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

# Fast Multi-Robot Motion Planning via Imitation Learning of Mixed-Integer Programs

Srinivasan, Mohit; Chakrabarty, Ankush; Quirynen, Rien; yoshikawa, nobuyuki; Mariyama, Toshisada; Di Cairano, Stefano

TR2021-134 November 13, 2021

**Abstract**

We propose a centralized multi-robot motion planning approach that leverages machine learning and mixed-integer programming (MIP). We train a neural network to imitate optimal MIP solutions and, during execution, the trajectories predicted by the network are used to fix most of the integer variables, resulting in a significantly reduced MIP or even a convex program. If the obtained trajectories are feasible, i.e., collision-free and reaching the goal, they can be used as-is or further refined towards optimality. Since maximizing the likelihood of feasibility is not the standard goal of imitation learning, we propose several techniques aimed at increasing such likelihood. Simulation results show the reduced computational burden associated with the proposed framework and the similarity with optimal MIP solutions.

*IFAC Modeling, Estimation and Control Conference (MECC) 2021*



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

# Fast Multi-Robot Motion Planning via Imitation Learning of Mixed-Integer Programs

**Mohit Srinivasan** [*,**] **Ankush Chakrabarty** [**] **Rien Quirynen** [**]
**Nobuyuki Yoshikawa** [***] **Toshisada Mariyama** [***]
**Stefano Di Cairano** [**]

[*] *Georgia Institute of Technology (mohit.s@ieee.org).*
[**] *Mitsubishi Electric Research Labs*
*({chakrabarty,quirynen,dicairano}@merl.com).*
[***] *Information Technology Center, Mitsubishi Electric Corporation*
*({Yoshikawa.Nobuyuki,Mariyama.Toshisada}@ab.MitsubishiElectric.co.jp).*

**Abstract:** We propose a centralized multi-robot motion planning approach that leverages machine learning and mixed-integer programming (MIP). We train a neural network to imitate optimal MIP solutions and, during execution, the trajectories predicted by the network are used to fix most of the integer variables, resulting in a significantly reduced MIP or even a convex program. If the obtained trajectories are feasible, i.e., collision-free and reaching the goal, they can be used as-are or further refined towards optimality. Since maximizing the likelihood of feasibility is not the standard goal of imitation learning, we propose several techniques aimed at increasing such likelihood. Simulation results show the reduced computational burden associated with the proposed framework and the similarity with the optimal MIP solutions.

*Keywords:* Machine Learning; Path Planning and Motion Control; Optimal Control.

## 1. INTRODUCTION

Real-time computation of centralized motion planning for multi-robot systems (Yan et al., 2013) becomes computationally challenging as the number of robots increases. An example is motion planning by mixed-integer programming (MIP) that allows one to tackle obstacle avoidance, inter-robot collision avoidance, and goal-reaching specifications, and as such has been proposed for a number of applications such as multi-vehicle routing (Schouwenaars et al., 2001), unmanned aerial vehicles trajectory planning (Albert et al., 2017), multi-robot task scheduling (Gombolay et al., 2013) However, mixed-integer linear programming (MILP) is $\mathcal{NP}$-hard, and this limits its applicability for real-time implementation.

Recently, there have been significant efforts in leveraging machine learning techniques to speed-up the solution of combinatorial problems, such as MILP; a detailed discussion and overview is available in (Bengio et al., 2020). For instance, in (Karg and Lucia, 2018), deep neural networks are used to approximate mixed-integer model predictive controllers, and supervised learning methods to warm start branch-and-bound (B&B) solvers are proposed in (Masti and Bemporad, 2019). Here, we follow the philosophy in (Bengio et al., 2020) that suggests not to solely rely on machine learning to solve combinatorial problems, but rather to learn offline predictors for certain components of the problem, and to use them to significantly reduce the online (real-time) computational burden. Specifically, here we learn a predictor, offline, for trajectories of an MILP-based centralized multi-robot motion planning, and we use the predicted trajectories online to determine most/all integer variables of the MILP. Thus, online, we solve a problem where the integer variables, which largely determine the computational burden, are significantly reduced or even completely eliminated, but the final motion plan results from an exact optimization algorithm. The motion plan could be further refined using the original MILP, for which the B&B algorithm (Floudas, 1995) may converge significantly faster because an integer feasible initial solution is provided.

Learning-based motion planners have been investigated using both reinforcement learning (RL) and imitation learning (IL). RL-based planners (Zhang et al., 2017; Faust et al., 2018) are particularly effective in uncertain environments where learning is achieved through interactions with the environment, while requiring careful handcrafting of reward functions and large amounts of data and computing resources (Johnson et al., 2020). On the other hand, IL exploits an available expert that can generate optimal plans, to "learn from demonstrations", that is, by reproducing the expert policy. Neural motion planners (NMPs) use deep neural networks to emulate an optimal planning algorithm (Qureshi and Yip, 2018) achieving near-optimal solutions with higher computational speeds, see, for example, MPNet (Qureshi et al., 2020) While based on similar rationale, our approach differs from NMPs as we use the learned policy to reduce the online computations of the expert rather than to fully replace it, thus achieving significantly faster computations, yet still generating the final trajectories from an exact optimization algorithm.

For our proposed approach to be effective, the trajectory from the predictor must be feasible. Concretely, it must avoid collisions and reach the goal, so that the online optimizer also computes a feasible trajectory that can be used directly or refined using the MILP in an anytime fashion. For feasibility, the approximation errors due to learning have a desirable direction: an error that steers away from the constraints is preferable to one that steers towards them. This preferred direction is not considered in standard learning approaches that aim at simply replicating the behavior of the data. In this paper, we modify conventional learning methods to increase the likelihood of obtaining feasible trajectories To this end, we customize the loss function to favor learning errors that steer the trajectory away from the constraints, and propose a "receding horizon" implementation to further recover feasibility. Supported by our numerical results, we posit that the proposed method can generalize to scenarios for which it has not been explicitly trained. For instance, we demonstrate the potential to generalize the IL-based approach to cases that are a subset of the training scenario, i.e., if we train for $N$ robots and $M$ obstacles, we show that the same neural network can handle an environment with $< N$ robots and $< M$ obstacles.

The rest of the paper is organized as follows. Section 2 provides background and describes the problem, and Section 3 presents the architecture of the proposed motion planning approach. Sections 4, 5, 6, and 7 describe the offline motion planning MILP, the training and operation of the predictor, and the online reduced motion planning problem, respectively. Simulation results are shown in Section 8 and conclusions are summarized in Section 9.

## 2. PRELIMINARIES AND PROBLEM DESCRIPTION

We define $\mathbb{R}$ and $\mathbb{Z}$ to be the sets of reals and integers, respectively. The cardinality of a set $\mathcal{S}$ is denoted by $|\mathcal{S}|$. For $a, b \in \mathbb{Z}$, we denote an interval of integers by $\mathbb{Z}_{[a,b)} = \{z \in \mathbb{Z} : a \leq z < b\}$. For a vector $a \in \mathbb{R}^n$, $[a]_i$ denotes the $i^{\text{th}}$ component of $a$, $i \in \mathbb{Z}_{[1,n]}$. Inequalities between vectors $a \leq b$, are intended component-wise.

### 2.1 Mixed-Integer Linear Programming

A binary mixed-integer linear program (MILP) is a linear program where some variables are restricted to be 0 or 1,

$$\begin{aligned} \min_{\xi} \quad & c^\top \xi \\ \text{subject to:} \quad & H\xi \leq K, \\ & [\xi]_h \in \mathbb{R}, \qquad \forall h \in \mathcal{H}_{\text{r}}, \\ & [\xi]_h \in \{0,1\}, \qquad \forall h \in \mathcal{H}_{\text{b}}, \end{aligned} \tag{1}$$

where $\xi$ is a vector of decision variables and $\mathcal{H}_{\text{r}}$, $\mathcal{H}_{\text{b}}$ are the set of indices of real and binary variables, respectively.

The discrete variables in MILPs allow modeling disjunctive and assignment constraints, e.g., this enables collision avoidance (stay outside of a region), alternative decisions (go left or right), assignments (allocate an agent to a task) and timing minimization (step counting) (Richards, 2002). Despite being non-convex, the MILP feasible domain, that is, the set $\Xi_F$ where (1) admits a finite solution, is the union of convex sets, so that the global optimum can be found in finite time, albeit combinatorial with respect to the number of discrete variables $|\mathcal{H}_{\text{b}}|$.

### 2.2 Motion Planning Problem Description

We consider $N$ robots, where the dynamics of the $i^{\text{th}}$ robot, $i \in \mathcal{I} = \mathbb{Z}_{[1,N]}$, is described by the discrete-time model

$$x^i_{k+1} = A^i x^i_k + B^i u^i_k, \tag{2a}$$
$$z^i_k = C^i x^i_k + D^i u^i_k, \tag{2b}$$

sampled with period $T_s$, where $x^i \in \mathcal{X}^i \subseteq \mathbb{R}^{n_i}$ is the state vector, $u^i \in \mathcal{U}^i \subseteq \mathbb{R}^{m_i}$ is the input vector, $z^i \in \mathcal{D} \subseteq \mathbb{R}^p$ is the output vector that describes the robot configuration, where $p = 2, 3$ for position-only configurations in 2D, 3D, respectively, $\mathcal{X}^i$, $\mathcal{U}^i$ are the sets of admissible states and inputs, and $\mathcal{D}$ is the environment, or workspace. The model (2) can represent heterogeneous and homogeneous multi-robot systems. In this paper, for simplicity, we consider a position-only 2D configuration space, $p = 2$, where $\mathcal{D} = \{z \in \mathbb{R}^2 : \ p_{\min} \leq z \leq p_{\max}\}$. In this case, the state vector can be defined as $x^i = \begin{bmatrix} z^{i\top} & v^{i\top} \end{bmatrix}^\top \in \mathbb{R}^4$ and $v^i \in \mathbb{R}^2$ is the velocity vector.

The workspace contains obstacles $o_j$, $j \in \mathcal{O} = \mathbb{Z}_{[1,N_{\text{o}}]}$, and the robots must avoid colliding with obstacles as well as with each other. A collision between robot $i$ and obstacle $j$ occurs if and only if $H^j_{\text{o}} z^i \leq K^j_{\text{o}}$: that is, if its configuration vector $z^i$ is inside a polytope. Similarly, robot $i$ collides with robot $j$ if $H^j_{\text{r}}(z^i - z^j) \leq K^j_{\text{r}}$: that is, if $z^i$ is in a polytope centered around $z^j$. Each robot $i \in \mathcal{I}$ must reach its corresponding goal state $x^i_{\text{goal}} \in \mathcal{X}^i$ in minimum time, within a maximum time horizon $T$, while avoiding the obstacles and other robots active in the environment.

As the goals and obstacles may change position, fast re-planning during robot operation may be needed. To this end, we propose an imitation learning framework to speedup the motion planning based on MILP for achieving these objectives, so that we can avoid solving an MILP with many binary variables online, during robot operation.

## 3. PROPOSED ARCHITECTURE FOR FAST REAL-TIME MOTION PLANNING

Our motion planning architecture is shown in Fig. 1, with an *offline training* and an *online planning* component.

The offline component includes a module that generates a dataset of optimal trajectories obtained by solving the full MILP for multi-robot motion planning for different initial conditions of the robots, target goals and obstacle positions. The obtained trajectories reach the goal optimally and are collision-free, thus they are referred to as *expert trajectories* and the module generating them is referred to as *Expert* 1. The dataset is used by the *training module* to train a neural network for predicting the robot trajectories.

In the online planning component, the robot states, the obstacle locations, and the target goal coordinates are given as input to the *predictor module* that uses the trained imitation learning network to return predictions for the robot trajectories. Since the predictions are approximations of the expert trajectories, we do not provide them directly to

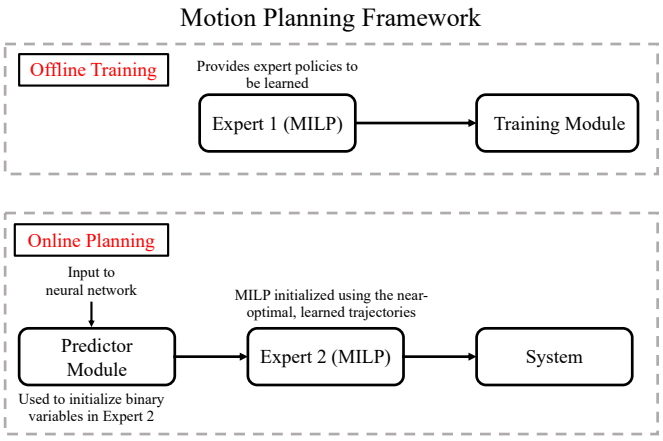


Fig. 1. The proposed IL-based motion planning framework that addresses the objectives in Section 2.2.

the robots. Instead, we extract information from the predictions to simplify the online motion planning problem. More specifically, based on the predicted trajectories, we fix the values of most (or even all) integer variables, which are the main source of complexity in MILPs. Then, the resulting simpler MILP is solved in the *Expert 2 module*, yielding the actual planning trajectories for the robots to follow. These can be further improved by solving the full MILP initialized at the solution of Expert 2, with a timeout for real-time feasibility.

For the proposed approach to be effective, the feasibility of the predicted trajectories is paramount, to ensure feasibility of Expert 2 so that its solution can be safely provided to the robots or refined based on the full MILP in an anytime fashion with an available backup for timeouts. However, since optimal trajectories often "ride" the constraints (Dantzig, 1998), it is likely that even a small perturbation to the imitation network predictions causes constraint violations. In addition, the trajectories for collision avoidance are not everywhere continuous with respect to the initial conditions (Panagou, 2014; Zhang et al., 2009). The continuous interpolating nature of standard activation functions may, therefore, result in planned trajectories that collide with an obstacle from some initial conditions. One possibility would be to use networks capturing such discontinuous behavior (Forti and Nistri, 2003), which however are harder to train since the loss functions are no longer always differentiable, and activation function selection requires knowledge of the type (e.g., jump) of discontinuities. Thus, in this paper we use feed-forward neural networks to imitate the MILP solver but we propose modifications that increase the likelihood of obtaining feasible trajectories. Hereafter, we describe the implementation of the different modules.

## 4. EXPERT 1: MILP FORMULATION (OFFLINE)

### *4.1 System Dynamics and Boundary Conditions*

For each robot $i \in \mathcal{I}$, the system dynamics are described by (2) with initial condition

$$x_0^i = x_{\text{init}}^i, \tag{3}$$

where $x_k^i \in \mathcal{X}^i$ is the state of the robot $i$ at time step $k$ and $x_{\text{init}}^i \in \mathcal{X}^i$ is the initial condition. For each robot $i \in \mathcal{I}$, reaching the corresponding goal state $x_{\text{goal}}^i$ is enforced as

$$\|x_k^i - x_{\text{goal}}^i\|_1 \leq M\,(1 - g_k^i)\ , \quad \sum_{k=1}^{T} g_k^i = 1, \tag{4}$$

where $M > 0$ is a constant that is always larger than the left-hand side (Conforti et al., 2014), and $g_k^i \in \{0,1\}$ for all $i \in \mathcal{I}$, $k \in \mathbb{Z}_{[1,T]}$, where $g_k^i = 1$ if robot $i$ reaches the goal at step $k$. In addition, path inequality constraints are included to restrict the position within the workspace $z_k^i \in \mathcal{D} = \{z \in \mathbb{R}^2 :\ p_{\min} \leq z \leq p_{\max}\}$, and to limit the velocity $v_k^i$ and input acceleration $u_k^i$ for each robot $i \in \mathcal{I}$ and each time step $k \in \mathbb{Z}_{[0,T]}$ as

$$\|v_k^i\|_\infty \leq v_{\max}^i, \quad \|u_k^i\|_\infty \leq u_{\max}^i. \tag{5}$$

### *4.2 Collision Avoidance Constraints*

The obstacle avoidance for each robot $i \in \mathcal{I}$ with respect to each obstacle $j \in \mathcal{O}$ is enforced by the constraints

$$H_{\text{o}}^j(z_k^i - \vartheta^j) \geq K_{\text{o}}^j - M\,(1 - b_k^{i,j}), \quad \sum_{h=1}^{n_j^{\text{o}}} [b_k^{i,j}]_h \geq 1, \tag{6}$$

where $\vartheta^j$ is the given position of the $j^{\text{th}}$ obstacle reference point, e.g., its center, $n_j^{\text{o}}$ is the number of inequalities of the polytope centered at $\vartheta^j$, and for each robot $i \in \mathcal{I}$, obstacle $j \in \mathcal{O}$, and time $k \in \mathbb{Z}_{[0,T]}$, $b_k^{i,j} \in \{0,1\}^{n_j^{\text{o}}}$ is a binary vector that enforce $z_k^i$ to lie outside of the polytope $H_{\text{o}}^j(z^i - \vartheta^j) \leq K_{\text{o}}^j$ where a collision occurs.

For any $i, p \in \mathcal{I}$, $i \neq p$, $k \in \mathbb{Z}_{[0,T]}$, the inter-robot collision avoidance is enforced by the constraints

$$H_{\text{r}}^p(z_k^i - z_k^p) \geq K_{\text{r}}^p - M\,(1 - d_k^{i,p}), \quad \sum_{h=1}^{n_p^{\text{r}}} [d_k^{i,p}]_h \geq 1, \tag{7}$$

where $n_p^{\text{r}}$ is the number of inequalities of the polytope around the $p^{\text{th}}$ robot, and $d_k^{i,p} \in \{0,1\}^{n_p^{\text{r}}}$ is a binary vector that enforce $z_k^i$ to lie outside of the polytope $H_{\text{r}}^p(z_k^i - z_k^p) \leq K_{\text{r}}^p$ where a collision occurs.

### *4.3 Minimum Time Cost Function*

The MILP cost function is

$$J^* = \sum_{i \in \mathcal{I}} \sum_{k=1}^{T} \left(k\, g_k^i + \gamma \|u_{k-1}^i\|_1\right), \tag{8}$$

where the first term minimizes the time to reach each robot's target, and the second term adds a small control penalty to make the B&B algorithm more efficient (Richards, 2002, pp. 75-76); here, $0 < \gamma \ll 1$. The 1-norm in (8) is encoded by adding auxiliary continuous variables as in (Richards, 2002).

### *4.4 MILP Problem Formulation*

The complete MILP is given by

$$\begin{aligned} \min_{x,u,g,b,d} \quad & J^*(g,u) \\ \text{subject to:} \quad & (2),\ (3),\ (4),\ (5),\ (6) \text{ and } (7), \end{aligned} \tag{9}$$

including $(T+1)\sum_{i \in \mathcal{I}} n_i$ state variables in $x$, $T \sum_{i \in \mathcal{I}} m_i$ control variables in $u$, $T\,N$ binary variables in $g$ for

reaching goals, $T\,N\sum_{j\in\mathcal{O}} n_j^{\mathrm{o}}$ binary variables in $b$ for obstacle avoidance and $T\,(N-1)\sum_{j\in\mathcal{I}} n_j^{\mathrm{r}}$ binary variables in $d$ for inter-robot collision avoidance. Assuming for simplicity, $n_j^{\mathrm{o}} = n^{\mathrm{o}}$ for all $j \in \mathcal{O}$, $n_j^{\mathrm{r}} = n^{\mathrm{r}}$ for all $j \in \mathcal{I}$, the total number of binary variables in (9) is

$$T\,N\,(1 + N_{\mathrm{o}}\,n^{\mathrm{o}} + (N-1)\,n^{\mathrm{r}})\,. \tag{10}$$

The number of binary variables increases quadratically with the number of robots $N$ and the number of obstacles $N_{\mathrm{o}}$ in the environment, making the solution of MILP in (9) generally intractable for fast re-planning during operation of the multi-robot system.

## 5. PREDICTION BY IMITATION LEARNING: DEEP NEURAL NETWORK (DNN)

The training module constructs the predictor by learning the parameters of a neural network with given architecture from Expert 1 trajectories for different robots' initial and goal conditions, and obstacle positions. While the architecture and hyperparameters are specific to our case study, the approach is general and extends to other cases.

### *5.1 Sampling For Dataset Generation*

Expert trajectories are generated by solving (9) using Expert 1, where each solution is computed for different $x_0^i$, $x_{\mathrm{goal}}^i$, $i \in \mathcal{I}$, and $\vartheta^j$, $j \in \mathcal{O}$. Concretely, we generate data for $x_0^i$, $x_{\mathrm{goal}}^i$, $\vartheta^j$ by sampling from a low-discrepancy sequence which ensures that the samples of initial conditions, goals, and obstacle positions are well-distributed within the workspace, rather than being clustered in subregions that could happen by sampling from a uniform distribution (Chakrabarty et al., 2017). These samples, along with the expert trajectories, constitute the training and label sets used for training, validation, and testing of the predictor module. In a 2D workspace ($p = 2$) with two robots and one obstacle, assuming $n_{\mathrm{x}} = 4$ state variables in the robot dynamics and the velocities are set to 0 for the target goal conditions, each sample in the training set is a vector with $2(n_{\mathrm{x}}+p)+p = 14$ elements comprising the initial state condition of each robot and the coordinates of its target goal, and of the obstacle center.

We make two modifications to the sampling method, inspired by practical considerations. A well-distributed sampling pattern will result in samples being drawn from within the obstacle, which we know *a priori* to result in Expert 1 to be infeasible. Therefore, first, we remove these infeasible samples from our dataset without even generating the labels. Second, we generate more samples in the immediate neighborhood of the obstacles, since more data increase learning precision and obstacle avoidance actions need to be more precise near the obstacles. Thus, we first randomly sample $\vartheta^j$, $j \in \mathcal{O}$, $x_{\mathrm{goal}}^i$, $i \in \mathcal{I}$, then select neighborhoods of the obstacles and randomly sample $x_0^i$, $i \in \mathcal{I}$ in such neighborhoods.

### *5.2 Imitation Learning Network Architecture*

The predictor is obtained by learning to imitate Expert 1 and implemented as a deep neural network. In our simulations of Section 8, the predictor is a five-layer (1 input, 3 hidden, 1 output) deep neural network with 50 neurons in the input layer and 100-100-50 neurons in the hidden layers. Each network layer is fully connected with Leaky ReLUs as activation functions, with slope 0.1 and no dropout. The output layer is linear. For $N_{\mathrm{o}}$ obstacles and $N$ robots, each with state dimension $n_{\mathrm{x}}$, the input to the predictor module has dimension $N(n_{\mathrm{x}} + p) + N_{\mathrm{o}}p$ and the output has dimension $N\,T n_{\mathrm{x}}$, including a trajectory of state values for each robot over the $T$-steps horizon.

### *5.3 Imitation Network Training*

We use a train-validation-test split of 80/10/10 and set the number of epochs to 2000. We describe the loss function in more detail in the next subsection. We use the Adam algorithm to optimize the network (Kingma and Ba, 2014) with a batch-size of 64, and an adaptive learning rate: $10^{-2}$ for the first 50 epochs, $10^{-3}$ for the next 150 epochs, and $10^{-4}$ afterwards. To avoid overfitting and promote uniqueness of solutions, we add a $\mathcal{L}_2$ regularizer to the loss function, with regularization parameter $10^{-4}$.

### *5.4 Tailored Loss Function Implementation*

In order to promote feasibility of predictor module trajectories, we include in the imitation learner's loss function a barrier term to bias each robot configuration to stay outside a neighborhood of the obstacles and other robots

$$\begin{aligned}\mathcal{L}_{\mathrm{obs}} &= \frac{1}{N\,N_{\mathrm{o}}\,T\,B}\sum_{i=1}^{N}\sum_{m\in\mathcal{O}}\sum_{j=1}^{B}\sum_{k=0}^{T}\phi_{\mathrm{o}}^m(z_k^{i,j}),\\ \mathcal{L}_{\mathrm{robots}} &= \frac{1}{N^2\,T\,B}\sum_{i=1}^{N}\sum_{m\in\mathcal{I}\setminus\{i\}}\sum_{j=1}^{B}\sum_{k=0}^{T}\phi_{\mathrm{r}}^m(z_k^{i,j}),\end{aligned} \tag{11}$$

where $\phi_{\mathrm{o}}^m(\cdot)$ and $\phi_{\mathrm{r}}^m(\cdot)$ denote the barrier functions for the obstacles and robots, respectively. The barrier terms (11) are imposed for each element in the batch size $B$ and for each time step $k \in \mathbb{Z}_{[0,T]}$. In addition to (11), given the state $x_{\mathrm{ref},k}^{i,j}$ of the Expert 1 trajectory for robot $i$ at time step $k$ in batch element $j$, we add a mean-squared error (MSE) loss term, to promote imitating such trajectories

$$\mathcal{L}_{\mathrm{MSE}} = \frac{1}{N\,T\,B}\sum_{i=1}^{N}\sum_{j=1}^{B}\sum_{k=0}^{T}\|x_k^{i,j} - x_{\mathrm{ref},k}^{i,j}\|_2^2. \tag{12}$$

Then, the complete loss function (without regularization) reads as

$$\mathcal{L} = w_{\mathrm{MSE}}\,\mathcal{L}_{\mathrm{MSE}} + w_{\mathrm{obs}}\,\mathcal{L}_{\mathrm{obs}} + w_{\mathrm{robots}}\,\mathcal{L}_{\mathrm{robots}}, \tag{13}$$

where $w_{\mathrm{MSE}}, w_{\mathrm{obs}}, w_{\mathrm{robots}} \in \mathbb{R}_{>0}$ are loss function weights. A candidate for the barrier function in (11) is

$$\phi_\star^m(z_k^{i,j}) = \frac{\pi}{2} - \tan^{-1}\left(\alpha\left(\|z_k^{i,j} - C_m\|_{P_m}^2 - 1\right)\right), \tag{14}$$

where $\|z_k^{i,j} - C_m\|_{P_m}^2 = (z_k^{i,j} - C_m)^\top P_m (z_k^{i,j} - C_m)$ and $\alpha > 0$ is a scaling factor. The barrier loss (14) is based on an ellipsoid enclosing the obstacle or robot $m$, with symmetric positive definite shape matrix $P_m \succ 0$ and center $C_m \in \mathcal{D}$. The weights in the loss (13) and the shape parameters in the barrier function (14) are hyperparameters, tuned using the validation dataset.

**Algorithm 1** Receding Horizon-based DNN Predictor.

1: **Input:** Initial $f_0 = \Big(\{x_0^i\}_{i\in\mathcal{I}}, \{x_{\text{goal}}^i\}_{i\in\mathcal{I}}, \{\vartheta^j\}_{j\in\mathcal{O}}\Big)$.
2: $\mathcal{T}^i \leftarrow \emptyset$, $\forall i \in \mathcal{I}$ and $f \leftarrow f_0$.
3: $\{x_k^i\}_{i\in\mathcal{I},k\in\mathbb{Z}_{[0,T]}} \leftarrow \texttt{DNN}(f)$.
4: $\delta_{obs} \leftarrow \texttt{ObstacleCollision}\left(\{x_k^i\}_{i\in\mathcal{I},k\in\mathbb{Z}_{[0,T]}}, \{\vartheta^j\}_{j\in\mathcal{O}}\right)$.
5: $\delta_{robot} \leftarrow \texttt{RobotCollision}\left(\{x_k^i\}_{i\in\mathcal{I},k\in\mathbb{Z}_{[0,T]}}\right)$.
6: **if** $\delta_{obs}$ is False **and** $\delta_{robot}$ is False **then**
7: $\quad\mathcal{T}^i \leftarrow \texttt{Concatenate}\left(\mathcal{T}^i, \{x_k^i\}_{k\in\mathbb{Z}_{[0,T]}}\right)$, $\forall i \in \mathcal{I}$.
8: $\quad$**Return** $\mathcal{T}^i$ for $i \in \mathcal{I}$.
9: **else**
10: $\quad\mathcal{T}^i \leftarrow \texttt{Concatenate}\left(\mathcal{T}^i, x_0^i\right)$, $\forall i \in \mathcal{I}$.
11: $\quad f \leftarrow \Big(\{x_1^i\}_{i\in\mathcal{I}}, \{x_{\text{goal}}^i\}_{i\in\mathcal{I}}, \{\vartheta^j\}_{j\in\mathcal{O}}\Big)$.
12: $\quad$**GoTo** Step 3.
13: **end if**
14: **Output:** Trajectories $x_k^i$ for $i \in \mathcal{I}$, $k \in \mathbb{Z}_{[0,T]}$.

## 6. RECEDING HORIZON-BASED DNN PREDICTOR (ONLINE)

In practical situations, even if the training set is well-distributed and the imitation learner is of sufficient complexity to imitate Expert 1 with high accuracy, open-loop prediction of trajectories can result in infeasibility, due to inter-robot collisions or collisions with obstacles. This is because the imitation learner cannot mimic Expert 1 with 100% accuracy for the reasons discussed in Section 3. To alleviate this issue, we propose a receding horizon implementation along the lines of (Chakrabarty et al., 2017), where only a part of the robot trajectory from the predictor is used, the state values are updated accordingly, and the learner re-predicts the remainder of the trajectory from such new states.

The approach is summarized in Algorithm 1. The algorithm first predicts trajectories of the robots, given the initial conditions (step 3). If the trajectories are feasible, then the algorithm terminates, and the predicted trajectory, concatenated to any previously computed trajectory prefix, is returned (step $7-8$). If infeasibility is detected, due to a collision with obstacles (`ObstacleCollision`) or due to a collision between robots (`RobotCollision`), then the initial state is concatenated to any existing trajectory prefix. The robots' state values are updated by one step, and used as the new initial condition from which a new trajectory is predicted. This process is repeated until feasible trajectories for all robots are obtained, or a pre-decided termination time is reached, after which one could apply a backup approach (e.g., solving the full MILP on-line). Due to concatenation, the trajectories provided to Expert 2 may be longer than $T$ time steps, which can be handled in post processing if the goal is reached in less than $T$ steps, or by expanding the horizon in Expert 2. Re-planning the trajectories adds a significant degree of robustness, as expected from receding horizon methods (Rawlings and Mayne, 2009).

## 7. EXPERT 2: REDUCED RE-OPTIMIZATION (ONLINE)

The Expert 2 module in Figure 1 aims at improving the trajectories from the predictor towards optimality. Expert 2 solves online a small-scale optimization problem obtained by fixing most or even all of the binary variables in the MILP (9). Given the trajectories $(z_k^i)_{k=0}^T$ for each $i \in \mathcal{I}$ provided by the predictor module, the values for the binary variables $b_k^{i,j}$ in (6) and $d_k^{i,j}$ in (7) are uniquely defined. Fixing these binary variables results in a set of convex constraints that enforce each robot $i$ to be on the outside of (at least) one half-space of the polytope $j \in \mathcal{O}$, $j \in \mathcal{I}\setminus\{i\}$ where a collision occurs with obstacles and other robots, respectively. The resulting reduced MILP is

$$\begin{aligned}\min_{x,u,g}\quad & J^*(g,u)\\ \text{subject to:}\quad & (2)\text{–}(7),\\ & [b_k^{i,j}]_h = 0 \text{ or } 1,\ h\in\mathbb{Z}_{[1,n_j^{\text{o}}]},\ i\in\mathcal{I},\ j\in\mathcal{O},\\ & [d_k^{i,j}]_h = 0 \text{ or } 1,\ h\in\mathbb{Z}_{[1,n_j^{\text{r}}]},\ i\in\mathcal{I},\ j\in\mathcal{I}\setminus\{i\}.\end{aligned} \tag{15}$$

The above MILP includes $T\,N$ binary variables, those related to reaching the goal state (4), which are considerably less than the binary variables in (10). A solution of (15) could be obtained by solving convex linear programs (LPs) for fixed values of the binary variables $g_k^i$, hence fixing the time for each robot to reach its goal state (4), until the minimum time is found. In addition, the trajectories from Expert 2 could be further refined by using them as an integer-feasible solution guess for the full MILP (9), to be solved with a fixed timeout for real-time feasibility.

## 8. NUMERICAL SIMULATION RESULTS

In this section, we illustrate our approach for a two-robot and a four-robot system in a 2D workspace $\mathcal{D}$ of size $5\times 5$. The robots are shaped as squares of width 0.6 and have decoupled $x$–$y$ double integrator dynamics (2), where

$$\begin{aligned}&A^i = \begin{bmatrix} I & T_s I \\ 0 & I \end{bmatrix}, \qquad B^i = \begin{bmatrix} \frac{T_s^2}{2} I \\ T_s I \end{bmatrix},\\ &C^i = [\,I\ 0\,] \qquad\qquad D^i = 0, \qquad\qquad \forall i\in\mathcal{I},\end{aligned} \tag{16}$$

$T_s = 0.1$ is the sampling period, and $I \in \mathbb{R}^{2\times 2}$ is the identity matrix. Thus, $z^i \in \mathcal{D} \subset \mathbb{R}^2$ is the 2D position vector, the state $x^i \in \mathcal{X}^i \subset \mathbb{R}^4$ contains the position vector $z^i \in \mathcal{D}$ and the velocity vector $v^i \in \mathcal{V}_i \subset \mathbb{R}^2$, $[v^i]_h \in [-1,1]$, $h \in \mathbb{Z}_{[1,2]}$, and the input $u^i \in \mathcal{U} \subset \mathbb{R}^2$ is the acceleration vector, $[u^i]_h \in [-1,1]$, $h \in \mathbb{Z}_{[1,2]}$. The obstacles are grown to account for the robot volume.

### *8.1 Case 1: Two robots and one fixed obstacle*

First, we consider a scenario where there are two robots and one fixed obstacle centered at $(2.5, 2.5)$. Here, our approaches significantly reduce the number of initial conditions from which predicted trajectories are infeasible. Table 1 reports the actual number of infeasible trajectories using only MSE loss (12), MSE loss with barrier term (13), and loss (13) and Algorithm 1. Averaging across the tests, the loss (13) reduces the infeasible trajectories by more than 30%, and adding also Algorithm 1 results in no more than 0.5% predicted trajectories to be infeasible.

### *8.2 Case 2: Two robots and two changing obstacles*

We extend the previous setup to a scenario with 2 robots and 2 static obstacles, where the center of each obstacle can be changed from one planning time step to the

Table 1. Number of infeasible trajectories predicted using MSE loss (12), the proposed loss with barrier terms (13), and (13) in combination with Algorithm 1.

| # Samples | MSE | MSE + Barrier | MSE + Barrier + Alg. 1 |
|---|---|---|---|
| 1000 | 146 | 91 | 5 |
| 2000 | 253 | 174 | 7 |
| 3000 | 411 | 291 | 9 |
| 4000 | 524 | 380 | 11 |
| 5000 | 660 | 459 | 17 |

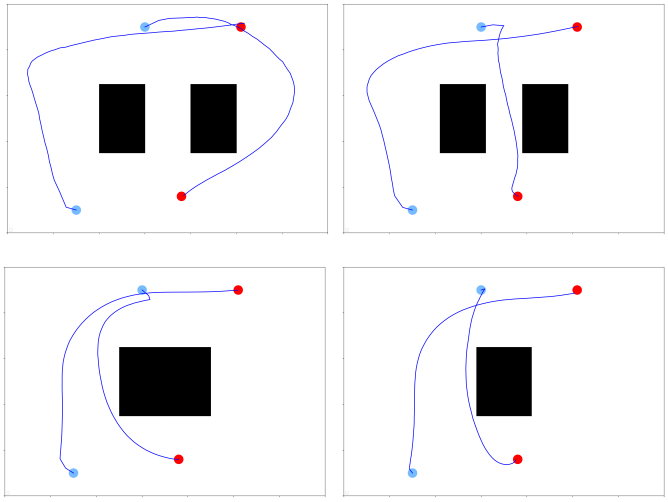

Fig. 2. Trajectories based on the receding horizon predictor in Algorithm 1 when the positions of the two obstacles change. The proposed approach predicts feasible trajectories for the different conditions, including when the obstacles partially or completely overlap.

next. Therefore, the inputs to the DNN predictor module include the current position for each of the 2 obstacles, which allows us to compute motion planning trajectories for different obstacle configurations. This is illustrated in Figure 2, which shows the trajectories by the receding horizon-based predictor in Algorithm 1 for different center positions of the obstacles. As illustrated in Figure 2, varying the positions of the obstacles in the training dataset allows us to compute online plans for the cases where multiple obstacles are merged into one larger obstacle (bottom left) or the number of obstacles is reduced by placing two obstacles on top of each other (bottom right).

### *8.3 Case 3: Four robots and three fixed obstacles*

Next, we consider a scenario with four robots and three obstacles, see Fig. 3, where the problem horizon is $T = 60$. In Fig. 3, the trajectories generated by Expert 1 are shown in dashed yellow, and the solid green lines are the trajectories generated by Expert 2, which is initialized by the predictor module. Using Gurobi (Gurobi Optimization, 2020) from its Python interface, the computing time for Expert 2 was 81 milliseconds, whereas the computation time for Expert 1 was 2207 milliseconds, both including the interface overhead which is approximately constant. Table 2 reports additional data of the computation time comparison between Expert 1 and our approach of predictor and Expert 2, for different initial conditions. Our approach speeds up the solution from 4 to 55 times, with more evident improvements in the more complex cases.

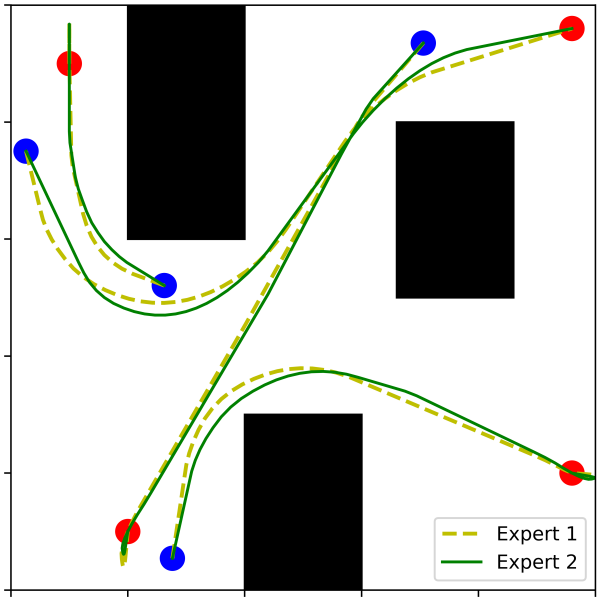


Fig. 3. Trajectories for four robots obtained from Expert 1 (yellow, dash) and by the proposed approach with receding horizon predictor and Expert 2 (green, solid). Initial conditions (blue circles), targets (red circles).

### *8.4 Case 4: Changing number of active robots*

Next, we show that the deep neural network does not appear to need specific training to handle cases that are simplifications of the training scenarios. We consider the case where we train a network with 4 robots, but during testing only a subset of robots in the environment are active. Here, for simplicity, we set the target positions of the inactive robots to their initial positions, and use the neural network to compute trajectories for all the robots. In Figure 4, we show the plan made by the neural network in a receding horizon manner for 4, 3, 2, and 1 active robots. The neural network restricts the inactive robots near their respective initial conditions while generating feasible paths for the active robots, indicating that it has learned that the robots motion can be somewhat decoupled when they are inactive, even though this behaviour was not explicitly part of the training dataset. The approach could be further refined by placing the inactive robots in a special section of the workspace that is disconnected from the main workspace, thus avoiding potential impact caused by their mere presence. The results in Figure 4 suggest that the DNN may be trained for a larger number of robots and obstacles, and then used without transfer learning or online adaptation in scenarios with fewer robots and obstacles, which increases the generalization capabilities of the proposed method.

## 9. CONCLUSIONS AND FUTURE WORK

We proposed a motion planning method for multi-robot systems that leverages machine learning to reduce the computational burden of MILP-based online trajectory

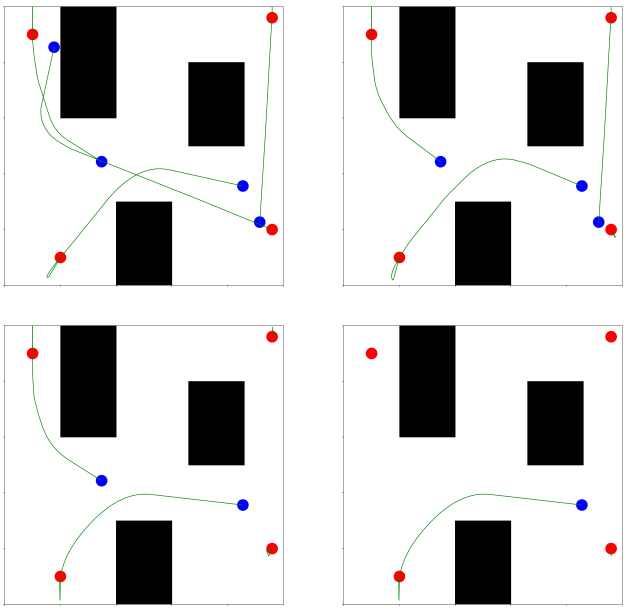

Fig. 4. Trajectories for varying number of active robots based on Algorithm 1 and Expert 2, by setting initial condition equal to target position for inactive robots.

Table 2. Computation time for the predictor module plus Expert 2 (ML-MILP) versus the MILP timing in Expert 1.

| Scenario of Robots | Proposed ML-MILP Framework (ms) | Traditional MILP of Expert 1 (ms) |
|---|---|---|
| 1 | 95 | 570 |
| 2 | 96 | 420 |
| 3 | 16 | 836 |
| 4 | 12 | 667 |
| 5 | 109 | 3929 |

generation while ensuring feasibility of the plan under operational and safety constraints. We construct a predictor by imitation learning from an expert that solves the computationally intensive MILP-based trajectory generation problem. Using the predictor, the online trajectory generation problem is reduced, so that it solves faster and requires fewer computational resources. We developed a receding horizon-based predictor method to increase the likelihood of returning feasible trajectories, i.e., collision-free and reaching the targets. We provided simulation results that showed the effectiveness and generalization capabilities of the approach.

## REFERENCES


Albert, A., Leira, F.S., and Imsland, L. (2017). UAV path planning using MILP with experiments. *Modeling, Identification and Control*, 38(1), 21–32.

Bengio, Y., Lodi, A., and Prouvost, A. (2020). Machine learning for combinatorial optimization: A methodological tour d'horizon. *Europ. Jour. Oper. Research*.

Chakrabarty, A., Dinh, V., Corless, M.J., Rundell, A.E., Żak, S.H., and Buzzard, G.T. (2017). Support vector machine informed explicit nonlinear model predictive control using low-discrepancy sequences. *IEEE Trans. Automatic Control*, 62(1), 135–148.

Conforti, M., Cornuéjols, G., and Zambelli, G. (2014). *Integer programming*, volume 271. Springer.

Dantzig, G.B. (1998). *Linear programming and extensions*, volume 48. Princeton university press.

Faust, A., Oslund, K., Ramirez, O., Francis, A., Tapia, L., Fiser, M., and Davidson, J. (2018). PRM-RL: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning. In *IEEE Int. Conf. Robotics and Automation*, 5113–5120.

Floudas, C.A. (1995). *Nonlinear and mixed-integer optimization: fundamentals and applications*. Oxford University Press.

Forti, M. and Nistri, P. (2003). Global convergence of neural networks with discontinuous neuron activations. *IEEE Trans. Circuits and Systems I*, 50(11), 1421–1435.

Gombolay, M., Wilcox, R., and Shah, J. (2013). Fast scheduling of multi-robot teams with temporospatial constraints. In *Proc. of the Robots: Science and Systems*.

Gurobi Optimization, L. (2020). Gurobi optimizer reference manual. URL `http://www.gurobi.com`.

Johnson, J.J., Li, L., Liu, F., Qureshi, A.H., and Yip, M.C. (2020). Dynamically constrained motion planning networks for non-holonomic robots. *arXiv preprint arXiv:2008.05112*.

Karg, B. and Lucia, S. (2018). Deep learning-based embedded mixed-integer model predictive control. In *European Control Conf.*, 2075–2080.

Kingma, D.P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Masti, D. and Bemporad, A. (2019). Learning binary warm starts for multiparametric mixed-integer quadratic programming. In *European Control Conf.*, 1494–1499.

Panagou, D. (2014). Motion planning and collision avoidance using navigation vector fields. In *IEEE Int. Conf. Robotics and Automation*, 2513–2518.

Qureshi, A.H. and Yip, M.C. (2018). Deeply informed neural sampling for robot motion planning. In *Int. Conf. Intelligent Robots and Systems*, 6582–6588.

Qureshi, A.H., Miao, Y., Simeonov, A., and Yip, M.C. (2020). Motion planning networks: Bridging the gap between learning-based and classical motion planners. *IEEE Transactions on Robotics*.

Rawlings, J.B. and Mayne, D.Q. (2009). *Model predictive control: Theory and design*. Nob Hill Pub.

Richards, A.G. (2002). *Trajectory optimization using mixed-integer linear programming*. Ph.D. thesis, Massachusetts Inst. Tech.

Schouwenaars, T., De Moor, B., Feron, E., and How, J. (2001). Mixed integer programming for multi-vehicle path planning. In *European Control Conf.*, 2603–2608.

Yan, Z., Jouandeau, N., and Cherif, A.A. (2013). A survey and analysis of multi-robot coordination. *International Journal of Advanced Robotic Systems*, 10(12), 399.

Zhang, J., Springenberg, J.T., Boedecker, J., and Burgard, W. (2017). Deep reinforcement learning with successor features for navigation across similar environments. In *Int. Conf. Intelligent Robots and Systems*, 2371–2378.

Zhang, L., LaValle, S.M., and Manocha, D. (2009). Global vector field computation for feedback motion planning. In *Int. Conf. Robotics and Automation*, 477–482.