Scalable and Safe Multi-Agent Motion Planning with Nonlinear Dynamics and Bounded Disturbances

Abstract

We present a scalable and effective multi-agent safe motion planner that enables a group of agents to move to their desired locations while avoiding collisions with obstacles and other agents, with the presence of rich obstacles, high-dimensional, nonlinear, nonholonomic dynamics, actuation limits, and disturbances. We address this problem by finding a piecewise linear path for each agent such that the actual trajectories following these paths are guaranteed to satisfy the reach-and-avoid requirement. We show that the spatial tracking error of the actual trajectories of the controlled agents can be precomputed for any qualified path that considers the minimum duration of each path segment due to actuation limits. Using these bounds, we find a collision-free path for each agent by solving Mixed Discrete-Continuous Linear Programs and coordinate agents by using priority-based search. We demonstrate our method by benchmarking in 2D and 3D scenarios with ground vehicles and quadrotors, respectively, and show improvements over the solving time and the solution quality compared to two state-of-the-art multi-agent motion planners.

1 Introduction

Multi-agent motion planning has a wide range of real-world applications, but it is notoriously difficult. Even navigating a single robot from an initial location to a goal location while avoiding collisions with obstacles is terribly challenging with the presence of rich obstacles, high-dimensional, nonlinear, nonholonomic dynamics, actuation limits, and disturbances. Not to say that when such complex robotic systems can interfere with each other in a shared environment, the scale of this problem is beyond the capability of most existing methods.

In this paper, we present Scalable and Safe Multi-agent Motion (S2M2) planner, a novel multi-agent motion planner that can fast and effectively generates provably safe plans for agent models with high-dimensional nonlinear dynamics and bounded disturbances in continuous time and space. Instead of directly planning dynamically-feasible trajectories, which are extremely computationally expensive, S2M2 exploits a separation-of-concerns approach: We first design piecewise linear (PWL) paths S_i for each dynamical agent $\mathcal{A}_i$ to follow, with the understanding that the agents cannot follow those PWL paths exactly. However, we show that with appropriate tracking controllers, the actual trajectories of each agent under disturbances can converge to S_i with guaranteed bounds on the spatial tracking error. More importantly, we show that such error bound can be pre-computed for any qualified PWL path. The secrete sauce behind the high efficiency of S2M2 is that we are able to formulate the problem of finding PWL paths for a single agent as a Mixed Integer-Linear Program (MILP), which can be solved efficiently using off-the-shelf MILP solvers.

To avoid inter-agent collisions, we wrap our single-agent motion planner with the priority-based search (Ma et al. 2019) that explores the space of priority orderings using systematic search. When priorities are specified, lower-priority agents replan their paths while treating higher-priority agents as moving obstacles. Together with the MILP-based path planner and guaranteed tracking controller, S2M2 can efficiently find plans that are provably safe and robust to disturbances during execution. Moreover, by planning paths for multiple agents on a continuous map over continuous time, our method finds high-quality solutions with low flowtime (i.e., makespan sum of all single-agent plans).

Consider an example of two disc-shaped vehicles making u-turns, as shown in Figure 1. Given the partially known initial locations and bounded disturbances, we first compute the maximum spatial error of tracking a path segment, which is a line, and then consider this error together with the agent shape to obtain the possible swept area of the path segment. The swept area of agent $\mathcal{A}_1$ during its second segment (p_{12}, p_{13}) is shown in light blue. We also consider the minimum duration for each segment such that the agent has enough time to adjust its position after turning, which is 3 seconds in our example. When our single-agent motion planner plans such paths for the agents, the obstacles are bloated with respect to the spatial tracking error and the agent shape, as shown in light red. The planner also constrains each segment to respect the minimum segment duration. As we can see, when passing the corridor, both agents slow down and use this 3 seconds to adjust its trajectory. After passing the corridor, agent $\mathcal{A}_2$ uses its full speed to approach its goal location. When potential collisions are detected, some agents are assigned higher priorities and thus are treated as moving obstacles for others. In our example, agent $\mathcal{A}_1$ is assigned

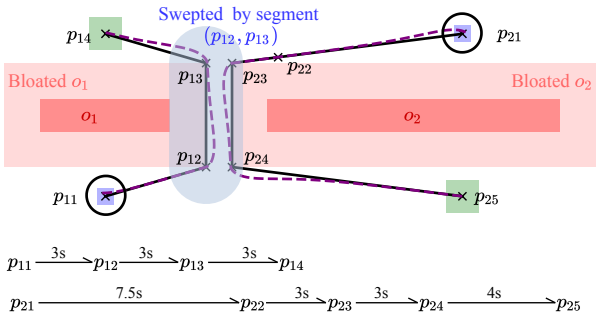


Figure 1: Example of two disc-shaped vehicles making u-turns. The initial and goal position sets are in blue and green, respectively. The obstacles are in red. The bloated obstacles are in light red. The paths of agents $\mathcal{A}_1$ and $\mathcal{A}_2$ are shown in solid black lines, and examples of their actual trajectories are shown in dashed purple lines. The swept area of agent $\mathcal{A}_1$ during its second segment is in light blue.

higher priority and thus passes the corridor before agent $\mathcal{A}_2$.

We compare the performance of S2M2 with ECBS-CT (Cohen et al. 2019) and MAPF/C+POST (Hönig et al. 2018) on 2D and 3D scenarios with ground vehicles and quadrotors, respectively. Results show that S2M2 outperforms both planners in terms of solution qualities with $15 \sim 70\%$ reduction for most instances. Moreover, S2M2 requires several magnitudes less time on pre-processing and also shows competitive runtime performance compared to the other two planners.

2 Related Work

Many works have approached the Multi-agent Motion Planning (MAMP) problem from the perspectives of AI or robotics. These works can be generally divided into two categories with discrete and continuous settings, respectively. In the discrete setting, time and space are discretized into time steps and grids, respectively. Each agent occupies exactly one grid and can only move to adjacent grids at each time step. This problem is known as Multi-agent Path Finding (MAPF) (Stern et al. 2019). Researchers in the past years have made substantial progress in finding high-quality solutions to various scenarios with hundreds of agents and high congestion as described in surveys (Ma et al. 2016; Felner et al. 2017). Prioritized planning is a popular and widely-used class of MAPF algorithms that coordinates multiple agents by specifying priorities among agents and forcing lower-priority agents to avoid collisions with higher-priority agents by treating their paths as dynamic obstacles (Velagapudi, Sycara, and Scerri 2010; Čáp, Vokřínek, and Kleiner 2015). The most recent prioritized planning algorithm Priority-based Search (PBS) (Ma et al. 2019) systematically explores the space of priority orderings, which can lead to near-optimal solutions and possibly scale to a thousand agents. Our method adapts PBS to a continuous setting and thus can efficiently coordinate a large number of agents.

While the classical MAPF assumes synchronized and discretized time, zero-volume shape, constant velocities, and rectilinear movement, several notable attempts have been made towards closing the gap between classical MAPF and MAMP using more realistic models. This includes considering the continuous timeline, different-size agent shapes, kinematic constraints, robustness, and any-angle moving directions (Walker, Sturtevant, and Felner 2018; Li et al. 2019; Cohen et al. 2019; Andreychuk et al. 2019; Ma, Kumar, and Koenig 2017; Atzmon et al. 2020; Yakovlev and Andreychuk 2017). To obtain robust, executable, and high-quality solutions, our method also supports these features.

In the continuous setting, sampling-based motion planners are often used (Le and Plaku 2018; Hönig et al. 2018). They first generate a probabilistic roadmap and then apply MAPF algorithms on it. These MAPF solutions are either used to guide the motion tree expansion (Le and Plaku 2018) or post-process to valid continuous trajectories (Hönig et al. 2018). Similar to our approach, these algorithms can handle high-order, nonlinear dynamics, and arbitrary complex geometries. Some optimization-based planners tend to solve a large optimization problem in which the decision variables define the trajectories for all agents (Augugliaro, Schoellig, and D'Andrea 2012; Mellinger, Kushleyev, and Kumar 2012), which are only demonstrated on small agent teams. Our motion planner also uses optimization problems for generating trajectories. However, we only coordinate different agents on demand. To find a feasible plan for each agent, we focus on finding a PWL path satisfying certain duration and boundary requirements, which are encoded as MILPs and solved efficiently.

As safety guarantee is important to successful execution, safe motion planning is receiving more attention recently. Several approaches are studied in a reference tracking framework, which uses the idea of bounding tracking errors through pre-computation based reachability analysis (Herbert et al. 2017; Singh et al. 2017; Vaskov et al. 2019; Fan, Miller, and Mitra 2020; Tedrake 2009; Majumdar and Tedrake 2017). Other safe motion planners employ barrier functions (Barry, Majumdar, and Tedrake 2012; Agrawal and Sreenath 2017) or use robust model predictive control with chance constraints (Blackmore, Ono, and Williams 2011; Jasour and Williams 2019; Yu et al. 2013). Some of these works have been extended to the multi-robot setting (Wang, Ames, and Egerstedt 2016; Panagou, Stipanović, and Voulgaris 2013; Srinivasan, Coogan, and Egerstedt 2018; Desai et al. 2019; Huang, Ayton, and Williams 2018; Richards and How 2004; Wagner and Choset 2017). Recent learning-based methods also show great potential to effectively navigate robot teams (Semnani et al. 2020; Liu et al. 2020). However, they cannot provide safety guarantees and may lead to collisions.

3 Preliminaries and Problem Statement

For an n-dimensional vector $x \in \mathbb{R}^n$, $x^{(i)}$ is its i^{th} entry, $\|x\|$ is its Euclidean norm, and $B_\epsilon(x) \equiv \{y \in \mathbb{R}^n \| \ \|y - x\| \leq \epsilon\}$ is the ϵ-ball centered at it with $\epsilon > 0$. Given a matrix $H \in \mathbb{R}^{n \times m}$ and a vector $b \in \mathbb{R}^n$, an (H, b)-polytope $\texttt{Poly}(H, b)$ is the set $\{x \in \mathbb{R}^m \| \ Hx \leq b\}$. Each row of the polytope

defines a halfspace $H^{(i)}x \leq b^{(i)}$, and each face is defined by $H^{(i)}x = b^{(i)}$. $\texttt{dP}(H)$ is the number of its faces.

Definition 1 (Agent Model)**.** An agent model $\mathcal{A}_i = \langle \mathcal{X}_i, \mathcal{U}_i, \mathcal{D}_i, f_i, \mathcal{P}_i \rangle$ is defined by its state space $\mathcal{X}_i \in \mathbb{R}^n$, input space $\mathcal{U}_i \subseteq \mathbb{R}^m$, disturbance space $\mathcal{D}_i \in \mathbb{R}^l$, dynamic function $f_i : \mathcal{X}_i \times \mathcal{U}_i \times \mathcal{D}_i \to \mathcal{X}_i$, and geometric shape $\mathcal{P}_i : \mathcal{X}_i \to 2^{\mathbb{R}^\delta}$.[1]

The semantics of agent dynamics are defined by trajectories, which describe the evolution of states over time. An input trajectory u over duration T is a continuous function u: $[0, T] \to \mathcal{U}_i$, which maps each time $t \in [0, T]$ to a control signal $u(t) \in \mathcal{U}_i$. Similarly, a disturbance trajectory d over duration T is a continuous function d: $[0, T] \to \mathcal{D}_i$. Given an input trajectory u over $\mathcal{U}_i$, a disturbance trajectory d over $\mathcal{D}_i$, and an initial state $x_0 \in \mathcal{X}_i$, its state trajectory ξ_i satisfies $\xi_i(x_0, u, d, 0) = x_0$ and for all $t > 0$,

$$\dot{\xi}_i(x_0, u, d, t) = f_i(\xi_i(x_0, u, d, t), u(t), d(t)).$$

Example 1. Consider a nonholonomic differential two-wheeled vehicle (Rodríguez-Seda et al. 2014) as an example. Its state $\xi_i(t)$ consists of three components: $p(t) = [p_x(t), p_y(t)]^T$ is the Cartesian coordinate of the center of inertia, $\theta(t)$ is the angular orientation, and $v(t)$ is the linear velocity. We also consider the bounded disturbances d_x on p_x, d_y on p_y, and d_θ on θ. The dynamic function f_i consists of five components: $\dot{p}_x(t) = v(t)\cos\theta + d_x(t)$, $\dot{p}_y(t) = v(t)\sin\theta + d_y(t)$, $\dot{v}(t) = u_1(t) - kv(t)$, $\dot{\omega}(t) = u_2(t) - k\omega(t)$, and $\dot{\theta}(t) = \omega(t) + d_\theta(t)$, where k is a constant, and $u_1(t)$ and $u_2(t)$ are control force and torque as inputs, which can be used to compute the torques for the left and right wheels.

Definition 2 (MAMP)**.** A multi-agent motion planning (MAMP) problem is defined by

$$\langle \mathcal{W}, O, \mathcal{A}_1, .., \mathcal{A}_N, \mathcal{X}_1^{\texttt{init}}, .., \mathcal{X}_N^{\texttt{init}}, \mathcal{X}_1^{\texttt{goal}}, .., \mathcal{X}_N^{\texttt{goal}} \rangle,$$

where workspace $\mathcal{W} \subseteq \mathbb{R}^\delta$ is a bounding box in $\mathbb{R}^\delta$; $\delta = 2$ for ground vehicles and $\delta = 3$ for aerial and underwater vehicles, and $\xi_i(t) \downarrow \mathcal{W}$ is the projection of $\xi_i(t)$ to the workspace; obstacles $O = \{o_i\}_i \subseteq \mathcal{W}$ are polytopes in $\mathcal{W}$; $\mathcal{A} = \{\mathcal{A}_1, .., \mathcal{A}_N\}$ is a set of agent models; $\mathcal{X}_i^{\texttt{init}} \subseteq \mathcal{X}_i$ and $\mathcal{X}_i^{\texttt{goal}} \subseteq \mathcal{X}_i$ are the initial set and the goal set of $\mathcal{A}_i$. The planning problem is to find inputs $(u_1, .., u_N)$ for every $(x_1^{\texttt{init}}, .., x_N^{\texttt{init}}) \in \mathcal{X}_1^{\texttt{init}} \times \mathcal{X}_2^{\texttt{init}} \times \cdots \mathcal{X}_N^{\texttt{init}}$ and every disturbance trajectories $(d_1, .., d_N)$ such that the state trajectories $(\xi_1, .., \xi_N)$ satisfy the reach-and-avoid requirement:

1. (Dynamics) $\forall \mathcal{A}_i \in \mathcal{A}$, $\xi_i(t) \equiv \xi_i(x_i^{\texttt{init}}, u_i, d_i, t)$;
2. (Reach goal set) $\exists t \geq 0$, $\forall \mathcal{A}_i \in \mathcal{A}$, $\xi_i(t) \in \mathcal{X}_i^{\texttt{goal}}$;
3. (Avoid obstacles) $\forall t \geq 0$, $\forall \mathcal{A}_i \in \mathcal{A}$, $\mathcal{P}_i(\xi_i(t)) \in \mathcal{W}$ and $\mathcal{P}_i(\xi_i(t)) \cap O = \emptyset$.
4. (Avoid inter-agent collisions) $\forall t \geq 0$, $\forall \mathcal{A}_i, \mathcal{A}_j \in \mathcal{A}$ and $i \neq j$, $\mathcal{P}_i(\xi_i(t)) \cap \mathcal{P}_j(\xi_j(t)) = \emptyset$

[1] A state is usually made up of positions, orientations, and velocities, while an input refers to the input of the controller, such as accelerations and steering rates.

In this work, we will solve the MAMP problem by finding a piecewise linear (PWL) path for each agent. The PWL paths for the agents will be sufficiently far away from the obstacles and from each other so that agents' tracking controllers can drive them to follow their PWL paths to reach their goals without collisions. Here we define PWL paths, tracking controllers, and reachability envelopes of each agent with a given tracking controller.

Definition 3 (Piecewise Linear Path)**.** A PWL path S_i in the workspace $\mathcal{W}$ for an agent $\mathcal{A}_i$ is a function $S_i : \mathbb{R}^{\geq 0} \to \mathcal{W}$ that maps a time point to a position $S_i(t) \in \mathcal{W}$, which can be constructed from a sequence of time-stamped waypoints $S_i = \texttt{Path}(\{(t_k, p_k)\}_k)$ such that $S_i(t) = p_{k-1} + \frac{p_k - p_{k-1}}{t_k - t_{k-1}}(t - t_{k-1})$ for $t \in [t_{k-1}, t_k]$. $(t_k, p_k) \in \mathbb{R}^{\geq 0} \times \mathcal{W}$ is called the k^{th} waypoint of path S_i, and $S_i(t)$ when $t \in [t_{k-1}, t_k]$ is called the k^{th} segment of S_i and denoted by $S_i^{(k)}$.

Definition 4 (Decentralized Tracking Controller)**.** A tracking controller for an agent $\mathcal{A}_i$ is a (state feedback) function $g_i : \mathcal{X}_i \times \mathcal{W} \to \mathcal{U}_i$. At any time t, a tracking controller takes in a current state of the system $x \in \mathcal{X}_i$ and a desired position $S_i(t) \in \mathcal{W}$, and gives an input $g_i(x, S_i(t)) \in \mathcal{U}_i$ for $\mathcal{A}_i$.

Fix a tracking controller g_i and a PWL path S_i for an agent $\mathcal{A}_i$, the resulting closed-loop controlled agent becomes an *autonomous system*. We use $\xi_i^{g_i}(x_0, S_i, d_i, t) = \xi_i(x_0, g_i(x_0, S_i(t)), d_i, t)$ to represent the trajectory of the controlled agent $\mathcal{A}_i$ starting from x_0 with disturbance d_i. The *reachablity envelope* of a controlled agent is a set of states around the PWL path that contains all possible actual trajectories of the controlled agent, defined as follows.

Definition 5 (Reachability Envelope)**.** Given an agent model $\mathcal{A}_i = \langle \mathcal{X}_i, \mathcal{U}_i, \mathcal{D}_i, f_i, \mathcal{P}_i \rangle$, an initial set $\mathcal{X}_i^{\texttt{init}} \subseteq \mathcal{X}_i$, a PWL path S_i, and a tracking controller g_i, the reachablity envelope at time t is $\texttt{Reach}_{\mathcal{A}_i}(\mathcal{X}_i^{\texttt{init}}, S_i, g_i, D_i, t) = \{\xi_i^{g_i}(x_0, S_i, d_i, t) \in \mathcal{X}_i \,|\, \exists x_0 \in \mathcal{X}_i^{\texttt{init}}, \exists d_i : \mathbb{R}^{\geq 0} \to \mathcal{D}_i\}$.

Example 2. Let a $S(t) = [p_x^*(t), p_y^*(t)]^T$ be a PWL path of waypoint sequence $\{(t_k, p_k)\}_k$. From (Rodríguez-Seda et al. 2014), a valid tracking trajectory for Example 1 can be constructed as

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta/L & \cos\theta/L \end{bmatrix} \begin{bmatrix} u_1' + v\omega\sin\theta + L\omega^2\cos\theta \\ u_2' - v\omega\cos\theta + L\omega^2\sin\theta \end{bmatrix},$$

where L is a positive constant and u_1', u_2' are computed as

$$\begin{bmatrix} u_1' \\ u_2' \end{bmatrix} = \begin{bmatrix} v^*\cos\theta^* - L\sin\theta^*\omega^* \\ v^*\sin\theta^* + L\cos\theta^*\omega^* \end{bmatrix} + G\frac{z^{2p-1}}{1 + \|z\|^{2p-1}},$$

where G is a positive constant, p is a positive integer, $z = \begin{bmatrix} (p_x - p_x^*) + L(\cos\theta - \cos\theta^*) \\ (p_y - p_y^*) + L(\sin\theta - \sin\theta^*) \end{bmatrix}$ and $\forall t \in [t_{k-1}, t_k]$, $v^*(t) = \frac{\|p_k - p_{k-1}\|}{t_k - t_{k-1}}$, $\theta^*(t) = \text{atan2}(p_k - p_{k-1})$. The dashed purple lines in Figure 1 illustrate such two trajectories of the closed-loop agents.

4 Approach

Figure 2 gives an overview of our approach S2M2, consisting of three key modules: (a). (Figure 2 left): Given a tracking controller g_i for each agent $\mathcal{A}_i$, pre-compute reachability

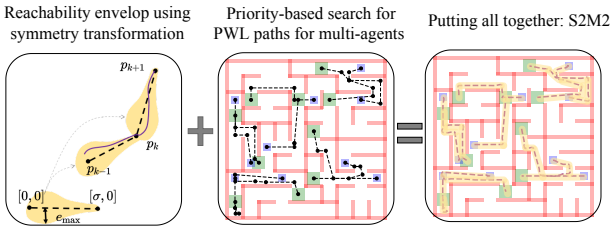


Figure 2: Schematic illustration of the approach.

envelopes for any PWL path S_i using symmetry transformations and cashed reachability envelopes, to get two key parameters: (1) an upper bound of the spatial tracking error between the actual trajectory $\xi_i^{g_i}(x_0, S_i, d_i, t) \downarrow \mathcal{W}$ and S_i, and (2) the minimum duration of each path segment $S_i^{(k)}$ such that the spatial tracking error bound is always valid (Section 4.1). (b). (Figure 2 middle): Given two parameters from (a), the safe motion planning problem for each agent is reduced to finding a PWL path that is sufficiently far from the obstacles and other agents, which is further encoded as a MILP problem (Section 4.2). (c). (Figure 2 middle): To coordinate multiple agents, employ priority-based search to avoid inter-robot collisions, in which some agents treat other agents as moving obstacles and replan their paths optimally (Section 4.3). Putting them all together (Figure 2 right), S2M2 finds a PWL path for each agent in the multi-agent system, so the closed-loop agents driven by tracking controllers can move along the PWL paths to safely achieve the reach-and-avoid requirement.

4.1 Computing Reachability Envelopes

Beyond the fact that a PWL path can be computed efficiently through solving a MILP problem (see Section 4.2), another key idea behind the use of PWL paths is that the reachability envelope can be pre-computed independent of the concrete values of the PWL paths' waypoints. In this way, we know in advance how far the actual trajectories of each agent are away from their reference PWL paths, which can be used as constraints in finding the PWL paths.

Various methods can be used to pre-compute the reachability envelope, including Contraction Metrics (Singh et al. 2017; Tsukamoto and Chung 2020), Lyapunov functions (Fan, Miller, and Mitra 2020), Funnels (Majumdar and Tedrake 2017), and Hamilton-Jacobi analysis (Herbert et al. 2017). In this section, we take an alternative approach and show that under mild assumptions, using symmetry transformation, we can construct the reachability envelope of each agent $\mathcal{A}_i$ following a PWL path $S_i = \texttt{Path}(\{(t_k, p_k)\}_k)$ from a finite number of reachability envelopes. These envelopes are of the agent with the same dynamics following a single straight line defined by only two waypoints $(0, [0, 0]^T)$ and $(T, [\sigma, 0]^T)$. As a result, from these envelopes around straight-line paths, we can obtain (1) an upper-bound of the distance between the actual trajectory projected to the workspace $\xi_i^{g_i}(t)(x_0, S_i, d_i, t) \downarrow \mathcal{W}$ and the reference PWL path S_i; and (2) the minimum duration bound for each path segment to guarantee such distance bound.

Symmetry transformations on dynamical systems are first introduced in (Russo and Slotine 2011) and defined as the ability to compute new trajectories (reachable states) of dynamical systems using existing trajectories (reachable states). Intuitively, the reachability envelope for each line segment are similar and can be constructed using translation and rotation transformations. Due to the space limit, we provide a detailed definition and steps to use symmetry transformations in the supplementary material.

For the rest of this section, we abuse the notation and use $\texttt{Reach}_{\mathcal{A}_i}(\mathcal{X}_{i,k}^{\texttt{init}}, S_i^{(k)}, g_i, D_i, t))$ to denote the reachability envelope of the agent following the path segment $S_i^{(k)}$ from (t_{k-1}, p_{k-1}) to (t_k, p_k), where trajectories start from initial set $\mathcal{X}_{i,k}^{\texttt{init}}$ at time t_{k-1}. The initial set is defined recursively as $\mathcal{X}_{i,k}^{\texttt{init}} = \texttt{Reach}_{\mathcal{A}_i}(\mathcal{X}_{i,(k-1)}^{\texttt{init}}, S_i^{(k-1)}, g_i, D_i, t_{k-1}))$.

Consider the ground vehicle model in Example 1 with the tracking controller in Example 2, we construct the following symmetry operator for this model, which encodes translation and rotation transformations:

$$\gamma([p_x, p_y, \theta, v, \omega]) = [p_x \cos\theta^* - p_y \sin\theta^* + p_{k-1,x}, \\ p_x \sin\theta^* + p_y \cos\theta^* + p_{k-1,y}, \theta + \theta^*, v, \omega],$$

where $\theta^* = \text{atan2}(p_k - p_{k-1})$. Using such γ, the reachability envelopes of the segment $S_i^{(k)}$ can be constructed from a reachability envelope of path $S_i^{x\text{-axis}}$, which moves and rotates $S_i^{(k)}$ as a line along the x-axis from the origin:

$$S_i^{x\text{-axis}} = \texttt{Path}\left(\left\{\left(0, \begin{bmatrix}0\\0\end{bmatrix}\right), \left(t_k - t_{k-1}, \begin{bmatrix}\|p_k - p_{k-1}\|\\0\end{bmatrix}\right)\right\}\right).$$

Then, the reachability envelope of $S_i^{(k)}$ is constructed as:

$$\texttt{Reach}_{\mathcal{A}_i}(\mathcal{X}_i^k, S_i^{(k)}, g_i, D_i, t)) = \\ \gamma\left(\texttt{Reach}_{\mathcal{A}_i}\left(\gamma^{-1}(\mathcal{X}_i^k), S_i^{x\text{-axis}}, g_i, D_i, t)\right)\right),$$

where for a set $\mathcal{Y} \subseteq \mathcal{X}$, $\gamma(\mathcal{Y}) = \{\gamma(x) \mid x \in \mathcal{Y}\}$ and $\gamma^{-1}(\mathcal{Y}) = \{y \mid \exists x \in \mathcal{Y}, \gamma(y) = x\}$.

Since the translation and rotation transformations preserve vector lengths, $\gamma\left(\texttt{Reach}(\cdot,\cdot,\cdot,\cdot,\cdot)\right)$ has the same size as $\texttt{Reach}(\cdot,\cdot,\cdot,\cdot,\cdot)$. Thus, as long as we pre-compute the envelope $\texttt{Reach}_{\mathcal{A}_i}(\mathcal{X}_i^{\texttt{init}}, S_i^{x\text{-axis}}, g_i, D_i, t))$ that follows a path $S_i^{x\text{-axis}} = \texttt{Path}(\{(0, [0, 0]^T), (T, [\sigma, 0]^T)\})$ for sufficiently long T and for all $\sigma \in [v_{\min}T, v_{\max}T]$, and also if $\gamma^{-1}(\mathcal{X}_i^k) \subseteq \mathcal{X}_i^{\texttt{init}}$ for all $k = 1, .., K$ (K is the number of path segments in S_i), we can always construct the reachability envelope of following PWL paths using the symmetry operator γ on the envelope following $S_i^{x\text{-axis}}$. Moreover, agents with the same dynamics and same tracking controller can also re-use the cached reachability envelope $\texttt{Reach}_{\mathcal{A}_i}(\mathcal{X}_i^{\texttt{init}}, S_i^{x\text{-axis}}, g_i, D_i, t))$. These envelopes can be computed using any nonlinear reachability toolbox, such as Flow* (Chen, Ábrahám, and Sankaranarayanan 2013), CORA (Althoff, Grebenyuk, and Kochdumper 2018), and DryVR (Fan et al. 2017). The construction of reachability envelope for aerial vehicles is similar.

To understand how far the reference path S_i needs to be away from the obstacles and each other, we define the maximum spatial tracking error $e_{i,\max}$ as follows:

$$e_{i,\max} = \text{argmin}_e \forall \sigma \in [v_{\min}T, v_{\max}T], \forall t \in [0,T],$$
$$B_e(S_i^{x\text{-axis}}(t)) \supseteq \texttt{Reach}_{\mathcal{A}_i}\left(\mathcal{X}_i^{\texttt{init}}, S_i^{x\text{-axis}}, g_i, D_i, t\right) \downarrow \mathcal{W},$$

where $S_i^{x\text{-axis}} = \texttt{Path}(\{(0,[0,0]^T),(T,[\sigma,0]^T)\})$. We implement this equation by first discretizing $[v_{\min}T, v_{\max}T]$ with Δ and calculating the error bound e for each reachability envelopes with length $\sigma_j = (v_{\min}T + j\Delta)$. Then, we choose the maximum value over all these errors as $e_{i,\max}$.

Notice is that to guarantee that, for any segment $S_i^{(k)}$, the trajectories can converge close enough to it before switching to the next line segment, the tracking controller needs to be applied sufficiently long. So we identify the minimum segment duration $T_{i,\min}$ such that $t_k - t_{k-1} > T_{i,\min}$ guarantees $\gamma^{-1}(\mathcal{X}_i^k) \subseteq \mathcal{X}_i^{\texttt{init}}$, for $k = 1,\ldots,K$. Similarly, for the last line segment $S_i^{(K)}$, we also find the minimum duration $T'_{i,\min} < t_K - t_{K-1}$ such that the final reachable states of the trajectory following path $S_i^{(K)}$ have enough time to be sufficiently small to fit into the goal set $\mathcal{X}_i^{\texttt{goal}}$.

4.2 Finding Paths for Single Agents

In this section, we describe the method for finding a PWL path for a single agent $\mathcal{A}_i$ with the presence of static obstacles and moving obstacles. To obtain a valid path, we solve a MILP problem. The decision variables of this MILP are $(p_0, p_1, .., p_K)$ with domain $\mathcal{W}$ and $(t_0, t_1, .., t_K)$ with domain $\mathbb{R}^{\geq 0}$, which represent the waypoint positions and their time stamps, respectively. The objective is to minimize the makespan t_K. We constrain the initial position to be at the center of the initial set $S_i^{\texttt{init}}$ and the initial time to be 0: $(p_0 = \texttt{Center}(S_i^{\texttt{init}} \downarrow \mathcal{W})) \wedge (t_0 = 0)$.

In the rest of this section, we introduce the other sets of the constraints that ensure the instantiated trajectory of the obtained path is valid with respect to the system dynamics, spatial tracking error $e_{i,\texttt{max}}$, minimum segment duration $T_{i,\texttt{min}}$, and minimum duration for the last segment $T'_{i,\texttt{min}}$.

Time-Position Constraints We first add constraints over the duration $(t_k - t_{k-1})$ and the spatial difference $(p_k - p_{k-1})$ for each segment to make sure its velocity $\frac{p_k - p_{k-1}}{t_k - t_{k-1}}$ respects the velocity bound. Let `vmin` and `vmax` be the minimum and maximum velocities, respectively. Then, the feasible velocity set is $B_{\texttt{vmax}}(0)/B_{\texttt{vmin}}(0)$. We further under-approximate $B_{\texttt{vmax}}(0)$ to polytope $\texttt{Poly}(H_{\texttt{vmax}}, b_{\texttt{vmax}})$, and over-approximate $B_{\texttt{vmin}}(0)$ to polytope $\texttt{Poly}(H_{\texttt{vmin}}, b_{\texttt{vmin}})$. The constraints to ensure each segment to satisfy such dynamic relations are:

$$(\vee_{j=1}^{\texttt{dP}(H_{\texttt{vmin}})} H_{\texttt{vmin}}^{(j)}(p_k - p_{k-1}) \geq b_{\texttt{vmin}}^{(j)}(t_k - t_{k-1}))$$
$$\wedge (H_{\texttt{vmax}}(p_k - p_{k-1}) \leq b_{\texttt{vmax}}(t_k - t_{k-1}))\ \forall k = 1,2,..,K$$

We handle disjunctive linear constraints $(\vee_{j=1}^{\texttt{dP}(H)} H^{(j)}x \leq b^{(j)}$ by using the big-M method. We define a $\texttt{dP}(H)$-vector of binary variables α, and $\alpha^{(j)} = 1$ if and only if $H^{(j)}x \leq b^{(j)}$ holds for x. Let M be a very large positive number, then the constraints is encoded as $(\wedge_{j=1}^{\texttt{dP}(H)} H^{(j)}x + M(1-\alpha^{(j)}) \leq b^{(j)}) \wedge (\sum_{j=1}^{\texttt{dP}(H)} \alpha^{(j)} \geq 1)$.

Reach-and-Avoid Constraints As the actual tracking trajectories deviate from the PWL paths due to inertia, actuation limits, disturbances, and uncertain initial position, we should consider this deviation when encoding constraints related to obstacles and goals. We have shown that the error between the actual trajectory and the PWL path can be bounded within $e_{i,\max}$ for each agent $\mathcal{A}_i$. In addition to the position deviation, we should consider the agent shape, and the swept area should not intersect with obstacles as well. Then, we know all the possible swept area at position p is $R = p \oplus B_l(0)$, where $l = e_{i,\max} + r_i$, and r_i is the radius of the ball containing the agent $\mathcal{A}_i$. For obstacle $o = \texttt{Poly}(H_o, b_o) \in O$, the bloated obstacle with respect to R is $\texttt{Poly}(H_o, b_o + \|H_o\|\, l)$. As long as the path is away from these bloated obstacles, the actual tracking trajectories are guaranteed to be collision-free.

To constrain the segments to be away from an obstacle, which is an polytope, we force the end points of every segment to be at least on one face of this polytope, which is a sufficient condition of being collision-free. The constraint is then as follows: $\forall k = 1,2,..,K, \forall o \in O$,

$$\vee_{j=1}^{\texttt{dP}(H_o)}((H_o^{(j)}p_{k-1} > b_o + \|H_o\|\, l) \wedge (H_o^{(j)}p_k > b_o + \|H_o\|\, l)).$$

For the moving obstacle $o \in O'$ defined by its occupied region $\texttt{Poly}(H_o, b_o)$ and the occupying duration $[lb_o, ub_o]$, we require the agent to either avoid colliding with o or moving through this region out of the duration $[lb_o, ub_o]$:

$$\vee_{j=1}^{\texttt{dP}(H_o)}((H_o p_{k-1} > b_o + \|H_o\|\, l) \wedge (H_o p_k > b_o + \|H_o\|\, l))$$
$$\vee (t_{k-1} < lb_o) \vee (t_k > ub_o),\ \forall k = 1,2,..,K, \forall o \in O'.$$

To make sure the spatial error are indeed bounded by $e_{i,\max}$, we add a constraint to force the nonzero-duration segment to be at least $T_{i,\texttt{min}}$ time:

$$(t_k - t_{k-1} = 0) \vee (t_k - t_{k-1} \geq T_{i,\texttt{min}})\ \forall k = 1,2,..,K.$$

We also require the agent to be at the goal set $S_i^{\texttt{goal}}$ at time t_K, and the last segment should be at least $T'_{i,\min}$ such that the agent has enough time to fit in: $(p_K = \texttt{Center}(S_i^{\texttt{goal}} \downarrow \mathcal{W})) \wedge (t_K - t_{K-1} \geq T'_{i,\texttt{min}})$.

4.3 Coordinating Multiple Agents

We deploy Priority-based Search (PBS) (Ma et al. 2019) to coordinate agents and avoid inter-agent collisions. PBS is an efficient two-level search algorithm designed for solving MAPF near-optimally. When it detects a collision between two agents, it constrains one of the agents to have a lower priority than the other and replans its path by treating the path of the higher-priority agent as dynamic obstacles. As we coordinate the agent trajectories over continuous timeline and continuous space, which is nontrivial or inefficient to summarizes collisions to conflicts, PBS is a more natural candidate to effectively coordinate agents than conflict-based algorithms such as CBS (Sharon et al. 2015).

Formally, we coordinate agents and resolve collisions by exploring a priority tree (PT). We start with the root PT node, that contains a set of individually optimal paths, not necessarily collision-free, and an empty priority ordering. We explore the PT in a depth-first manner, breaking ties by in favor of the node with smaller flowtime. During expansion, we identify the pair of agents $\mathcal{A}_i$ and $\mathcal{A}_j$ with the earliest collision and generate two child PT nodes that inherit the priority ordering of the current PT node plus an additional ordered pairs $(j \prec i)$ and $(i \prec j)$, respectively. For each child PT node, we pick the lower-priority agent, i.e., $\mathcal{A}_i$ or $\mathcal{A}_j$, and replans an individually optimal path for it by treating all agents that have higher priorities than it as moving obstacles. This procedure is terminated when we find a PT node whose paths are collision-free.

Compared to the original PBS in (Ma et al. 2019), we make two modifications: (1) the single-agent planner in S2M2 uses our MILP encoding since our sub-problem is to find a sequence of time-stamped waypoints in a continuous map over continuous timeline rather than a graph over discrete time steps; (2) when a new priority is added to resolve a collision, we lazily update the paths by replanning for only the lower-priority agent involved in this collision instead of all the lower-priority agents that have collisions, which shows better scalability in our practical experiments.

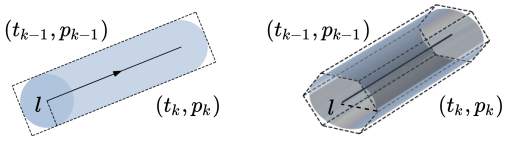


Figure 3: Examples of moving obstacles in 2D and 3D workspaces. All the possible swept area is in blue, and is over approximated to polytopes as shown with dashed line boundaries.

During replanning, some agents are treated as moving obstacles for other agents. Here we introduce our method for encoding all the possible swept area of a path segment as a moving obstacle. Consider a segment from (t_{k-1}, p_{k-1}) to (t_k, p_k). As we know all the swept area at position p is $R = p \oplus B_l(0)$, we can calculate its moving obstacle as $(t_{k-1}, t_k, \texttt{Poly}(H_k, b_k))$, where $\texttt{Poly}(H_k, b_k)$ is a polygon containing all the possible swept area during duration (t_{k-1}, t_k). Thus, if other agents do not swept $\texttt{Poly}(H_k, b_k)$ during (t_{k-1}, t_k), their paths are guaranteed to be collision-free. The central axis of this moving obstacle is in the direction atan2$(p_k - p_{k-1})$ with length $2l + \|p_k - p_{k-1}\|$. In a 2D workspace, the cross section of this tube is a line that is vertical to the central axis, and its width is $2l$. In a 3D workspace, the cross section is a circle with radius l. We further over approximate this circle as a polygon such as a octagon. Figure 3 shows examples in 2D and 3D workspace.

5 Experimental Results

We present experimental results on both 2D and 3D environments with ground vehicles and quadrotor models, respectively. We used DryVR (Fan et al. 2017) to generate reachability envelopes and Gurobi 9.0.1 (Gurobi Optimization 2020) as the MILP solver. We compared S2M2 with the state-of-the-art 2D planner ECBS-CT (Cohen et al. 2019) and 3D planner MAPF/C+POST (Hönig et al. 2018). All experiments were run on a 3.40GHZ Intel Core i7-6700 CPU with 36GB RAM with a runtime limit of 100s. We repeated each experiment 25 times for each agent number using different randomly generated initial and goal locations for the agents. We report the average runtime, success rates, and flowtime for each agent number on each map. We also provide additional experiments and videos in the supplementary materials.

(a) Arena. (b) Den502d. (c) Wall.

Figure 4: Maps and scenarios.

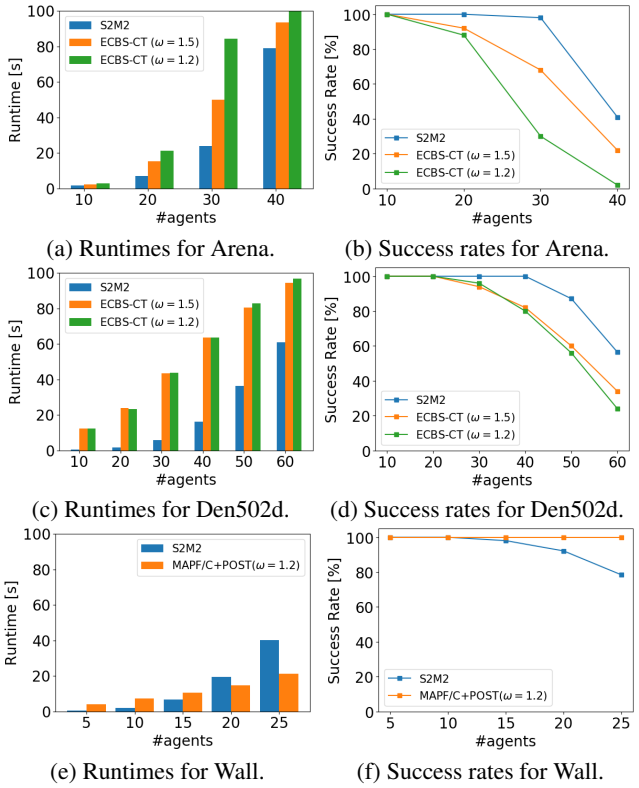
2D Experiment We compare S2M2 against ECBS-CT on two benchmark maps from the Grid-Based Path Planning Competition (GPPC)[2], namely Arena (49×49) and Den502d (251×211). We assume a disc-shaped agent with a radius 0.5. While agent dynamics for ECBS-CT is approximated with 16 discrete orientations and 5 primitives, which is taken from the Search-based Planning Laboratory (SBCL)[3], S2M2 considers a vehicle model with continuous, nonholonomic, nonlinear dynamics from (Rodríguez-Seda et al. 2014) with additional bounded disturbances. Both the cost multiplier for the motion primitive model and the maximum velocity for our model are set to 1. We set the focal weight ω (i.e., suboptimality ratio) for ECBS-CT to 1.2 and 1.5. The average runtime and success rate of our method and ECBS-CT on these two maps are given in Figure 5(a)-(d). The corresponding solution qualities are given in Table 1.

First of all, we observe that S2M2 is several magnitudes faster than ECBS-CT in terms of the pre-processing time. While S2M2 takes $0.33s$ to pre-compute the spatial error bound and the minimum duration for all agents, the time for ECBS-CT to pre-compute the perfect heuristic is $0.27s$ on Arena and $18.18s$ on Den502d for each agent, which make the total pre-processing time for ECBS-CT very large. For instance, for each instance on Den502d with 60 agents, ECBS-CT spends roughly $1,100s$ to compute these heuristics. This makes S2M2 a much better candidate to real-time navigate large agent teams.

Furthermore, S2M2 outperforms ECBS-CT in terms of both runtime and success rates, as shown in Figure 5(a)-(d). While S2M2 halves the runtime of ECBS-CT on Arena for most instances when $\omega = 1.5$, it is roughly one-third of that when $\omega = 1.2$ (Figure 5(a)). In Figure 5(c), the runtime of ECBS-CT does not change much with different weights on Den502d. While S2M2 takes half the time than ECBS-CT for most Den502d instances, S2M2 is one magnitude faster when the agent number is less 40. As shown in Figure 5(b)(d), the success rates of ECBS-CT drop much sharper than S2M2. When ECBS-CT fails half the instances,

[2]GPPC: https://movingai.com/GPPC

[3]SBCL: http://sbpl.net



(a) Runtimes for Arena. (b) Success rates for Arena.

(c) Runtimes for Den502d. (d) Success rates for Den502d.

(e) Runtimes for Wall. (f) Success rates for Wall.

Figure 5: Runtimes and success rates.

S2M2 still solves more than 90% of them. We do not include the case when $\omega = 1$ (i.e., search for optimal solutions) since ECBS-CT merely solves instances even for 10 agents. We also test with larger focal weights, but it does not show significant improvement over runtimes or success rates.

In Table 1, we can see S2M2 reduces at least half flowtime for Arena instances, and this reduction is up to 70% as the agent number increases. This is because S2M2 directly plans high-fidelity models on a continuous map over continuous timeline, which provides more flexibility in avoiding collisions, especially in smaller maps. On the larger map Den502d, we can still observe roughly 15% cost reduction.

3D Experiment We use map Wall ($13 \times 13 \times 5$) from (Hönig et al. 2018). In this scenario, a nano-quadrotor team (Preiss et al. 2017) starts from one side of the wall with three windows and is asked to fly to the other side. We compare S2M2 with MAPF/C+POST, which performs a generalized-MAPF algorithm called MAPF/C on a graph with sparse samples and then post-processes the discrete solution to valid continuous trajectories. We use the same parameters for sampling and post-processing as the original paper. The focal weight of MAPF/C is chosen to be 1.2. In post-processing, we set the total iterations to 7 and continuity degree to 4. The average runtime and success rate of our method and MAMP/C+POST are given in Figure 5(e)-(f), while the solution qualities are given in Table 1.

While it takes only $0.83s$ for S2M2 to pre-compute the error bound and minimum duration, the time to generate

	#agents	S2M2	ECBS-CT ($\omega = 1.2$)	ECBS-CT ($\omega = 1.5$)
Arena	10	382.02	867.83	868.00
	20	741.20	1848.07	1948.07
	30	1062.60	2929.95	3147.36
	40	1366.23	NA	4554.60
Den502d	10	1292.01	1566.78	1567.87
	20	2569.11	3135.57	3134.96
	30	3962.72	4621.11	4620.03
	40	5517.90	6292.29	6202.304
	50	7289.37	7798.79	7821.35
	60	8681.73	9452.26	9480.53

	#agents	S2M2	MAPF/C	MAPF/C+POST
Wall	5	48.00	50.32	80.35
	10	102.77	103.08	142.27
	15	162.37	152.38	200.36
	20	230.75	200.67	230.00
	25	299.29	265.09	315.20

Table 1: Solution quality (i.e., average flowtime) for Arena and Den502d (above) and Wall (below).

roadmaps for MAPF/C by using SPARS is $668.94s$ and annotating conflicts takes $902.00s$, which leads to $1570.94s$ in total. This pre-processing time takes such long because the sampling and conflict-annotating procedures, which is critical to the efficiency and solution qualities of MAPF/C, requires computationally expensive distance checking on an exponentially increasing number of edges. Thus, MAPF/C is sensitive to the map size and only scales to small maps. We also tested the SPARS sampling module with the 3D Arena map ($49 \times 49 \times 5$), in which the obstacle height is 5. We failed to get a reasonably connected roadmap in hours.

Although MAPF/C is efficient on sparse, well-connected roadmaps, S2M2 can still solve most instances faster when #agents $<$ 20. The runtime is less than 1s for instances with 10 agents, which makes S2M2 a better candidate to coordinate agents in real-time. In Table 1, we also observe that S2M2 has up to 40% cost reduction compared to MAPF/C+POST trajectories. Even though the discrete MAPF/C solution is bounded suboptimal, the quality of its valid continuous trajectory is not guaranteed.

6 Conclusions

We presented S2M2, a fast and effective multi-agent motion planner that generates provably safe plans for agent models with high-dimensional, nonlinear dynamics and bounded disturbances. S2M2 plans piecewise linear paths for each agent that satisfies certain safe bounds and coordinates multiple agents using priority-based search. We show that S2M2 improves both the solving time and the solution quality compared to two state-of-the-art multi-agent motion planners ECBS-CT and MAPF/C+POST. Especially, S2M2 saves much time on pre-processing either for computing heuristics or sampling roadmaps compared to the existing methods. In the future, we intend to explore the search-based method, such as any-angle pathfinding to improve the efficiency of our single-agent motion planner.

References

Agrawal, A.; and Sreenath, K. 2017. Discrete Control Barrier Functions for Safety-Critical Control of Discrete Systems with Application to Bipedal Robot Navigation. In *Robotics: Science and Systems*.

Althoff, M.; Grebenyuk, D.; and Kochdumper, N. 2018. Implementation of Taylor Models in CORA 2018. In *Proc. of the 5th International Workshop on Applied Verification for Continuous and Hybrid Systems*, 145–173. doi:10.29007/zzc7.

Andreychuk, A.; Yakovlev, K. S.; Atzmon, D.; and Stern, R. 2019. Multi-Agent Pathfinding with Continuous Time. In Kraus, S., ed., *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, 39–45. ijcai.org. doi:10.24963/ijcai.2019/6. URL https://doi.org/10.24963/ijcai.2019/6.

Atzmon, D.; Stern, R.; Felner, A.; Wagner, G.; Barták, R.; and Zhou, N. 2020. Robust Multi-Agent Path Finding and Executing. *J. Artif. Intell. Res.* 67: 549–579.

Augugliaro, F.; Schoellig, A. P.; and D'Andrea, R. 2012. Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach. In *2012 IEEE/RSJ international conference on Intelligent Robots and Systems*, 1917–1922. IEEE.

Barry, A. J.; Majumdar, A.; and Tedrake, R. 2012. Safety verification of reactive controllers for UAV flight in cluttered environments using barrier certificates. In *2012 IEEE International Conference on Robotics and Automation*, 484–490. IEEE.

Blackmore, L.; Ono, M.; and Williams, B. C. 2011. Chance-constrained optimal path planning with obstacles. *IEEE Transactions on Robotics* 27(6): 1080–1094.

Čáp, M.; Vokřínek, J.; and Kleiner, A. 2015. Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures. In *Proceedings of the Twenty-Fifth International Conference on International Conference on Automated Planning and Scheduling*, 324–332.

Chen, X.; Ábrahám, E.; and Sankaranarayanan, S. 2013. Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer Aided Verification*, 258–263. Springer.

Cohen, L.; Uras, T.; Kumar, T. S.; and Koenig, S. 2019. Optimal and bounded-suboptimal multi-agent motion planning. In *Twelfth Annual Symposium on Combinatorial Search*.

Desai, A.; Ghosh, S.; Seshia, S. A.; Shankar, N.; and Tiwari, A. 2019. SOTER: A Runtime Assurance Framework for Programming Safe Robotics Systems. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 138–150. IEEE.

Fan, C.; Miller, K.; and Mitra, S. 2020. Fast and Guaranteed Safe Controller Synthesis for Nonlinear Vehicle Models. In *International Conference on Computer Aided Verification*, 629–652. Springer.

Fan, C.; Qi, B.; Mitra, S.; and Viswanathan, M. 2017. D ry VR: data-driven verification and compositional reasoning for automotive systems. In *International Conference on Computer Aided Verification*, 441–461. Springer.

Felner, A.; Stern, R.; Shimony, S. E.; Boyarski, E.; Goldenberg, M.; Sharon, G.; Sturtevant, N.; Wagner, G.; and Surynek, P. 2017. Search-based optimal solvers for the multi-agent pathfinding problem: Summary and challenges. In *Tenth Annual Symposium on Combinatorial Search*.

Gurobi Optimization, I. 2020. Gurobi optimizer reference manual. *URL http://www. gurobi. com* .

Herbert, S. L.; Chen, M.; Han, S.; Bansal, S.; Fisac, J. F.; and Tomlin, C. J. 2017. FaSTrack: A modular framework for fast and guaranteed safe motion planning. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, 1517–1522. IEEE.

Hönig, W.; Preiss, J. A.; Kumar, T. S.; Sukhatme, G. S.; and Ayanian, N. 2018. Trajectory planning for quadrotor swarms. *IEEE Transactions on Robotics* 34(4): 856–869.

Huang, A.; Ayton, B. J.; and Williams, B. C. 2018. RADMPC: A Fast Decentralized Approach for Chance-Constrained Multi-Vehicle Path-Planning. *CoRR* abs/1811.09914. URL http://arxiv.org/abs/1811.09914.

Jasour, A. M.; and Williams, B. C. 2019. Sequential Chance Optimization For Flow-Tube Based Control Of Probabilistic Nonlinear Systems. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, 5392–5399. IEEE.

Le, D.; and Plaku, E. 2018. Cooperative, dynamics-based, and abstraction-guided multi-robot motion planning. *Journal of Artificial Intelligence Research* 63: 361–390.

Li, J.; Surynek, P.; Felner, A.; Ma, H.; Kumar, T. S.; and Koenig, S. 2019. Multi-agent path finding for large agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 7627–7634.

Liu, Z.; Chen, B.; Zhou, H.; Koushik, G.; Hebert, M.; and Zhao, D. 2020. MAPPER: Multi-Agent Path Planning with Evolutionary Reinforcement Learning in Mixed Dynamic Environments. *arXiv preprint arXiv:2007.15724* .

Ma, H.; Harabor, D.; Stuckey, P. J.; Li, J.; and Koenig, S. 2019. Searching with consistent prioritization for multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 7643–7650.

Ma, H.; Koenig, S.; Ayanian, N.; Cohen, L.; Hönig, W.; Kumar, T. K. S.; Uras, T.; Xu, H.; Tovey, C.; and Sharon, G. 2016. Overview: Generalizations of Multi-Agent Path Finding to Real-World Scenarios. In *IJCAI-16 Workshop on Multi-Agent Path Finding*.

Ma, H.; Kumar, T. K. S.; and Koenig, S. 2017. Multi-Agent Path Finding with Delay Probabilities. In *AAAI Conference on Artificial Intelligence*, 3605–3612.

Majumdar, A.; and Tedrake, R. 2017. Funnel libraries for real-time robust feedback motion planning. *The International Journal of Robotics Research* 36(8): 947–982.

Mellinger, D.; Kushleyev, A.; and Kumar, V. 2012. Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams. In *2012 IEEE international conference on robotics and automation*, 477–483. IEEE.

Panagou, D.; Stipanovič, D. M.; and Voulgaris, P. G. 2013. Multi-objective control for multi-agent systems using Lyapunov-like barrier functions. In *52nd IEEE Conference on Decision and Control*, 1478–1483. IEEE.

Preiss, J. A.; Honig, W.; Sukhatme, G. S.; and Ayanian, N. 2017. Crazyswarm: A large nano-quadcopter swarm. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 3299–3304. IEEE.

Richards, A.; and How, J. 2004. A decentralized algorithm for robust constrained model predictive control. In *Proceedings of the 2004 American control conference*, volume 5, 4261–4266. IEEE.

Rodríguez-Seda, E. J.; Tang, C.; Spong, M. W.; and Stipanović, D. M. 2014. Trajectory tracking with collision avoidance for nonholonomic vehicles with acceleration constraints and limited sensing. *The International Journal of Robotics Research* 33(12): 1569–1592.

Russo, G.; and Slotine, J.-J. E. 2011. Symmetries, stability, and control in nonlinear systems and networks. *Physical Review E* 84(4): 041929.

Semnani, S. H.; Liu, H.; Everett, M.; de Ruiter, A.; and How, J. P. 2020. Multi-Agent Motion Planning for Dense and Dynamic Environments via Deep Reinforcement Learning. *IEEE Robotics and Automation Letters* 5(2): 3221–3226.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219: 40–66.

Singh, S.; Majumdar, A.; Slotine, J.-J.; and Pavone, M. 2017. Robust online motion planning via contraction theory and convex optimization. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 5883–5890. IEEE.

Srinivasan, M.; Coogan, S.; and Egerstedt, M. 2018. Control of multi-agent systems with finite time control barrier certificates and temporal logic. In *2018 IEEE Conference on Decision and Control (CDC)*, 1991–1996. IEEE.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. K. S.; Barták, R.; and Boyarski, E. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *Proceedings of the 12th International Symposium on Combinatorial Search (SoCS)*, 151–159.

Tedrake, R. 2009. LQR-Trees: Feedback motion planning on sparse randomized trees .

Tsukamoto, H.; and Chung, S.-J. 2020. Neural Contraction Metrics for Robust Estimation and Control: A Convex Optimization Approach. *arXiv preprint arXiv:2006.04361* .

Vaskov, S.; Kousik, S.; Larson, H.; Bu, F.; Ward, J.; Worrall, S.; Johnson-Roberson, M.; and Vasudevan, R. 2019. Towards provably not-at-fault control of autonomous robots in arbitrary dynamic environments. *arXiv preprint arXiv:1902.02851* .

Velagapudi, P.; Sycara, K.; and Scerri, P. 2010. Decentralized prioritized planning in large multirobot teams. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 4603–4609. IEEE.

Wagner, G.; and Choset, H. 2017. Path planning for multiple agents under uncertainty. In *Twenty-Seventh International Conference on Automated Planning and Scheduling*.

Walker, T. T.; Sturtevant, N. R.; and Felner, A. 2018. Extended Increasing Cost Tree Search for Non-Unit Cost Domains. In *IJCAI*, 534–540.

Wang, L.; Ames, A.; and Egerstedt, M. 2016. Safety barrier certificates for heterogeneous multi-robot systems. In *2016 American Control Conference (ACC)*, 5213–5218. IEEE.

Yakovlev, K. S.; and Andreychuk, A. 2017. Any-Angle Pathfinding for Multiple Agents Based on SIPP Algorithm. In Barbulescu, L.; Frank, J.; Mausam; and Smith, S. F., eds., *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017, Pittsburgh, Pennsylvania, USA, June 18-23, 2017*, 586. AAAI Press. URL https://aaai.org/ocs/index.php/ICAPS/ICAPS17/paper/view/15652.

Yu, S.; Maier, C.; Chen, H.; and Allgöwer, F. 2013. Tube MPC scheme based on robust control invariant set with application to Lipschitz nonlinear systems. *Systems & Control Letters* 62(2): 194–200.