RLSS: Real-time Multi-Robot Trajectory Replanning using Linear Spatial Separations

Baskın Şenbaşlar, Wolfgang Hönig, and Nora Ayanian

***Abstract*— Trajectory replanning is a critical problem for multi-robot teams navigating dynamic environments. We present RLSS (Replanning using Linear Spatial Separations): a real-time trajectory replanning algorithm for cooperative multi-robot teams that uses linear spatial separations to enforce safety. Our algorithm handles the dynamic limits of the robots explicitly, is completely distributed, and is robust to environment changes, robot failures, and trajectory tracking errors. It requires no communication between robots and relies instead on local relative measurements only. We demonstrate that the algorithm works in real-time both in simulations and in experiments using physical robots. We compare our algorithm to a state-of-the-art online trajectory generation algorithm based on model predictive control, and show that our algorithm results in significantly fewer collisions in highly constrained environments, and effectively avoids deadlocks.**

I. Introduction

Effective collaboration of multiple robots is key to emerging industries such as warehouse automation [2], autonomous driving [3], and automated intersection management [4]. One of the core robotic challenges in such domains is multi-robot trajectory planning, which entails navigating a team of robots in environments with dynamic obstacles. In practice, robots must operate safely in dynamic environments even if there is only partial observability and limited communication available. While there is a large body of research on multi-robot trajectory planning, none of the existing solutions are practical for dynamic environments with high robot- and obstacle-density, which makes the problem particularly challenging.

Existing algorithms employ centralized [5]–[8] or distributed strategies [9]–[13] to address the multi-robot trajectory planning problem. Centralized algorithms can provide theoretical guarantees, but they cannot react to changes in the environment in real-time and they require a reliable communication link to all robots. Robots using them do not deadlock, and follow trajectories successfully so long as the underlying assumptions hold. Distributed algorithms delegate the computation of trajectories: each robot plans for itself and reacts to changes in the environment in real-time. We propose

Baskın Şenbaşlar and Nora Ayanian are with the Department of Computer Science, University of Southern California, Los Angeles, CA, USA.
Email: {baskin.senbaslar, ayanian}@usc.edu
Wolfgang Hönig is with the Department of Aerospace, California Institute of Technology, Pasadena, CA, USA. Part of this work was done while he was at the Department of Computer Science, University of Southern California, Los Angeles, CA, USA. Email: whoenig@caltech.edu
This paper is a short version of our work submitted to IEEE Transactions on Robotics [1].
This work was supported by NSF awards IIS-1724399, IIS-1724392, and CPS-1837779. B. Şenbaşlar was supported by a USC Annenberg Fellowship.

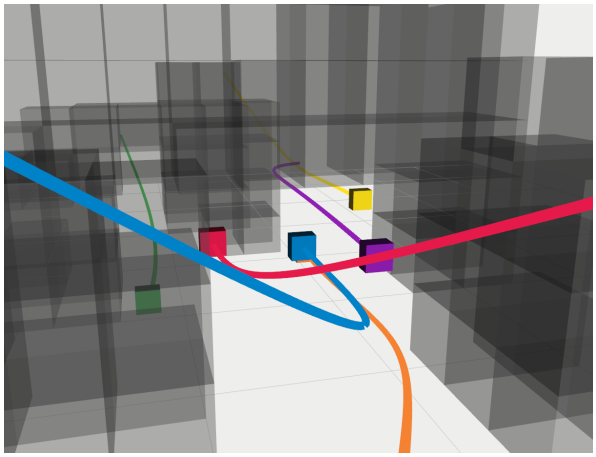


Fig. 1. RLSS calculates trajectories in real-time where robots move in close proximity to each other. Computations are done in a distributed way that each robot plans for its own trajectory.

an algorithm that employs real-time trajectory optimization using receding horizon planning that is fully distributed while exhibiting some advantages typical of centralized solutions that existing distributed algorithms do not have, yet avoiding the shortcomings of existing distributed algorithms that make them impractical. To our knowledge, compared to other state-of-the-art distributed multi-robot trajectory planning algorithms [9]–[13], our approach is the only one that provides the following simultaneously. It i) explicitly handles the dynamic limits of the robots; ii) is completely distributed with no reliance on a central computer; iii) is robust to environmental changes, agent failures, and poorly performing controllers; iv) requires no communication between robots; v) does not deadlock; vi) enforces hard safety constraints and reports failure when the constraints are infeasible, thus guaranteeing collision avoidance for computed trajectories; and vii) works in the presence of obstacles. Our approach uses a combination of discrete planning and trajectory optimization presented as a convex quadratic program that can be solved in real-time.

We utilize linear spatial separations for collision avoidance during the optimization step, hence we call our algorithm RLSS (Replanning using Linear Spatial Separations). RLSS can replan trajectories for robots in dense environments (Fig. 1) where robots have to move in close proximity to each other and navigate through an environment with hundreds of obstacles. Each robot plans a trajectory for itself without any reliance on the trajectories of other robots.

We show using simulations that RLSS works in real-

time in highly constrained environments. We compare our algorithm to a state-of-the-art method based on distributed model predictive control (DMPC) [13]. DMPC requires the approximation of the robot's own controller behaviour under given desired states as a linear system, and uses it during planning. Our method results in significantly fewer collisions and no deadlocks compared to DMPC. Furthermore, our algorithm does not require communication to ensure safety unlike the baseline algorithm. We demonstrate our algorithm's robustness to external disturbances in physical robot experiments using a heterogeneous differential drive robot team and a homogeneous quadrotor team.

II. Problem Statement

Consider a —possibly heterogeneous—team of N robots. We assume the robots are differentially flat [14], i.e., the robots' states and inputs can be expressed in terms of output trajectories and a finite number of derivatives thereof. Differential flatness is common for many kinds of mobile robots, including differential drive robots [15], car-like robots [16], omnidirectional robots [17], and quadrotors [18].

To provide the collision avoidance guarantee for the computed trajectories, we assume that robots are able to sense the positions of other robots and obstacles around them perfectly. We assume that obstacles have convex shapes[1]. Many existing, efficient, and widely-used mapping tools, including occupancy grids [19] and octomaps [20], internally store obstacles as convex shapes; such maps can be updated in real-time using visual or RGBD sensors, and use unions of convex axis aligned boxes to approximate the obstacles in the environment. We use $\mathcal{O}(t)$ to denote the set of obstacles in the environment at time t.

Optionally, we permit that each robot i has a desired trajectory $\mathbf{o}_i(t) : [0, T_i] \rightarrow \mathbb{R}^d$, $d \in \{2, 3\}$ that it should follow. We do not require that this trajectory is collision-free or even feasible to track by the robot. If no such desired trajectory is known, it can be initialized, for example, with a straight line to the goal location.

Our goal is that each robot i computes and tracks a Euclidean trajectory $\mathbf{f}_i(t) : [0, T_i] \rightarrow \mathbb{R}^d$ such that $\mathbf{f}_i(t)$ is collision-free, executable according to the robot's dynamics, and is as close as possible to the desired trajectory $\mathbf{o}_i(t)$. Since our robots are differentially-flat, we can account for their dynamics by i) imposing C^c continuity on the trajectories for any given c, and ii) imposing constraints on the maximum k^{th} derivative magnitude of trajectories $\mathbf{f}_i(t)$ for any desired k.

Let $\mathcal{S}_{R_i}(\mathbf{p})$ be the convex collision shape of robot i located at position $\mathbf{p} \in \mathbb{R}^d$, i.e. the convex region occupied by robot i located at position $\mathbf{p}$. Let $\mathcal{W}_i \subseteq \mathbb{R}^d$ be the convex workspace that robot i should stay inside. It can be set to a bounding box that defines a room that a ground robot must not leave, a half-space that contains vectors with positive z coordinates so that a quadrotor does not hit the ground or be simply set to $\mathbb{R}^d$. Formally, each robot must solve the following optimization problem:

$$
\begin{aligned}
\min \int_0^{T_i} \|\mathbf{f}_i(t) - \mathbf{o}_i(t)\|_2^2 \, dt \text{ s.t.} \\
\mathbf{f}_i(t) &\in C^{c_i} \\
\frac{d^c \mathbf{f}_i(0)}{dt^c} &= \frac{d^c \mathbf{p}_i^0}{dt^c} && \forall c \in \{0, \ldots, c_i\} \\
\mathcal{S}_{R_i}(\mathbf{f}_i(t)) \cap \left(\cup_{\mathcal{Q} \in \mathcal{O}(t)} \mathcal{Q}\right) &= \emptyset && \forall t \in [0, T_i] \\
\mathcal{S}_{R_i}(\mathbf{f}_i(t)) \cap \mathcal{S}_{R_j}(\mathbf{f}_j(t)) &= \emptyset && \forall j \neq i \; \forall t \in [0, T_i] \\
\mathcal{S}_{R_i}(\mathbf{f}_i(t)) &\in \mathcal{W}_i && \forall t \in [0, T_i] \\
\max_{t \in [0, T_i]} \left\| \frac{d^k \mathbf{f}_i(t)}{dt^k} \right\|_2 &\leq \gamma_i^k && \forall k \in \{1, \ldots, K_i\}
\end{aligned}
\tag{1}
$$

where $\mathbf{p}_i^0$ is the initial position of robot i; γ_i^k is the maximum k^{th} derivative magnitude that the i^{th} robot can execute; K_i is the maximum derivative degree that robot i has a derivative magnitude limit on; and c_i is the order of derivative up to which the trajectory of the i^{th} robot must be continuous.

The cost of the optimization problem denotes the deviation from the desired trajectory; it is the sum of position differences between the planned and the desired trajectories.

III. Preliminaries

Here, we introduce essential mathematical concepts we use.

A. Parametric Curves and Splines

We use curves $\mathbf{f} : [0, T] \rightarrow \mathbb{R}^d$ that are parametrized by time, where T is the duration of the curve, as trajectories. Mathematically, we adopt splines, i.e. piecewise polynomials, where each piece is a Bézier curve defined by a set of control points and a duration.

A Bézier curve $\mathbf{f} : [0, T] \rightarrow \mathbb{R}^d$ of degree h is defined by $h + 1$ control points $\mathbf{P}_0, \ldots, \mathbf{P}_h \in \mathbb{R}^d$ as follows:

$$
\mathbf{f}(t) = \sum_{i=0}^{h} \mathbf{P}_i \binom{h}{i} \left(\frac{t}{T}\right)^i \left(1 - \frac{t}{T}\right)^{(h-i)}. \tag{2}
$$

Since any Bézier curve $\mathbf{f}$ is a polynomial of degree h, it is smooth, meaning $\mathbf{f} \in C^\infty$.

We focus on Bézier curves as pieces because of their *convex hull property*: the curves themselves lie inside the convex hull of their control points, i.e., $\mathbf{f}(t) \in ConvexHull\{\mathbf{P}_0, \ldots, \mathbf{P}_h\} \; \forall t \in [0, T]$ [21]. Using the convex hull property, we can constrain a curve to be inside a convex region by constraining its control points to be inside the same convex region.

B. Linear Spatial Separations: Half-spaces, Convex Polytopes, and Support Vector Machines

A hyperplane $\mathcal{H}$ in $\mathbb{R}^d$ can be formulated by a normal vector $\mathbf{n}$ and an offset a as $\mathcal{H} = \{\mathbf{x} \in \mathbb{R}^d \mid \mathbf{n}^\top \mathbf{x} + a = 0\}$. A half-space $\tilde{\mathcal{H}}$ in $\mathbb{R}^d$ is a subset of $\mathbb{R}^d$ that is bounded by a hyperplane such that $\tilde{\mathcal{H}} = \{\mathbf{x} \in \mathbb{R}^d \mid \mathbf{n}^\top \mathbf{x} + a \leq 0\}$.

[1]For the purposes of our algorithm, concave obstacles can be described or approximated by a union of finite number of convex shapes provided that the union contains the original obstacle. Using our algorithm with approximations of concave obstacles results in trajectories that avoid the approximations.

A convex polytope is the intersection of a finite number of half-spaces.

Our approach relies heavily on computing safe convex polytopes and constraining spline pieces to be inside these polytopes. Specifically, we compute support vector machine (SVM) [22] hyperplanes between curve pieces and the obstacles/robots in the environment, and use these hyperplanes to create safe convex polytopes for robots to navigate in.

IV. Approach

To solve (1) in one shot, it is required for the robot to know the trajectories $\mathbf{f}_j(t)$ of the other robots and the future obstacle positions $\mathcal{O}(t)$ ahead of time. While the first requirement could be realized by some form of inter-robot communication, the second requirement is unrealistic for dynamic environments. Therefore, instead of solving this problem exactly in one shot, we solve it iteratively by replanning at each timestep.

Each robot $i \in \{1, \ldots, N\}$ replans at every timestep u to calculate a piecewise trajectory $\mathbf{f}_i^u(t)$ that is safe for duration δt, where each piece is a Bézier curve. Then, it executes $\mathbf{f}_i^u(t)$ for δt and replans. Henceforth, we will use superscript u to denote an object at timestep u and subscript i to denote an object used or computed by robot i.

RLSS fits into the planning part of the classical robotics pipeline using perception, planning, and control. The inputs from the perception for each robot i at each timestep u are:

- $\mathcal{R}_i^u$: Convex collision shapes of other robots[2]. $\mathcal{R}_{i,j}^u \in \mathcal{R}_i^u$ where $j \in \{1, \ldots, i-1, i+1, \ldots, N\}$ is the convex collision shape of robot j sensed by robot i at timestep u.
- $\mathcal{O}_i^u$: The set of convex obstacles robot i sensed up to timestep u.
- $\mathbf{p}_i^u$: Position of robot i at timestep u estimated by itself.

There are 4 main stages of RLSS: 1) goal selection, 2) discrete search, 3) trajectory optimization, and 4) validity check. The replanning pipeline is summarized in Fig. 2. At each timestep u, each robot i executes the four stages independently of the other robots.

In the goal selection stage, a goal position $\mathbf{g}_i^u$ on the robot's desired trajectory $\mathbf{o}_i(t)$ is selected. Given the desired time horizon τ_i, which denotes the desired duration of the resulting trajectory, we adjust τ_i to the actual time horizon τ_i^u so that the position $\mathbf{g}_i^u = \mathbf{o}_i(u\delta t + \tau_i^u)$ is a reachable position for the robot without considering kinematics. The computed goal position $\mathbf{g}_i^u$ of robot i at timestep u and the actual time horizon τ_i^u of robot i at timestep u is given as input to the discrete search. Details of the goal selection stage are presented in our journal submission.

During the discrete search stage, we compute a discrete path with no collisions within the workspace $\mathcal{W}_i$ from the robot's current position $\mathbf{p}_i^u$ towards its goal position $\mathbf{g}_i^u$.

[2]Practically, if robot i cannot sense a particular robot j at timestep u because it is not within the sensing range of i, robot j can be omitted by robot i. As long as the sensing range of robot i is more than the maximum distance that can be travelled by robots i and j in duration δt, omitting robot j does not affect the safety of the algorithm.

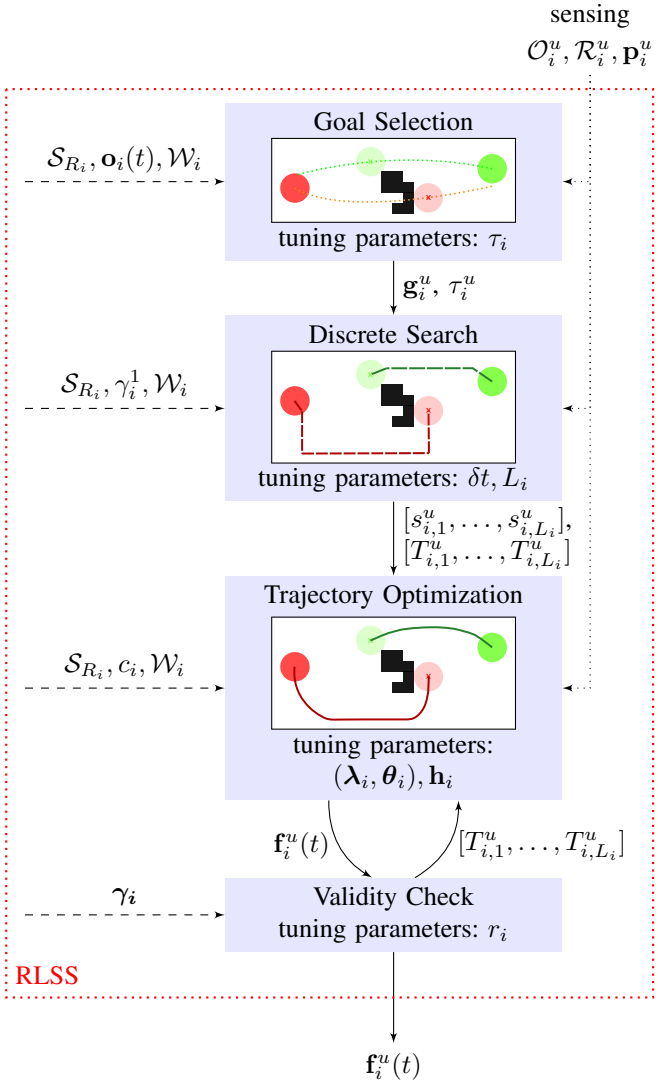


Fig. 2. The overall data flow pipeline of RLSS. Based on the sensed obstacles $\mathcal{O}_i^u$, robots $\mathcal{R}_i^u$, and estimated position $\mathbf{p}_i^u$ we compute the trajectory $\mathbf{f}_i^u$ at every timestep u. The problem instance constants $S_{R_i}, \mathbf{o}_i(t), \mathcal{W}_i, \boldsymbol{\gamma_i}$, and c_i are given with dashed lines to stages that use them as inputs. More details about each stage are explained in our journal submission.

Also, discrete search calculates the total duration T_i^u for the path using τ_i^u, δt, γ_i^1, and distributes the total duration to each of the segments in the discrete path proportional to the segment lengths. The desired number of segments of the discrete path is set by the parameter L_i. The outputs are the resulting path segments $s_{i,1}^u, \ldots, s_{i,L_i}^u$, and path segment durations $T_{i,1}^u, \ldots, T_{i,L_i}^u$, where $s_{i,j}^u$ is the j^{th} segment of the discrete path for robot i at timestep u and $T_{i,j}^u$ is that segment's duration. Details of the discrete search stage are presented in our journal submission.

At the trajectory optimization stage, we compute convex polytopes between i) obstacles $\mathcal{O}_i^u$ and each volume swept by the robot with collision shape function $\mathcal{S}_{R_i}$ while traversing a discrete path segment, and ii) robot collision shapes $\mathcal{R}_i^u$ and each volume swept by the robot while traversing a discrete path segment using support vector machines. The

optimization smooths the path segments within these convex polytopes to a piecewise trajectory where each piece is a Bézier curve so that the resulting trajectory is safe in terms of both robot-to-robot and robot-to-obstacle collisions. The smoothing procedure is formulated as a convex quadratic optimization problem. Let $T_i^u = \sum_{j=1}^{L_i} T_{i,j}^u$ denote the total duration of the planned trajectory where $T_{i,j}$ is the duration assigned to the j^{th} segment. The Bézier piece degrees, and hence the number of control points of pieces, are tuned with the parameter $\mathbf{h}_i$, where the j^{th} entry of $\mathbf{h}_i$, namely $h_{i,j}$, is the degree of the j^{th} piece of the trajectory for each $j \in \{1, \ldots, L_i\}$. Let $\mathbf{P}_{i,j,k}^u \in \mathbb{R}^d$ be the k^{th} control point of the j^{th} Bézier piece of robot i at timestep u. Let $\mathcal{P}_i^u = \{\mathbf{P}_{i,j,k}^u \mid j \in \{1, \ldots, L_i\}, k \in \{0, \ldots, h_{i,j}\}\}$ be the set of all control points of robot i at timestep u.

The cost function we use in the trajectory optimization stage is a weighted combination of energy usage and deviation from the discrete segments.

We use the sum of integrated squared derivative magnitudes as a metric for energy usage, similar to prior work [8], [23]. The energy usage $E(\mathcal{P}_i^u)$ is computed as

$$E(\mathcal{P}_i^u) = \sum_j \lambda_{i,j} \int_0^{T_i^u} \left\| \frac{d^j \mathbf{f}_i^u(t)}{dt^j} \right\|_2^2 dt, \tag{3}$$

where $\lambda_{i,j}$ are weight parameters. $E(\mathcal{P}_i^u)$ is a quadratic function of the control points $\mathcal{P}_i^u$ [23].

We use the Euclidean distance between trajectory piece endpoints and segment endpoints as a metric for the deviation from the discrete segments. The deviation $D(\mathcal{P}_i^u)$ from the discrete segments is

$$D(\mathcal{P}_i^u) = \sum_{j=1}^{L_i} \theta_{i,j} \left\| \mathbf{P}_{i,j,h_{i,j}}^u - \mathbf{e}_{i,j}^u \right\|_2^2, \tag{4}$$

where $\theta_{i,j}$ are the weight parameters, $\mathbf{e}_{i,j}^u$ is the endpoint of the segment $s_{i,j}^u$, and $\mathbf{P}_{i,j,h_{i,j}}^u$ is the last control point of the j^{th} piece.

In addition to the convex polytope constraints we impose for safety, we add constraints for differential continuity between pieces, differential continuity between planning iterations and navigation within the workspace.

The output of the trajectory optimization is the resulting trajectory $\mathbf{f}_i^u(t)$.

During the validity check, we check i) whether the trajectory optimization succeeded and ii) whether the trajectory obeys the robot's maximum derivative magnitudes $\boldsymbol{\gamma_i} = (\gamma_i^1, \ldots, \gamma_i^{K_i})^\top$. If any of the checks fail, temporal rescaling is applied. Each piece duration is multiplied by r_i, and trajectory optimization is executed again with the new durations. This process is repeated until the trajectory passes the validity check. Details of the validity check stage are presented in our journal submission.

The output of one iteration of RLSS is the final output of the trajectory optimization stage which has duration τ_i^u and is safe for duration δt.

We provide no theoretical guarantees on the completeness of RLSS for arbitrary robot dynamics. Therefore, replanning may fail. In our usage of the algorithm, we use the trajectory calculated in the previous iteration when replanning fails. However, one can also choose to stop the robot immediately depending on the robot dynamics.

TABLE I
DESCRIPTIONS OF SETUPS

#	Dim.	# Robots	Cell Size	# Cells	# Obstacles
1	3D	6	50x50x50 cm^3	32k	114
2	3D	6	40x40x40 cm^3	62k	172
3	3D	6	30x30x30 cm^3	148k	246
4	3D	6	20x20x20 cm^3	500k	601
5	3D	6	10x10x10 cm^3	4m	2736
7	3D	12	50x50x50 cm^3	32k	114
8	3D	24	50x50x50 cm^3	32k	114
9	3D	48	50x50x50 cm^3	32k	114
10	3D	6	50x50x50 cm^3	32k	0
11	3D	12	50x50x50 cm^3	32k	0
12	3D	24	50x50x50 cm^3	32k	0
13	3D	48	50x50x50 cm^3	32k	0

V. EXPERIMENTS

We implement and release an implementation of our algorithm in C++[3] with its ROS integration[4]. Our implementation of the algorithm requires the occupancy grid representation of the environment. It regards each occupied cell as a separate obstacle.

We test our algorithm in simulations and on physical robots. During each type of experiment, we feed the pre-initialized occupancy grid of the environment directly to the algorithm and we do not integrate a mapping system to the experiments.

A supplemental video for our experiments is available at `https://youtu.be/xsnWs-85knA`.

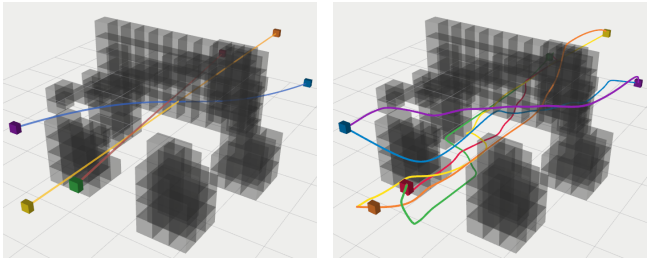
A. Experiments on Simulated Robots

We simulate our algorithm on a desktop computer (Intel i9-9900R @ 3.10GHz CPU, 32GB memory) with Ubuntu 18.04 LTS as the operating system. We use CPLEX [24] to solve the trajectory optimization problem in our implementation.

Descriptions of the setups we use during the simulations are summarized in Table I. Setups 1, 2, 3, 4, 5, 7, 8, and 9 contain the same set of obstacles shown in Fig. 3 differing in the cell size they use for occupancy grids or the number of robots they contain. Setups 10, 11, 12, and 13 contain no obstacles. They differ in the number of robots navigating in the environment.

All of the experiments presented here are conducted in 3D. 2D experiments exists in our journal submission. We use two strategies for solving optimization problems in the experiments. In our first strategy, which we call HARD, we solve the hard constraint formulation of the optimization problem. If the hard constraint formulation fails, trajectory optimization stage fails. In our second strategy, which we call HARD-SOFT, we first solve the hard constraint formulation

[3] `https://github.com/usc-actlab/rlss`
[4] `https://github.com/usc-actlab/rlss_ros`



(a) Desired Trajectories (b) Executed Trajectories

Fig. 3. The environment and desired trajectories in setups $1, 2, 3, 4$, and 5. There are 6 robots that needs to cross a tight environment in opposite directions. Desired trajectories of the robots are given in (a). The executed trajectories in setup 2 calculated by RLSS in real-time are given in (b).

TABLE II

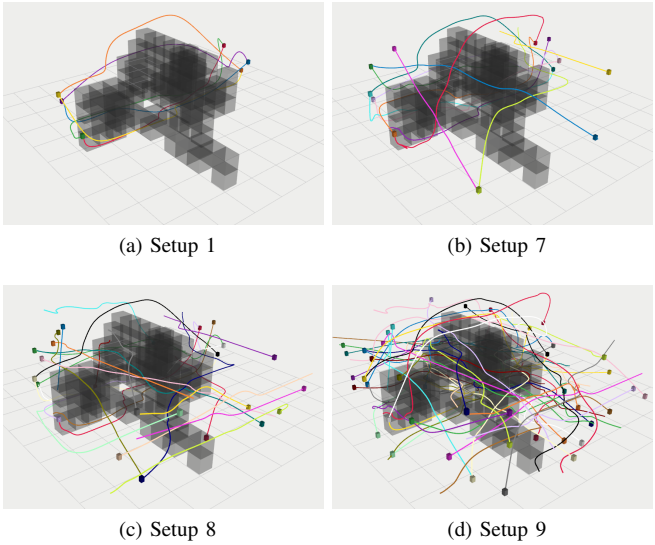
COMPUTATION TIME PER ITERATION WITH VARYING CELL SIZES

		HARD		HARD-SOFT	
#	C. Size [cm^3]	max [ms]	avg [ms]	max [ms]	avg [ms]
1	50x50x50	88.83	13.57	88.77	13.08
2	40x40x40	89.31	12.34	76.19	12.32
3	30x30x30	100.56	14.26	167.59	14.48
4	20x20x20	103.49	19.72	120.13	19.69
5	10x10x10	4598.22	159.40	4597.91	159.44

of the problem and fall back to soft constraint formulation of the problem whenever the hard version fails. If the soft constraint formulation fails, trajectory optimization stage fails.

We check how the occupancy grid cell size affects the performance of our planning pipeline. We use the scenario shown in Fig. 3 in which 6 robots cross a tight 3D environment. We conduct our experiments using both HARD and HARD-SOFT strategies. We plan for 4-piece splines with degree 6 Bézier curves as pieces. The results of our experiments are summarized in Table II. Falling back to the soft formulation when the hard formulation fails does not significantly change the computation time per iteration on average as the hard formulation succeeds most of the time. However, as seen in setups 3 and 4, the soft formulation of the optimization problem can decrease the worst case performance of the RLSS pipeline. While we can run RLSS at more than $5\,\mathrm{Hz}$ in the worst case in setups 1, 2, 3, and 4, our algorithm performs poorly for setup 5 in which the cell size is significantly smaller.

Next, we test the scalability with respect to the number of robots. We use the environment in Fig. 3 with 50x50x50 cm^3 cell sizes. We test with 6, 12, 24, and 48 robots corresponding to setups 1, 7, 8, and 9. We conduct our experiments using both HARD and HARD-SOFT strategies. We plan for 4-piece splines with degree 6 Bézier curves as pieces. The resulting executed trajectories for the HARD strategy are shown in Fig. 4. The results of our experiments are summarized in Table III. As it is the case for varying occupancy grid cell sizes, falling back to soft formulation when the hard formulation fails does not significantly change the performance of our pipeline on average. However, the



(a) Setup 1 (b) Setup 7

(c) Setup 8 (d) Setup 9

Fig. 4. Executed trajectories in setups 1, 7, 8, and 9 when the HARD strategy is used.

TABLE III

COMPUTATION TIME PER ITERATION WITH VARYING NUMBER OF ROBOTS

		HARD		HARD-SOFT	
#	# Robots	max [ms]	avg [ms]	max [ms]	avg [ms]
1	6	88.83	13.57	88.77	13.08
7	12	101.17	11.60	200.78	12.60
8	24	105.63	11.84	216.50	11.88
9	48	142.38	15.94	532.56	18.38

worst case performance of RLSS decreases significantly when the soft formulation is introduced. While we can run our algorithm at $10\,\mathrm{Hz}$ for experiments 7 and 8 with the HARD strategy, we can only run it at $5\,\mathrm{Hz}$ with the HARD-SOFT strategy in the worst case. In experiment 9, the HARD-SOFT strategy takes more than 3 times more time than the HARD strategy in the worst case.

B. Comparison with DMPC in Simulations

We compare the performance of our algorithm with a state-of-the-art online trajectory generation algorithm based on distributed model predictive control, which we will call DMPC from now on [13]. DMPC uses a quadratic optimization formulation where safety is enforced with soft constraints. It requires communication between robots; robots exchange their intended future states via a communication channel whenever they replan, and robots receiving the future states add constraints that enforce a minimum distance to the future states of other robots. Hence, safety depends on the quality of the communication link. The optimization problem DMPC formulates is goal oriented in which robots try to go as close as possible to their goals in each iteration. During comparisons, we use the HARD-SOFT strategy in our algorithm because it is the strategy we would employ in a real-life deployment.

There are two sets of setups we use during the comparison.

TABLE IV

COMPUTATION TIME PER ITERATION COMPARISON OF RLSS AND DMPC

	RLSS		DMPC	
#	max [ms]	avg [ms]	max [ms]	avg [ms]
10	41.02	8.25	**10.97**	**1.05**
11	135.66	9.49	**11.17**	**0.99**
12	165.98	11.44	**11.31**	**0.96**
13	272.33	14.78	**6.18**	**1.05**
1	167.37	17.60	**13.43**	**2.55**
7	213.97	14.11	**6.66**	**2.11**
8	226.79	14.07	**12.51**	**1.68**
9	270.46	17.15	**7.13**	**1.64**

In the first set, we use setups 10, 11, 12, and 13 in which there are no obstacles and the number of robots are 6, 12, 24, and 48, respectively. In the second set, we use setups 1, 7, 8, and 9 in which there are 114 obstacles and the number of robots are 6, 12, 24, and 48 respectively.

During the comparisons, we use Bézier curves of degree 3 since DMPC optimization fails for larger degrees in its authors' implementation. We plan for 4-piece splines with degree 3 Bézier curves as pieces using both algorithms.

First, we compare RLSS and DMPC in terms of computation time per iteration to quantify performance. The results of computation time comparison are summarized in Table IV. DMPC takes considerably less time both in the worst case and in the average case than RLSS.

Then, we compare RLSS and DMPC in terms of number of deadlocks, number of collisions, and total distance traveled by all robots to quantify solution quality. The number of collisions is calculated by sampling the trajectories traversed by robots in 0.01 second intervals and incrementing a collision counter by 1 for each robot/robot and robot/obstacle collision. If there is more than 1 collision in the same 0.01 second interval, collision counter is incremented by the number of collisions. The results of our comparisons are summarized in Table V. In our experiments, robots using RLSS do not deadlock unlike robots that employ DMPC. Also, in all of the cases, DMPC causes collisions. This stems from the fact that DMPC uses a soft QP formulation. On the other hand, the number of collisions with RLSS are considerably lower, because RLSS enforces safety with hard constraints, and falls back to a soft formulation only if the hard version fails. Lastly, the distance travelled is lower in DMPC than RLSS in 3 out of 4 experiments without deadlocks (experiments with setups 10, 11, 12, 13). This happens because DMPC utilizes communication to exchange planned trajectories, which results in a more efficient use of the free space. On the other hand, since RLSS depends on sensing only, robots using it share the available space conservatively.

C. Experiments on Physical Robots

We test our algorithm in 2D using iRobot Create2s, Turtlebot3s, Turtlebot2s, and in 3D using Crazyflie 2.0s. The details of our physical experiments are provided in the journal submission.

TABLE V

QUALITY COMPARISON OF RLSS AND DMPC IN TERMS OF NUMBER OF DEADLOCKS, NUMBER OF COLLISIONS AND TOTAL DISTANCE TRAVELLED

	RLSS			DMPC		
#	# deadl.	# coll.	dist. [m]	# deadl.	# coll.	dist. [m]
10	**0**	**0**	44.31	**0**	33	**43.89**
11	**0**	**0**	90.39	**0**	1047	**87.63**
12	**0**	**11**	163.26	**0**	1158	**156.31**
13	**0**	**46**	358.42	**0**	1633	**340.08**
1	**0**	**0**	55.46	4	3386	32.15
7	**0**	**0**	100.66	5	3844	69.96
8	**0**	**3**	173.65	5	6837	141.52
9	**0**	**3**	389.16	7	20131	323.68

The recordings for our physical robot experiments are included in the supplemental video.

VI. CONCLUSION

In this work, we present RLSS, an algorithm for distributed real-time trajectory replanning that i) considers dynamic limits of the robots explicitly, ii) enforces safety as hard constraints, iii) can work in real-time, iv) requires sensing only the positions of other agents and obstacles, v) does not use communication, and vi) empirically avoids deadlocks. Our approach enables robots to navigate in cluttered environments with high reactivity. Our conservative requirements of only requiring to sense the positions of other robots and no explicit communication, increase the applicability of RLSS to a wide range of settings, including cases where velocity estimates are very noisy and communication is lossy or unavailable.

In empirical comparison to DMPC, our algorithm takes more time per iteration and computes slightly less distance-efficient solutions. However, DMPC sacrifices completeness and correctness: RLSS has considerably fewer collisions and avoids deadlocks effectively. We show our algorithm's applicability to physical robots by demonstrating its behavior on Turtlebot2s, Turtlebot3s, iRobot Create2s, and Crazyflie 2.0s.

In future work, we would like to extend the algorithm with communication to improve the quality of the resulting trajectories. Also, in the future, we would like to incorporate the noise in the sensing systems within our algorithm to bring it closer to real world deployment.

REFERENCES

[1] B. Şenbaşlar, W. Hönig, and N. Ayanian, "RLSS: Real-time Multi-Robot Trajectory Replanning using Linear Spatial Separations," *arXiv e-prints*, p. arXiv:2103.07588, Mar. 2021.

[2] P. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses." *AI Magazine*, vol. 29, pp. 9–20, 03 2008.

[3] A. Furda and L. Vlacic, "Enabling safe autonomous driving in real-world city traffic using multiple criteria decision making," *IEEE Intelligent Transportation Systems Magazine*, vol. 3, no. 1, pp. 4–17, 2011.

[4] K. Dresner and P. Stone, "A multiagent approach to autonomous intersection management," *Journal of artificial intelligence research*, vol. 31, pp. 591–656, 2008.

[5] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40 – 66, 2015.

[6] E. Lam, P. Le Bodic, D. D. Harabor, and P. J. Stuckey, "Branch-and-cut-and-price for multi-agent pathfinding," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, 7 2019, pp. 1289–1296.

[7] S. Tang and V. Kumar, "Safe and complete trajectory generation for robot teams with higher-order dynamics," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 1894–1901.

[8] W. Hönig, J. A. Preiss, T. K. S. Kumar, G. S. Sukhatme, and N. Ayanian, "Trajectory planning for quadrotor swarms," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 856–869, 2018.

[9] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P. Beardsley, and R. Siegwart, "Optimal reciprocal collision avoidance for multiple non-holonomic robots," in *Distributed Autonomous Robotic Systems (DARS)*. Springer, 2013, pp. 203–216.

[10] L. Wang, A. D. Ames, and M. Egerstedt, "Safety barrier certificates for collisions-free multirobot systems," *IEEE Transactions on Robotics*, vol. 33, no. 3, pp. 661–674, 2017.

[11] G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. K. Kumar, S. Koenig, and H. Choset, "Primal: Pathfinding via reinforcement and imitation multi-agent learning," *IEEE Robotics and Automation Letters*, vol. 4, pp. 2378–2385, 2019.

[12] D. Zhou, Z. Wang, S. Bandyopadhyay, and M. Schwager, "Fast, online collision avoidance for dynamic vehicles using buffered voronoi cells," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 1047–1054, 2017.

[13] C. E. Luis, M. Vukosavljev, and A. P. Schoellig, "Online trajectory generation with distributed model predictive control for multi-robot motion planning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 604–611, 2020.

[14] R. M. Murray, M. Rathinam, and W. Sluis, "Differential flatness of mechanical control systems: A catalog of prototype systems," in *ASME international mechanical engineering congress and exposition*. Citeseer, 1995.

[15] G. Campion, G. Bastin, and B. Dandrea-Novel, "Structural properties and classification of kinematic and dynamic models of wheeled mobile robots," *IEEE transactions on robotics and automation*, vol. 12, no. 1, pp. 47–62, 1996.

[16] R. M. Murray and S. S. Sastry, "Nonholonomic motion planning: steering using sinusoids," *IEEE Transactions on Automatic Control*, vol. 38, no. 5, pp. 700–716, 1993.

[17] S. Jiang and K. Song, "Differential flatness-based motion control of a steer-and-drive omnidirectional mobile robot," in *IEEE International Conference on Mechatronics and Automation (ICMA)*, 2013, pp. 1167–1172.

[18] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 2520–2525.

[19] F. Homm, N. Kaempchen, J. Ota, and D. Burschka, "Efficient occupancy grid computation on the gpu with lidar and radar for road boundary detection," in *IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2010, pp. 1006–1013.

[20] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous robots*, vol. 34, no. 3, pp. 189–206, 2013.

[21] R. T. Farouki, "The bernstein polynomial basis: A centennial retrospective," *Computer Aided Geometric Design*, vol. 29, no. 6, pp. 379–419, 2012.

[22] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[23] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Int. Symposium of Robotic Research (ISRR)*, 2013, pp. 649–666.

[24] I. I. Cplex, "V12. 1: User's manual for cplex," *International Business Machines Corporation*, vol. 46, no. 53, p. 157, 2009.