Model Predictive Control with Bayesian Last Layer Trust Regions

Johannes Gaus

***Abstract*—Model Predictive Control (MPC) is an advanced control technique used in various industrial processes and systems. It involves predicting future system behavior based on a mathematical model and optimizing control actions over a finite time horizon to minimize a cost function, subject to constraints. We discuss the use of Bayesian last layer trust regions for Model Predictive Control algorithms to address uncertainty estimation. In the context of MPC, which traditionally relies on mathematical models, there is a growing interest in leveraging machine learning techniques due to their adaptability and robustness. We discuss the implementation of BLL trust regions in MPC, highlighting both advantages and challenges. Furthermore, we present a case study to evaluate a proposed approach, revealing its effectiveness in managing uncertainty across diverse scenarios. Our findings underscore the potential of BLL trust regions to enhance control performance, robustness, and uncertainty quantification in real-world applications.**

I. Introduction

Traditionally, control methods like Model Predictive Control (MPC) rely on complex mathematical models of system behavior. These models describe how systems change over time in response to inputs and disturbances, turning control into an optimization problem. But now, there is a shift towards machine learning (ML) methods, which learn directly from data, making them suitable for complex systems [12].

The shift from traditional control methods to machine learning methods marks a significant evolution in the field of control systems. Traditional methods, while robust in well-defined scenarios, often work worse in the face of complex, dynamic systems where precise mathematical models are either unavailable or insufficient. ML methods, on the other hand, thrive in such environments due to their inherent adaptability and robustness. They excel at learning from data in real-time, allowing them to adjust to new patterns and anomalies that rigid mathematical models might miss. [12] They are also able to handle changes and uncertainties in the environment and can deal with nonlinear or complex systems without simplifying them. By combining ML with control methods, we can improve system performance and adaptability across different industries. [8, 12]

Additionally, uncertainty plays a crucial role in MPC. Uncertainty quantification allows MPC to make decisions under varying degrees of uncertainty, ensuring robust performance in real-world scenarios. By incorporating uncertainty into the control framework, MPC can provide more reliable and adaptable control strategies, especially in dynamic and uncertain environments. [4]

Paper Overview: In the following work we first introduce the model predictive control principle in II and discuss related work in III. Then in IV we introduce the Bayesian neural network approach and the Bayesian last layer model with trust regions. In V we talk about using Bayesian last layer NN for MPC and we present and test a small case study in VI. In VII we point out different advantages and challenges and in VIII we give a conclusion and an outlook.

Notation: In the subsequent sections, we utilize the following notation: s_k for state, a_k for control action, and d_k for external inputs. The system dynamics function is denoted as $f_{sys}(\cdot)$. The prediction horizon is represented by N. State cost and terminal cost are denoted as $J(\cdot)$ and $J_N(\cdot)$. Stage and terminal constraints are represented by $g(\cdot)$ and $g_N(\cdot)$. Bayesian Neural Network (BNN) parameters are indicated by ω and BNN hyperparameters by Θ. The dataset used for training is represented by $\mathcal{D}$. The posterior weights distribution is denoted as $p(w \mid \mathcal{D}, \Theta)$, while the prior distribution of BNN parameters is indicated by $p(\omega)$. Hidden layer activations are represented by Φ and output predictions by F. The vector notation is used for neural network inputs and outputs. For the BLL neural network, a distribution function is used only for the last layer, denoted as $p(w \mid \mathcal{D}, \Theta)$.

II. Problem Formulation

A. MPC

The core principle of Model Predictive Control revolves around utilizing a dynamic model to predict future system behavior and optimizing this forecast to determine the best control action at the current time step. Models play a central role in all forms of MPC due to their crucial role in decision-making. Additionally, MPC addresses the challenge of determining the initial state of the system by leveraging past measurements to estimate the most probable initial state. This process, known as state estimation, involves reconciling past data with the model to infer the current state accurately. Both the control problem, which focuses on optimizing control actions based on model forecasts, and the estimation problem, which aims to produce an optimal state estimate using past measurements, rely on dynamic models and optimization techniques. [11] Therefore MPC continuously updates predictions based on real-time data, enabling adaptive and responsive control in complex systems. To ensure robustness and constraint satisfaction, MPC involves solving a constrained optimization problem. Additionally, achieving stability often involves formulating the cost function as a Lyapunov function and incorporating a terminal set constraint. We can determine an optimal output by solving a constrained optimization problem. [12]

The subsequent problem formulation is presented in [4]. We consider a discrete-time dynamic system:

$$s_{k+1} = f_{sys}(s_k, a_k, d_k),$$

where $s \in \mathbb{R}^{n_s}$ represent states of the system, $a \in \mathbb{R}^{n_a}$ represent actions, and $d \in \mathbb{R}^{n_d}$ the external inputs. The subscript k denotes the time step t_k.

For the control task, we define the optimal control problem (OCP) with the horizon N, which represents the prediction horizon. In MPC, the prediction horizon determines the length of time into the future over which the control actions are optimized. A longer prediction horizon allows for more accurate forecasting but may increase computational complexity. Therefore, selecting an appropriate prediction horizon is crucial for balancing prediction accuracy with computational efficiency in MPC. We define the OCP as:

$$\begin{aligned}\min_{\{s_i\}_{i=k}^{k+N}, \{a_i\}_{i=k}^{k+N-1}} \quad & \sum_{i=k}^{k+N-1} J(s_i, a_i, d_i) + J_N(s_{k+N}) \\ \text{subject to:} \quad & s_k = s_{k,\text{init}} \\ & s_{i+1} = \hat{f}_{\text{sys}}(s_i, a_i, d_i) \quad \forall i \\ & g(s_i, a_i, d_i) \le g_{\text{ub}} \quad \forall i \\ & g_N(s_{k+N}) \le g_{N,\text{ub}},\end{aligned}$$

where $J(\cdot)$ and $J_N(\cdot)$ are arbitrary state and terminal cost, and $g(\cdot)$ and $g_N(\cdot)$ are arbitrary stage and terminal constraints.

The OCP is solved at each time step k with the initial state $s_{k,init}$, providing the optimal sequence of actions $\{a_i^*\}_{i=k}^{k+N-1}$. The action a_k is then applied to the true system. For good control performance, an accurate system model $\hat{f}_{sys}(\cdot)$ is crucial.

To overcome the challenges associated with traditional MPC approaches, researchers have increasingly turned to neural networks (NNs) for more adaptive control strategies. This approach has found applications across a wide range of engineering sectors, including process industries, power electronics, building management, climate and energy systems and manufacturing. [12] One of the primary challenges in using MPC is the requirement for a reasonably accurate system model. While physical modeling is a traditional approach, another common method involves system identification from data. However, many established techniques mainly focus on identifying linear system models, which might not adequately capture highly nonlinear systems. So there is a growing interest in machine learning for extracting nonlinear relationships from data. [4]

A persistent challenge lies in the fact that NN predictions can be highly inaccurate without any indication of uncertainty. When employing NN system models for MPC with tracking cost functions, they are typically trained with samples around the desired set-point to ensure sufficient prediction accuracy. A prominent theoretical framework for addressing this challenge is Bayesian neural networks. [4]

BNNs operate under the assumption that the weights of a neural network follow an unknown distribution. However, BNNs are complex. A less frequently used technique within approximate Bayesian neural networks involves focusing solely on a distribution function for the last layer of the deep neural network. This method, denoted as Bayesian last layer, strikes a balance between ease of computation and expressive power. The BLL method provides a straightforward interpretation: The neural network learns a finite feature space, which is then linearly mapped onto the output in the final layer. This approach can also involve estimating the covariance from the Bayesian last layer to define a trust region. This trust region is subsequently employed as a constraint within the framework of Model Predictive Control problem. By using this approach, uncertainty quantification can lead to better and more precise results. [4]

III. Related Work

Traditional MPC relies on dynamic models of the process, often based on physical relation and parameter identification. It allows optimization over a finite time horizon while considering future time slots. Unlike classical control methods such as PID controllers and state feedback control, MPC can anticipate future events and take control actions accordingly [1].

Standard MPC approaches often rely on simple first-principle models due to real-time constraints on embedded platforms. These limitations can hinder performance, safety, and operation close to the system's physical limits. Machine learning, specifically neural networks, offers an alternative to simple models. Neural networks can accurately model complex dynamic effects, but their computational complexity has historically limited their use in real-time MPC. However, recent advancements have allowed for the integration of large-capacity neural network architectures within MPC pipelines, enabling more accurate dynamics modeling and improved performance. [12]

This paper [8] reviews the current landscape of integrating Machine Learning with Model Predictive Control in advanced control systems applications. Especially in the automotive area. It categorizes integration methods into five main categories.

1) ML in the model structure of MPC
2) ML in control structure
3) ML in imitation of MPC
4) ML in optimization of MPC
5) MPC for safe learning controller

ML in the model structure of MPC discusses offline and online learning methods' application in various subsystems such as engine emission modeling and vehicle dynamics prediction. ML in control structure explores integrating ML as a high-level supervisory controller and embedding MPC inside ML, suggesting that combining these methods improves performance in robustness, convergence, and predictability. ML in imitation of MPC discusses how ML models learn optimal actions from MPC by imitating controller behavior. ML in optimization of MPC explores how ML methods can enhance MPC optimization accuracy and reduce computational time. MPC for safe learning controller discusses using MPC to enforce safety constraints in learning-based controllers, with examples like autonomous control path generation and emission control in internal combustion engines.

In terms of future directions, the paper suggests several areas for further research and development. These include exploring online learning methods further for real-time deployment in connected vehicles and exploring the combination of MPC with learning-based controllers to ensure safe operation while meeting control requirements. For our work we would suggest that the method of Model Predictive Control with Bayesian last layer trust regions for uncertainty estimation would fall under the category of ML in control structure. Specifically, it aligns with the concept of ML as a high-level controller. In this approach, the Bayesian last layer trust regions serve as a mechanism to quantify uncertainty associated with neural network predictions, which is crucial for decision-making processes.

Uncertainty estimation is a cornerstone of robust Model Predictive Control systems. The integration of Bayesian last layer trust regions offers a novel approach to quantify and manage this uncertainty. By providing a probabilistic framework, these trust regions enable the MPC to account for the inherent variability in neural network predictions. The trust regions are integrated into the control structure to enhance robustness and reliability by adjusting control actions based on the estimated uncertainty. Therefore, this method fits within the broader framework of incorporating machine learning techniques as a supervisory controller to improve the performance and adaptability of MPC-based control systems. [4]

In summary, machine learning techniques enhance MPC by providing more accurate dynamics models, overcoming traditional limitations, and improving control performance for autonomous systems. Researchers continue to explore future trends in this field. [12]

IV. BNN-BASED DYNAMIC MODELS

A. Bayesian Neural Networks

Neural network models are very popular and often implemented in a frequentist scheme [5]. The basic form of NN can be defined as the standard Multi-Layer Perceptron (MLP). The MLP is foundational to neural networks, with modern architectures such as convolutional networks having an equivalent MLP representation. The following formulation in based on [5]. For a network with an input x of dimension N, the output of the network can be modeled as:

$$\Phi = a(X^T W^{(1)})$$

$$F = g(\Phi W^{(2)})$$

In the first equation X represents the input data matrix, where each row corresponds to a single input example, and $W^{(1)}$ represents the weight matrix for the connections between the input layer and the hidden layer. X^T denotes the transpose of the input data matrix, effectively rearranging the data to have each column correspond to a single input example. The product $X^T W^{(1)}$ computes the weighted sum of inputs for each neuron in the hidden layer. The activation function $a(\cdot)$ is then applied element-wise to this weighted sum, producing the activations of the neurons in the hidden layer. The resulting matrix Φ contains the activations of all neurons in the hidden layer for all input examples.

Once we have the activations of the neurons in the hidden layer, we can compute the forward propagation of the output layer in the second function. $W^{(2)}$ represents the weight matrix for the connections between the hidden layer and the output layer. The product $\Phi W^{(2)}$ computes the weighted sum of inputs for each neuron in the output layer. The activation function $g(\cdot)$ is then applied element-wise to this weighted sum, producing the final output of the neural network. The resulting matrix F contains the output predictions for all input examples. In summary, these equations describe how the inputs are transformed through the neural network layers via matrix multiplication and activation functions to produce the final output predictions.

A bias value is often added during each layer but is left out for simplicity. The activation function $a(\cdot)$ is an affine transform followed by a non-linear element-wise transform, commonly known as an activation. When using the Sigmoid function, the expression for the hidden layer's output becomes equivalent to logistic regression, meaning that the network's output is the sum of multiple logistic regression models. For a regression model, the function applied to the output $g(\cdot)$ will be the identity function, while for binary classification, it will be a Sigmoid function. [5]

Initially, the perceptron employed the sign$(\cdot)$ function as its activation function, but nowadays we can use more advanced activation functions like the Sigmoid, the Hyperbolic Tangent or the Rectified Linear Unit function. Different activation functions have distinct characteristics that make them suitable for various tasks [6]. We can expand this scheme to include many hidden layers, with the input of each layer being the output of the last layer and as mentioned before a bias value can also be added during each layer. [5]

Despite their popularity NNs can not reason about uncertainty in their predictions. To add this usefull addition and use NNs in a Bayesian scheme we can use Bayesian neural networks. [5] In the frequentist approach described earlier, the model weights are perceived as having an unknown true value, while the data is treated as a random variable. But with a Bayesian approach we want to recognize the unknown model weights based on the information available to us. Since we lack precise values for our weights, it feels natural to regard them as random variables. The Bayesian statistical perspective aligns with this notion, where unknown parameters are viewed as random variables, and our aim is to learn a distribution of these parameters given what we can observe in the training data.

In the process of "learning" within Bayesian neural networks, we infer the unknown model weights based on observable information. This scenario contains the problem of inverse probability, which is tackled through Bayes' theorem. We cannot directly perceive the true distribution in our model, denoted as ω. However, Bayes' theorem enables us to represent a distribution over these weights using probabilities we can observe, resulting in the posterior distribution of model parameters given the data we have encountered, denoted as $p(\omega|\mathcal{D})$. We can call them the posterior distribution. [5]

The joint distribution between our weights and our data is determined by our prior beliefs regarding our latent variables, represented by $p(\omega)$, and our chosen model, denoted as $p(\mathcal{D}|\omega)$, which combines to form:

$$p(\omega, \mathcal{D}) = p(\omega)p(\mathcal{D}|\omega).$$

This likelihood term is shaped by our selection of network architecture and loss function. The likelihood can be characterized by a distribution with the mean value dictated by the network's output:

$$p(\mathcal{D}|\omega) = \mathcal{N}(f^{\omega}(\mathcal{D}), \sigma^2).$$

Within this modeling framework, it is commonly assumed that all samples from the dataset $\mathcal{D}$ are independent and identically distributed (i.i.d.). Consequently, the likelihood can be expressed as a product of the contributions from the individual terms in the dataset, totaling N terms: [5]

$$p(\mathcal{D}|\omega) = \prod_{i=1}^{N} \mathcal{N}(f^{\omega}(\mathcal{D}), \sigma^2).$$

Because neural networks operate as black boxes, determining a meaningful prior can be difficult. In practical NNs trained using the frequentist approach, the weights of the network often have low magnitudes and tend to be centered around zero. Based on this, one approach is to employ a zero-mean Gaussian distribution with a small variance as our prior. With the prior and likelihood, Bayes theorem is then applied to get the posterior distribution over the model weights,

$$p(\omega \mid \mathcal{D}) = \frac{p(\omega)p(\mathcal{D} \mid \omega)}{\int p(\omega)p(\mathcal{D} \mid \omega)d\omega} = \frac{p(\omega)p(\mathcal{D} \mid \omega)}{p(\mathcal{D})}.$$

The denominator in the posterior distribution, known as the marginal likelihood or evidence, includes the entire distribution of the data given the model parameters. This distribution integrates over all possible values of the model parameters, essentially representing the uncertainty in our knowledge about these parameters. It serves an important role in Bayesian inference by normalizing the posterior distribution, ensuring that it reflects a valid probability distribution over the model parameters. By leveraging this posterior distribution, we can generate predictions for any quantity of interest. These predictions are expressed as an expectation computed with respect to the posterior distribution, encapsulating not only the expected value but also the entire distribution's uncertainty [5]:

$$\mathbb{E}_p[f] = \int f(\omega)p(\omega \mid \mathcal{D})d\omega.$$

Predictive quantities, such as means, variances, or intervals can all be computed as expectations over the posterior distribution. This involves averaging the function $f(\omega)$ with respect to the posterior distribution $p(\omega)$, where $f(\omega)$ represents the function used for the specific prediction task. In essence, prediction involves weighing the function f's values by the posterior distribution $p(\omega)$, which captures the model's uncertainty by considering the full distribution over model parameters.

The posterior distribution in Bayesian neural networks represents our updated beliefs about the model parameters after observing the data. For example, consider a scenario where we use a BNN to predict the behavior of an autonomous car under various driving conditions. Initially, we have some prior belief about the car's performance based on historical data. As we collect new data from the car's sensors during test drives, we update our beliefs, resulting in a posterior distribution that reflects our updated understanding of the car's behavior. This posterior distribution helps us make more informed predictions about how the car will perform in future scenarios. The predictive distribution is the culmination of the Bayesian inference process, where we use the posterior distribution to make predictions about new, unseen data. It is crucial for decision-making as it not only provides a point estimate but also quantifies the uncertainty of the prediction. For instance, in autonomous driving, the predictive distribution can be used to determine the probability of different maneuvers the car might need to take in response to sudden obstacles, along with the uncertainty of each maneuver. This allows for safer and more reliable decision-making under uncertainty, as the system can choose maneuvers with not only the highest likelihood of success but also the lowest risk. [7]

This approach allows us to gain insights into the generative process of a model, which contrasts with the optimization methods used in the frequentist setting. By leveraging this generative model, our predictions are expressed as valid conditional probabilities. In high-dimensional models, computing the integral through quadrature approximation can become computationally challenging. Therefore, to handle this complexity, approximations for the posterior distribution are necessary. [5]

B. *BNN with Bayesian Last Layer*

Understanding uncertainty is pivotal in machine learning, particularly in scenarios where safety is critical. As mentioned before, conventional neural networks often fall short in handling this aspect. Bayesian neural networks offer a promising solution by assuming probability distributions for all parameters, resulting in predictions that reflect this uncertainty. Yet, training and inference with Bayesian neural networks are complex processes that often necessitate approximations. One such promising approximation involves using neural networks with a Bayesian last layer. These networks assume distributed weights only in the linear output layer, yielding predictions with a normal distribution [3]. For this we can reformulate the weight w of the last layer as

$$p(w \mid \mathcal{D}, \Theta) = \frac{p(\mathcal{D} \mid w, \Theta)p(w \mid \Theta)}{p(\mathcal{D} \mid \Theta)}.$$

Then the neural network with Bayesian last layer has deterministic weights in all hidden layers and distributed weights in the output layer. Neural networks with a Bayesian last layer strike a balance between manageability and expressive power within the Bayesian neural network domain. This BLL approach has shown promising results, particularly when compared to Bayesian linear regression with features from a pre-trained neural network as seen in [3].

C. Trust Regions

We explored the Bayesian Last Layer approach, which simplifies the network while preserving its ability to quantify uncertainty. Now, we shift our focus to another critical concept: Trust Regions. This section will discuss how trust regions can further enhance the reliability and robustness of BNNs in uncertain environments. Bayesian neural networks present an elegant way for determining the reliability of neural network predictions by enabling the estimation of predictive distributions. Due to the complexity of training and prediction, approximations are typically used, often leading to inaccurate uncertainty estimations. To address this challenge, we can try to identify input space regions with dependable predictions. [13] Different approaches already exist to identify these trust regions. For example [13] used a method where 2 BNNs and complex testing strategies are used to determine the trust regions. In [4] they used an approach where they define trust regions based on the computed covariance.

Trust regions in Bayesian neural networks provide a means to quantify uncertainties effectively, aiming for well-calibrated predictions that align closely with the data-generating process. Instead of strictly confining predictions to predefined confidence intervals, trust regions focus on achieving reliable predictions that reflect the inherent uncertainties in the data. This approach is important for assessing the impact of noisy or outlier data points, fostering robust predictions that are less susceptible to perturbations in the input data. The evaluation of trust regions enables the identification of regions where BNNs can sufficiently correct the uncertainty associated with their predictions. Moreover, trust regions contribute to enhancing the generalization capability of BNNs by mitigating overfitting tendencies and make more accurate predictions on unseen data instances. [13]

V. MPC with BLL Trust Regions

Traditional MPC formulations often overlook process uncertainty and stochasticity. BNN or BLL approaches may sacrifice some performance, but prioritize constraint satisfaction which is crucial in safety-critical scenarios. Typically, a learned model provides uncertainty estimates, which are then integrated into robust or stochastic MPC. [10] Incorporating historical data into control systems through learning-based techniques can improve their performance and ensuring both effectiveness and safety challenges by considering uncertainties. [10, 2] While Gaussian processes are commonly employed for this purpose, they have limitations with large models and datasets and their computational complexity limits their applicability to relatively small datasets. [4] Bayesian neural networks achieve comparable performance, suggesting they could serve as a viable alternative for control designs, especially when managing extensive datasets. [10]

So overall Bayesian neural networks aim to address the issue with large computation complexity, we can also use a Bayesian last layer NN. This approach offers a simplified version of BNN that strikes a balance between tractability and expressiveness and preserves the training and point estimation procedures of regular neural networks while incorporating uncertainty quantification. Although BLL may not provide probabilistic guarantees in most cases, we can use the computed covariance to define a trust region as shown in [4]. By leveraging the BLL covariance to define a trust region and incorporating it as a soft constraint within the MPC framework, the controller in the case study presented in [4] surpasses the performance of its counterpart lacking the BLL constraint. Crucially, this improvement is achieved without adding complexity to NN training, and the computation time for solving the control problem remains unaffected by the number of training samples.

In summary, the exploration of BNNs and the Bayesian Last Layer approach has shown their potential to enhance predictive models by incorporating uncertainty quantification. While traditional neural networks provide point estimates, BNNs and BLL offer a probabilistic perspective, crucial for decision-making in uncertain environments. As we have seen, BLL simplifies the complexity of BNNs without significantly compromising performance, making it a practical choice for many applications. With these theoretical foundations in place, we now turn our attention to a real-world application that illustrates the efficacy of these concepts. The following case study delves into the integration of BLL trust regions within a Model Predictive Control framework. This practical example is from [4] and demonstrates how the theoretical principles discussed are applied to achieve robust and reliable control in the presence of uncertainty. We take a closer look into the threshold value defining the trust regions in the results.

VI. Case Study

Trust regions play a critical role in managing uncertainty in system dynamics. They define a space where the system's behavior is considered acceptable, helping to guide the optimization process within reasonable limits. In this study [4], the trust region, identified through Bayesian last layer uncertainty assessment, significantly add to the robustness and stability of the MPC framework.

The choice of threshold value directly impacts the performance and robustness of the MPC framework in real-world scenarios. A lower threshold value, such as 0.02, may lead to a higher number of trust region violations, indicating significant deviations between predicted and actual trajectories. While this higher sensitivity to deviations could enhance the system's responsiveness to uncertainty, it may also result in a higher frequency of false alarms, potentially leading to unnecessary adjustments in control actions. Conversely, a higher threshold value might reduce the number of trust region violations, offering a more conservative approach to managing uncertainty. However, this could also lead to a decreased sensitivity to deviations, compromising the system's ability to react promptly to unforeseen changes in the environment. Thus, the choice of the threshold value needs careful consideration, balancing the trade-off between sensitivity to uncertainty and robustness in control performance.

The trust region, derived from the estimated BLL covariance, acts as a soft constraint in the MPC problem formulation.

Mathematically, it is defined as:

$$\mathbb{T}_{BLL} := \{x \in \mathbb{R}^n_x | \phi(x)^T C \phi(x) \leq c_{ub}\}.$$

Here the trust region is defined as a region with variance smaller than c_{ub}. In practice, tuning this trust region involves comparing values within regions with and without training data. The threshold value c_{ub} serves as a threshold to identify instances where predicted trajectories deviate significantly from true trajectories, considering the system's uncertainty level. Visualizing and analyzing trust violations in predicted trajectories aid in assessing prediction reliability.

The threshold value in the mentioned paper, seemingly arbitrary, is set to 0.02 for neural networks and 0.3 for Gaussian processes. These values guide the generation of plots to evaluate the trust region approach. According to the paper the trust violations are marked based on methodologies detailed in [14, 9].

In our exploration, we trained six additional models, similar to those in [4], with different random seeds for initial NN weights. Everything else remained constant, except for the initial seeds. This exploration aimed to test the robustness of the 0.02 threshold choice. Each model evaluated 20 cases, consistent with the methodology of the referenced study, where 9 out of the 20 test cases exhibited trust violations.

Table I: Summary of different threshold values tested and the number of cases outside the defined threshold for each model (M1 to M6) and the associated mean value.

c_{ub}	M1	M2	M3	M4	M5	M6	mean
0.0001	20	20	20	20	20	20	20
0.0005	20	20	20	20	20	20	20
0.001	20	20	20	20	20	20	20
0.005	20	20	20	20	20	20	20
0.0075	20	20	20	19	20	20	19.8
0.01	20	16	18	15	18	18	17.5
0.02	14	8	9	5	10	7	8.8
0.03	7	3	5	2	5	4	4.3
0.04	6	2	5	2	3	4	3.6
0.05	4	2	5	1	3	3	3.0
0.06	3	2	3	2	1	2	2.2
0.07	2	1	3	2	1	2	1.8
0.08	0	1	3	2	1	1	1.4
0.09	0	1	3	1	0	1	1.0

Our models were subjected to the same 20 test cases. The results of our exploration are summarized in Table I and illustrated in figure 1. The table describes the number of cases outside the threshold value for each model (M1 to M6) across different threshold values ranging from 0.0001 to 0.09. We also provide our used code here [1]. Figure 1 plots 10 random selected test cases visualized with the trust region threshold set to 0.02. On the x-axis we can see the time steps and on the y-axis we can see the covariance as the uncertainty. And as we can see some test cases exceed the threshold and are way above, but it appears that the majority persist beyond it.

After analyzing the results, we determined that a threshold value of 0.02 strikes a well-balanced middle ground between trust region violations and adherence. In comparison, the

[1]https://github.com/jo997/BLL-Trust-Regions

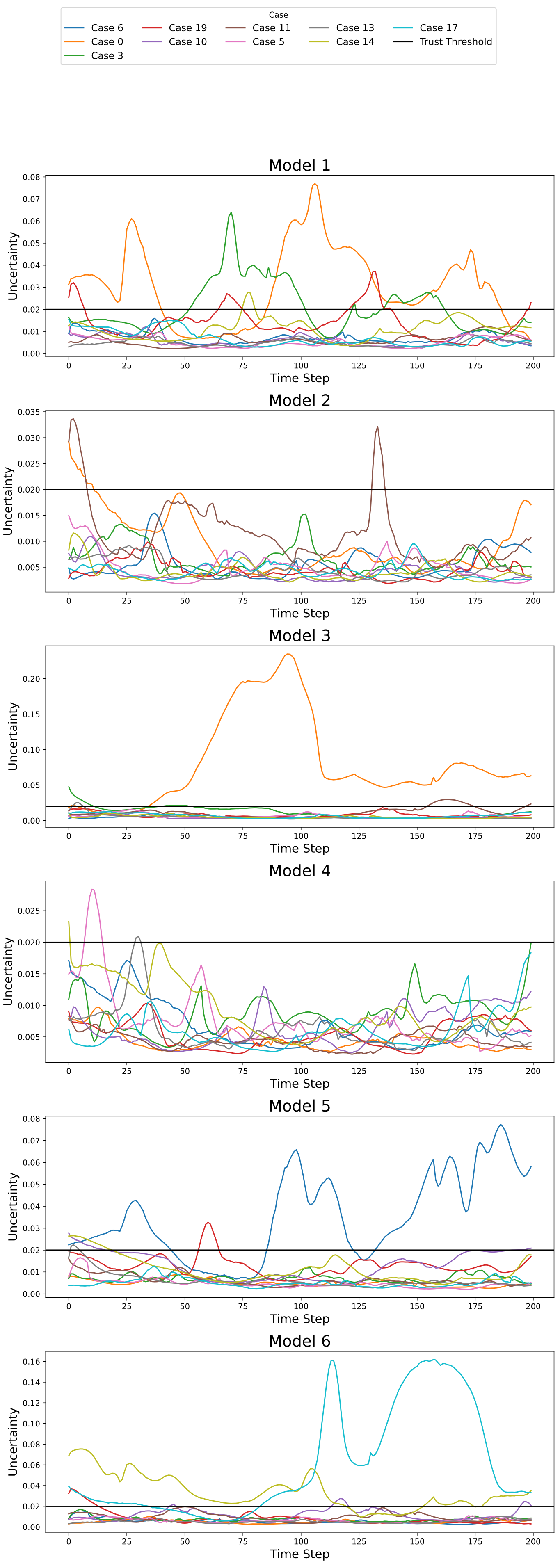

Figure 1: Model 1 to Model 6 with all test cases visualized with the trust region threshold of 0.02.

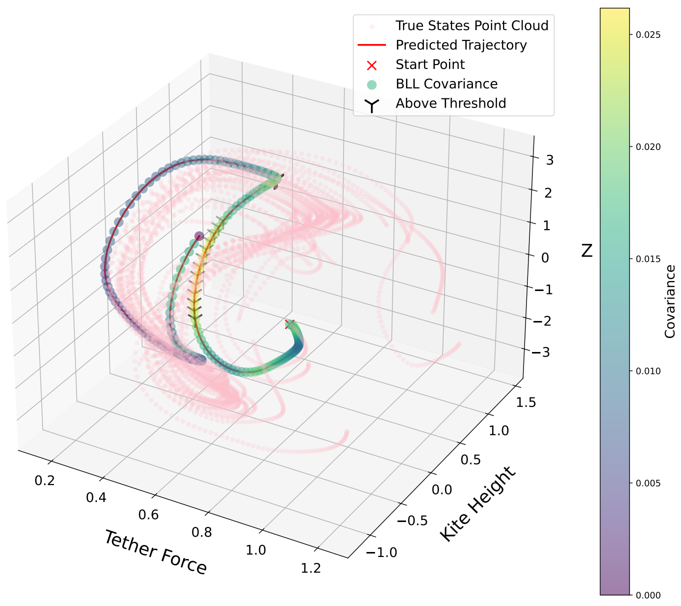

Figure 2: Model one test case four visualized, showcasing the predicted trajectory alongside the true data presented as a scatter plot. Additionally, the uncertainty surrounding the trajectory and the locations surpassing the threshold value are also depicted.

0.03 threshold exhibits approximately half as many violations on average, while the 0.01 threshold shows roughly twice as many. This suggests that utilizing trust regions with a threshold of 0.02 effectively enhances prediction performance by providing a favorable compromise. Moving beyond this threshold, we observe a total rejection of all test cases starting at 0.005, or the rejection of only three or fewer cases at a threshold of 0.05 or larger. However, further experimentation with a broader range of models is necessary to validate this observation.

We consider test case ten from model one as an illustrative example. In Figure 2, this case is depicted, showcasing the predicted trajectory in red, with the starting point denoted by a red cross. The plot includes the entire dataset used for prediction, represented as pink data points. Notably, the covariance, serving as a measure of uncertainty, increases as the trajectory diverges from the actual data points, as shown in the mapping. Furthermore, we have identified locations where the threshold of 0.02 is exceeded, marked as a tri down in black. This alignment underscores the reliability of our uncertainty assessment.

Our observation underscores the impact of data density on prediction certainty. In regions with sparse data points, our confidence in the prediction diminishes. While this analysis focuses on a singular test case, it highlights the importance of conducting further observations across various cases. Nonetheless, consistent trends have emerged in our exploration of different scenarios, emphasizing that moving away from densely populated data regions tends to elevate uncertainty. Moreover, the effectiveness of the 0.02 threshold value in managing uncertainty is confirmed.

VII. Discussion

Introducing Bayesian last layer trust regions into control algorithms comes with certain disadvantages. Firstly, this integration can add complexity to control systems, potentially increasing computational overhead and complicating implementation. Balancing the trade-off between performance improvement and computational efficiency becomes crucial. Additionally, neural networks, including those incorporating BLL trust regions, often function as black boxes, making it challenging to interpret model predictions. This lack of interpretability makes the understanding of the underlying decision-making processes a significant challenge in critical applications where transparency is essential. Moreover, while BLL trust regions provide uncertainty estimates, they may not always offer probabilistic guarantees, limiting their applicability in scenarios requiring precise risk assessment. However, it is important to note that while BLL can offer a degree of probabilistic assurance, it may not always equate to absolute certainty due to the inherent approximations and limitations of the model.

On the other hand, integrating BLL trust regions brings several advantages to control systems. Using BLL trust regions lead to more reliable and effective control strategies, particularly in safety-critical environments. Furthermore, these trust regions enable the quantification of uncertainty associated with neural network predictions, a critical factor in decision-making processes where understanding prediction confidence is important.

In our case study chapter, we conducted a comparative analysis where we experimented with different initial neural network weights to investigate the resilience of the chosen trust region threshold. Variations in the threshold value significantly influence the occurrence of trust region violations across diverse models. We found that setting the threshold at 0.02 strikes a good balance between having too many violations and having no violations, which helped make the predictions more accurate. However, while this threshold appeared effective in our specific scenarios, it is important to approach these findings with critical thinking. The threshold value's optimality might not generalize across all possible contexts and models. Therefore, we cannot say for sure whether this approach definitively works.

We attempted to define a clearer threshold, one that distinctly marks when a violation becomes significant, to provide a better understanding of deviations from the trust region. This effort underscored the complexity and variability inherent in setting such parameters. Additionally, we recognized an opportunity for broader exploration. More extensive testing across different datasets and model configurations is necessary to validate the robustness of the 0.02 threshold and to potentially discover more universally applicable guidelines. This critical reflection highlights the need for ongoing investigation and refinement in this area.

Furthermore, it is important to acknowledge that while the integration of BLL trust regions into control systems shows promise, there remains a considerable need for further scientific research to comprehensively evaluate their use across a diverse range of Model Predictive Control problems in other areas. Future studies should aim to investigate how trust regions perform in various control scenarios, considering different system dynamics, control objectives, and environmental conditions. Additionally, there is a pressing need to delve deeper into the underlying mechanisms of trust regions and discover how they work in different settings. Moreover, several open questions persist regarding the practical implementation and optimization of trust regions. For instance, researchers need to explore the most effective methods for calculating trust regions, considering factors such as system complexity, data availability, and computational resources. Additionally, determining the optimal utilization of trust regions remains an open area. This includes investigating strategies for dynamically adjusting trust region parameters based on real-time feedback or adapting trust region formulations to accommodate evolving system dynamics.

In summary, while the integration of BLL trust regions holds considerable potential for enhancing control system performance and managing uncertainty, ongoing scientific work is essential to fully understand their capabilities and limitations. By addressing these open questions through scientific investigation and experimentation, we can gain new insights into the practical application of trust regions and their widespread adoption in diverse real-world scenarios.

VIII. Conclusion

Despite the promising benefits of BLL trust regions, several challenges remain. One significant challenge is the need to enhance the interpretability of neural network predictions. Our recent case study, which explored the robustness of threshold choices in trust regions across various models, underscores the importance of addressing these challenges to foster trust and acceptance in real-world applications, particularly in safety-critical domains.

To make trust regions more precise and reliable, it is important to explore different methods for calculating and refining them. Additionally, we suggest expanding our research. This entails broadening our case study with a wider range of models and experimenting with different threshold values. It would also be beneficial to examine various Model Predictive Control problems to gain a better understanding of how trust regions function in different scenarios.

In summary, by exploring new ways to improve trust regions and broadening the scope of our research, we can make significant improvements in enhancing control algorithms and their practical applications. Furthermore, further advancements in machine learning techniques and control theory are likely to lead to the development of more sophisticated and efficient control algorithms incorporating BLL trust regions.

References

[1] E. F. Camacho and C. Bordons. *Introduction to Model Predictive Control*, pages 1–11. Springer London.

[2] Francesco Cursi, Valerio Modugno, Leonardo Lanari, Giuseppe Oriolo, and Petar Kormushev. Bayesian neural network modeling and hierarchical mpc for a tendon-driven surgical robot with uncertainty minimization. *IEEE Robotics and Automation Letters*, 6(2):2642–2649, 2021.

[3] Felix Fiedler and Sergio Lucia. Improved uncertainty quantification for neural networks with bayesian last layer. 11:123149–123160.

[4] Felix Fiedler and Sergio Lucia. Model predictive control with neural network system model and bayesian last layer trust regions. In *2022 IEEE 17th International Conference on Control & Automation (ICCA)*, pages 141–147. IEEE.

[5] Ethan Goan and Clinton Fookes. Bayesian neural networks: An introduction and survey.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[7] Laurent Valentin Jospin, Wray Buntine, Farid Boussaid, Hamid Laga, and Mohammed Bennamoun. Hands-on bayesian neural networks – a tutorial for deep learning users. 17(2):29–48.

[8] Armin Norouzi, Hamed Heidarifar, Hoseinali Borhan, Mahdi Shahbakhti, and Charles Robert Koch. Integrating machine learning and model predictive control for automotive applications: A review and future directions. 120:105878.

[9] Sebastian W. Ober and Carl Edward Rasmussen. Benchmarking the neural linear model for regression. *2nd Symposium on Advances in Approximate Bayesian Inference*, abs/1912.08416, 2019.

[10] J. Pohlodek, H. Alsmeier, B. Morabito, C. Schlauch, A. Savchenko, and R. Findeisen. Stochastic model predictive control utilizing bayesian neural networks.

[11] James Blake Rawlings, David Q. Mayne, and Moritz Diehl. *Model predictive control: theory, computation, and design*. Nob Hill Publishing, 2nd edition edition. OCLC: on1242934507.

[12] Max Schwenzer, Muzaffer Ay, Thomas Bergs, and Dirk Abel. Review on model predictive control: an engineering perspective. 117(5):1327–1349.

[13] Markus Walker, Marcel Reith-Braun, Peter Schichtel, Mirko Knaak, and Uwe D. Hanebeck. Identifying trust regions of bayesian neural networks. In *2023 IEEE Symposium Sensor Data Fusion and International Conference on Multisensor Fusion and Integration (SDF-MFI)*, pages 1–8. ISSN: 2767-9357.

[14] J. Watson, Lin J. A., P. Klink, J. Pajarinen, and J. Peters. Latent derivative bayesian last layer networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2021.