Safe Data-Driven Model Predictive Control of Systems with Complex Dynamics

Ioanna Mitsioni\textsuperscript{1}, Pouria Tajvar\textsuperscript{1}, Danica Kragic, Jana Tumova, and Christian Pek

Abstract

In this paper, we address the safety and efficiency of data-driven model predictive controllers (DD-MPC) for systems with complex dynamics. First, we utilize safe exploration of dynamical systems to learn an accurate model for the DD-MPC. During training, we use rapidly exploring random trees (RRT) to collect a uniform distribution of data points in the state-input space and overcome the common distribution shift in model learning. This model is also used to construct a tree offline, which at test time is used in the cost function to provide an estimate of the predicted states’ distance to the target. Additionally, we show how safe sets can be approximated using demonstrations of exclusively safe trajectories, i.e. positive examples. During test time, the distances of the predicted trajectories to the safe set are used as a cost term to encourage safe inputs. We use a broken version of the inverted pendulum problem where the friction abruptly changes in certain regions as a running example. Our results show that the proposed exploration algorithm and the two proposed cost terms lead to a controller that can effectively avoid unsafe states and displays higher success rates than the baseline controllers with models from controlled demonstrations and even random actions.

1 INTRODUCTION

Data-driven control is increasingly applied in robotic applications and has enabled tackling a plethora of tasks that were previously intractable \cite{13}. However, providing performance guarantees such as robustness, stability, and safety is particularly challenging in data-driven approaches \cite{6}. Data-Driven Model Predictive Control (DD-MPC) can potentially combine the benefits of data-driven control with performance guarantees such as

\textsuperscript{1}Equal contribution
Figure 1: The broken pendulum is a more complicated variation of the pendulum swing-up. Compared to its vanilla counterpart (Fig. 1c), a broken pendulum with 2 segments of different friction coefficients requires almost three times as many abstraction cells to linearize (Fig. 1d). In each cell the dynamics is approximated with a linear function, the cells are decomposed into smaller cell until the linearization error in each cell is smaller than a predefined threshold. The addition of more segments does not dramatically increase the numbers of cells. A DD-MPC can easily display undesired behavior in such tasks and fail to solve them with input constraints. The discontinuities will affect the predictions and in order to make the system react faster, we make it more aggressive. More aggressive inputs might drive the pendulum in another high-friction region where it will slow down and will need more aggressive inputs, thus getting trapped and failing the task.

robustness and stability that typically rely on analytic models [4]. Several challenges remain to be addressed towards achieving full versatility of data-driven approaches together with performance guarantees.

In this paper we study three important challenges in DD-MPC: safety verification, model learning, and myopic predictions. Firstly, safety verification [21] is a complex task when the set of safe behaviors is not explicitly given and has to be inferred from data. Approaches that address this challenge by learning behavior classifiers (e.g. [15]), rely on both positive (i.e. safe) and negative (i.e. unsafe) examples. Obtaining unsafe behaviors however, can be very costly or, worst case, infeasible for many applications. Secondly,
model learning is an integral part of DD-MPC, as DD-MPC relies on previously collected data from the system to construct a dynamical model. While there are closed-form approaches to collect informative data from a linear system, i.e. persistently exciting data, doing so for a non-linear system is not trivial. Lastly, by myopic predictions we refer to the difficulty of assessing the distance to the target state in systems with complex dynamics. Estimating this distance is not-trivial and action optimization using short-term predictions of the state evolution can be misleading. For example, consider the swing-up task for a cart-pendulum system with bounded inputs. Even in the presence of an analytical model, if the input is not sufficient to bring the pendulum to the upright position in one go, a simple distance function based on the pendulum angle cannot lead to a successful swing-up. Similarly, classical continuous-time approaches such as Lyapunov-based control cannot be directly applied to this problem as finding a globally stabilizable Lyapunov function becomes substantially challenging.

All of the aforementioned challenges are even more prominent when the system has temporally or spatially varying dynamics that change suddenly, such as a broken pendulum, i.e. when the pendulum’s base joint has different friction coefficients depending on the angle like in Fig. 1. To complete a full swing-up, the pendulum will cross either one of the shaded regions with a friction coefficient that is two or four times larger than in the non-shaded areas. The simplest strategy would be to increase the control effort in order to exit these zones and reach the goal position. However, high control efforts will overshoot and potentially enter the opposite high-friction area. When in the high-friction area, if the control effort is too small, the pendulum will not compensate gravity and fall, while if too high, it will oscillate between the upper bounds of the two areas, overshooting the goal position. To address this problem efficiently, one would need to create a switching dynamics model for the different regions of the state-space, which will not scale well for higher dimensions or more complicated problems. On the other hand, DD-MPC methods address this problem by learning the model based on collected data and then using model predictions to optimize the control inputs in a receding horizon manner. However, the aforementioned challenges of DD-MPC need to be addressed in order to exploit its advantage, while ensuring desired behaviors.

In this work, we take inspiration from unsupervised outlier detection to detect unsafe behaviors. To this end, we exploit that collected datasets to train the dynamics model of a DD-MPC, are chosen to display good enough performance (e.g., as evaluated by a human worker). Given a clustering of the collected trajectories, we detect the regions of high sample density that correspond to safe states, as well as the outliers which will correspond to unsafe/undesired states. This information is used with the controller to avoid unsafe states and drive the system towards the safe ones. We increase the informativeness of the data collection process by incorporating short intervals of (bounded magnitude) random actions in predefined trajectories. Finally, we address myopic predictions by incorporating information about the long-term dynamics in the controller through RRT\(^*\). In order to make RRT\(^*\) applicable to this setting, we provide a steer function for nonlinear systems with dynamical uncertainty. In addition, instead of constructing trees from an initial state, we use inverse-time dynamical models to construct backward reachable trees for a given target state. The benefit is that the same tree is reusable for any initial state. For steering,
we construct locally stabilizing controllers using the multi-step feedback control design formulation we proposed in [22].

We exemplify our method through a broken pendulum. The original version of the swing-up pendulum is a well-known task that has been extensively used to benchmark control methods from different communities. It is a popular benchmark because it strikes a balance with complex dynamics that are still fully-observable and, depending on the state-space representation, a manageable dimensionality. By including the regions of different friction, the dynamics are more complex and the task becomes an excellent test-bed to showcase the potential and limitations of DD-MPC.

Related work

Sampling based control synthesis

A sampling based motion planning method, namely rapidly-exploring random trees (RRT), is proposed as a solution to control problems with non-convex reachable sets in [23]. The main idea is to construct a tree in the state space with its root being an initial state. The tree is expanded through random sampling in the state space and adding a child to the closest existing tree node with a random input. It has been shown that the probability of having a tree node in any open subset of the reachable set approaches 1 as the number of samples approaches infinity resulting a full coverage of the reachable set. In addition in [23] local LQR controllers are designed along tree vertices to ensure robustness of the constructed trajectories. While RRT is shown to be an efficient method for trajectory planning in a nonlinear dynamical system, the obtained trajectory can be arbitrarily sub-optimal. As a result methods such as [18] have been proposed to guarantee asymptotic optimality as the number of samples increases. RRT*, as asymptotically optimal counterpart of RRT was initially introduced for geometric motion-planning problems without dynamics but can be extended to any system as long as a steer function can be constructed that steers a node to a potentially better parent node.

Safety in robotics is usually expressed through the satisfaction of safety constraints that may refer to the control input, the state of the system or its stability properties. Considering the constraint satisfaction, safety can be categorized in the following groups, starting from the strictest [6]: (i) guaranteed safety (ii) safety with probability p, (iii) safety encouragement. Safety levels i and ii to a smaller degree, are only achieved with models of very specific structures (e.g. linear, control-affine), which are not advantageous for the changing dynamics we consider. Data-driven methods that set no prior on the dynamics model can mostly encourage safety or offer a chance constraint that still implies some unsafe actions. A thorough analysis of the three levels can be found in the survey paper [6].

Three common approaches towards control safety are: manual design of fail-safe mechanisms, incorporation of safety terms in the cost/reward function and restriction of the possible control inputs. Fail-safe mechanisms, like the ones in [3], rely on setting empirical conditions for task interruption. Their applicability is limited and they do not scale well, but can be very efficient in combination with the other methods. The incorporation of the safety constraint in the cost function that is optimized usually aims to modify the behavior as a whole. e.g. to encourage gentle contacts with unknown objects as in [11]. Lastly,
modifying the control inputs can be implemented as pre-filtering \[14,15\] to reduce the feasible input space or even as an adjustment of the input a-posteriori \[7\]. Often, the inputs are restricted to maintain the system in a safe set. When the dynamics of the system are known, safe sets can be computed with Control Barrier Functions (CBFs) \[2,8\], through Invariant Sets \[17\] or based on the Regions of Attraction (ROA) \[5\]. Most of these methods however require strong priors on the dynamics models in order to make the calculations feasible, even when they are approximated through data. For example, in the data-driven method presented in \[12\], safe exploration and deployment are tackled through ROAs, the model prior is assumed to be known and the model error Lipschitz continuous. A less restrictive approach is presented in \[27\] where model order reduction is used instead for an initial estimate of the safe set.

An efficient way of alleviating some of the modelling assumptions, is to rely on data for the estimation of the safe sets as well. This can be done by learning binary classifiers on data collected by demonstrations \[25\] or successful task executions \[24\]. In this work, we follow the same approach and utilize datasets that complete the task while satisfying safety to learn safe sets in the latent space. A similar approach to ours is presented in \[26\] where the safe set is learned from images, that are encoded in the latent space, as a binary classification task. Demonstrations and online environment interactions are used to learn iteratively improving policies in a reinforcement learning scheme. In contrast to us, the authors rely on both safe and unsafe samples to learn the classifier. Furthermore, the latent space is used to sample datapoints in order to train the dynamics model to plan and estimate whether those plans will likely complete the task, as opposed to using the latent space to calculate the distance of a state from the safe set.

2 PRELIMINARIES AND PROBLEM DEFINITION

Preliminaries

We indicate the set of real numbers by \(\mathbb{R}\), an \(n\)-dimensional vector space by \(\mathbb{R}^n\), and an \(m \times n\) matrix space by \(\mathbb{R}^{m \times n}\). We use \(v_i\) to refer to the \(i\)-th element of a vector \(v \in \mathbb{R}^n\). A closed interval between two values \(a, b \in \mathbb{R}\), \(a \leq b\) is denoted as \([a, b]\).

The Minkowski sum \(A \oplus B\) of two subsets of an \(n\)-dimensional vector space \(A \subset \mathbb{R}^n\) and \(B \subset \mathbb{R}^n\) refers to the following set:

\[
A \oplus B = \{a + b \mid a \in A, b \in B\}.
\]  

A zonotope \(Z(\mu, G) \subset \mathbb{R}^n\), where \(\mu \in \mathbb{R}^n\) and \(G \in \mathbb{R}^{m \times n}\), refers to the following convex and centrally symmetric set:

\[
Z(\mu, G) = \{x \in \mathbb{X} \mid x = \mu + G\omega, \omega \in [-1, 1]^n\},
\]  

where \([-1, 1]^n \subset \mathbb{R}^n\) represents the Cartesian product of closed \([-1, 1]\) intervals. We refer to \(\mu\) as the center of the zonotope and \(G\) as the set of generators. \(Z.\mu\) and \(Z.G\) are used to refer to the corresponding parameters of the zonotope \(Z\).
We express the time discretization of system dynamics subject to zero-order hold (ZOH) control input

\[ x_{t+1} = f(x_t, u_t). \]  

(3)

where \( x \) and \( u \) correspond to system state and control input respectively.

**Problem Definition**

We consider a system with unknown dynamics that are approximated by a non-parametric nonlinear function and tasks with a fixed goal. We define a *transition* as a finite trace in the measurement-input space. We formulate the property of safety as a function of transitions, noting that the classical notion of safety as a function of state is a special case of this formulation. Considering a transition, in contrast to a state, allows for a richer safety specification that can, for example, include limiting oscillatory behaviors. Furthermore, utilizing transitions, improves the applicability of safety evaluations as transitions are also beneficial to systems without full state observability. Given sets of safe transitions, safety is judged by how similar new trajectories are to them.

**Running Example**

We first present a normal cart-pendulum that is controlled to the upright position. For this system, \( x_1 = \theta \text{ (rad)} \) is the angle to the desired position, \( x_2 = \dot{\theta} \text{ (rad/s)} \) the rate of angle change, \( u \) the control input, \( m = 0.2 \text{kg} \) the mass of the pendulum, \( l = 0.3 \text{m} \) the length of the pendulum, \( J=0.006 \text{kgm}^2 \) the mass moment of inertia and \( b = 0.1 \text{N/m/s} \) the friction coefficient at the base joint. The dynamics are then given by:

\[
\begin{aligned}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= \frac{mg}{J} \sin x_1 - \frac{b}{J} x_2 + \frac{l}{J} \cos x_1 u.
\end{aligned}
\]  

(4)

The broken pendulum that will be the running example in the following sections occurs when the friction coefficient is not constant but depends on the angle, i.e. \( \tilde{b} = \tilde{b}(\theta) \). More specifically, we double and quadruple the friction coefficient for 2 segments around the upright position as seen in Fig. 1, making the swing-up task significantly harder from the positive side:

\[
\tilde{b} = \begin{cases} 
4 \cdot b, & 0.5 \leq \theta \leq 2.0 \\
2 \cdot b, & -2.0 \leq \theta \leq -0.5 
\end{cases}.
\]  

(5)

**3 METHOD**

**Overview**

We collect system trajectories from a set of initial states using a controller that, for short intervals throughout the trajectory execution, employs random inputs of bounded magnitude and is verified to produce safe results. The collected dataset \( D_{train} \) comprises the desired
3. METHOD

Figure 2: System overview. Given current states $x_t$ and potential DD-MPC inputs $u_{t:H}$, the dynamics model $f$ is used to predict the next states. The safety of these predictions is estimated by the safety layer through their distance $\tilde{d}_S$ to the safe sets and the RRT tree $\mathcal{N}$ is used to evaluate the predictions’ distance from the goal through its dynamics.

In an offline step, using $f$ as system dynamics, we construct a tree $\mathcal{N}$ in the state space using an optimal variation of rapidly exploring random trees RRT*. Because of the potential discrepancy between the learned model and actual system dynamics, this tree cannot be used to control the system, however, it can be used to provide a distance estimate between the root of the tree and any state within the tree’s support.

During test time, the dynamics model is used with DD-MPC to predict future states and optimize control inputs, the distance estimate from the tree is incorporated in the DD-MPC cost function offering foresight for the long-term dynamics. The predicted future states

$$\hat{x}_{t+1:t:H_t} = f(x_t, u_{t:t+H_t-1}).$$  \hspace{1cm} (6)
to reach $\mathbf{x}_{t+H_t}$ form a trajectory that is mapped to the latent point $\mathbf{z}_{t+H_t}$. The control inputs are also penalized proportionally to the distance $d_{S'}$ between $\mathbf{z}_{t+H_t}$ and the safe set’s boundary, encouraging the controller to move towards, or remain, within the safe set. The overview of this procedure can also be seen in Fig. 2.

**Data-Driven Model Predictive Control**

For DD-MPC, we require a model of the system’s dynamics $f$ that maps current system state and input $\mathbf{x}_t \in \mathbb{R}^n$, $\mathbf{u}_t \in \mathbb{R}^m$, to the next state $\mathbf{x}_{t+1}$:

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t).$$  \hspace{1cm} (7)

Given the model $f$, the DD-MPC determines the optimal control input $\mathbf{u}^*_{t:H_t-1}$ over a horizon $H_t$, by minimizing a cost function $C$ that imposes constraints on states and inputs and reflects the desired system behavior:

$$\mathbf{u}^*_{t:H_t-1} = \arg \min_{\mathbf{u}} \sum_{k=0}^{H_t-1} C(\mathbf{x}_{t+k+1}, \mathbf{u}_{t+k}, \mathbf{z}_{t+H_t}),$$

subject to $\mathbf{x}_{t+k+1} = f(\mathbf{x}_{t+k}, \mathbf{u}_{t+k})$,

$$\mathbf{x}_t = x(t),$$

$$\mathbf{z}_{t+H_t} = f_{enc}(\mathbf{x}_t, \ldots, \mathbf{x}_{t+k+1}).$$ \hspace{1cm} (8)

To solve the optimization problem Eq. (8), we employ random shooting [19] which is a numerical method. With random shooting, every optimization cycle consists of generating a set of $k$ feasible inputs $\mathcal{U}$ to acquire the model’s next states. Next, the cost for every input is evaluated for the desired prediction horizon and we ultimately determine $\mathbf{u}^*_{t:H_t-1}$ based on the lowest cost in the set of feasible inputs. The optimal input $\mathbf{u}^*_{t:H_t-1}$ is then applied for the immediate next time-step and we proceed to the next optimization cycle. By only applying the first step of the input and re-sampling from the environment, DD-MPC reduces potential plant-model mismatches and is able to adapt faster to changing dynamics as opposed to a control scheme that optimizes the input for the entire task duration.

The performance of an DD-MPC is strictly tied to the model’s accuracy despite re-sampling from the environment at every cycle. Cost evaluations based on an inaccurate model will not reflect the true outcome of the candidate input and will degrade the system’s performance, even leading it to undesired behaviors. There is always a trade-off to consider between longer horizons that will help the system be less myopic, but with a less accurate learned dynamics model, and short horizons that help with the accuracy by re-sampling the real system, but only have a local understanding of the task. This trade-off is especially prominent in tasks with complex dynamics that may even change in some regions of the state-space, such as with the broken pendulum.

**Example 1**

*Mismatches between the model and the actual system are even more common in tasks with changing dynamics. For example in Fig. 3 the friction coefficient changes at angle $\theta_f$.***
3. METHOD

Any model estimates before time $t + 1$ will be invalid, leading to wrong trajectories (black). However, re-sampling from the system at $t + 2$ can adjust the prediction and lead the system closer to the actual trajectory (green). The ability to easily correct the control inputs based on observations from the environment is what makes receding horizon DD-MPC a good choice for these tasks.

Safe sets in the latent space

Latent Space Transitions  We learn a compressed representation of the system trajectories that distills the desired system traits. This representation is learned through a $\beta$-Variational Autoencoder ($\beta$-VAE) [10] that maps system transitions into a regular two-dimensional latent space through $f_{\text{enc}}$. Every latent point $z_t$ corresponds to the trajectory $X_{t:t+H_t}$, that starts from the current state $x_t$ and ends at the desired prediction horizon $x_{t+H_t}$: $X_{t:t+H_t} = x_t, \ldots, x_{t+i}, i \in [1, H_t]$. Namely, $z_t = f_{\text{enc}}(X_{t:t+H_t})$ with $f_{\text{enc}} : IR^{n \cdot (H+1)} \rightarrow Z$. The control input is omitted to remove the dependency on the controller used to gather data and allow the system to generalize better. This is particularly important on tasks with partially observable dynamics where different inputs could lead to similar trajectories. The $\beta$-VAE first encodes the input trajectory into the latent variable $z_t$ with prior distribution $p(z)$ and learns to approximate the posterior distribution $q(z|X)$ that is then used by the decoder that reconstructs the input into $\hat{X}$. The optimization criterion is a combination of the reconstruction loss, expressed by the Mean Squared Error (MSE), and the Kullback-Leibler (KL) divergence between the posterior and a normal distribution $N(0, 1)$, weighed by a factor $\beta$:

$$L_{\text{vae}} = MSE(\hat{X}) + \beta \cdot KL(q(z|X), N(0, 1))$$  \hspace{1cm} (9)

The scaling factor $\beta$ controls the trade-off between accurate reconstruction and regularization of the latent space.
The regions of high sample density in the latent space coincide with system transitions that are consistently safe, as the $\beta$-VAE is trained only on pre-approved datasets. Using a density-based clustering algorithm like DBSCAN \cite{9}, can distinguish these clusters, $C_i$, $i \in \mathbb{N}$, in the presence of noise and detect outliers and thus approximate the safe sets $\tilde{S}$. The points that are close in the latent space are similar once decoded, as the encoding is regularized. Therefore, the euclidean distance of a point $z_t$ to a cluster $C_i$ can be used to approximate the distance of $z_t$ from the corresponding safe set:

$$
    d_{\tilde{S}}(z_t) = d_{C_i}(z_t)
$$

(10)

**Safe Set** Using the obtained safe set, we determine whether a given system transition is classified as safe by checking if the encoded version is enclosed in our safe sets:

$$
    X_{t:t+H_t} \text{ is safe} \iff z_t \in \tilde{S}.
$$

(11)

Since our safe sets usually under-approximate the maximum safe set, we will eventually encounter transitions that are not enclosed in the safe sets. To steer the system back to the safe sets, we compute the signed distance of the encoded transition to the closest safe set.

$$
    d_{\tilde{S}}(z_t) = \min(d_{C_i}(z_t)), \ i \in \mathbb{N}
$$

(12)

By adding the distance as a cost to the DD-MPC formulation, we encourage solutions that transition the system back to the safe sets.

**Example 2**

*In Fig. 4, the system starts from the latent point $z_{t_1}$ which lies outside the safe set. The trajectory $X_{t_2:t_2+H_1}$ is bringing the system closer to the target angle and the corresponding latent point $z_{t_2}$ is also approaching the safe set. Finally, the trajectory $X_{t_3:t_3+H_1}$ is the desired one and the latent point $z_{t_3}$ is now inside the safe set $C_1$.***

**Sampling based control synthesis**

We introduce a dynamic version of the RRT* algorithm to synthesize a controller for a given system dynamics. In this section we discuss how certain functions such as near, steer, and rewire are adapted for our problem setting. Throughout this paper we refer to the dynamic implementation of RRT* as $\mathcal{N} =$ DynamicRRT(...) where the inputs are the system dynamics and the starting state respectively and the returned value $\mathcal{N}$ is a tree construct.

Similar to the geometric RRT and RRT* approaches, in the dynamic RRT the expansion of the tree happens through random sampling of the state space and selecting the closest existing tree. This approach tends to favor nodes with larger Voronoi regions and is the key to encourage uniform distribution in tree nodes. In our implementation of the dynamic RRT the new node will be generated by applying a random input to the selected near node rather than using a control-based steer function to ensure feasible expansion.
Starting from the target state $x_{target}$, we construct a tree using reverse-time dynamics in RRT, i.e. from the target to the initial state. The reverse-time dynamics of the discrete-time model can be approximated as follows:

$$x_{t-1} = x_t - f(x_t, u_t) := f^-(x_t, u_t).$$  \(13\)

Since we are using reverse-time dynamics, we refer to this tree as backward reachable tree $\mathcal{R}^- = \text{DynamicRRT}(f^-, x_{target})$. In a backward tree we obtain trajectories going from the tree node to the target state following system dynamics. Given the probabilistic completeness of RRT, we will eventually find such a trajectory from any subset of the backward reachable set, i.e. the set of states that can reach the target. Therefore, we can use the backward reachable tree both as an approximation of the backward reachable set and as a guide for how to reach the target.

Example 3
In Fig. 5 we see the backward reachable tree for the broken pendulum starting from the upright position as tree root. We note that the backward reachable set in this case is indeed non-convex and does not cover the entire state-space; as a result we cannot synthesize a global controller for the system using classical continuous time control approaches or short-term predictive controllers. Furthermore, we observe that tree nodes are sparser on the high-friction side, as expected. The pendulum can only converge from the high-friction side if it enters the high-friction area with a high enough velocity, creating a dynamically imposed...
narrow passage in that region. A given trajectory in the constructed tree, however, can be arbitrarily sub-optimal. The swing up trajectory using the unrewired tree includes several avoidable turns before convergence as seen by the red dashed trajectories in Fig. 5.

We present a method that enables tree rewiring for nonlinear systems using locally stabilizing controllers. Recalling that for tree rewiring we require a steer function, we construct such a function for a node as follows:

1. Computing the a local linearization of the dynamics around the node
2. Constructing the multi-step controller
3. Computing the backward reachable set
4. Rewiring nodes that are in the backward reachable set and will have a lower cost after rewiring.

In contrast to geometrical planning, in dynamical planning both the near function and the steer function depend on the reachability properties of the system. The backward reachable set of a state in a linear system subject to disturbance is the following Zonotope. Checking whether a point belongs to a Zonotope is a linear programming problem; to reduce the computational complexity of inclusion checking we propose a conservative alternative for checking the inclusion. We compute the state $x$ as a convex combination of the generators $Z.G$ of a zonotope $Z$:

$$x_Z = Z.G^#x,$$

where $Z.G^#$ represents the pseudo-inverse of the generator matrix.

If $||x_Z||_{\infty} \leq 1$, then the state $x$ is inside the zonotope as this is a valid combination of the zonotope generators that corresponds to the state $x$. However if $||x_Z||_{\infty} > 1$ the state may still be inside the zonotope since the pseudo-inverse matrix is only one of the possible convex combinations the generators; as a result the proposed approximation is conservative.

**Example 4**
The same backward reachable tree after rewiring is shown in Fig. 6. Rewiring reduces the average swing-up convergence time from different states in the state space by 40% (from 234 to 140 time steps). The red dashed trajectory is the swing-up motion from the same initial state.

**Data collection for model identification**

One of the challenges in model-based data-driven control is data collection. Ideally, the collected data points are uniformly distributed in the reachable subset of the $n + m$-dimensional state-input space, where $n$ is the dimension of the state space and $m$ is the dimension of the input space. Data points from such a distribution are representative of the
3. METHOD

**Figure 5:** A backward reachable tree for the broken pendulum system. As the number of tree nodes increases, areas where there are no nodes are concluded to be outside the target’s backward reachable set. The red dashed trajectory is an example of the pendulum swing-up motion from $\theta = -\pi$ using this tree; due to the sub-optimality of RRT, we observe that the state takes several turns in the state-space before converging to the upright position.

**Figure 6:** Same tree after rewiring. The red dashed trajectory shows the swing-up motion from $\theta = -\pi$ using the rewired tree; The resulting trajectory is shorter compared to the unreiwred counterpart and avoids unnecessary turns.
Algorithm 1: Iterative data-collection / model identification

1. **Input:** Initial model \( \hat{f}_0(x, u) \), Uncertainty bound \( \omega_0(x, u) \);
2. **Output:** Refined model and uncertainty bound \( \hat{f}_k(x, u), \omega_k(x, u) \);
3. **for** \( i \in [1 : k] \) **do**
   4. Generate tree with \( \hat{f}_{i-1}(x, u) \);
   5. \( ([x_0 : x_T], [u_0, u_T]) \) find safe branch;
   6. execute branch record data;
   7. \( \hat{f}_i(x, u) = \text{FitModel}(D) \) \( x_0 \leftarrow x(T) \)

system dynamics. However, since the process of data collection affects the system state, such distribution cannot be guaranteed, depending on the system dynamics. The most common approach to data collection is using pre-designed controllers. Since the action at a given state is pre-determined in this approach, the data is collected from an \( n \)-dimensional manifold in the \( n + m \)-dimensional space, resulting in the distribution shift problem in model identification. Alternatively, dynamical data may also be collected through random actions; however, even disregarding the potential safety violation, the collected data through random actions is also subject to distribution shift. For example, in a simple single integrator dynamical system, random velocity commands generates data similar to random-walk, i.e. normal distribution around the starting state.

In this section, we propose an iterative data-collection and model identification method to improve model accuracy through increased variety in data while respecting safety specifications. We build on two observations: Control based data-collection from random initial states provides variety in the state space and data-collection from random actions provides variety in the input space.

Let us assume to be given an initial estimated model of the dynamics \( \hat{f}_0(x, u) \). Such model can be obtained either through physical parameters of the system or constructed using a first set of data-points collected from a local controller. We define the residual model, i.e. the discrepancy between the actual dynamics and the initial model as follows:

\[
\hat{f}_0(x, u) = f(x, u) - \hat{f}_0(x, u).
\]  

(15)

We further assume that the residual model is bounded and the bound can be conservatively approximated as follows:

\[
|\hat{f}_0(x, u)| < \omega_0(x, u).
\]  

(16)

4 RESULTS

In this work, our goal is to address three key limiting factors in DD-MPC:

1. The accuracy of the dynamics model
2. The often myopic predictions of a DD-MPC
3. The safety of the system

To address the first point, we propose a more effective method of collecting data (Sec. 3) by balancing pre-described controllers with random, safe, actions that cover a larger part of the state-input space and lead to more accurate models. The second point is addressed by enhancing the DD-MPC by incorporating foresight about the nonlinear task dynamics, through the proposed RRT and steering function (Sec. 3). The last objective is addressed by the proposed safety layer that, using only safe data, encodes the desired system traits into a latent space that is then used to extract the approximated safe sets for the system. To demonstrate these points we conduct the following experiments:

- In Sec. 4 we demonstrate how the distribution shift can affect the performance of the dynamics model and how incorporating randomness in the actions can improve it.
- In Sec. 4 we discuss implicit and explicit methods for outlier detection and present the approximated safe set.
- In Sec. 4 we evaluate the accuracy of the safety layer and discuss how the combined method improves it.
- Finally, in Sec. 4 we conclude by comparing the overall performance of our method against baselines.

Setup of Our Framework

For our experiments, we use the broken pendulum as described in Sec. 2. The dynamics model in Sec. 3 and the $\beta$-VAE in Sec. 3 are networks with fully-connected layers. More details on their architecture and training can be found in the Appendix, Sec. ???. The dynamics model is used to predict $H_t = 10$ timesteps in the future. During the DD-MPC execution, we sample $k = 10$ feasible inputs from $U$ that are bounded between $[-10, 10]$. For all the comparisons, the initial position range for the trials is $\theta_0 \in [-1.5\pi, 1.5\pi]$ rad and the initial velocity is in the range $\dot{\theta}_0 \in [-10, 10]$ rad/s and all of the baselines start from the same initial state (position and velocity).

Full Rundown for Pendulum Example

We first collect data with the procedure described in Sec. 3. The trials that converge with constrained inputs, comprise the training set $D_{\text{train}}$ for the dynamics model Eq. (6) and the safety layer described in Sec. 3 that are used with the controller Eq. (8). The cost function used for the DD-MPC is:

$$C(\theta_{t+k}, \dot{\theta}_{t+k}, u_{t+k}, z_{t+k}) = c_\Theta \|\theta_{t+k} - \theta_{\text{target}}\|^2 + c_U \|u_{t+k}\|^2 + c_D \left(e^{-\min(0,d)} - 1\right) + c_N \left(\mathcal{N}.dist(\theta_{t+k}, \dot{\theta}_{t+k})\right),$$

(17)
where $\theta$ is the angle of the pendulum, $d_{\tilde{S}}$ is the distance to the approximated safe set and $c_\Theta, c_U, c_D, c_\aleph$ are scaling constants for the different cost terms. The first term encodes the desired angle for the pendulum, the second term encourages solutions of small magnitude and the third term assigns costs proportionally to the distance between $z_{t+k}$ and the safe set $\tilde{S}$ if $z_{t+k}$ is originally outside of it. If the $z_{t+k}$ is already in the safe set ($d_{\tilde{S}} \geq 0$), that solution is not penalized. The last term incorporates the distance estimate from the state to the target using the constructed tree $\aleph$. When the state is far from the desired upright angle, $c_\aleph$ is the dominant term compared to $c_\Theta$ to avoid potential local optima and close to the target angle $c_\Theta$ becomes the dominant factor, enabling local regulation.

Lastly, if the controller is unable to lead the system closer to the safe set within a number of steps $N_{\text{max}}$, the trial is ultimately unsafe and aborted. Both the DD-MPC and the safety accuracy results are averaged over 1000 trials.

**Distribution shift in data collection**

In this section we show how the distribution shift effect can be reduced through different randomized action approaches and compare their effectiveness. For the broken pendulum problem, let’s compare acceleration prediction accuracy between when the model is trained on data coming from trajectories generated using different system interaction policies and when it is trained on randomized data in the state-input space. We are particularly interested in acceleration prediction as the most challenging variable since it non-linearly depends on the state and instantaneously changes with input.

Even though the latter case is practically impossible to collect on a real system, it can serve as an upper limit on the achievable accuracy given a fixed network architecture and data size.

In Fig. 7 we observe that the lowest prediction error is achieved when the model is trained on randomized data in the state-input space, i.e. FRAND, as expected. Looking at the random walk data-collection method, we observe that starting from the upright position results in slightly higher accuracy since the equilibrium is unstable and as a result a larger part of the state space can be reached; however in both cases the error is relatively high as the data is collected locally and cannot be generalized to the dynamics. We observe using the RRT based controller from different initial states has provided a richer data-set; recalling that data points in this method come from a 2D subspace of the 3D state-input space, we observe that model learning can be further improved by adding randomized inputs during the trajectory execution.

**Outlier Detection**

To define the safe sets $\tilde{S}$ in $Z$, we first need to detect the regions of high density. This can be done either implicitly, using density estimators and isometric contours of the estimated log-likelihood, or explicitly, through clustering algorithms directly on $Z$. For the implicit approach, we are exploiting the fact that the regions that resemble the training set the most will form peaks with high log-likelihoods. Around the peaks, we can calculate the isometric contours that correspond to different thresholds of log-likelihoods and
4. RESULTS

Figure 7: Prediction error of models trained using different data collection methods: 

- **RW $\theta = \pi$:** Random walk from the fallen state,
- **RW $\theta = 0$:** Random walk from the upright state,
- **RRTC**: RRT based control from random initial states,
- **RRTC + rand**: 50% random actions during RRT based control,
- **FRAND**: Fully random sampling of the state/input space,

as a consequence, different degrees of "in-distribution", or safe, samples. Given a desired threshold, the safe set is the combination of the contours. The advantage of this approach is that density estimation methods can adapt to distributions of arbitrary complexities, as is the case with Normalizing Flows [20]. However, setting thresholds for log-likelihoods is not intuitive and is very task-dependent, which limits the applicability of these methods. On the other hand, for the explicit approach we rely on the fact that the latent points $z$ will naturally form clusters which are straightforward to detect with clustering algorithms. The parameter to choose for this approach is the maximum neighborhood size which can be more intuitive than a threshold for the log-likelihood of a sample. Based on our definition of safety (safe samples are the ones that are consistently good and often observed), it is very important that the clustering algorithm is able to perform with noisy samples and detect outliers that are not part of the clusters' core samples. Furthermore, to make our method applicable to a multitude of tasks, we do not heavily regularize the latent space which would result in more convex shapes and we do not enforce even cluster sizes or a specific number of clusters. For all of the above reasons, we chose to perform cluster-
Figure 8: The approximated safe sets based on the samples of $D_{\text{train}}$ using DBSCAN.

ing with DBSCAN [9] which detects core samples of high density and constructs clusters around them.

The resulting safe sets of the latent mapping of $D_{\text{train}}$ can be seen in Fig. 8. For the remainder of this paper, we will visualize only the boundaries of the approximated safe sets and omit the training samples for clarity of presentation.

**Safety Check**

Tables 0.1 and 0.2 present the safety layer’s performance averaged over 1000 trials on its own and with the RRT cost, respectively. From Table 0.1, we see that the safety layer has an accuracy of 79.8. Its Recall (how good it is at detecting safe trials in general) is 91.6. However, the Precision (how well it detects safe executions and does not mistake unsafe ones as safe) is at 73, implying that the standalone safety layer can overestimate the safety of a trial. By incorporating the RRT cost, the performance is greatly improved and reaches a Precision of 98.7 and a Recall of 97.9, effectively minimizing the overestimation of safety
4. RESULTS

<table>
<thead>
<tr>
<th>Actual</th>
<th>Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive</td>
<td>0.916</td>
</tr>
<tr>
<td>Negative</td>
<td>0.310</td>
</tr>
</tbody>
</table>

Table 0.1: Normalized confusion matrix for safety check performance without RRT cost. Positive examples correspond to safe ones and negative examples to unsafe ones accordingly. **Accuracy:** 80.3, **Precision:** 73, **Recall:** 91.6.

<table>
<thead>
<tr>
<th>Actual</th>
<th>Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive</td>
<td>0.978</td>
</tr>
<tr>
<td>Negative</td>
<td>0.020</td>
</tr>
</tbody>
</table>

Table 0.2: Normalized confusion matrix for safety check performance with RRT cost. Positive examples correspond to safe ones and negative examples to unsafe ones accordingly. **Accuracy:** 97.9, **Precision:** 98.7, **Recall:** 97.9.

and providing an overall accurate DD-MPC approach. Note that the True Negative rate corresponds to the unsafe trajectories our approach will preemptively terminate without letting the system diverge.

The reason for this improvement can be explained by observing the differences in the latent mappings of the equivalent trajectories. False Positives occur because the definition of safety we use is not overly strict or when the latent mapping of the states is not enough to characterize them as unsafe, i.e. the trials seem to be converging and are inside a safe set but their terminal state is not at the swing-up angle. The first case can be observed in Fig. 9 for the DD-MPC with only the safety cost term. The trial starts away from any safe set but crosses two of them along the way. Even though it did not terminate inside one of the sets, it kept on approaching set 3 within the $N_{max}$ allowed which is accepted as a safe execution based on our definition of safety. When the RRT cost is incorporated Fig. 10, the system starts closer to a safe set, enters it for while but then exits again and does not re-enter one within the allotted time, making the trial a True Negative.

For the second case, we can see from Fig. 11 that the trial starts and ends inside safe set 0. The system trajectories are wrongly mapped inside safe set 0 and the controller tries to regulate the pendulum at $\theta = -2\text{rad}$. Since the system never leaves the safe set, this trial is considered safe, making it a False Positive. In this case, incorporating the RRT in Fig. 12 cost helps the system escape the initial safe set and leads it towards set 4, which contains most of the converging terminal states, by traversing the remainder of the safe
**Figure 9:** False positive due to oscillation between sets.

**Figure 10:** True Negative as the RRTC cost does not manage to lead it closer to the safe set.
4. RESULTS

**Figure 11:** False Positive because the latent encoding maps it to a safe set based on its settling trend and not the actual value of the angle.

**Figure 12:** True Positive as the RRT cost leads it to the correct safe set and it converges.
Overall Performance

In this experiment we evaluate the overall performance of the proposed method. For the comparisons we consider the following baselines:

- **RRTC** with perfect dynamics, serves as an upper bound of achievable performance
- **Local**, a local multistep controller that serves as the lower bound of achievable performance
- **MPC_RRTC**, a DD-MPC using a model trained on samples from an RRT based controller
- **MPC_FRAND**, a DD-MPC using a model trained on fully random samples of the state/input space
- **MPC_RRTC+rand_ij**, ablations of the proposed method

RRTC is the RRT based controller described in Sec. 3 and it is using the real dynamics of the broken pendulum Eq. (4) where $\dot{b}$ is the state-dependent $\dot{b}$ of Eq. (5). The dynamics models for MPC_RRTC, MPC_FRAND and MPC_RRTC+rand_ij are the models compared in Sec. 4. Finally, for MPC_RRTC+rand_ij we use the following naming scheme: $i = 1$ denotes that safety is employed for this controller in the form of the safety cost and the safety layer while $i = 0$ denotes there are no safety considerations. Accordingly, $j = 1$ means that the RRT cost is used for the controller and $j = 0$ that it is not.

Table 0.3 contains the convergence rate for the baselines, averaged over 1000 trials. It should be noted that because of the nature of this task, it is not always possible to converge to the upright position, as seen in Fig. 5,6 and discussed in Sec 3. As expected, RRTC has the highest rate and Local the lowest. For the next 2 baselines, the
4. RESULTS

The prediction accuracy of their models in Sec. 4 seems to carry over to their DD-MPC convergence rate with MPC_FRAND having a higher rate than MPC_RRTC. Interestingly, MPC_RRTC+rand_00 has a better rate than the controller based on the fully random model, even though the model RRTC+rand had a lower accuracy. Furthermore, we observe that incorporating the safety cost alone in the controller (MPC_RRTC+rand_10) leads to slightly worse convergence rate than omitting it altogether. This is a result of the False Positives discussed in the previous section and shown in Fig. 9, 11, where the wrong safety estimation results in the controller regulating around the wrong \( \theta \) or oscillating between sets. Finally, incorporating the RRT cost in MPC_RRTC+rand_01 and MPC_RRTC+rand_11 leads to almost the same convergence rate. The addition of the safety cost term in the proposed controller MPC_RRTC+rand_11 does not visibly improve the performance however, it is important to note that the proposed controller preemptively aborts predicted unsafe trials with high accuracy, as seen in Table 0.2, which are normally between 30-40% of the trials. In conclusion, the proposed combination of the safety layer and the foresight the RRT cost provides results in a controller that can complete the task with a success rate that approaches the upper bound while also effectively terminating trials that are unsafe and would not succeed.

Discussion

In this work, we proposed a framework to encourage safety in DD-MPC approaches on systems with complex dynamics. Our method relies on rapidly exploring random trees (RRTs) to collect informative data through safe interactions with the environment and learn a dynamics model for the DD-MPC. The data collected this way are also used to approximate safe sets by encoding them in a latent space. During executions, information from the RRT and the latent space is used to increase the foresight of the controlled and to lead the system to the its goal as safely as possible.

Data collection  An important consideration for applying our method to different systems is how data collection/dynamics exploration will be carried out safely. For simulated tasks with low computational cost, like the broken pendulum, it is fairly easy to do an a posteriori evaluation of whether the trajectories were safe or not. This is not however true when simulating more complicated tasks or working with real systems. The simplest solution to this is to employ bounded randomness during data-collection. Namely, ensure the safety of the system by bounding the inputs/states to acceptable limits e.g. push the object in a random direction but the acceleration/contact force should never be higher than a threshold. A more conservative solution, is to start with a non-informative but safe controller and iteratively built the backward reachable tree and safe sets or, if the application is safety-critical, to underestimate the safe sets and slowly extend them with exploration towards their boundaries.

Latent safe sets  In our experiments, we presented two cases of False Positive safety checks when we included only the safety term in the cost function. In the first one, the
safety definition we use was too lenient and allowed an oscillation between safe sets to be considered as "approaching safety". If necessary, this can be minimized through a terminal constraint. The other example of False Positive was due to a sub-optimal latent encoding of the states. One way to improve this, without using the RRT cost, would be to include the temporal aspect in the encoding and use a contrastive loss \[^1\] in order to avoid getting "trapped" based on the settling trend. Lastly, we should consider how the latent space safe sets scale with dimensions. In our examples, we have used a 2-dimensional latent space but not all applications can be sufficiently encoded in just 2 dimensions. The encoding itself will not prove to be problematic for higher dimensions, however the inclusion check we are currently using will not scale and will need to be implemented differently.

5 CONCLUSIONS

This paper addressed three challenges in DD-MPC and aimed to increase its overall safety. First, we presented a sampling-based search for safe data-collection that increases the informativeness of the data by exploring the dynamics in a larger part of the state-input space. Our results showed that the datasets collected this way led to controllers with better success rates than baselines that utilized controlled trajectories or even completely random actions to collect data for model learning. Furthermore, we presented an approach to approximate safe sets through safe trajectories only, eliminating the need for unsafe samples. During test time, the distance of the predicted trajectories to the safe sets was used as a metric of safety and was incorporated into the cost function of the controller. Lastly, we addressed the often myopic predictions of a DD-MPC that are a side-effect of its receding-horizon nature. By constructing an RRT that encodes long-term information about the system, we can enhance the cost function of the controller and improve its foresight. In our experiments we showed that the combination of the two cost terms resulted in a controller with excellent safety prediction accuracy. The article concluded with a discussion of some critical points of our implementation, as well as suggestions for how this method can be adapted to other tasks.
Bibliography


[9] Martin Ester, Hans-Peter Kriegel, Sander Jörg, Evangelos Xu, Xiaowei Simoudis, Jiawei Han, and Usama M Fayyad. A density-based algorithm for discovering clusters in large spatial databases with noise. In Proceedings of the Second International
Conference on Knowledge Discovery and Data Mining (KDD-96), page 226–231, 1996.


