Multi-Robot Coordinated Planning in Confined Environments under Kinematic Constraints

Clayton Mangette and Pratap Tokekar

***Abstract*— We investigate the problem of multi-robot coordinated planning in environments where the robots may have to operate in close proximity to each other. We seek computationally efficient planners that ensure safe paths and adherence to kinematic constraints. We extend the central planner dRRT* with our variant, fast-dRRT (fdRRT), with the intention being to use in tight environments that lead to a high degree of coupling between robots. Our algorithm is empirically shown to achieve the trade-off between computational time and solution quality, especially in tight environments. We also demonstrate the ability of our algorithm to be adapted to the online planning problem while maintaining computational efficiency. The software implementation is available online at `https://github.com/CMangette/Fast-dRRT`.**

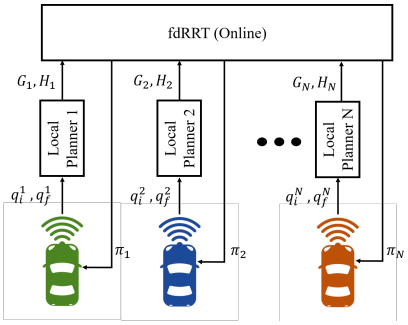


Fig. 1: The central planner returns collision-free path queries by referencing pre-computed roadmaps from a local planner.

I. Introduction

Computationally efficient multi-robot motion planning algorithms are highly sought after for their numerous applications. In a time when automotive manufacturers are quickly approaching the advent of self-driving cars, centralized motion planners in lieu of traditional traffic control structures open the possibility of increased traffic flow in busy urban environments [6], [19]. With an increase in automation in warehouses [1], efficient path planning of robots designed to move inventory has become another important use case. Beyond ground vehicles, traffic management of Unmanned Aerial Vehicles (UAVs) is identified as an important area of research to ensure safe integration of aerial vehicles into the airspace [8].

In each of the aforementioned applications, the algorithms used must be robust to planning in tight, confined environments while eliminating the possibility of robot collisions. In the case of automated driving, urban traffic structures such as intersections and highway merging ramps constrain vehicles to a narrow set of paths. Similarly, robots operating in a warehouse environment must conform to the facility infrastructure to avoid storage and shelving units. While not subject to high clutter, high volume air traffic can artificially restrict paths for UAVs.

The planning algorithms available for such problems can be classified as centralized or decoupled. Centralized algorithms plan in the joint space of all robots whereas decoupled approaches only consider the space for each individual robot [9]. Centralized frameworks already exists in each use case. The intersection manager in [16] is a candidate replacement to traffic lights that can control when autonomous vehicles enter an intersection. A task allocation and path planning system in [5] demonstrates how to automate warehouse stock movement with kiva robots. The Unmanned Aerial System Traffic Management uses a centralized service supplier to manage requests and conflicts between UAVs operating within the same space [8].

The main challenge in centralized planning is time-efficiency. State-of-the-art planners feature algorithms that improve efficiency while preserving completeness. Recognizing the shortcomings of previous algorithms that rely on explicit computation of the composite planning space, discrete rapidly expanding random trees (dRRT) [12] and its optimal variant dRRT* [11] improve efficiency by delegating computations to offline tasks when possible and relying on implicit representations of the planning space. These algorithms do not encode steering constraints, but provide a general framework for fast multi-robot planning.

This paper presents a variant to dRRT / dRRT*, which we call fast-dRRT (fdRRT), that returns sub-optimal trajectories quickly in tight environments that require significant coordination between robots. We also extend these planners to account for robots with kinematic constraints.

II. Related Work

Extending motion planning to the multi-robot domain has been challenging due an increase in search space size. A sequential process in [20] splits the problem into local path planning using D* and coordination between robots to avoid collisions. Instead of handling spatial and velocity planning separately, Wagner and Choset developed M*, a multi-robot analogue to A* that resolves local path collisions by coupling paths only when they are found to overlap [17].

Van den Berg et al. provide a framework for planning in a roadmap that determines a sequential ordering for each robot

C. Mangette is with the Department of Electrical and Computer Engineering, Virginia Tech, U.S.A. {mangettecj}@vt.edu

P. Tokekar is with the Department of Computer Science at the University of Maryland, U.S.A. {tokekar}@umd.edu

to execute its path [15]. The coordinated path planner in [21] searches for collision-free paths over an explicitly computed multi-robot work space, but is limited in scope due to the memory required to build its roadmap. Using the principle of sub-dimensional expansion, Wagner et al. designed sub-dimensional RRT and sub-dimensional PRM to plan paths for multiple robots with integrator dynamics [18].

Solovey et al. also use sub-dimensional expansion in discrete RRT (dRRT) [12]. The idea is to build collision-free road maps for each robot, and then use them to build a search tree implicitly embedded in the road maps. dRRT draws samples from each road map and combines them into a composite sample to which the tree is extended towards by selecting a composite neighboring vertex. Collision-free composite motions are added as vertices to the tree until a goal is reached. The optimal variant of dRRT, dRRT*, improves upon computation time further by carefully choosing neighbors to expand towards the goal state [11].

This paper presents a centralized planning strategy for kinematically constrained robots in tight environments. The central planning algorithm, which we call fast-dRRT (fdRRT), is designed to switch between randomly exploring the state space and driving greedily towards the goal state in a manner similar to dRRT*. The difference in our algorithm is how expansion failures due to collisions are adjudicated. Instead of returning an expansion failure if no collision-free connection can be established, fdRRT forces an expansion success by commanding some robots to stay in their previous configurations while permitting others to move forward. It is empirically demonstrated that this makes fdRRT faster than dRRT* in tight work spaces, but at the cost of solution quality. We also demonstrate the feasibility of fdRRT in dynamic planning scenarios. Unlike dRRT*, fdRRT makes no guarantee of probabilistic optimality, thus imposing an trade-off between solution efficiency and quality when choosing between the two algorithms. Additionally, fdRRT's incorporation of kinematic constraints makes it a flexible planner that can be used in different systems.

III. Problem Formulation

The inputs to this problem are a set of initial and goal configurations for N robots in the planning space. $q \in (x, y, \theta, \kappa)^T$ denotes a configuration of a robot. $Q = \{q_1, q_2, ..., q_N\}$ denotes a set of robot configurations, which is referred to as a joint configuration. The initial and goal configurations are denoted as Q_{init} and Q_{goal}, respectively. Furthermore, environmental obstacles are known and denoted as C_{obst}. The goal is to find a joint trajectory $\Pi = \{\pi_1, \pi_2, ..., \pi_N\}$ that connects Q_{init} to Q_{final} such that all local trajectories $\pi \in \Pi$ are collision-free with respect to the environment and other robots.

We are interested in multi-robot systems that operate under kinematic constraints. Each robot is assumed to follow the dynamics given by: $\left[\dot{x}, \dot{y}, \dot{\theta}, \dot{\kappa}\right]^T = \left[\cos\theta, \sin\theta, \kappa, \sigma\right]$. For simplicity, each robot is constrained to moving forward. The dynamics are an extension of Dubins' steering constraints [2] with an additional variable κ for curvature. The control signal in this model is σ, which is the angular acceleration. The sum of path lengths of the multi-robot trajectory is the cost metric chosen for evaluating solution quality.

IV. Algorithm Overview

Our system is illustrated in Figure 1. A roadmap for each robot is constructed locally to definte its possible configurations and motions. Similar to dRRT*, the central planner receives path queries in the form of initial and final configurations and local roadmaps from the robots in the planning space. To avoid re-planning due to new requests, the central planner accepts requests until a deadline $T_{deadline}$ and relegate new requests to the next planning cycle. Given the local roadmaps and initial and final configurations of each robot, the central planner returns composite path $\Pi = (\pi_1, \pi_2, ..., \pi_R)$ that guarantees collision free trajectories between robots. Each local trajectory is sent to its corresponding robot as a list of time-parameterized waypoints $w_i(t) = [x_i(t), y_i(t), \theta_i(t), \kappa_i(t)]^T$ and connecting paths $\pi_i(s) = [x_i(s), y_i(s), \theta_i(s), \kappa_i(s)]^T$. The intent is for the central planner to determine the sequence of configurations that each robot visits as well as its arrival time. Each robot must then determine appropriate speed and steering commands to execute its motions.

The central planning algorithm takes the work in [11] and applies it to a system of multiple kinematically constrained robots. Two modifications are made to improve upon its speed in finding a solution. First, when finding the best node in the tree to connect to a new configuration, both the lowest cost configuration and the lowest cost free configuration are returned. Then, in the event that no collision-free configuration is found, the `ForceConnect` subroutine is invoked, and attempts to find a hybrid configuration in which a subset of robots move forward towards the new joint configuration and the rest maintain their positions. The intuition behind this subroutine is that there is a small set of paths that take a robot to its goal in confined environments, making the likelihood of overlapping paths high. The only feasible path may involve some robots momentarily holding their positions while others move forward, similar to the pebble motion problem [13]. Our modification to dRRT* explicitly searches for these types of solutions when collisions occur to increase the computational efficiency of the algorithm. Completeness is preserved as this modification does not significantly change the structure of the algorithm.

The local controller on each robot determines the speed profile to follow from $w_i(t)$ and the distance travelled between consecutive waypoints. $\pi_i(s)$ is re-parameterized to $\pi_i(t)$ from the distance traveled over time, which can be tracked by a local controller using a technique such as pure-pursuit or nonlinear Model Predictive Control (MPC) [7].

A. Roadmap Generation

Solovey et al. suggest using probabilistic roadmaps (PRMs) as approximations to local configuration spaces [12]. The PRM algorithm builds a roadmap as a graph $G = (V, E)$, with each vertex $v \in V$ being a unique configuration

and each edge $e(v_i, v_j) \in E$ a path in free space connecting two adjacent vertices v_i and v_j [4]. Configurations q_{rand} are randomly sampled in the configuration space C and connected to any vertices in G within a connection distance d, $\{v \in V | \texttt{dist}\,(q_{rand}, v) \leq d \wedge e(q_{rand}, v) \in C_{free})\}$. Roadmap construction continues until $|V| = n$, after which paths between configurations can be queried.

Connections in [4] are line segments, which are sufficient under the assumption of single-integrator dynamics, but not for the dynamics we consider. Scheuer and Fraichard extend Dubins' paths to continuous curvature paths using clothoids to transition between changes in curvature that, while less computationally efficient than Dubins' segments, are a feasible connection method [10]. Additionally, G in [4] is an undirected graph, implying that motions between connected vertices are bi-directional. Due to Dubins' steering constraints and the non-holonomic constraints, this is not necessarily true, and the existence of a collision-free path connecting v_i to v_j does not guarantee the reverse. To address this, Svestka and Overmars demonstrate that making G a directed graph is sufficient to impose this restriction [14].

The local planning method is similar to the Probabilistic Path Planner (PPP) in [14] with additional sampling and connection constraints to build a road map biased towards the optimal path that discriminates against undesirable connections (Algorithm 1). G is initialized with a

Algorithm 1: `RoadmapGeneration`$(q_i, q_f, n, r, C_{obst})$

```
1  G ← q_i;
2  π_sample ← ReferencePath(q_i, q_f);
3  while |V| < n do
4      q_rand ← RandomConfig(π_sample);
5      for v ∈ V do
6          π_local ← Steer(v, q_rand);
7          if Reachable(π_local , d) &
             CollisionFree(π_local, C_obst) then
8              if q_rand ∉ G then
9                  (G, v_new) ← Insert(q_rand);
10             end
11             G ← Connect(v, v_new, π_local);
12         end
13     end
14 end
15 H ← CostToGoal(G, q_f);
16 G ← PruneDeadNodes(G, H);
17 return G , H
```

configuration q_i (Line 1), which represents a robot's starting configuration. A base path π_{sample} is computed as the ideal path to follow from q_i to q_f and is used when sampling configurations (Line 2). G expands to size n by sampling random configurations q_{rand} (Line 4), finding a continuous curvature path π_{local} connecting each existing node in G to q_{rand} (Line 6), and evaluating its feasibility (Line 7). A continuous curvature path π_{12} connecting two configurations q_1 and q_2 is considered feasible if it does not lead to a collision with any environmental obstacles and satisfies the both of the reach-ability constraints: (1) the path length $l(\pi_{local}) \leq d$; (2) q_2 is in front of q_1. This subroutine uses the check in [3] to determine if q_2 is in the half-space of q_1 and then determine the orientation of q_2 with respect to q_1.

The reachability check prunes complex maneuvers such as turning 360° to connect two adjacent configurations. After G reaches sufficient size, a cost-to-goal array H is computed storing the shortest path cost to go from every configuration in G to q_f (Line 15). Any configurations in G that do not have a path to q_f are removed from G to prevent useless exploration during central planning (Line 16).

B. Central Planner

Algorithm 2: `fdRRT`$(Q_{init}, Q_{goal}, \mathbb{G}, \mathbb{H})$

```
1  T ← Q_init;
2  V_last ← Q_init;
3  while Q_goal ∉ T do
4      (T, V_last) = Expand(T, G, H, V_last, Q_goal);
5      if Q_goal ∈ T then
6          Π ← FindPath(T, Q_goal);
7          return Π
8      end
9  end
```

The structure from dRRT* (Algorithm 2) is preserved with the initialization of $\mathbb{T}$ with Q_{init} (Line 1). The algorithm then expands, while remembering the most recent expansion node V_{last} to determine how it expands in the next expansion call (Line 4). `FindPath` queries $\mathbb{T}$ for a path to Q_{goal} and returns a composite path Π if successful (Lines 5–6). A notable difference is the omission of a local connector present in [11], [12], whose purpose is to solve the coordination problem when close to Q_{goal}. We found this to be unnecessary in our environments. For traffic intersections, once all vehicles have passed through the physical intersection of the two roads, $\mathbb{T}$ expands greedily towards Q_{goal}. We instead re-purpose this subroutine during node expansion for resolving collision conflicts.

Algorithm 3: `Expand`$(\mathbb{T}, \mathbb{G}, \mathbb{H}, V_{last}, Q_{goal})$

```
1  if V_last = ∅ then
2      Q_rand ← RandomConfig(G);
3      V_near ← Nearest(T, Q_rand);
4  else
5      Q_rand ← Q_goal;
6      V_near ← V_last;
7  end
8  V_new ← I_d(V_near, G, H, Q_goal);
9  N ← NeighborsInTree(V_new, T);
10 (V_best^free, V_best) ← BestParent(V_new, N);
11 if V_best^free = ∅ then
12     V_H ← ForceConnect(V_new, V_best);
13     if V_H = ∅ then
14         return ∅
15     else
16         T ← Connect(V_best, V_H);
17         return V_H;
18     end
19 else
20     T ← Connect(V_best^free, V_new);
21     return V_new;
22 end
```

The method for expanding $\mathbb{T}$ is detailed in Algorithm 3.

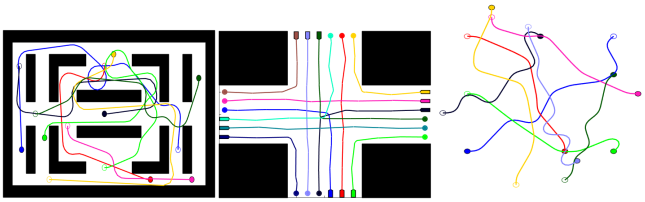
Expansion begins with selecting a node to expand from. If V_{last} was added during the previous call, then a new expansion vertex V_{new} is chosen by selecting a neighbor of V_{last} (Lines 2–3). Otherwise, the closest neighbor V_{near} of a random configuration Q_{rand} is chosen (Lines 5–6). The direction oracle subroutine selects an expansion node V_{new} based on the success of the previous expansion (Line 8). If $Q_{rand} = Q_{goal}$, V_{new} is chosen as the tuple of individual vertices $v^i \in V$ that are neighbors to v^i_{near} and have the lowest path cost to $q^i_f \in Q_{goal}$. It is otherwise chosen as a tuple of randomly selected neighbors to v^i_{near} [11].

All composite parents to V_{new} that have already been added to $\mathbb{T}$ are expansion candidates to connect to V_{new} (Line 9). Each candidate is evaluated based on whether the composite path between N and V_{new} results in a collision-free motion and the composite path cost. The lowest cost collision-free node, V^{free}_{best}, and the lowest cost node V_{best} are selected. V_{best} is simply found by selecting the candidate node with the lowest overall cost, and $V^{f}_{best}ree$ is selected as the lowest cost node that whose joint path connecting to V_{new} is collision-free. If no such V^{free}_{best} exists, the subroutine `ForceConnect` (Algorithm 4) attempts forcing $\mathbb{T}$ to expand by creating a new hybrid node, V_H, that restricts some robots to hold their position at $v_{best} \in V_{best}$, and allows others to move forward towards $v_{new} \in V_{new}$. While forcing some vehicles to stop increases traffic delays, `ForceConnect` increases computational efficiency in practice by restricting random sampling to a last resort.

Algorithm 4: `ForceConnect`(V_1, V_2)

```
1  H ← ∅;
2  L ← ∅;
3  Π12 ← LocalPaths(V1, V2);
4  for π_i ∈ Π12 do
5  |   for π_j ∈ Π12, i ≠ j do
6  |   |   (H_i, L_i, A_i) ← LocalPriority(π_i, π_j);
7  |   end
8  end
9  S ← ∅;
10 for i = 1, 2, .., N do
11 |   if H_i = ∅ & A_i = ∅ then
12 |   |   S ← S ∪ i;
13 |   else if H_i = ∅ & A_i ≠ ∅ then
14 |   |   if cost(i) ≤ min(cost(j ∈ A_i)) then
15 |   |   |   S ← S ∪ i;
16 |   |   end
17 |   end
18 end
19 V_H ← {v_2^i | i ∈ S} ∪ {v_1^j | j ∉ S};
20 return V_H;
```

`ForceConnect`: When forcing a connection between two composite nodes V_1 and V_2, the i^{th} robot either holds its position at $v^i_1 \in V_1$ or moves forward towards $v^i_2 \in V_2$. Three sets are initialized for each robot $r_i \in R$: H_i, the set of robots with higher local priority than r_i, L_i, the set of robots with lower priority than r_i, and A_i, the set of robots that conflict with r_i but have no local priority assigned. Each interaction is checked and $H, L,$ and A are populated by `LocalPriority`. The local priority of r_i with respect to r_j is assigned according to the rules, which originate from the logic in [12] and [15]: (1) If $\pi_i(0)$ blocks π_j, then robot i is given priority. (2) If $\pi_j(0)$ blocks the path of π_i, then robot j is given priority. (3) If π_i and π_j do not overlap, then there is no interaction and no priority is assigned. (4) Otherwise, the local priority cannot be determined. This occurs when π_i and π_j overlap, but the starting positions of robots i and j do not block each other. Either robot can be given priority, but the decision is deferred.



(a) (b) (c)

Fig. 2: Simulation outputs from each environment from the most confined space (Warehouse) to the least confined environment (UAV air traffic)

A solution set S is then initialized to pick robots that should move forward (Line 9). Each robot is added to or rejected from S based on its own H_i, L_i and A_i sets. If no other robots have a higher local priority and no robots have an undetermined priority, then r_i is added to S. If any vehicle has a higher priority, then r_i is rejected from S. If no robots have a higher priority, but some have undetermined priorities, then the cost of adding r_i is assessed. Here, the cost refers to number of vehicles that would be excluded from S if r_i was added to S. The cost of adding r_i is compared to the cost of adding any of $r_j \in A_i$ and will be added to S if the trade-off from adding r_i is lower than the trade-off from adding any other member of A_i. After all robots are either added to or rejected from S, a hybrid node V_H is constructed (Line 19).

C. Extensions to Online Planning

To accommodate new robots entering the planning space, algorithms 2 and 3 are modified to include an additional input $\Pi_{current}$, the current joint trajectory being executed. $\Pi_{current}$ is treated as an obstacle when planning Π_{new}, so collision checking joint configurations in Π_{new} also involves collision checking against the configurations in $\Pi_{current}$. Re-planning of $\Pi_{current}$ is not allowed in the current implementation, so the planner that searches for Π_{new} must plan around $\Pi_{current}$. The only modification required for accommodating $\Pi_{current}$ when planning is when finding V_{best} (algorithm 3, line 10). Because $\Pi_{current}$ is given priority over new path queries, V_{best} must be selected such that collisions between new robots entering the environment are permitted, but collisions with robots executing $\Pi_{current}$ are prohibited. If no configuration satisfies this constraint, then the algorithm must report an expansion failure.

V. Simulations and Results

Our algorithm was implemented and tested in MATLAB using three environments (Figure 2), ordered from the most to least confined spaces. The first environment considers disk-shaped robots with radius $1.6m$ planning in a warehouse space with ten obstacles (Figure 2a). Environment 2 (Figure 2b) is a three-line, four-way traffic intersection. Each lane is $3m$ wide, and each robot is rectangular-shaped with length $3.6m$ and $1.6m$. Environment 3 (Figure 2c) has no obstacles, but has numerous circular UAV robots with disk radius $0.25m$ planning multiple overlapping paths.

The number of robots n is increased incrementally, and n individual queries are drawn randomly in 1000 test cases. Average search tree size, solution time, and path lengths are the chosen evaluation metrics to illustrate the trade-offs between dRRT* and fdRRT (Figures 3–5).

We observe that fdRRT performs better than dRRT* in terms of efficiency in the intersection and warehouse spaces. In test cases with maximum traffic, fdRRT returned solutions 57% faster in the traffic intersection and around 200% faster in the warehouse. However, dRRT* is 12% faster in the UAV environment. This may be due to the lack of clutter within the UAV space, and thus reduced number of choke points. Under these conditions, the added computational time in fdRRT when forcing connections may degrade performance.

Solution quality metrics show the opposite trend. As more robots are added to each environment, the path quality in fdRRT degrades, with paths being 22% and 54% longer in the intersection and warehouse spaces, respectively. Paths in the UAV space are nearly identical with a 0.2% discrepancy.

The trends in solution times across the different test cases may be attributed to the degree of clutter populating each environment and the number of overlapping paths. For instance, the environment in Figure 2a has numerous obstacles scattered throughout the environment, which dramatically limits the number of alternative paths that each robot can take. In contrast, the robots in Figure $2c$ are closely spaced and have a number of overlapping paths, but are less confined due to the lack of clutter in the environment.

Additional tests were run to evaluate the performance of the online planner. We compare its solution time to that of the offline planner when planning in a fully populated version of environment 2 (Figure 6). In the first test case $n = 10$ robots act as the initial plan request, and after one iteration, $m = 2$ robots enter the space and request a joint trajectory. In the second test case, $n = 9$ robots are initially in the environment and $m = 3$ robots enter afterwards. We continue this until two robots are members of the initial query and ten robots enter after the initial planning phase. The average solution times with standard deviations in Figure 6 show that while both plans exhibit exponential growth in computation times as more robots are considered, the online planner's execution time grows faster with a higher variance.

VI. Conclusions and Future Work

We developed a planner for multiple robots to plan paths when operating in confined environments. Our implementation has demonstrated its advantage in computational time over dRRT* when planning in confined environments, at the cost of solution quality. The results from this study are promising, but several challenges remain. Testing the feasibility of fdRRT in a real system is one goal we would like to reach. We also plan to explore extending the planner to incorporate vehicle dynamics in addition to vehicle kinematics. In its current form, we only consider sampling configurations $q \in (x, y, \theta, \kappa)$ and ignore constraints on vehicle speed and acceleration. Adding constraints on vehicle dynamics makes connecting between configurations more difficult, but carries the benefit of ensuring that all paths are feasible for robots with both kinematic and dynamic constraints.

References

[1] E. Ackerman, "Amazon uses 800 robots to run this warehouse," Jun 2019. [Online]. Available: https://spectrum.ieee.org/automaton/robotics/industrial-robots/amazon-introduces-two-new-warehouse-robots

[2] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, no. 3, p. 497, jul 1957. [Online]. Available: https://doi.org/10.2307

[3] J. h. Jeon, S. Karaman, and E. Frazzoli, "Anytime computation of time-optimal off-road vehicle maneuvers using the rrt*," in *2011 50th IEEE Conference on Decision and Control and European Control Conference*, Dec 2011, pp. 3276–3282.

[4] L. E. Kavraki, P. Svestka, J. . Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, Aug 1996.

[5] J.-T. Li and H.-J. Liu, "Design optimization of amazon robotics," 2016.

[6] B. Liu and A. El Kamel, "V2x-based decentralized cooperative adaptive cruise control in the vicinity of intersections," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 3, pp. 644–658, March 2016.

[7] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[8] J. L. Rios, L. Martin, and J. Mercer, "Use of UAS Reports (UREPs) during TCL3 Field Testing," National Aeronautics and Space Administration,, Tech. Rep., 07 2017.

[9] G. Sanchez and J. . Latombe, "Using a prm planner to compare centralized and decoupled planning for multi-robot systems," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, vol. 2, May 2002, pp. 2112–2119 vol.2.

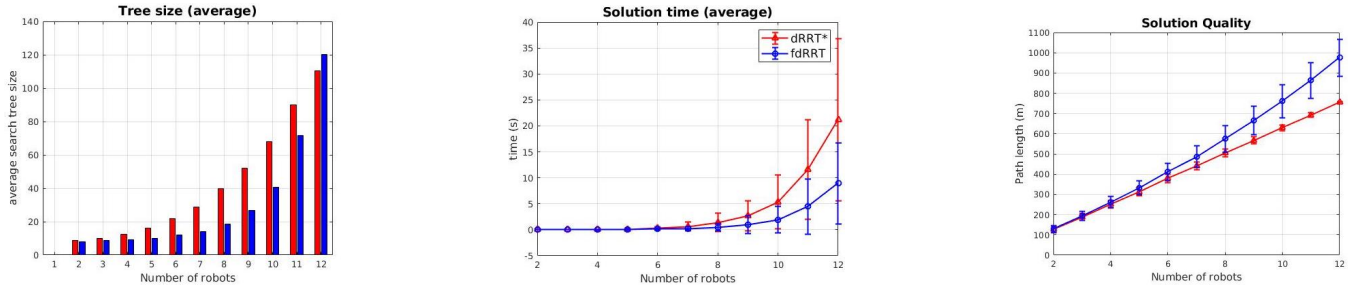
[10] A. Scheuer and T. Fraichard, "Continuous-curvature path planning for car-like vehicles," 10 1997, pp. 997 – 1003 vol.2.

[11] R. Shome, K. Solovey, A. Dobson, D. Halperin, and K. E. Bekris, "drrt*: Scalable and informed asymptotically-optimal multi-robot motion planning," *Autonomous Robots*, vol. 44, no. 3, pp. 443–467, Mar 2020. [Online]. Available: https://doi.org/10.1007/s10514-019-09832-9

[12] K. Solovey, O. Salzman, and D. Halperin, *Finding a Needle in an Exponential Haystack: Discrete RRT for Exploration of Implicit Roadmaps in Multi-robot Motion Planning*. Cham: Springer International Publishing, 2015, pp. 591–607. [Online]. Available: https://doi.org/10.1007/978-3-319-16595-0_34

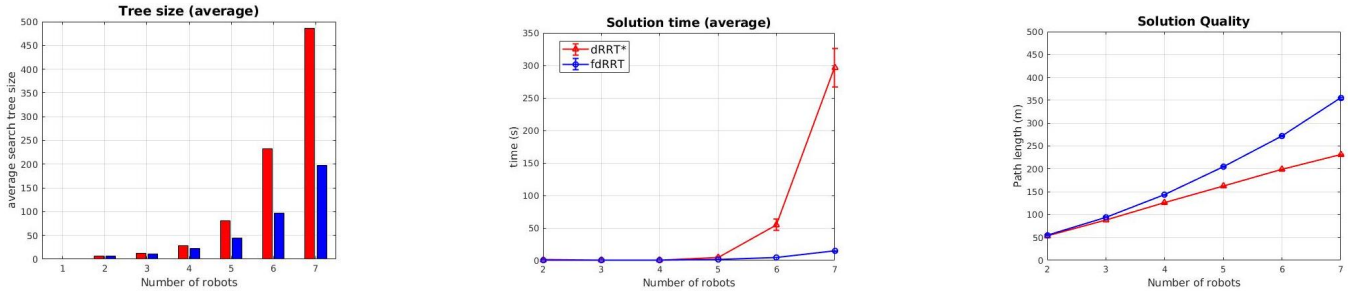
[13] P. Surynek, "An application of pebble motion on graphs to abstract multi-robot path planning," in *2009 21st IEEE International Conference on Tools with Artificial Intelligence*, Nov 2009, pp. 151–158.

[14] P. Svestka and M. H. Overmars, "Motion planning for carlike robots using a probabilistic learning approach," *The International Journal of Robotics Research*, vol. 16, no. 2, pp. 119–143, 1997. [Online]. Available: https://doi.org/10.1177/027836499701600201



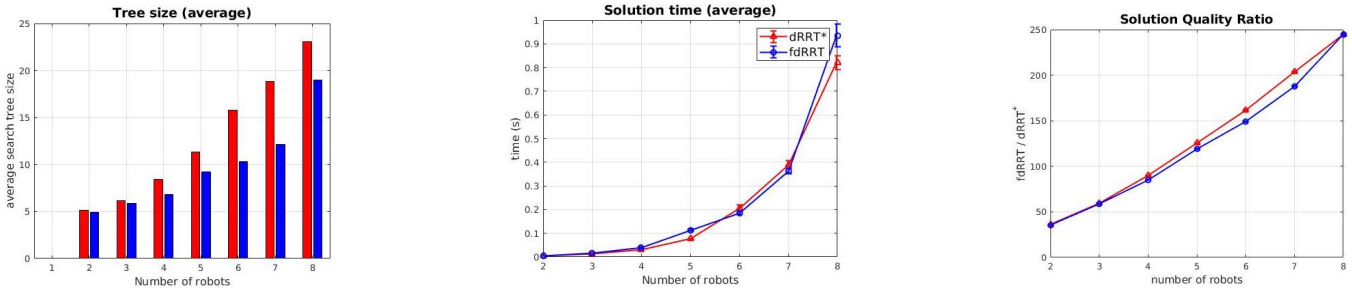
(a) (b) (c)

Fig. 3: Performance comparison in traffic intersection.



(a) (b) (c)

Fig. 4: Performance comparison in a warehouse space.



(a) (b) (c)

Fig. 5: Performance comparison in UAV environment.

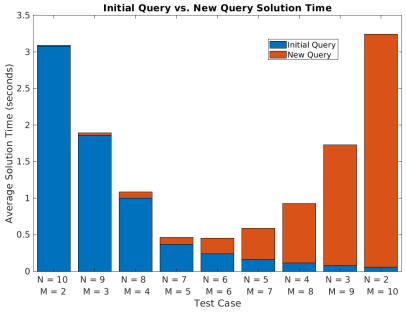


Fig. 6: A comparision of the computation time required for an initial plan of N robots and a new request as M robots enter the environment.

[15] J. P. van den Berg, J. Snoeyink, M. C. Lin, and D. Manocha, "Centralized path planning for multiple robots: Optimal decoupling into sequential plans," in *Robotics: Science and Systems*, 2009.

[16] J. J. B. Vial, W. E. Devanny, D. Eppstein, and M. T. Goodrich, "Scheduling autonomous vehicle platoons through an unregulated intersection," *CoRR*, vol. abs/1609.04512, 2016. [Online]. Available: http://arxiv.org/abs/1609.04512

[17] G. Wagner and H. Choset, "M*: A complete multirobot path planning algorithm with performance bounds," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2011, pp. 3260–3267.

[18] G. Wagner, Minsu Kang, and H. Choset, "Probabilistic path planning for multiple robots with subdimensional expansion," in *2012 IEEE International Conference on Robotics and Automation*, May 2012, pp. 2886–2892.

[19] L. Ye and T. Yamamoto, "Modeling connected and autonomous vehicles in heterogeneous traffic flow," *Physica A: Statistical Mechanics and its Applications*, vol. 490, pp. 269 – 277, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0378437117307392

[20] Yi Guo and L. E. Parker, "A distributed and optimal motion planning approach for multiple mobile robots," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, vol. 3, May 2002, pp. 2612–2619 vol.3.

[21] P. Švestka and M. H. Overmars, "Coordinated path planning for multiple robots," *Robotics and Autonomous Systems*, vol. 23, no. 3, pp. 125 – 152, 1998. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092188909700033X