MITSUBISHI ELECTRIC RESEARCH LABORATORIES
https://www.merl.com

Learning Disagreement Regions with Deep Neural Networks to Reduce Practical Complexity of Mixed-Integer MPC

Chakrabarty, Ankush; Quirynen, Rien; Romeres, Diego; Di Cairano, Stefano

TR2021-126 October 21, 2021

Abstract

Efficiently computing solutions to mixed-integer optimization-based control problems, such as in model predictive control (MPC) of hybrid systems, is extremely challenging due to the exponential worst-case complexity. The practical time-complexity of computing good control actions can be reduced by using a combination of two solvers: a strong solver that generates optimal or near-optimal closed-loop solutions with a large number of iterations, and a weak solver that converges quickly to suboptimal closed-loop solutions. In this paper, we propose the use of deep neural networks to learn sub-regions of the admissible state-space where replacing the strong solver with the weak solver maintains constraint satisfaction properties and does not result in a significant deterioration of performance. We illustrate the practical time-complexity reduction of the proposed solver selection mechanism on a station-keeping problem for a satellite.

IEEE International Conference on Systems, Man, and Cybernetics 2021

Mitsubishi Electric Research Laboratories, Inc.
201 Broadway, Cambridge, Massachusetts 02139

Learning Disagreement Regions with Deep Neural Networks to Reduce Practical Complexity of Mixed-Integer MPC

Ankush Chakrabarty[‡], Rien Quirynen, Diego Romeres, Stefano Di Cairano[*]

***Abstract*—Efficiently computing solutions to mixed-integer optimization-based control problems, such as in model predictive control (MPC) of hybrid systems, is extremely challenging due to the exponential worst-case complexity. The practical time-complexity of computing good control actions can be reduced by using a combination of two solvers: a *strong* solver that generates optimal or near-optimal *closed-loop* solutions with a large number of iterations, and a *weak* solver that converges quickly to suboptimal closed-loop solutions. In this paper, we propose the use of deep neural networks to learn sub-regions of the admissible state-space where replacing the strong solver with the weak solver maintains constraint satisfaction properties and does not result in a significant deterioration of performance. We illustrate the practical time-complexity reduction of the proposed solver selection mechanism on a station-keeping problem for a satellite.**

***Index Terms*—Learning-enabled model predictive control, non-convex optimization, integer optimization, solver selection, deep neural networks.**

I. Introduction

Model predictive control (MPC) allows handling of performance optimization and constraints by solving a constrained optimal control problem at each time step [1]. This framework can be extended to hybrid dynamical systems [2] that include both continuous and discrete decision variables, providing a powerful model-based control design for a large class of problems, for example: switched dynamical systems [3], discrete or quantized actuation [4], logic rules and temporal logic specifications [5].

For a linear-quadratic objective, (piecewise) linear dynamics and linear constraints, the optimization problem can be formulated as a mixed-integer quadratic program (MIQP). Mixed-integer MPC for hybrid dynamical systems needs to solve this MIQP at every sampling time instant. Exact optimization algorithms to find globally optimal solutions for MIQPs are typically based on a variant of the branch-and-bound (B&B) method [6]. Even though pre-solve [7] and branching [8] techniques have been developed to improve B&B algorithms, resulting in powerful software tools, e.g., `GUROBI` [9], mixed-integer programming problems are NP-hard in general.

Given the combinatorial complexity of MIQPs, fast heuristic approaches could be used to find feasible but typically suboptimal solutions instead. Examples include rounding schemes, the feasibility pump, approximate optimization algorithms, or the use of machine learning [10], [11], [12].

[‡]Corresponding author. Email: `chakrabarty@merl.com`

[*]All authors are affiliated with Mitsubishi Electric Research Laboratories, Cambridge, MA, USA.

In this paper, we propose a framework where a fast heuristic technique is used when possible, while relying on an exact optimization algorithm when necessary. In what follows, we refer to the exact optimization solver as a *strong*, usually expensive, solver and the heuristics-based solver as a *weak*, typically cheap, solver. We posit that one can reduce the dependency on the expensive solver by identifying *disagreement regions*: sub-regions in the search space where the weak and strong solvers disagree, according to some predefined logic or metric. If such a disagreement region is identified, then this is where the strong solver is required: for the rest of the feasible region, the weak solver can replace the strong solver, since the two solvers are in agreement there.

In general, computing such a disagreement region analytically could be difficult as the geometries of such regions are rarely well-behaved [13]. Therefore, we resort to supervised learning approaches that have performed well in identifying complex sets in control applications such as reachable and invariant sets [13], [14], [15] or even control policies [16] for nonlinear systems, without requiring complete knowledge of the underlying dynamical system. This is a major advantage, since the weak and strong solvers could be implemented in a high-fidelity simulation environment where one cannot access the internal mathematical representations. We propose the use of efficient supervised learning algorithms that are capable of inducing functions whose level sets can approximate the complex geometries of disagreement regions. When the true disagreement region is not large our practical complexity can potentially be reduced from the complexity of the strong solver to the complexity of the weak solver, with the added inferential complexity of the learner which has to identify which solver has to be queried. This motivates our use of deep neural networks, since they are cheap to evaluate at inference time, and they are capable of learning complex sub-region boundaries [17], [18].

The rest of the paper is organized as follows. Section II describes the MIQP-MPC problem and the weak and strong solvers for this problem. The disagreement region and the deep learning pipeline is described in more detail in Section III, and some preliminary theoretical results are provided in Section IV. We demonstrate the potential of the proposed approach on a station-keeping problem in Section V and provide our conclusions in Section VI.

II. Background and Motivation

In this section, we introduce the MIQP problem formulation and describe our proposed framework for reducing

practical complexity of MI-MPC via disagreement regions.

A. MIQP Formulation for Mixed-integer MPC

We consider a class of dynamical systems represented by

$$x_{t+1} = Ax_t + Bu_t + a, \tag{1a}$$
$$y_t = Cx_t + Du_t, \tag{1b}$$

where t is the time index, $x_t \in \mathbb{X} \subset \mathbb{R}^{n_x}$ is the state variable, $u_{t,j} \in \mathbb{Z}, j \in \mathcal{I}_t$ is an integer control action, $u_{t,j} \in \mathbb{R}, j \notin \mathcal{I}_t$ is a continuous control action, $\mathcal{I}_t$ denotes a set of indices, $a \in \mathbb{R}^{n_x}$ is an affine known term, and $y_t \in \mathbb{R}^{n_y}$ is the output of the system. The matrices A, B, C, and D have appropriate dimensions. We solve a finite-horizon MI-MPC problem from an initial condition $\hat{x}_0 \in \mathbb{X}$ of the form

$$\min_{X,\,U} \quad \sum_{k=0}^{N-1} \frac{1}{2}\begin{bmatrix} x_k \\ u_k \end{bmatrix}^\top H \begin{bmatrix} x_k \\ u_k \end{bmatrix} + \begin{bmatrix} q \\ r \end{bmatrix}^\top \begin{bmatrix} x_k \\ u_k \end{bmatrix} \tag{2a}$$
$$+ \frac{1}{2} x_N^\top P x_N + p^\top x_N \tag{2b}$$
$$\text{s.t.} \quad x_0 = \hat{x}_0, \qquad \hat{x}_0 \in \mathbb{X}, \tag{2c}$$
$$x_{k+1} = Ax_k + Bu_k + a, \qquad k \in \mathbb{Z}_0^{N-1}, \tag{2d}$$
$$\underline{y}_k \le Cx_k + Du_k \le \bar{y}_k, \qquad k \in \mathbb{Z}_0^{N-1}, \tag{2e}$$
$$u_{k,j} \in \mathbb{Z}, \quad j \in \mathcal{I}_k, \qquad k \in \mathbb{Z}_0^{N-1}, \tag{2f}$$
$$\underline{y}_N \le Ex_N \le \bar{y}_N, \tag{2g}$$

where the notation $\mathbb{Z}_0^N$ denotes the range of integers $0, 1, \ldots, N$, and the optimization variables are the state $X = [x_0^\top, \ldots, x_N^\top]^\top$ and control $U = [u_0^\top, \ldots, u_{N-1}^\top]^\top$ sequences. For simplicity of notation, the set $\mathcal{I}_k$ denotes the indices of the discrete control variables for each step $k \in \mathbb{Z}_0^N$ of the prediction horizon. The Hessian matrices in (2a) and (2b) are assumed to be positive semi-definite, i.e., $H \succeq 0$ and $P \succeq 0$. The vector a is an affine term in the model, and the vectors q, r, p define the linear component of the cost. Note that the inequality constraints in (2e) typically include simple bounds on some of the optimization variables; for instance, in order to define binary optimization variables. Also, the admissible state space $\mathbb{X}$ could be defined within (2e).

B. Strong versus Weak Solvers for MIQPs

Our strong solver is based on an exact B&B optimization algorithm, for example, as implemented in `GUROBI` [9]. The use of advanced, pre-solve, branching and cutting plane techniques [19] allows the strong solver to be computationally efficient for many practical MIQPs. However, the worst-case computational complexity scales exponentially with the number of integer optimization variables in (2f).

Different heuristic techniques can be used to compute solutions of (2) that are (approximately) feasible and suboptimal, which could be used as a *weak* solver. For example, in the numerical results of Section V, we rely on the approximate solution of (2) in which part of the integer variables are relaxed, that is, they are allowed to have real values. More specifically, we can replace the integer-valued constraints in (2f) to generate the relaxed problem

$$\min_{X,\,U} \quad \sum_{k=0}^{N-1} \frac{1}{2}\begin{bmatrix} x_k \\ u_k \end{bmatrix}^\top H \begin{bmatrix} x_k \\ u_k \end{bmatrix} + \begin{bmatrix} q \\ r \end{bmatrix}^\top \begin{bmatrix} x_k \\ u_k \end{bmatrix} \tag{3a}$$
$$+ \frac{1}{2} x_N^\top P x_N + p^\top x_N \tag{3b}$$
$$\text{s.t.} \quad x_0 = \hat{x}_0, \qquad \hat{x}_0 \in \mathbb{X}, \tag{3c}$$
$$x_{k+1} = Ax_k + Bu_k + a, \qquad k \in \mathbb{Z}_0^{N-1}, \tag{3d}$$
$$\underline{y}_k \le Cx_k + Du_k \le \bar{y}_k, \qquad k \in \mathbb{Z}_0^{N-1}, \tag{3e}$$
$$u_{k,j} \in \mathbb{Z}, \quad j \in \mathcal{I}_k, \qquad k \in \mathbb{Z}_0^{M-1}, \tag{3f}$$
$$\underline{y}_N \le Ex_N \le \bar{y}_N, \tag{3g}$$

where $M \le N$ and practically $M \ll N$ such that the computational cost for solving the resulting relaxed MIQP in (3) is considerably lower than for solving (2). Note that at least the first control input values u_0 satisfy the integer constraints when $M \ge 1$ in the weak solver. Clearly, any feasible solution to the MIQP in (2) satisfies the relaxed integer-valued constraints in (3), although the optimal objective value of the MIQP in (2) will be greater than or equal to the optimal objective value for the relaxed problem. Due to state feedback, we expect that using the weak solver in closed-loop can result in acceptable performance for some initial values $\hat{x}_0$ in (3c) within $\mathbb{X}$.

C. Disagreement region for closed-loop MI-MPC

In order to define the disagreement region, we define two MPC control policies. The first MPC policy involves iteratively solving (2) using the strong solver and applying the first control input of the optimal solution at every time step k. For a simulation horizon of T time steps, this generates a trajectory of states and inputs $\xi^{\mathrm{S}}(\hat{x}_0, T) := \{x_{k+1}^{\mathrm{S}}, u_k^{\mathrm{S}}\}_{k=0}^{T}$ of the closed-loop system, initialized at $x_0^{\mathrm{S}} = \hat{x}_0$. Another MPC policy is implemented similar to the first, but involves iteratively solving the relaxed MI-MPC problem (3) based on the weak solver, which yields the trajectory $\xi^{\mathrm{W}}(\hat{x}_0, T) := \{x_{k+1}^{\mathrm{W}}, u_k^{\mathrm{W}}\}_{k=0}^{T}$, initialized at the same initial state $x_0^{\mathrm{W}} = \hat{x}_0$.

We refer to a closed-loop trajectory ξ^{S} (or ξ^{W}) as *infeasible* if there exists any $k \in \mathbb{Z}_0^T$ such that any corresponding constraint in (2) (or (3), respectively) is violated at that time step k. Conversely, a trajectory is feasible if it satisfies constraints for every $k \in \mathbb{Z}_0^T$. With this definition in mind, we select a simulation horizon $T \gg N$, and use the corresponding closed-loop state and input trajectories to partition the admissible state-space $\mathbb{X}$ into three regions.

To begin with, we define an infeasible region

$$\bar{\mathbb{X}}(T) := \{\hat{x}_0 \in \mathbb{X} : \ \xi^{\mathrm{S}}(\hat{x}_0, T) \text{ infeasible}\}, \tag{4a}$$

which comprise initial conditions from which closed-loop trajectories generated by the strong solver are infeasible. We also define a *disagreement region*

$$\mathbb{X}_D(T) := \{\hat{x}_0 \in \mathbb{X} : \xi^{\mathrm{S}}(\hat{x}_0, T) \text{ feasible}, \ \xi^{\mathrm{W}}(\hat{x}_0, T) \text{ infeasible}\}, \tag{4b}$$

that contains initial states from which only the strong solver can generate feasible closed-loop trajectories, but the weak cannot. What remains is the *agreement region*, given by

$$\mathbb{X}_A(T) := \{\hat{x}_0 \in \mathbb{X} : \xi^{\mathrm{S}}(\hat{x}_0, T) \text{ and } \xi^{\mathrm{W}}(\hat{x}_0, T) \text{ are both feasible}\}, \quad (4c)$$

where both solvers generate feasible closed-loop trajectories. By construction, $\mathbb{X} = \bar{\mathbb{X}}(T) \cup \mathbb{X}_D(T) \cup \mathbb{X}_A(T)$, and all sets are mutually disjoint.

Remark 1. We have defined $\bar{\mathbb{X}}(T)$ as the region where the closed-loop trajectory obtained using the strong solver $\xi^{\mathrm{S}}(\hat{x}_0, T)$ is infeasible. There may be some $\hat{x}_0 \in \bar{\mathbb{X}}(T)$ such that the weak solver trajectory $\xi^{\mathrm{W}}(\hat{x}_0, T)$ is feasible. Our focus is how to obtain the constraint satisfaction of the strong solver (2) by using the weak solver (3) when possible, assuming (2) is properly designed, and not on determining which solver between (2) or (3) is to be used. Hence, we still call $\hat{x}_0 \in \bar{\mathbb{X}}(T)$ as infeasible based on the infeasibility of the strong solver, even when $\xi^{\mathrm{W}}(\hat{x}_0, T)$ is feasible.

In the next section, we propose a supervised learning framework that learns the infeasible, disagreement and agreement regions in order to effectively decide when to use the strong versus the weak solver at any initial condition in the admissible state-space $\mathbb{X}$. Switching from the strong to the weak solver on-the-fly when possible is expected to significantly curtail the practical complexity of implementing the MI-MPC online.

III. MI-MPC with Disagreement Learning

A. Offline: Training the disagreement network

For training the disagreement network, we compute labels for state samples extracted from the admissible state-space; concretely, we extract N_s samples $\{\hat{x}_0^i\}_{i=0}^{N_s} \in \mathbb{X}$. With each sample used as an initial condition, the *closed-loop* system is simulated forward for a simulation horizon of T time steps, and the problem (2) is iteratively solved with the strong solver at each time step to obtain $\xi^{\mathrm{S}}(\hat{x}_0^i, T)$. If the closed-loop trajectory obtained is infeasible, then the sample $\hat{x}_0^i$ is labeled as infeasible. If the sample is feasible, a second forward simulation is run from the same initial condition $\hat{x}_0^i$ with the weak solver, which yields $\xi^{\mathrm{W}}(\hat{x}_0^i, T)$. If both trajectories are feasible, $\hat{x}_0^i$ is labeled as agreement, otherwise as disagreement. To summarize, the label for the sample $\hat{x}_0^i$ is given by

$$\ell(\hat{x}_0^i) = \begin{cases} \text{agreement} & \text{if } \hat{x}_0^i \in \mathbb{X}_A(T) \\ \text{disagreement} & \text{if } \hat{x}_0^i \in \mathbb{X}_D(T) \\ \text{infeasible} & \text{otherwise.} \end{cases} \quad (5)$$

The samples $\{\hat{x}_0^i\}_{i=0}^{N_s}$ can be sampled uniformly over the admissible state-space $\mathbb{X}$, although prior work has demonstrated the effectiveness of low-discrepancy sampling and active sampling to reduce sample complexity [13]. Since labeling is done offline, there is no strict restriction on real-time feasibility or computational expenditure.

Once labeling is complete, we construct a classifier that classifies any $\hat{x}_0 \in \mathbb{X}$ into one of the three categories described in (5). While there are several techniques popularized by the machine learning community to solve multi-class problems, we employ deep neural networks (DNN) owing to their flexibility in inducing complex decision boundaries and their ability to handle a large number of samples [17]. This quality is essential to implement this method for systems with high-dimensional state-spaces. We refer to this disagreement learning network as DisNet.

Since it is impractical to assume that DisNet can provide perfect accuracy when trained with finite samples, we try to promote misclassifications in directions that are preferable from a safety viewpoint. In particular, we use a cost-sensitive categorical cross entropy function as the training loss. Concretely, classifying a sample in the disagreement region incorrectly with the agreement label is penalized more heavily than a sample in the agreement region classified as disagreement. This is because an initial condition that is truly within $\mathbb{X}_A(T)$ misclassified to be within $\mathbb{X}_D(T)$ will result in the strong solver being used to compute trajectories, which will incur computational expenditure, but result in desired closed-loop behaviour. Conversely, if an initial condition is truly within the disagreement region but DisNet assigns it to the agreement region, the weak solver will not induce feasible closed-loop trajectories and will result in constraint violations. Similarly, classifying infeasible samples as disagreement is more harmful than classifying samples within the disagreement region as infeasible. This is because if an initial condition truly is infeasible, using the strong solver (because it is misclassified to be within $\mathbb{X}_D(T)$) will not result in useful trajectories, whereas labeling a disagreement region sample as infeasible will trigger a fail-safe mechanism that is typically in place for most practical engineering systems. Any misclassification can result in loss of optimality of operation, but DisNet errs on the side of caution and promotes safety via asymmetry in the training loss.

B. Online: Solver selection with DisNet-in-the-loop

Let the initial state during on-line operation be denoted x_0, and the state at the t-th time step be denoted x_t. At each time step t, we query DisNet to determine whether x_t is infeasible or belongs to $\mathbb{X}_D(T)$ or $\mathbb{X}_A(T)$. Accordingly, the strong or weak solver is called to compute a sequence of control actions for the prediction horizon length N. Following the philosophy of MPC, we only implement the first control action in the sequence obtained. Thus, the control law has the form

$$u^\star(x_t) = \begin{cases} u_0^W(x_t) & \text{if } \ell(x_t) = \text{agreement}, \\ u_0^S(x_t) & \text{if } \ell(x_t) = \text{disagreement}, \\ u_0^S(x_t) & \text{if } \ell(x_t) = \text{infeasible}, \end{cases} \quad (6)$$

where $\ell(x_t)$ is output obtained by evaluating DisNet for the state x_t, and u_0^W (u_0^S) is the first control action computed by the weak (respectively, strong) solver with the initial state x_t and a prediction horizon of length N. Pseudocode for

implementing the proposed MI-MPC with DISNET in the loop is provided in Algorithm 1.

Algorithm 1 MI-MPC with Disagreement Learning

Require: Initial condition, x_0
Require: Trained disagreement network, DisNet
Require: Exact (strong) solver, u^S
Require: Approximate (weak) solver, u^W
for $t = 0 : \infty$ **do**
  $\ell_t \leftarrow \mathsf{DisNet}(x_t)$ ▷ evaluate DNN (fast)
  if ℓ_t is `agreement` **then**
    $U_t \leftarrow u^W(x_t)$ ▷ use weak solver (fast)
  else if ℓ_t is `disagreement` **then**
    $U_t \leftarrow u^S(x_t)$ ▷ use exact solver
  else
    if $u^S(x_t)$ has feasible solution **then**
      $U_t \leftarrow u^S(x_t)$ ▷ use exact solver
    else
      $U_t \leftarrow *$ ▷ handle infeasibility
    end if
  end if
  $u_t \leftarrow U_t[:, 0]$ ▷ apply first control action
  $x_{t+1} \leftarrow Ax_t + Bu_t + a$ ▷ collect updated state
end for

The use of the strong solver when a state is classified as infeasible is a safety mechanism in case there is a misclassification, and the state is actually in the disagreement region; in such a case, the strong solver can actually compute a feasible solution. If the strong solver cannot compute a feasible solution (that is, the state is indeed in the infeasible region), then a fail-safe mechanism specific to the application is deployed to avoid catastrophic failure. Since it is typical for state-of-the-art MIQP solvers to attain feasibility information more quickly compared to the optimal solution, this approach is practical as the system will not remain in an unsafe mode of operation for very long when the state is infeasible.

The next result is immediate for characterizing the probability of Algorithm 1 yielding a feasible future closed-loop trajectory, at each time step.

Proposition 1. *Consider a generic time t such that $x_t \notin \bar{\mathbb{X}}(T)$. Then, by applying Algorithm 1, the probability π_f that $x_{t+1} \notin \bar{\mathbb{X}}(T)$, that is, that there exists a feasible trajectory of length T from $t+1$, is at least*

$$1 - \mathbb{P}\left[\ell(x_t) = \textit{agreement} | x_t \in \mathbb{X}_D(T)\right] \mathbb{P}\left[x_t \in \mathbb{X}_D(T)\right].$$

Proposition 1 follows from the fact that the strong solver is used unless $\ell(x_t) = \mathsf{agreement}$. Hence, the weak solver is used incorrectly only if $\ell(x_t) = \mathsf{agreement}$ but actually $x_t \in \tilde{\mathbb{X}}_D(T)$. The purpose of considering two different solvers and a classifier to solve the MI-MPC problem, is to reduce the computational effort while maintaining a feasible trajectory. Indeed, even if the classifier requires additional computational time online, we design a learner such that the time for obtaining a classification plus the time to find an optimal solution with the weak solver is considerably lower than the time required by standard methods that consider only a strong solver. The smaller the volume of the disagreement region (which is problem-dependent), the more the potential speedup of on-line computations.

Remark 2. In this paper, we selected our disagreement metric to be based on feasible regions induced by the two solvers. However, this approach can be generalized to other criteria for disagreement: for instance, one could select a disagreement region based on whether the first control action of a strong and weak solver are equal or not[1].

IV. PRELIMINARY STUDY OF THEORETICAL GUARANTEES

While perfectly learning $\mathbb{X}_D$ and $\mathbb{X}_A$ will rarely happen in practice, assuming perfect learning allows us to provide some insights into the closed-loop behavior. In particular, this section explains how the control law (6) can result in recursively feasible trajectories, and asymptotic stability.

For brevity, let us rewrite the constraints (2e), (2f) respectively by

$$g_c(x_t, u_t) \leq 0, \tag{7a}$$
$$u_{t,j} \in \mathbb{Z}, \ \forall j \in \mathcal{I}_t. \tag{7b}$$

We start with conditions for achieving recursive feasibility.

Theorem 1. *Let $T \to \infty$, and $\mathbb{X}_D(\infty)$ and $\mathbb{X}_A(\infty)$ be perfectly learned. If at time t, $x_t \in \mathbb{X}_A(\infty) \cup \mathbb{X}_D(\infty)$, then the closed-loop system* (6)–(1) *satisfies* (7) *for all $t_1 \geq t$.*

Proof: We prove the statement by induction, starting from $x_t \in \mathbb{X}_A(\infty) \cup \mathbb{X}_D(\infty)$. By the definition of $\mathbb{X}_D(\infty)$, $x_t \in \mathbb{X}_D(\infty)$ only if there exists a closed-loop trajectory $\xi^S(x_t, \infty)$ that satisfies (7) for all $t_1 \geq t$. Thus, if $x_t \in \mathbb{X}_D(\infty)$ then the control law $u_0^S(x_t)$ ensures that (7) is satisfied and $x_{t+1} \notin \bar{\mathbb{X}}(\infty)$, i.e., $x_{t+1} \in \mathbb{X}_A(\infty) \cup \mathbb{X}_D(\infty)$. By the definition of $\mathbb{X}_A(\infty)$, $x_t \in \mathbb{X}_A(\infty)$ only if there exists a closed-loop trajectory $\xi^W(x_t, \infty)$ that satisfies (7) for all $t_1 \geq t$. When $T \to \infty$, this means that $\mathbb{X}_A(\infty)$ is forward invariant under the control law $u_0^W(x_t)$. Thus, if $x_t \in \mathbb{X}_A(\infty)$, then due to such invariance and the control law (6), (7) is satisfied and $x_{t+1} \in \mathbb{X}_A(\infty)$. Thus, if $x_t \in \mathbb{X}_A(\infty) \cup \mathbb{X}_D(\infty)$, (7) is satisfied and $x_{t+1} \in \mathbb{X}_A(\infty) \cup \mathbb{X}_D(\infty)$, from which the same argument can be repeated. ■

In practice, we can construct $\mathbb{X}_A(\infty)$ and $X_A(\infty) \cup \mathbb{X}_D(\infty)$ by considering trajectories of finite duration $\bar{T}$. Since for any T, $\mathbb{X}_A(T+1) \subseteq \mathbb{X}_A(T)$, the sequence $\{\mathbb{X}_A(T)\}_T$ is non-increasing and bounded, and hence it converges under fairly standard assumptions [20]. The following result is stated for $\mathbb{X}_A(\infty)$ yet it is immediate to apply also for $X_A(\infty) \cup \mathbb{X}_D(\infty)$.

Proposition 2. *Let $\bar{T} \in \mathbb{Z}_0^\infty$ be such that $\mathbb{X}_A(\bar{T}+1) = \mathbb{X}_A(\bar{T})$, where we assume again exact reconstruction from available data. Then, $\mathbb{X}_A(\infty) = \mathbb{X}_A(\bar{T})$.*

[1]Empirical results showed that the agreement regions using this criterion had relatively small volumes.

Proof: First, $\mathbb{X}_A(\bar{T}) \supseteq \mathbb{X}_A(\infty)$ since if $x \in \mathbb{X}_A(\infty)$, the closed-loop trajectory is feasible for all future steps, which means that it is feasible for any finite number $\bar{T}$ of steps, i.e., $x \in \mathbb{X}_A(\bar{T})$.

Next, we show that, under the stated assumptions, if $x \in \mathbb{X}_A(\bar{T})$ then $x \in \mathbb{X}_A(\infty)$. Since $\mathbb{X}_A(\bar{T}) = \mathbb{X}_A(\bar{T}+1)$, there exists a feasible trajectory $\xi^{\mathrm{W}}(x, \bar{T}+1) = \{x^{\mathrm{W}}_{k+1}, u^{\mathrm{W}}_k\}_{k=0}^{\bar{T}+1}$. Taking all the elements of $\xi^{\mathrm{W}}(x, \bar{T}+1)$ but the first one, we obtain the feasible trajectory of length $\bar{T}$, $\xi^{\mathrm{W}}(x^{\mathrm{W}}_1, \bar{T})$. Thus, $x^{\mathrm{W}}_1 \in \mathbb{X}_A(\bar{T})$, but since then $x^{\mathrm{W}}_1 \in \mathbb{X}_A(\bar{T}+1)$, which means that there exists $\xi^{\mathrm{W}}(x^{\mathrm{W}}_1, \bar{T}+1)$ feasible. Appending $\xi^{\mathrm{W}}(x^{\mathrm{W}}_1, \bar{T}+1)$ to the first element of $\xi^{\mathrm{W}}(x, \bar{T}+1)$, we obtain the feasible trajectory of length $\bar{T}+2$, $\xi^{\mathrm{W}}(x, \bar{T}+2)$, proving that $x \in \mathbb{X}_A(\bar{T}+2)$. Since we can repeat these steps an arbitrary number of times, $x \in \mathbb{X}_A(\infty)$. Therefore, $\mathbb{X}_A(\bar{T}) \subseteq \mathbb{X}_A(\infty)$ and from the previous inclusion, we obtain $\mathbb{X}_A(\bar{T}) = \mathbb{X}_A(\infty)$. ∎

In practice, $\mathbb{X}_A(\bar{T})$ that satisfies the assumptions of Proposition 2 is an invariant set [20], specifically, the region where MPC is recursively feasible, which, under fairly standard assumptions, is bounded [1]. The properties $\mathbb{X}_A(T+1) \subseteq \mathbb{X}_A(T)$ can also be used as a quick, necessary yet non-sufficient, test on whether the reconstruction is correct or more data is needed, and on where such additional data may be needed. Next, we consider asymptotic stability, under assumptions on the properties of $u^S_0(x_t)$, $u^W_0(x_t)$.

Theorem 2. *Let x^e be an equilibrium for* (1) *in the interior of $\mathbb{X}_A(\infty)$. Let* (1) *in closed-loop with $u^W_0(x_t)$, be asymptotically stable on x^e with domain of attraction $\mathbb{X}_A(\infty)$, and* (1) *in closed-loop with $u^S_0(x_t)$ be convergent to x^e with domain of attraction $\mathbb{X}_D(\infty) \cup \mathbb{X}_A(\infty)$. Then* (1) *in closed-loop with* (6) *is asymptotically stable on x^e with domain of attraction $\mathbb{X}_D(\infty) \cup \mathbb{X}_A(\infty)$.*

Proof: The constraints (7) are satisfied for $x \in \mathbb{X}_A(\infty)$ by (6)–(1), $\mathbb{X}_A(\infty)$ is invariant for (6)–(1), and there exists a ball with radius $\rho > 0$, centered at x^e such that $\mathbb{B}(x^e, \rho) \subseteq \mathbb{X}_A(\infty)$. Due to the control policy (6), for $x \in \mathbb{X}_A(\infty)$, the closed loop (6)–(1) is equal to the closed loop (1) and $u^W_0(x_t)$.

Since (1) in closed-loop with $u^S_0(x_t)$ is convergent on x^e, there will be a finite time $\tilde{t}$ such that $x_{\tilde{t}} \in B(x^e, \rho) \subseteq \mathbb{X}_A(\infty)$, and hence there will be a finite time $\ell(x_{\tilde{t}}) =$ agreement. Due to the invariance of $\mathbb{X}_A(\infty)$, $\ell(x_t) =$ agreement for all $t \geq \bar{t}$, and hence the closed-loop (6)–(1) is asymptotically stable on x^e with domain of attraction $\mathbb{X}_D(\infty) \cup \mathbb{X}_A(\infty)$. ∎

Remark 3. While achieving convergence of (1) in closed-loop with $u^S_0(x_t)$ is relatively simple, c.f. [2], achieving asymptotic stability of (1) in closed loop with $u^W_0(x_t)$ is difficult, since (2f) is enforced only at the first $M \leq N$ steps, which causes prediction errors for steps $M+1, \ldots, N$. However, methods based on enforcing a control Lyapunov function as constraint on the first step of the prediction horizon, e.g., see [21], [22], can be applied to this end.

V. Case Study: Satellite Station Keeping

We illustrate the proposed approach based on numerical simulation results for a case study of MI-MPC for satellite station keeping [23]. The MIQPs in (2) and (3) are solved using `Gurobi`, and `PyTorch 1.8.1` is used for constructing DisNet, on a `Python 3.8` base.

A. MI-MPC Problem Formulation

The case study is based on a real-world application, the control of a satellite in a circular low-Earth orbit 400 km from Earth's surface. We consider a re-centering maneuver in which the satellite, previously drifting due to perturbations such as 3rd-body gravity, drag, and solar pressure, is re-centered in its station keeping window around its orbit. The station keeping window defines the hard constraints on the satellite actual position relative to its target orbit:

$$-100 \leq p_{\mathrm{x}} \leq 100 \text{ and } -100 \leq p_{\mathrm{y}} \leq 100,$$

where p_{x} and p_{y} are the orbital plane relative coordinates in the Hill's frame, that is, the frame moving on the satellite target orbit. The Hill's frame $(p_{\mathrm{x}}, p_{\mathrm{y}})$-axes are the radial (along the orbit radius) and in-track (along the orbit path) axes, respectively. The satellite should be kept inside the station keeping window for its mission to be successful. The dynamics are obtained from the linearized HCW equations as [23]

$$\begin{bmatrix} \dot{p_{\mathrm{x}}} \\ \dot{p_{\mathrm{y}}} \\ \dot{v_{\mathrm{x}}} \\ \dot{v_{\mathrm{y}}} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 3n^2 & 0 & 0 & 2n \\ 0 & 0 & -2n & 0 \end{bmatrix} \begin{bmatrix} p_{\mathrm{x}} \\ p_{\mathrm{y}} \\ v_{\mathrm{x}} \\ v_{\mathrm{y}} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1/m & 0 \\ 0 & 1/m \end{bmatrix} \begin{bmatrix} F_{\mathrm{x}} \\ F_{\mathrm{y}} \end{bmatrix}, \tag{8}$$

where the state vector $x \triangleq \begin{bmatrix} p_{\mathrm{x}} & p_{\mathrm{y}} & v_{\mathrm{x}} & v_{\mathrm{y}} \end{bmatrix}^\top$ includes the relative positions and corresponding velocities, F_{x} and F_{y} denote the thrust forces along the Hill's frame axes, m is the satellite's mass and n denotes the orbital frequency. Our problem is a simplification of [4], where only the orbital plane dynamics are considered and the attitude dynamics are neglected. Therefore, we control a simpler propulsion system with two on/off gimbaled thrusters along the p_{y}-axis, one towards the positive and one towards the negative direction.

Thus, the forces are defined by four control inputs

$$F_{\mathrm{x}} = F_{\max}\left(F^+_{\mathrm{x}} - F^-_{\mathrm{x}}\right), \tag{9a}$$
$$F_{\mathrm{y}} = F_{\max}\left(F^+_{\mathrm{y}} - F^-_{\mathrm{y}}\right), \tag{9b}$$

where $F^+_{\mathrm{y}}, F^-_{\mathrm{y}} \in \{0, 1\}$ are binary variables, F^+_{x}, F^-_{x} are continuous variables and $F_{\max} = 0.4$ denotes the maximum thrust value. The gimbal ranges result in the (approximated) thrust triangle [4] constraints

$$-\gamma_1 F^i_{\mathrm{y}} \leq F^i_{\mathrm{x}} \leq \gamma_1 F^i_{\mathrm{y}}, \quad \text{for } i \in \{+, -\}, \tag{10a}$$
$$0 \leq F^+_{\mathrm{y}} + F^-_{\mathrm{y}} \leq 1, \tag{10b}$$

where $\gamma_1 = \sin(\pi/32)$ is the maximum deflection of the thruster gimbals, and (10b) imposes that at most one thruster is fired at any one time, for power considerations. In addition to the input constraints, the MI-MPC problem formulation includes the following state constraints

$$-100 \leq p_{\mathrm{x}}, p_{\mathrm{y}} \leq 100, \quad -0.2 \leq v_{\mathrm{x}}, v_{\mathrm{y}} \leq 0.2, \tag{11}$$

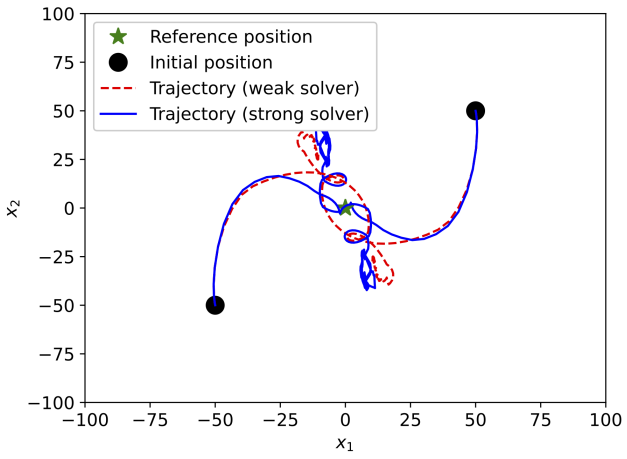

Fig. 1. Illustration of the MI-MPC state evolution for satellite station keeping around the origin, based on either the strong or weak solver (see Section II).

and the objective function reads as

$$\frac{1}{2}\sum_{k=0}^{N-1}\left(x_k^\top Q\, x_k + u_k^\top R\, u_k\right) + \frac{1}{2}x_N^\top P\, x_N, \qquad (12)$$

where the matrix P denotes the solution to the discrete-time algebraic Riccati equation.

In Figure 1, we show the closed-loop MI-MPC trajectories for the relative position of the satellite, where the origin is the desired satellite position along the orbit. The depicted area in the figure corresponds to the station keeping window, in which the satellite should be kept. The diagonal weighting matrices in (12) are

$$Q = \mathrm{diag}(0.001, 0.001, 1, 1), \quad R = \mathrm{diag}(1, 1, 1, 1), \quad (13)$$

in order to penalize the thrust forces and velocities more strongly than the position of the satellite, since this reduces fuel consumption and hence increases the satellite lifespan [23]. Thus, our design of the MI-MPC controller is focused on satisfying the constraints with minimal use of the on/off thrusters. For this particular case study, we use $M = 1$ for the MI-MPC formulation in (3), such that the weak solver, in the worst case, needs to enumerate only 3 possible assignments to the binary variables $F_{\mathrm{y}}^+, F_{\mathrm{y}}^- \in \{0, 1\}$ to satisfy the constraints in (10).

B. Disagreement Learning: Architecture and Training

As described before, we use DisNet to identify the agreement, disagreement, and infeasible regions. DisNet takes the state vector of the satellite as input, and generates three logits, one for each of $\bar{\mathbb{X}}$, $\mathbb{X}_A$ and $\mathbb{X}_D$. The architecture comprises four (hidden) fully connected layers with Leaky ReLU activation functions to prevent vanishing gradients. The fully connected layers have 4096, 1024, 256 and 64 units, respectively.

We use the Adam optimizer to train DisNet with a cost-sensitive cross entropy as the training/validation loss function. For the reasons discussed in Section III, the cost function has weight $10\times$ higher on the logit corresponding to the disagreement region. That is, we heavily penalize misclassifying states in the disagreement region. In order to avoid over-fitting and to obtain better generalization properties, the fully connected layers are equipped with dropout. The training data set consists of $75,000$ unique labeled samples drawn randomly from the admissible state-space (11), of which 20% are used for validation; for more details on labeling, see Section III-A. We notice a significant imbalance between the data points within each class: in particular, far fewer samples lie in the disagreement region (only 6k out of 75k) than in the other regions. To counteract this, we employ a weighted sampler (`torch.utils.data.WeightedRandomSampler`) in the mini-batch selection to obtain a more balanced distribution of points in each mini-batch.

Finally, it is known that several parameters need to be tuned in deep learning. In this case study, we performed a grid-search to select the optimal values of the batch size and of the learning rate in the optimizer. The grid-search consists in training the classifier for a grid of values and select the set of parameters that result in the lowest value of the validation loss function. The grid we considered is generated by the values $[16, 32, 64, 258, 1024, 4096]$ for the batch size and $[10^{-2}, 5\times10^{-3}, 10^{-3}, 10^{-4}]$ for the learning rate. The best mini-batch size and learning rate we obtain are 64 and 10^{-3}, respectively. All the simulations consisted of 2000 epochs, and we save the weights that correspond to the lowest loss value on the validation set over all epochs.

C. Closed-loop Simulation Results

Next, we describe the results obtained with the proposed method in closed-loop simulations of the satellite orbit control application. We report the performance of the disagreement learning network using the confusion matrix[2] depicted in Fig. 2. The confusion matrix is generated using an unseen test dataset comprising 100,000 labeled samples drawn from a different distribution than the training data.

Fig. 2 shows that DisNet correctly estimates the classes for 94.61% of the test data set; see element (4,4) of the confusion matrix. We note that the classification of the infeasible and the agreement regions are learned most accurately (see elements (1,4) and (2,4)) and that almost no test samples are misclassified in between these two regions (see elements (1,2) and (2,1)). This is quite intuitive because the infeasible region and the agreement region do not intersect for this example, so they are nonlinearly separable. As expected, the boundary of $\mathbb{X}_D$ is hardest to identify, and this is where most misclassification errors occur (see elements (2,3) and (3,2)). It is interesting to note that our cost-sensitive training loss results in more agreement samples being classified as disagreement than vice-versa, which is desirable from a safety perspective, since (2) admits a solution in $\mathbb{X}_A$, while (3) is infeasible in $\mathbb{X}_D$. The section for $v_{\mathrm{x}} = v_{\mathrm{y}} = 0$ of

[2]The confusion matrix is a convenient representation that shows the classification accuracy comparing the predicted classes against the actual classes in a matrix-type visualization.

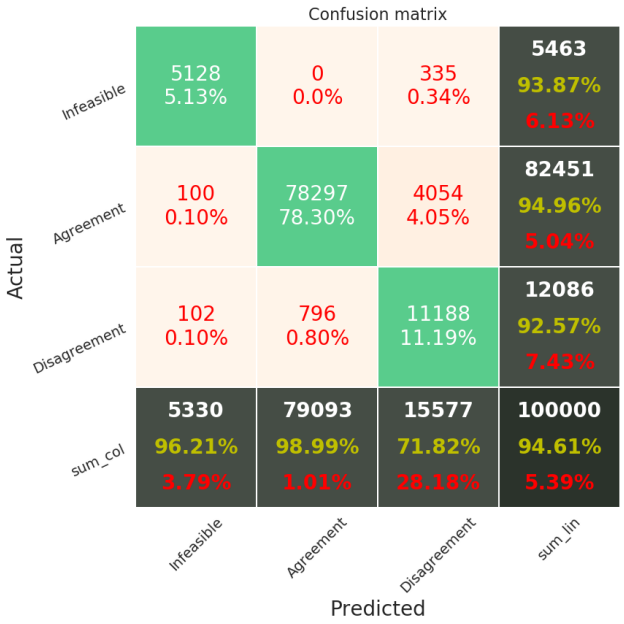

Fig. 2. Confusion matrix for DISNET on the test data set. The confusion matrix is the top left 3×3 submatrix, the 4th column and row represent the sum of the statistics per row and per column, respectively.

the regions in consideration for this case study is shown in Fig. 3, whose boundary is the station keeping window.

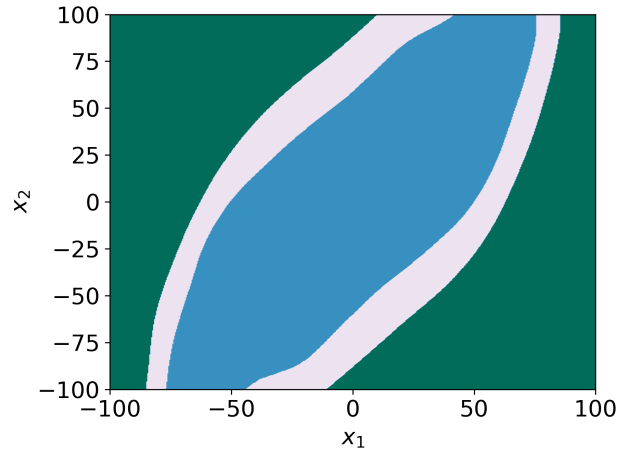

Fig. 3. Partition of the station-keeping window when the satellite relative velocity in Hill's frame is zero. The green outer region is the infeasible region, the middle is the disagreement region, and the innermost blue depicts the agreement region.

From Figure 3, we deduce that during online implementation, the number of times the strong solver is queried will be limited, especially in a neighborhood near the origin. This is corroborated in Figure 4, where we compare the computational performance of the proposed algorithm with the MI-MPC running only the strong solver. We have performed 1000 simulations starting from initial conditions taken in the test dataset, for $T = 1000$ time-steps. The median of the exact MI-MPC simulation time is around 32s over 1000 solver calls, whereas our proposed approach takes < 10s, thus saving more than 68% of the computation time, which is relevant in a spacecraft where power, including that for computers, must be saved as much as possible.

For longer simulation times, the savings increase even further in the cases where the satellite enters the agreement region and remains there. However, this is not always the case, as illustrated in Figure 5. For this initial condition (the black square), the exact and proposed MI-MPC induce different trajectories. Encouragingly, there are no constraint violations in either case. The proposed MI-MPC closed-loop trajectory switches between strong (exact) and weak (approximate) solvers. This might seem in conflict with Figure 3, but one should consider that Figure 3 shows the section for zero relative velocities, while the solver switching in Figure 5 occurs at different velocities, for which the disagreement region has different shape. In fact, Figure 5 demonstrates how complex the disagreement region can be, thus explaining why high-dimensional function approximators (such as deep neural networks) are needed to represent it.

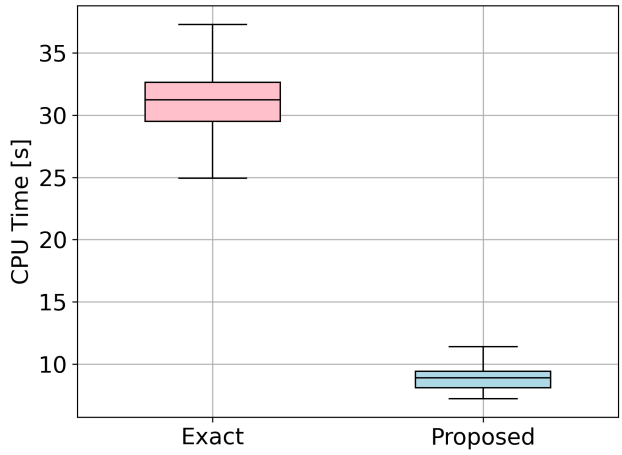

Fig. 4. Comparison of computational time incurred by an MI-MPC with strong solver vs. our proposed method.

VI. CONCLUSIONS

In this paper, we developed an algorithm for reducing the practical complexity of mixed-integer MPC by partitioning the admissible state-space into regions where an exact mixed-integer solver can be replaced by a faster sub-optimal solver without incurring constraint violations (with high probability). A deep neural network, DISNET, is trained off-line to characterize the partitioning into sub-regions and it is queried on-line to decide which solver to employ at the current time to solve a finite-horizon optimal control problem. A simulation example demonstrates the complexity reducing potential of the proposed method. Future work will consider updating the learned sub-regions on-the-fly and derivation of probabilistic guarantees on closed-loop performance with the more practical consideration of imperfect learning.

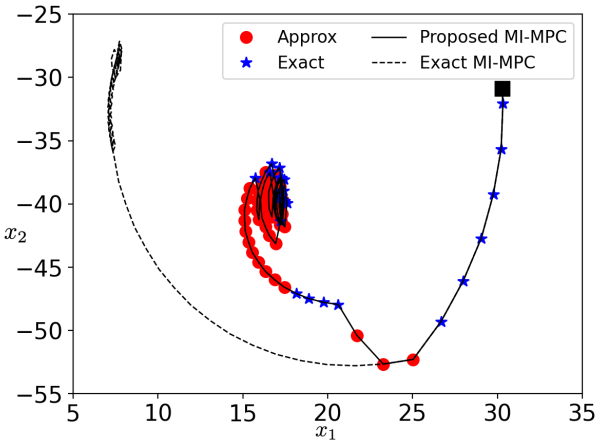

Fig. 5. Comparison of a sample closed-loop trajectory from the same initial condition generated by an exact MI-MPC and the proposed MI-MPC. The strong solver is called at the '⋆' state samples, and the weak solver is called at the '○' state samples.

REFERENCES

[1] D. Mayne and J. Rawlings, Model Predictive Control. Nob Hill, 2013.

[2] A. Bemporad and M. Morari, "Control of systems integrating logic, dynamics, and constraints," Automatica, vol. 35, pp. 407–427, 1999.

[3] T. Marcucci and R. Tedrake, "Mixed-integer formulations for optimal control of piecewise-affine systems," in Proc. of the 22nd ACM Inter. Conf. on Hybrid Systems, New York, NY, USA, 2019, p. 230–239.

[4] A. Walsh, S. Di Cairano, and A. Weiss, "MPC for coupled station keeping, attitude control, and momentum management of low-thrust geostationary satellites," in Proc. American Control Conference (ACC), 2016, pp. 7408–7413.

[5] Y. E. Sahin, R. Quirynen, and S. Di Cairano, "Autonomous vehicle decision-making and monitoring based on signal temporal logic and mixed-integer programming," in Proc. American Control Conference (ACC), 2020, pp. 454–459.

[17] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016.

[6] C. A. Floudas, Nonlinear and mixed-integer optimization: fundamentals and applications. Oxford University Press, 1995.

[7] T. Achterberg, R. E. Bixby, Z. Gu, E. Rothberg, and D. Weninger, "Presolve reductions in mixed integer programming," ZIB Report, pp. 16–44, 2016.

[8] T. Achterberg, T. Koch, and A. Martin, "Branching rules revisited," Operations Research Letters, vol. 33, no. 1, pp. 42–54, 2005.

[9] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2020. [Online]. Available: http://www.gurobi.com

[10] T. Achterberg and T. Berthold, "Improving the feasibility pump," Discrete Optimization, vol. 4, no. 1, pp. 77–86, 2007.

[11] R. Takapoui, N. Moehle, S. Boyd, and A. Bemporad, "A simple effective heuristic for embedded mixed-integer quadratic programming," International Journal of Control, vol. 93, no. 1, pp. 2–12, 2020.

[12] D. Bertsimas and B. Stellato, "Online mixed-integer optimization in milliseconds," arXiv preprint arXiv:1907.02206, 2019.

[13] A. Chakrabarty, C. Danielson, S. Di Cairano, and A. Raghunathan, "Active learning for estimating reachable sets for systems with unknown dynamics," IEEE Transactions on Cybernetics, 2020.

[14] Z. Wang and R. M. Jungers, "Scenario-based set invariance verification for black-box nonlinear systems," IEEE Control Systems Letters, vol. 5, no. 1, pp. 193–198, 2020.

[15] M. Turchetta, F. Berkenkamp, and A. Krause, "Safe exploration for interactive machine learning," in Advances in Neural Information Processing Systems, 2019, pp. 2891–2901.

[16] A. Chakrabarty, G. T. Buzzard, and S. H. Żak, "Output-tracking quantized explicit nonlinear model predictive control using multiclass support vector machines," IEEE Transactions on Industrial Electronics, vol. 64, no. 5, pp. 4130–4138, 2016.

[18] B. Hanin, "Universal function approximation by deep neural nets with bounded width and ReLu activations," Mathematics, vol. 7, no. 10, p. 992, 2019.

[19] G. L. Nemhauser and L. A. Wolsey, Integer and Combinatorial Optimization. New York, NY, USA: Wiley-Interscience, 1988.

[20] F. Blanchini and S. Miani, Set-theoretic methods in control. Springer, 2008.

[21] P. O. Scokaert, D. Q. Mayne, and J. B. Rawlings, "Suboptimal model predictive control (feasibility implies stability)," IEEE Trans. on Automatic Control, vol. 44, no. 3, pp. 648–654, 1999.

[22] M. Lazar, "Flexible control lyapunov functions," in Proc. American Control Conf., 2009, pp. 102–107.

[23] A. Weiss, U. V. Kalabić, and S. Di Cairano, "Station keeping and momentum management of low-thrust satellites using mpc," Aerospace Science and Technology, vol. 76, pp. 229–241, 2018.