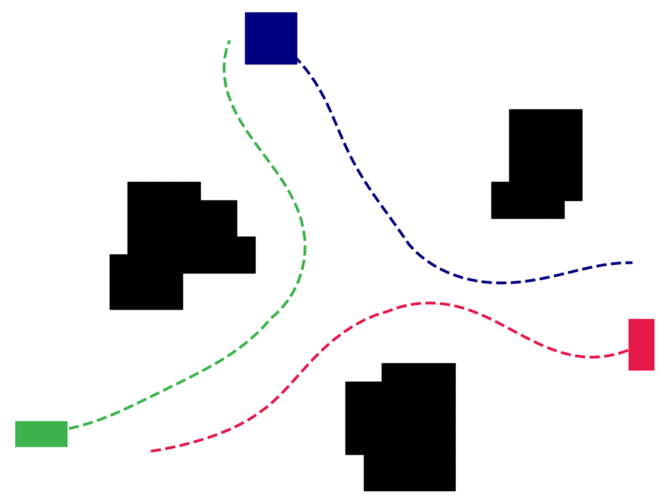
RLSS: Real-time Multi-Robot Trajectory Replanning using Linear Spatial Separations

Baskın Şenbaşlar[1], Wolfgang Hönig[1,2], Nora Ayanian[1]

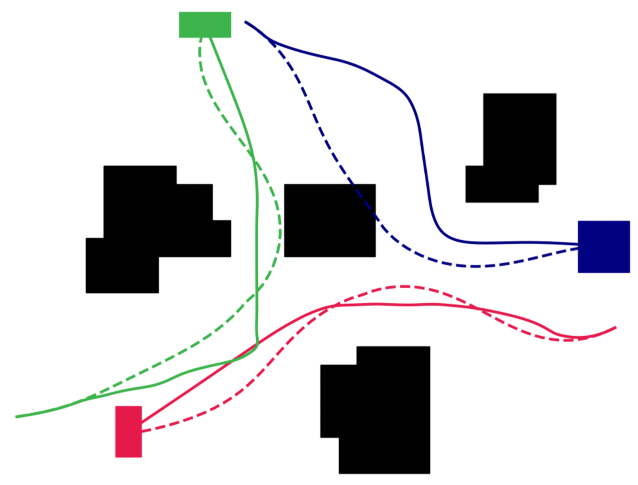
[1] University of Southern California, [2] California Institute of Technology

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. **Full paper is available on arXiv**: https://arxiv.org/abs/2103.07588

Introduction



Desired Trajectories

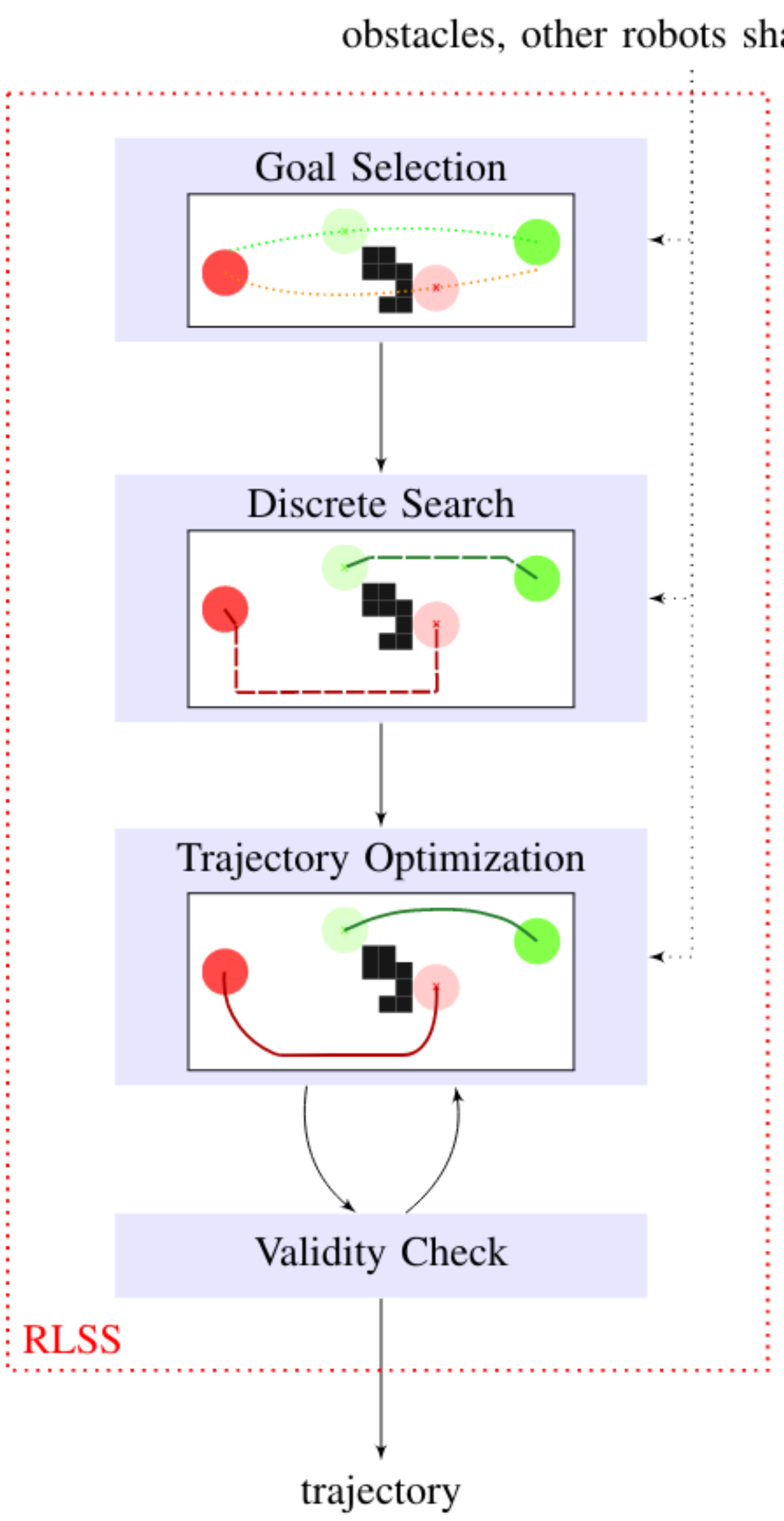


Executed Trajectories

RLSS computes trajectories in real-time. It

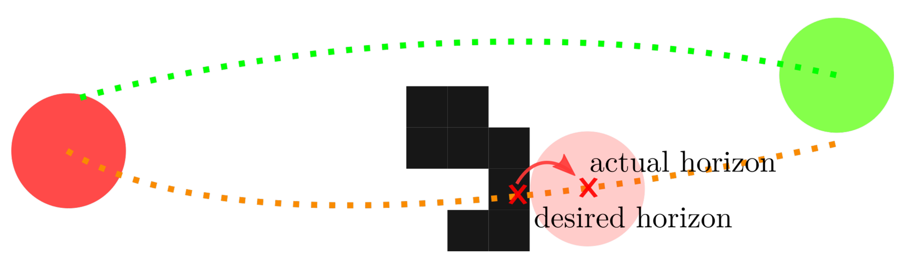
1. explicitly considers dynamic limits of robots,
2. does not rely on a central computer,
3. handles agent failures and compansates poorly performing controllers,
4. does not rely on communication for safety,
5. requires perfect sensing of positions of other robots and obstacles,
6. avoids deadlocks (only experimentally),
7. enforce safety using hard constraints abd reports back if it can't ensure safety,
8. and works in the presence of obstacles.

Approach



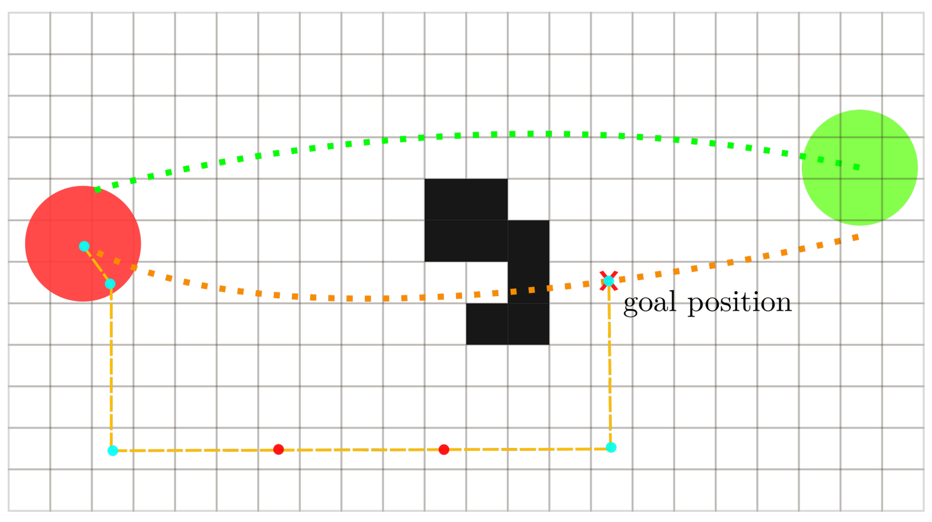
RLSS pipeline runs in every time step. We use periods between 100ms and 500ms in our experiments.

Goal Selection



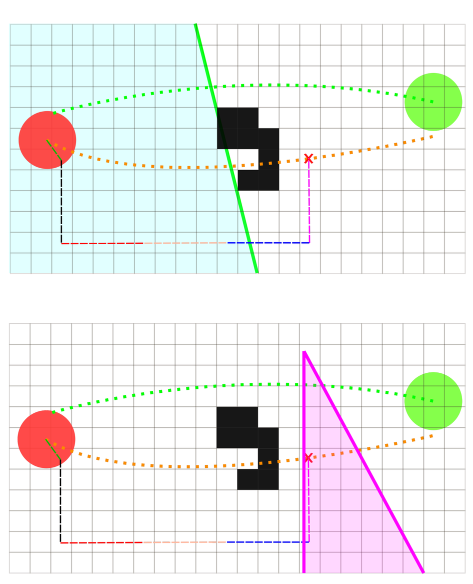
Select a goal on the desired trajectory to plan towards: Given a desired planning horizon, compute the actual planning horizon such that the desired trajectory is collision-free at the actual time horizon.

Discrete Search



Plan a discrete path to the selected goal position: We use A* search to compute a discrete path from the current position to the goal position. In A* search, forward moves and rotations have costs of 1. There are also actions to enter the grid and leave the grid. The desired number of segments is given as a hyperparameter and the segments are split if there are fewer segments than desired. (Segments with blue end points in the image are computed by A* search. Since the desired number of segments is 6, red end points are introduced by segment splitting.) If there are more segments than desired, segments from the end are omitted. We also compute segment durations in discrete search.

Trajectory Optimization



Compute a smooth trajectory over discrete segments: We fit a piecewise Bezier curve on discrete segments using one piece per segment.

Enforcing Robot-to-Robot Collision Avoidance: To enforce robot to robot collision avoidance we compute SVM hyperplanes among robot collision shapes. We constrain the first piece to stay inside the SVM half-space. We enforce that the duration of the first piece is more than the planning period.

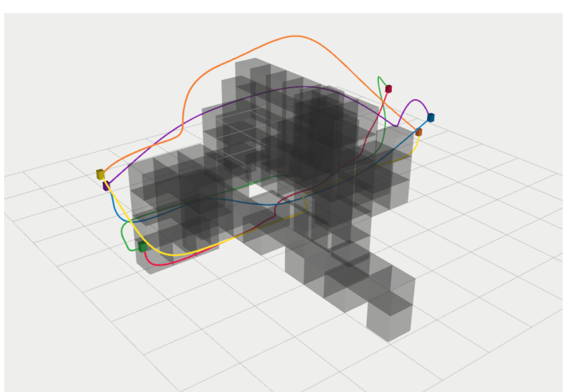
Enforcing Robot-to-Obstacle Collision Avoidance: We compute the SVM hyperplanes between each discrete segment and obstacle. We constrain trajectory pieces to stay in the computed SVM half-spaces. We also add constraints for continuity between trajectory pieces and between planning iterations.

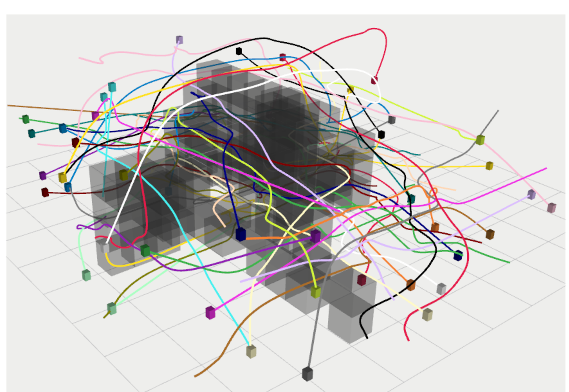
The cost function of the optimization problem is a combination of energy usage and deviation from discrete segments.

Validity Check

Check if the resulting trajectory is valid: We impose derivative magnitude limits on the trajectory in the validity check. If the magnitude of any derivative of any desired degree is more than the maximum amount, the trajectory is invalid. To compute the maximum derivative magnitude along the trajectory, we execute linear search with small increments over the duration of the trajectory. If the resulting trajectory is invalid, we increase the piece durations and re-run the optimization.

Evaluation



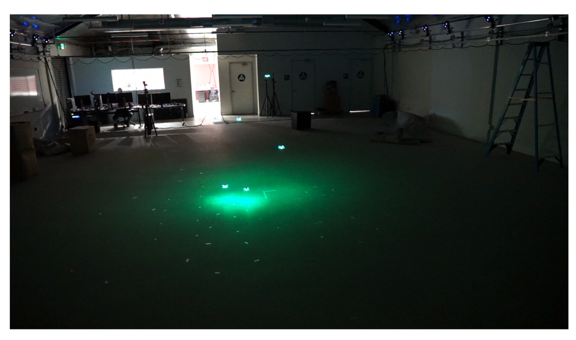


Computation Time per Iteration Comparison of RLSS and DMPC

	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**

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



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

https://youtu.be/xsnWs-85knA

USC University of Southern California   act lab   Caltech