Safety Based Machine Learning for Mechanical Systems

Luís Melo
Instituto Superior Técnico, Lisboa, Portugal

November 2022

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

1. Introduction

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.

The main challenge approached in this report 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 motivates 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 report 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 report 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 report 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

2. Background

2.1. Nonlinear Model Predictive Control

Nonlinear Model Predictive Control (NMPC) is an optimization based method used in nonlinear systems for feedback control. [4].

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.

The Algorithm

To track the robot in a given trajectory an algorithm for a time varying reference has to be considered. In [4], 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 [4] 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.

$$C(n, x^{ref}(n), u^{ref}(n)) = 0 \text{ for all } k \in \mathbb{N}_0,$$
$$C(n, x(n), u(n)) > 0 \text{ for all } n \in \mathbb{N}_0 \text{ with } x \neq x^{ref}(n). \quad (1)$$

The basic NMPC algorithm, taken from [4], 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

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

   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 [9, 8].

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.

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:

$$x_{k+1}^j = f(x_k^{p(j)}, u_k^j, z_k^{p(j)}, d_k^{r(j)}) \quad (2)$$

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.

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 1 shows the scenario tree with the proposed simplification.

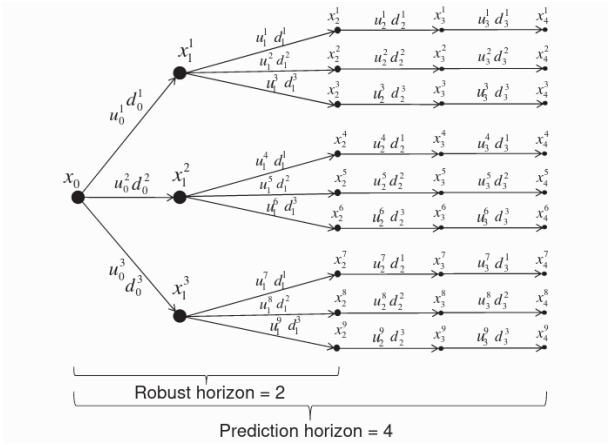


Figure 1: Scenario tree for the multi-stage approach with a robust horizon. x_k^j represents the state in stage k and position j, u_k^j the control inputs and $d_k^{r(j)}$ is the uncertainty parameter, with the realization of the uncertainty r being a function of j.

2.2. Mechanical System Dynamics

Describing the equations of motion for mechanical systems through the use of Lagrangian mechanics has been extensively studied in literature. This paper will follow the demonstrations and notation used in [1] which does a good job explaining this subject.

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 3, is a function of generalized coordinates **q** that completely describe the dynamics of a given system.

$$\mathcal{L}(q, \dot{q}, t) = T - V \tag{3}$$

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

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}}\right) - \frac{\partial \mathcal{L}}{\partial q} = Q. \tag{4}$$

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 3 where the Kinetic energy T is:

$$T(q, \dot{q}) = \frac{1}{2}\dot{q}^T H(q)\dot{q}. \tag{5}$$

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 [10]. Replacing the kinetic energy in 3 with 5 results in:

$$\mathcal{L}(q, \dot{q}, t) = \frac{1}{2}\dot{q}^T H(q)\dot{q} - V(q). \tag{6}$$

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 4 and by replacing L with 6 and then dV/dq = g(q) yields the second order ordinary differential equation (ODE) described by:

$$H(q)\ddot{q} + \dot{H}(q)\dot{q} - \frac{1}{2}\left(\frac{\partial}{\partial q}(\dot{q}^T H(q)\dot{q})\right)^T + g(q) = U. \tag{7}$$

The equation in 7 can also be rewritten as:

$$H(q)\ddot{q} + C(q, \dot{q}) + g(q) = U. \tag{8}$$

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

2.3. Related Work

The research and interest in safety based learning methods in control and artificial intelligence has been rapidly increasing over the last decade, [2] 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, [12] 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 (MPC) has established itself as the primary control method for the systematic handling of system constraints, [6] 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: [7], 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; [3] for modelling and controlling of unmanned quadrotors; [5]

to safely increase the performance of autonomous miniature race cars; and [11] 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.

3. Deep Lagrangian Networks and Learning Based Control Model

The problem presented in this paper 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. Instead of a white-box or black-box approach, the approach that was taken in this paper was a grey-box one. 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.

3.1. Incorporating Lagrangian Mechanics into Deep Learning

The work in this paper was inspired by the work done in [10] 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 (8), 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 later. The matrices H(q) and g(q) are represented in the neural network as:

$$\hat{H}(q) = \hat{L}(q;\theta)\hat{L}(q;\theta)^T \quad and \quad \hat{g}(q) = \hat{g}(q;\psi) \tag{9}$$

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 (8). The optimization problem is the following [10]:

$$(\theta^*, \psi^*) = \underset{\theta,\psi}{\operatorname{argmin}} \quad L\left(\hat{f}^{-1}(q,\dot{q},\ddot{q};\theta,\psi),\tau\right) \tag{10}$$

$$\begin{aligned} with \quad \hat{f}^{-1}(q,\dot{q},\ddot{q};\theta,\psi) &= \hat{L}(q;\theta)\hat{L}(q;\theta)^T\ddot{q} \\ &+ \frac{d}{dt}(\hat{L}(q;\theta)\hat{L}(q;\theta)^T)\dot{q} \\ &- \frac{1}{2}\left(\frac{\partial}{\partial q}(\dot{q}^T\hat{L}(q;\theta)\hat{L}(q;\theta)^T\dot{q})\right)^T + \hat{g}(q;\psi) \end{aligned} \tag{11}$$

$$s.t. \quad 0 < x^T\hat{L}(q;\theta)\hat{L}(q;\theta)^T x \quad \forall x \in \mathbb{R}_0^n \tag{12}$$

where $\hat{f}^{-1}$ is the inverse model and L a differentiable loss function, figure 2 shows the computational graph of $\hat{f}^{-1}$.

It is also possible to obtain the forward model by solving equation (11) for $\ddot{q}$ described as:

$$\begin{aligned} \ddot{q} = f(q,\dot{q},\tau;\theta,\psi) &= \left(\hat{L}(q;\theta)\hat{L}(q;\theta)^T\right)^{-1} \\ &\left(\tau - \frac{d}{dt}(\hat{L}(q;\theta)\hat{L}(q;\theta)^T)\dot{q}\right. \\ &\left. + \frac{1}{2}\Big(\frac{\partial}{\partial q}(\dot{q}^T\hat{L}(q;\theta)\hat{L}(q;\theta)^T\dot{q})\Big)^T - \hat{g}(q;\psi)\right) \end{aligned} \tag{13}$$

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 (12), which will be further explained in the next later. Solving the optimization problem in (10) 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 (10)

$$\begin{aligned} (\theta^*, \psi^*) = \underset{\theta,\psi}{\operatorname{argmin}} \quad & L\left(\hat{f}^{-1}(q,\dot{q},\ddot{q};\theta,\psi),\tau\right) \\ & + \lambda\Omega(\theta,\psi) \end{aligned} \tag{14}$$

- Non-violation of the constraint in (12): 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 (11): 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 [10].

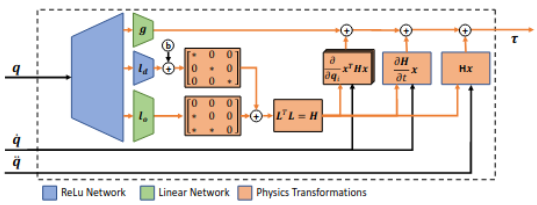


Figure 2: The computational graph of the Deep Lagrangian Network (DeLaN). Shown in blue and green is the neural network with the three separate heads computing g(q), ld (q), lo(q). The orange boxes correspond to the reshaping operations and the derivatives contained in the Euler-Lagrange equation. For training the gradients are backpropagated through all vertices highlighted in orange [10].

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 [10]. 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 (13), but also that the kinetic energy always remains positive for all possible velocity values. The condition is represented by the contraint in (12)

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 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 2 also shows that l_o and g share the same linear activation function.

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 (11). 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 [10]. The first derivative $\frac{d}{dt}(LL^T)$ yields:

$$\frac{d}{dt}H(q) = \frac{d}{dt}(LL^T) = L\frac{d}{dt}(L^T) + \frac{d}{dt}(L)L^T \quad (15)$$

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

$$\frac{d}{dt}l = \frac{\partial l}{\partial q}\frac{\partial q}{\partial t} + \sum_{i=1}^{N}\frac{\partial l}{\partial W_i}\frac{\partial W_i}{\partial t} + \sum_{i=1}^{N}\frac{\partial l}{\partial b_i}\frac{\partial b_i}{\partial t} \quad (16)$$

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 (16) with $\frac{dW_i}{dt} = 0$ and $\frac{db_i}{dt} = 0$:

$$\frac{d}{dt}l = \frac{\partial l}{\partial q}\dot{q} \quad (17)$$

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:

$$\frac{\partial l}{\partial q} = \frac{\partial l}{\partial h_{N-1}}\frac{\partial h_{N-1}}{\partial h_{N-2}} \quad ... \quad \frac{\partial h_1}{\partial q} \quad (18)$$

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

$$\frac{\partial h_i}{\partial h_{i-1}} = diag\Big(g'(W_i^T h_{i-1} + b_i)W_i\Big) \quad (19)$$

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

$$\frac{\partial}{\partial q_i}(\dot{q}^T LL^T \dot{q}) = \dot{q}^T\Big(\frac{\partial L}{\partial q_i}L^T + L\frac{\partial L^T}{\partial q_i}\Big)\dot{q} \quad (20)$$

where $\frac{\partial L}{\partial q_i}$ is computed with the demonstration shown in equation (18).

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 where:

$$\begin{aligned} a_i &= W_i h_{i-1} + b_i \quad h_i = g_i(a_i) \\ \frac{\partial h_i}{\partial h_{i-1}} &= diag(g_i'(a_i))W_i \end{aligned} \quad (21)$$

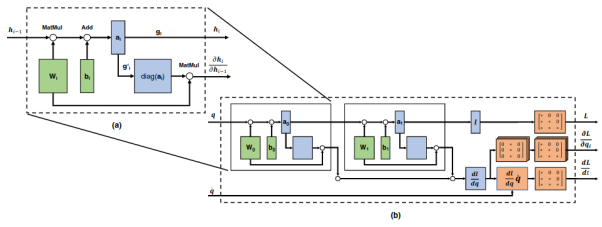


Figure 3: (a) Computational graph of the Lagrangian layer. The green boxes represent the learnable parameters. The upper computational sub-graph corresponds to the standard network layer while the lower sub-graph is the extension of the Lagrangian layer to simultaneously compute $\frac{\partial h_i}{\partial h_{i-1}}$. (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 [10].

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 [10] 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 report 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 [10], 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 4.

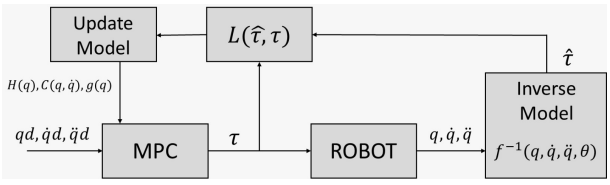


Figure 4: Full learning based control model arquitecture

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.

4. Results

The full model was implemented in python and the MPC was implemented using the "do mpc" library [9].

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 5 is displayed the simulated environment.

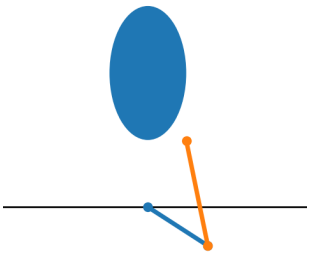


Figure 5: Simulation environment. The blue and orange lines represent links 1 and 2 of the arm. The obstacle is represented by the blue circle.

4.2. Model Predictive Control Problem

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

$$X = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} \quad \dot{\theta} = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} \tag{22}$$

$$U = \begin{bmatrix} \tau \end{bmatrix} \quad \tau = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} \tag{23}$$

$$Z = \begin{bmatrix} \ddot{\theta} \end{bmatrix} \quad \ddot{\theta} = \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} \tag{24}$$

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:

$$\min_{\boldsymbol{X}_{0:N-1}, \boldsymbol{U}_{0:N-1}, \boldsymbol{Z}_{0:N-1}} m(x_{N+1}) + \sum_{k=0}^{N} l(x_k, z_k, u_k, p_k) \tag{25}$$

$$\begin{aligned} \textit{subject to}: \quad & x_0 = \hat{x}_0 && (26)\\ & x_{k+1} = f(x_k, u_k, z_k, p_k) && (27)\\ & g(x_k, u_k, z_k, p_k) \leq 0 && (28)\\ & x_{lb} \leq x_k \leq x_{ub} && (29)\\ & u_{lb} \leq u_k \leq u_{ub} && (30)\\ & z_{lb} \leq z_k \leq z_{ub} && (31)\\ & g_{terminal}(x_{N+1}) \leq 0 && (32) \end{aligned}$$

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 $g_{terminal}(\cdot)$ 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:

$$l(\cdot) = m(\cdot) = (\theta - \theta_d)^2 \tag{33}$$

$$l(\cdot) = m(\cdot) = (e - e_d)^2 \tag{34}$$

The model of the robotic arm system, $f(\cdot)$, is defined by ODE obtained in 8 and the nonlinear constraint, $g(\cdot)$ is defined as the distance from the robotic arm to an obstacle, defined as a circular region

$$f(\theta, \dot{\theta}, \ddot{\theta}) = H(\theta)\ddot{\theta} + C(\theta, \dot{\theta}) + g(\theta) - \tau \tag{35}$$

$$g(\cdot) = (p_{arm} - p_{obs})^2 - r \tag{36}$$

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.

4.3. 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 [10]. The data set was created by [10] 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 [10]. 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.

4.4. 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, 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 6 the graphics for the predicted torque and real torque are displayed.

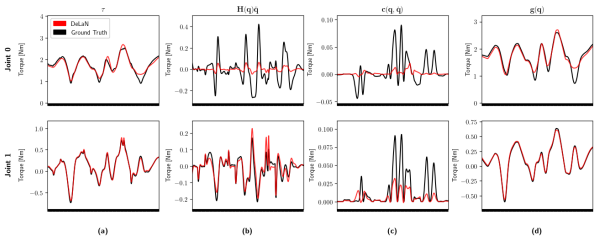


Figure 6: Neural network predicted torque (red) and test torque (black). (b), (c) and (d) represent the torque decomposed in inertial, Coriolis and centrifugal and gravitational matrices, respectively

Figure 6 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 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 7, while also ensuring that the system remains safe at all times and does not violate any constraints.

4.5. 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 1 and figures 8 and 10 show the performance evolution of the neural network model, after each training. The table compares the applied torque and the respective

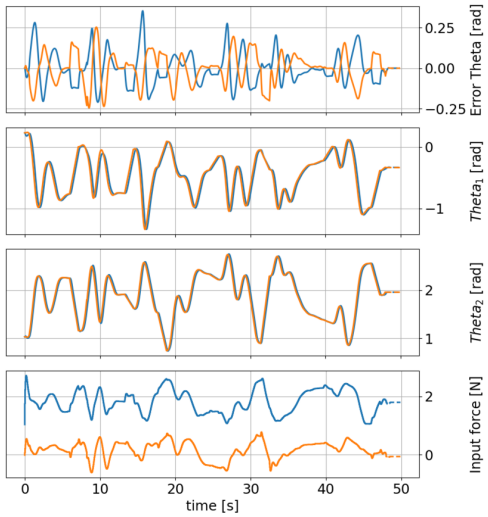


Figure 7: The first graphic represents the error between the desired angles and the actual angle of the arm in joint 1 (blue line) and joint 2 (orange line). The two middle graphics show the desired angles (orange line) and the actual angle of the arm (blue line). The fourth graphic shows the input torque to joint 1 (blue line) and joint 2 (orange line)

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 1: Results obtained for the online training.

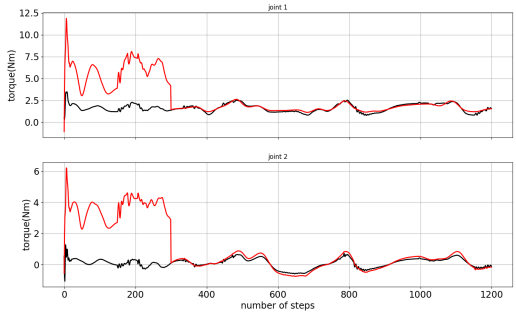


Figure 8: Comparison between the torque applied to the system by the controller (black line) and the predicted torque by the neural network model (red line).

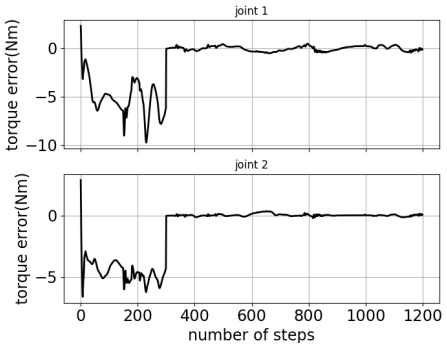
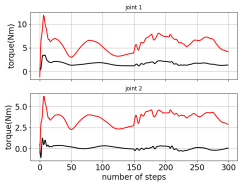
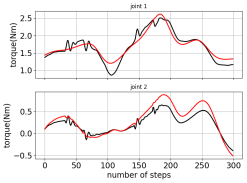


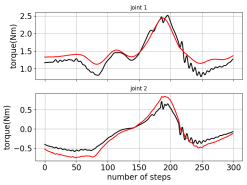
Figure 9: Error between the applied torque and predicted torque.



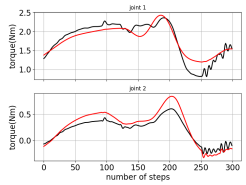
(a) Between step 0 and 300



(b) Between step 300 and 600

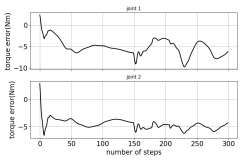


(c) Between step 600 and 900

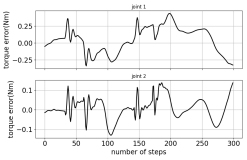


(d) Between step 900 and 1200

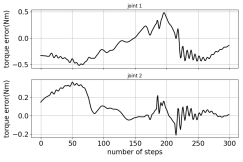
Figure 10: Comparison between the torque applied to the system by the controller (black line) and the predicted torque by the neural network model (red line) for each time the model was updated.



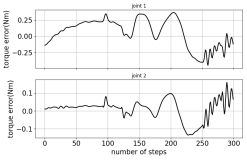
(a) Between step 0 and 300



(b) Between step 300 and 600



(c) Between step 600 and 900



(d) Between step 900 and 1200

Figure 11: 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 10 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 12 and 13 is possible to see that the system remains stable and is able to keep track of its objective.

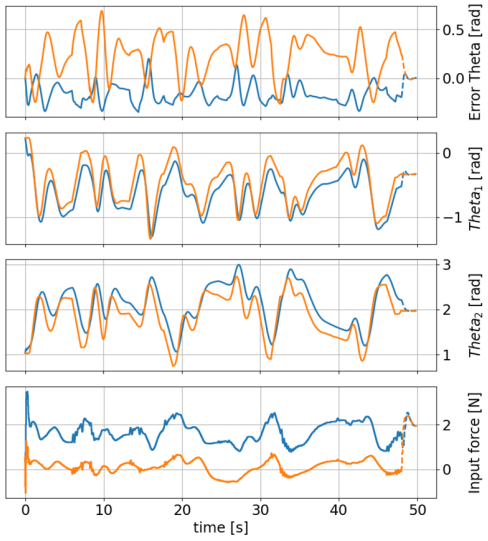


Figure 12: The first graphic represents the error between the desired angles and the actual angle of the arm in joint 1 (blue line) and joint 2 (orange line). The two middle graphics show the desired angles (orange line) and the actual angle of the arm (blue line). The fourth graphic shows the input torque to joint 1 (blue line) and joint 2 (orange line)

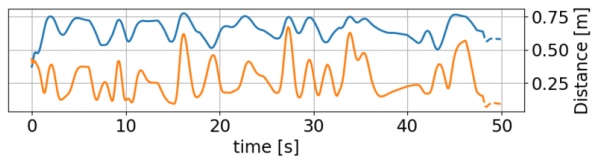


Figure 13: Distance between joint 1 (blue line) and joint 2 (orange line) to the obstacle. The arm remains safe if the blue and orange line remain above 0

5. Conclusions

This report introduced a safety based learning control method that combines MPC with a deep Lagrangian network. The results obtained in this paper 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 validate the performance of the neural network model. 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 paper may open a number possible ideas that can be developed on top of the work that was done, with special focus in the area of space robotics.

- **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.

References

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

[2] 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.

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

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

[11] 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.

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