.

[6] E. Papadopoulos and S. Dubowsky. Dynamic Singularities in Free-Floating Space Manipulators. *Journal of Dynamic Systems, Measurement, and Control*, 115(1):44–52, 03 1993.

[7] D. Nenchev, K. Yoshida, P. Vichitkulsawat, and M. Uchiyama. Reaction null-space control of flexible structure mounted manipulator systems. *IEEE Transactions on Robotics and Automation*, 15(6):1011–1023, 1999. doi: 10.1109/70.817666.

[8] K. Yoshida. Experimental study on the dynamics and control of a space robot with experimental free-floating robot satellite. *Advanced Robotics*, 9(6):583–602, 1994.

[9] K. Yoshida, H. Nakanishi, H. Ueno, N. Inaba, T. Nishimaki, and M. Oda. Dynamics, control and impedance matching for robotic capture of a non-cooperative satellite. *Advanced Robotics*, 18(2):175–198, 2004.

[10] S. E. Fredrickson, L. W. Abbott, S. Duran, J. D. Jochim, J. W. Studak, J. D. Wagenknecht, and N. M. Williams. Mini aercam: development of a free-flying nanosatellite inspection robot. In *Space Systems Technology and Operations*, volume 5088, pages 97–111. SPIE, 2003.

[11] H. Nakanishi and K. Yoshida. Impedance control for free-flying space robots-basic equations and applications. In *2006 IEEE/RSJ international conference on intelligent robots and systems*, pages 3137–3142. IEEE, 2006.

[12] S. Abiko, R. Lampariello, and G. Hirzinger. Impedance control for a free-floating robot in the grasping of a tumbling target with parameter uncertainty. In *2006 IEEE/RSJ international conference on intelligent robots and systems*, pages 1020–1025. IEEE, 2006.

[13] K. Yoshida, D. Dimitrov, and H. Nakanishi. On the capture of tumbling satellite by a space robot. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4127–4132. IEEE, 2006.

[14] R. B. Friend. Orbital express program summary and mission overview. In *Sensors and Systems for space applications II*, volume 6958, page 695803. International Society for Optics and Photonics, 2008.

[15] R. Lampariello and G. Hirzinger. Generating feasible trajectories for autonomous on-orbit grasping of spinning debris in a useful time. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5652–5659, 2013. doi: 10.1109/IROS.2013.6697175.

[16] J. Virgili-Llop, C. Zagaris, R. Zappulla, A. Bradstreet, and M. Romano. A convex-programming-based guidance algorithm to capture a tumbling object on orbit using a spacecraft equipped with a robotic manipulator. *The International journal of robotics research*, 38(1):40–72, 2019.

[17] R. Correia and R. Ventura. Payload transportation in microgravity with single and multiple cooperative free-flyer robots. In *2021 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 173–178, 2021. doi: 10.1109/ICARSC52212.2021.9429795.

[18] M. Ekal and R. Ventura. On inertial parameter estimation of a free-flying robot grasping an unknown object. In *2018 5th International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 815–821, 2018. doi: 10.1109/CoDIT.2018.8394883.

[19] M. Ekal and R. Ventura. On the accuracy of inertial parameter estimation of a free-flying robot while grasping an object. *Journal of Intelligent & Robotic Systems*, 98(1):153–163, 2020.

[20] W. Rackl and R. Lampariello. Parameter identification of free-floating robots with flexible appendages and fuel sloshing. In *Proceedings of 2014 International Conference on Modelling, Identification & Control*, pages 129–134. IEEE, 2014.

[21] W. Rackl, J. Gerstmann, and R. Lampariello. Analysis of liquid fuel sloshing on free-floating robot dynamics under low-gravity condition. In *2018 IEEE Aerospace Conference*, pages 1–12. IEEE, 2018.

[22] T. Smith, J. Barlow, M. Bualat, T. Fong, C. Provencher, H. Sanchez, and E. Smith. Astrobee: A new platform for free-flying robotics on the iss. *iSAIRAS 2016*, 3:5, 2016.

[23] B. Doerr, K. Albee, M. Ekal, R. Linares, and R. Ventura. Safe and uncertainty-aware robotic motion planning techniques for agile on-orbit assembly. *arXiv preprint arXiv:2102.10348*, 2021.

[24] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. the MIT Press, 2006. ISBN:026218253X.

[25] L. Grüne and J. Pannek. *Nonlinear Model Predictive Control: Theory and Algorithms*. Springer, 2006. ISBN:978-0-85729-500-2.

[26] S. Lucia, A. Tătulea-Codrean, C. Schoppmeyer, and S. Engell. Rapid development of modular and sustainable nonlinear model predictive control solutions. *Control Engineering Practice*, 60:51–62, 2017.

[27] S. Lucia. *Robust multi-stage nonlinear model predictive control*. Shaker Dortmund, 2015.

[28] H. K. Khalil. *Nonlinear Systems*. Pretice Hall, 2002. ISBN:0-13-067389-7.

[29] F. Berkenkamp, R. Moriconi, A. P. Schoellig, and A. Krause. Safe learning of regions of attraction for uncertain, nonlinear systems with gaussian processes. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 4661–4666, 2016. doi: 10.1109/CDC.2016.7798979.

[30] F. Berkenkamp, M. Turchetta, A. P. Schoellig, and A. Krause. Safe model-based reinforcement learning with stability guarantees. *arXiv preprint arXiv:1705.08551*, 2017.

[31] S. Bansal, M. Chen, S. Herbert, and C. J. Tomlin. Hamilton-jacobi reachability: A brief overview and recent advances. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 2242–2253. IEEE, 2017.

[32] A. Akametalu. *A Learning-Based Approach to Safety for Uncertain Robotic Systems*. PhD thesis, EECS Department, University of California, Berkeley, May 2018.

[33] J. F. Fisac, A. K. Akametalu, M. N. Zeilinger, S. Kaynama, J. Gillula, and C. J. Tomlin. A general safety framework for learning-based control in uncertain robotic systems. *IEEE Transactions on Automatic Control*, 64(7):2737–2752, 2018.

[34] A. K. Akametalu, J. F. Fisac, J. H. Gillula, S. Kaynama, M. N. Zeilinger, and C. J. Tomlin. Reachability-based safe learning with gaussian processes. In *53rd IEEE Conference on Decision and Control*, pages 1424–1431, 2014. doi: 10.1109/CDC.2014.7039601.

[35] I. M. Mitchell and J. A. Templeton. A toolbox of hamilton-jacobi solvers for analysis of nondeterministic continuous and hybrid systems. In *International workshop on hybrid systems: computation and control*, pages 480–494. Springer, 2005.

[36] S. Osher and R. P. Fedkiw. *Level set methods and dynamic implicit surfaces*, volume 1. Springer New York, 2005.

[37] J. A. Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, 93(4):1591–1595, 1996.

[38] K.-I. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural networks*, 2(3):183–192, 1989.

[39] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2): 251–257, 1991.

[40] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[41] N. E. Cotter. The stone-weierstrass theorem and its application to neural networks. *IEEE transactions on neural networks*, 1(4):290–295, 1990.

[42] V. Y. Kreinovich. Arbitrary nonlinearity is sufficient to represent all functions by neural networks: a theorem. *Neural networks*, 4(3):381–383, 1991.

[43] C. Bishop. Bishop-pattern recognition and machine learning-springer 2006. *Antimicrob. Agents Chemother*, pages 03728–14, 2014.

[44] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1): 1929–1958, 2014.

[45] D. Harrison. A brief introduction to automatic differentiation for machine learning. *arXiv preprint arXiv:2110.06209*, 2021.

[46] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *Journal of Marchine Learning Research*, 18:1–43, 2018.

[47] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[48] T. A. Battista. *Lagrangian mechanics modeling of free surface-affected marine craft*. PhD thesis, Virginia Tech, 2018.

[49] M. Lutter, C. Ritter, and J. Peters. Deep lagrangian networks: Using physics as model prior for deep learning. *arXiv preprint arXiv:1907.04490*, 2019.

[50] L. Brunke, M. Greeff, A. W. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. P. Schoellig. Safe learning in robotics: From learning-based control to safe reinforcement learning. *arXiv preprint arXiv:2108.06266*, 2021.

[51] T. J. Perkins and A. G. Barto. Lyapunov design for safe reinforcement learning. *Journal of Machine Learning Research*, 3(Dec):803–832, 2002.

[52] L. Hewing, K. P. Wabersich, M. Menner, and M. N. Zeilinger. Learning-based model predictive control: Toward safe learning in control. *Annual Review of Control, Robotics, and Autonomous Systems*, 3:269–296, 2020.

[53] T. Koller, F. Berkenkamp, M. Turchetta, and A. Krause. Learning-based model predictive control for safe exploration. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 6059–6066, 2018. doi: 10.1109/CDC.2018.8619572.

[54] G. Cao, E. M.-K. Lai, and F. Alam. Gaussian process model predictive control of unmanned quadrotors. In *2016 2nd International Conference on Control, Automation and Robotics (ICCAR)*, pages 200–206, 2016. doi: 10.1109/ICCAR.2016.7486726.

[55] L. Hewing, A. Liniger, and M. N. Zeilinger. Cautious nmpc with gaussian process dynamics for autonomous miniature race cars. In *2018 European Control Conference (ECC)*, pages 1341–1348, 2018. doi: 10.23919/ECC.2018.8550162.

[56] C. J. Ostafew, A. P. Schoellig, and T. D. Barfoot. Robust constrained learning-based nmpc enabling reliable mobile robot path tracking. *The International Journal of Robotics Research*, 35(13):1547–1563, 2016. doi: 10.1177/0278364916645661.

[57] A. M. Bayen, I. M. Mitchell, M. M. Oishi, and C. J. Tomlin. Aircraft autolander safety analysis through optimal control-based reach set computation. *Journal of Guidance, Control, and Dynamics*, 30(1): 68–77, 2007.

[58] J. Ding, J. Sprinkle, S. S. Sastry, and C. J. Tomlin. Reachability calculations for automated aerial refueling. In *2008 47th IEEE Conference on Decision and Control*, pages 3706–3712. IEEE, 2008.

[59] H. Huang, J. Ding, W. Zhang, and C. J. Tomlin. A differential game approach to planning in adversarial scenarios: A case study on capture-the-flag. In *2011 IEEE International Conference on Robotics and Automation*, pages 1451–1456. IEEE, 2011.

[60] S. L. Herbert, M. Chen, S. Han, S. Bansal, J. F. Fisac, and C. J. Tomlin. Fastrack: A modular framework for fast and guaranteed safe motion planning. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 1517–1522. IEEE, 2017.

[61] M. Chen, S. L. Herbert, M. S. Vashishtha, S. Bansal, and C. J. Tomlin. Decomposition of reachable sets and tubes for a class of nonlinear systems. *IEEE Transactions on Automatic Control*, 63(11): 3675–3688, 2018.

[62] M. Chen, S. Herbert, and C. J. Tomlin. Fast reachable set approximations via state decoupling disturbances. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 191–196. IEEE, 2016.

[63] Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh. A lyapunov-based approach to safe reinforcement learning. *arXiv preprint arXiv:1805.07708*, 2018.

[64] S. Huh and I. Yang. Safe reinforcement learning for probabilistic reachability and safety specifications: A lyapunov-based approach. *arXiv preprint arXiv:2002.10126*, 2020.

[65] A. B. Jeddi, N. L. Dehghani, and A. Shafieezadeh. Lyapunov-based uncertainty-aware safe reinforcement learning. *arXiv preprint arXiv:2107.13944*, 2021.

[66] S. M. Richards, F. Berkenkamp, and A. Krause. The lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems. In *Conference on Robot Learning*, pages 466–476. PMLR, 2018.

[67] J. Choi, F. Castañeda, C. J. Tomlin, and K. Sreenath. Reinforcement learning for safety-critical control under model uncertainty, using control lyapunov functions and control barrier functions, 2020.

[68] C. Dawson, Z. Qin, S. Gao, and C. Fan. Safe nonlinear control using robust neural lyapunov-barrier functions. In *Conference on Robot Learning*, pages 1724–1735. PMLR, 2022.

[69] J. K. Gupta, K. Menda, Z. Manchester, and M. J. Kochenderfer. A general framework for structured learning of mechanical systems. *arXiv preprint arXiv:1902.08705*, 2019.