Online Plan Repair in Multi-robot Coordination with Disturbances

Adem Coskun and Jason M. O'Kane

***Abstract*— This paper addresses the problem of multi-robot coordination in scenarios where the robots may experience unexpected delays in their movements. Prior work by Čáp, Gregoire, and Frazzoli introduced a control law, called RMTRACK, which enables robots in such scenarios to execute pre-planned paths in spite of disturbances in the execution speed of each robot, while guaranteeing that each robot can reach its goal without collisions and without deadlocks. We extend that approach to handle scenarios in which the disturbance probabilities are unknown at the start and non-uniform across the environment. The key idea is to 'repair' a plan on-the-fly, by swapping the order in which a pair of robots passes through a mutual collision region (i.e. a coordination space obstacle), when making such a change can be estimated to improve the overall performance of the system. We introduce a technique based on Gaussian Processes to estimate future disturbances, and propose two algorithms for testing, at appropriate times, whether a swap of a given obstacle would be beneficial. Tests in simulation demonstrate that our algorithm achieves significantly smaller average travel time than RMTRACK at only a modest computational expense.**

I. Introduction

As multi-robot systems become more reliable and more widespread, it is becoming increasingly common for those systems to share their operating environments with humans. Coordinating multi-robot systems in such environments can be a challenging problem for several reasons. One specific issue is that the motions of the robots may be interrupted or delayed by humans. In such a scenario, the robot may be prevented from progressing along its path for some period of time, an event we refer to as a *disturbance*.

Prior work by Čáp, Gregoire, and Frazzoli [2] showed how to handle these kinds of unexpected disturbances effectively, by introducing an approach that first generates a suite of trajectories that would be collision free in the absence of disturbances, and then controls the forward movements of the robots to ensure that the coordinated motions remain free of both collisions between robots and of deadlocks, even if some or all of the robot experience disturbances. The essential idea of the RMTRACK approach is for a robot to stop and wait, before entering a portion of its path on which a collision with another robot might occur, for which the other robot would have passed through this collision region first, according to the original (undisturbed) trajectories. See Figure 1.

The authors are with the Department of Computer Science and Engineering, University of South Carolina, 550 Assembly St, Columbia, SC 29201 USA. acoskun@email.sc.edu, jokane@cse.sc.edu. This material is based upon work supported by the National Science Foundation under Grants 1513203, 1526862, and 1659514.)

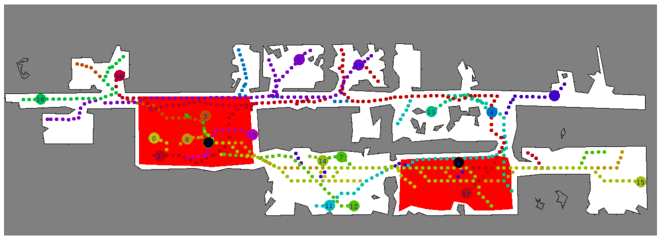

Fig. 1. An office environment in which 20 robots move. The robots are attempting to move from randomly generated origins to randomly generated destinations. The probability of having disturbances in the red zones are assigned as 0.7. The disturbance probability everywhere else is 0.05. (The map is based on laser rangefinder data from a building at the University of Bremen [9].)

The RMTRACK approach is very effective, especially in scenarios where the expected amount of disturbance experienced by each robot is approximately equal. The approach shows its limitations, however, in scenarios where the disturbance probabilities are unknown at the start and non-uniform across the environment. For example, consider the simple problem illustrated in Figure 2. Two robots attempt cross the room, one from left to right and the other from right to left. The robots are named R_1 and R_2 respectively. Without loss of generality, suppose that the initial planned trajectory instructs R_1 to pass through the narrow central region first. However, unbeknown to the robots and the trajectory planner, at the time of plan execution the left side of the room is filled with human workers that interrupt the motion of that robot. Then, R_2 reaches the collision region first. Using the RMTRACK algorithm, the robot R_2 would wait at that position until R_1 fully navigated the left side and passed through the collision region.

We propose to resolve these kinds of problems by repairing the plan on-the-fly. In the example of Figure 2, when robot the R_2 arrives the intersection and the robot R_1 has not yet cleared the intersection, the robot R_2 has a choice: Does it wait until the robot R_1 clears the intersection, or does it continue forward, hoping to pass through the collision region before the robot R_1 arrives? We propose a two-phase approach to answering this question: First we estimate the probability of each robot experiencing disturbances along the relevant section of its path, based on disturbances observed by the robots in those regions during the current execution. Then, using those estimated probabilities, the robot R_2 can determine whether it is likely to safely pass through the collision region before the arrival of the robot R_1. If so, we 'flip' that obstacle and resume the RMTRACK controller, at which point the robot R_2 will continue through the collision

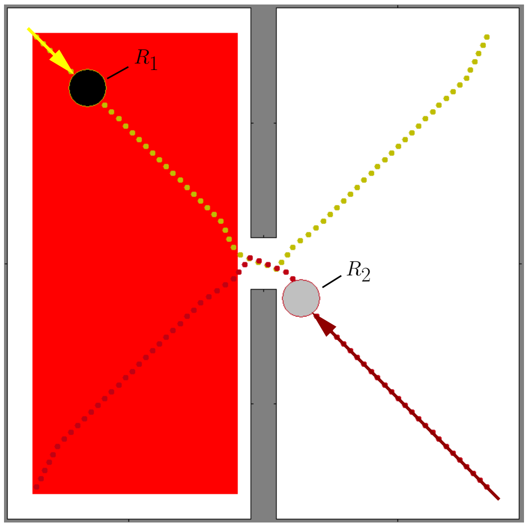

Fig. 2. Two robots, R_1 and R_2, are attempting to move from one side to another side of the environment. Dotted lines show the path of the robots, and arrows points from each robot's start position to current position. The probability of having disturbance in the red zones is 0.8.

region immediately.

The specific contributions of this paper are:

1) A formulation of the multi-robot coordination problem in which disturbances to the robots' progress are likely, but with probability distributed non-uniformly across the environment.
2) A pair of algorithms, designed to run in parallel with RMTRACK, for determining whether the pre-planned trajectory should be modified to allow a robot to pass through a collision region immediately, rather than waiting for another robot to cross first.
3) A series of simulations demonstrating that this approach offers a significant improvement in the robot's travel time compared to RMTRACK in some scenarios.

The remainder of this paper is organized as follows. First, we review some related work (Section II) and give a precise statement of the problem (Section III). Then we review the essential ideas behind RMTRACK (Section IV), describe our new algorithm that executes alongside RMTRACK (Section V), and present experiments demonstrating the improvements realized by our approach (Section VI). Concluding discussion appears in Section VII.

II. Related Work

The problem of coordinating multiple robots in a shared workspace is one of the best studied problems in the field. Approaches for this problem are generally classified as either decoupled or centralized. In decoupled methods, each robot's trajectory is computed individually, and robots resolve the collisions afterward [1], [15]–[17]. Though this approach is computationally efficient and practically applicable, because of the local view used for avoiding collisions, many such approaches can be susceptible to deadlocks.

In centralized approaches, a collision-free joint trajectory for all the robots is generated by a central decision maker. One well known technique in this category is prioritized planning [3], [7]. Geometry-based approaches [4], [5], [8], [11], [13], which typically leverage coordination diagrams to reason about possible collisions, are also common. However, the computational complexity of centralized methods often increases exponentially with the number of robots in the environment. In addition to centralized and decoupled approaches, there are also hybrid methods, which enjoy some of the benefits of both [6], [14], [12].

The most closely related work, that of Čáp, Gregoire, and Frazzoli [2] is specifically targeted to enable the reliable execution of centrally-generated joint plans, in cases where the robots cannot necessarily follow those paths without temporal disruption. In our work, we extend from that antecedent, considering a similar problem, but allowing the robots greater freedom to adjust the plan based conditions observed during the actual execution.

III. Problem Statement

This section formalizes the problem we address in this paper. The treatment is based upon, but generalizes, the model used by Čáp, Gregoire, and Frazzoli [2].

A. Environment, robots, and trajectories

We assume that n identical holonomic robots, indexed $1, \ldots, n$, operate in a shared 2d environment, $\mathcal{W} \subseteq \mathbb{R}^2$. The robots are disc-shaped with body radius r. We model time as a sequence of discrete stages indexed by $t \in \mathbb{N}$. Each robot starts at a *start position* and travels within $\mathcal{W}$ to a *goal position*. We assume that feasible collision-free trajectories for each robot, $\pi_1, \ldots, \pi_n$, from their respective start positions to their goals are generated by a multi-robot trajectory planner, such as prioritized planning [3]. Each trajectory $\pi_i : \{1, \ldots, K_i\} \rightarrow \mathcal{W}$ is a function mapping integers to locations in the environment, in which trajectory π_i for robot i has K_i steps. We model the robots' execution of these paths in discrete time, writing $x_i(t)$ to denote number of steps of π_i executed by robot i up to time t. If robot i experiences a disturbance or a delay in its execution, we will have $x_i(t) < t$. Thus, the actual position of robot i at time t is $\pi_i(x_i(t)) \in \mathcal{W}$.

B. Coordination spaces

For each pair of distinct robots (i, j), we define the *coordination space* $C_{ij} \subseteq C$ as

$$C_{ij} = \{(k_i, k_j) \mid ||\pi_i(k_i) - \pi_j(k_j)|| \geq 2r\}. \qquad (1)$$

The intuition is that a single point in C_{ij} is determined by the positions of robots i and j along their paths, and that pairs of positions that would place robot i in collision with robot j are excluded from the coordination space. See Figure 3 for an example. Within each coordination space C_{ij}, we can identify the obstacle region $O_{ij} = \{1, \ldots K_i\} \times \{1, \ldots, K_j\} - C_{ij}$. We partition O_{ij} into maximal connected regions, $o_1^{ij}, \ldots, o_m^{ij}$, so that $O_{ij} = o_1^{ij} \cup \cdots \cup o_m^{ij}$. Each o_k^{ij} is called a *coordination space obstacle*.

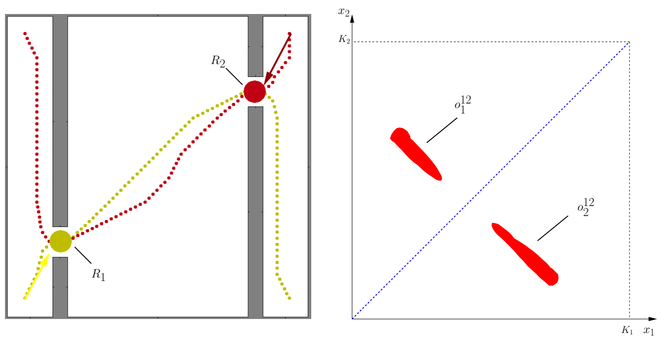

Fig. 3. Two robots, R_1 and R_2, in an environment where depicted on the left have two collision regions. The dotted lines are representing their paths. The coordination space of these two robots are depicted on the right with two obstacle, o_1^{12} and o_2^{12}. The blue dotted line represent the planned path. The label of $\ell(o_1^{12})$ is 1, which means R_1 should pass the obstacle o_1^{12} first. Also, the label of $\ell(o_2^{12})$ is 2, which means R_2 should pass the obstacle o_2^{12} first.

Each coordination space obstacle represents a region in the environment that both robots must pass through, but in which a collision may possibly occur if both robots occupy it at the same time. Notice the execution of robots i and j generates a path through C_{ij} from $(1,1)$ to (K_i, K_j). For each obstacle o_k^{ij}, this path must pass either above o_k^{ij} or below o_k^{ij}. The former case corresponds, in the workspace, to robot j passing through the collision region before robot i; in the latter case, robot i passes through the collision region before robot j.

In addition to the trajectories $\pi_1, \ldots, \pi_n$, we also assume that the trajectory planner assigns to each coordination space obstacle o_k^{ij} a *label* $\ell(o_k^{ij}) \in \{i, j\}$, indicating which of the two robots is planned to pass through the collision region first.

C. Commands and disturbances

At each time step, each robot may decide to attempt to either move forward along its path, or to voluntarily remain where it is. If the robot decides to move forward, that movement may be prevented by a *disturbance* of some kind from within the environment. We model these options using a control variable $a_i : \mathbb{N} \to \{0,1\}$ and a disturbance variable $\delta_i : \mathbb{N} \to \{0,1\}$ for each robot. Then each robot's progress through its path is governed by the transition equation

$$x_i(t+1) = x_i(t) + a_i(t)\delta_i(t). \qquad (2)$$

We assume that the disturbances are governed by some probability distribution that varies across $\mathcal{W}$, written as $p : \mathcal{W} \to [0,1]$, so that at any point $q \in W$, the probability of any robot experiencing a disturbance at position q is $p(q)$. The function p is unknown to both the robots and the trajectory planner. For simplicity, we assume that the same p governs the disturbances for all of the robots, and that p does not vary as time passes.

D. Objective

The goal is to establish an efficient control strategy for each robot i to select $a_i(t)$ at each time t. The control strategy should ensure that the robots do not collide with each other, and that all of the robots reach their goals, that is, there exist some time t such that $x_i(t) = K_i$ for all robots $i = 1, \ldots, n$. Given such a control strategy, we quantify its success in any particular execution by measuring the average travel time across all of the robots.

IV. Summary of RMTRACK

In this section, we summarize the existing RMTRACK algorithm, which executes in parallel with our algorithm, emphasizing the details that are important to understand how the complete system proposed in this paper operates. Our description of RMTRACK necessarily differs from that of Čáp, and Gregoire, Frazzoli because their formulation parameterizes the configuration space in a way that ensures that its diagonal is collision-free, which implies that the obstacle labels can be inferred by whether each obstacle is above or below the diagonal. Since we intend to modify the obstacle labels during execution, we introduce the following functionally equivalent presentation of RMTRACK.

The control law for RMTRACK, which for robot i at time t selects $a_i(t)$ is:

$$a_i(t) = \begin{cases} 0 & \exists j \neq i, \text{ s.t. } \exists k : \ell(o_k^{ij}) = j \text{ and} \\ & o_k^{ij} \cap (\{x_i(t)+1\} \times \{x_j(t), \ldots, K_j\} \neq \emptyset \\ 0 & \text{if } x_i(t) = K_i \\ 1 & \text{otherwise} \end{cases} \qquad (3)$$

The top portion of Figure 4 illustrates the intuition. If there exists at least one coordination space obstacle o_k^{ij} with robot j and representing a collision region that robot j should pass through first, that is, for which $\ell(o_k^{ij}) = j$, then robot i may need to wait for robot j to pass. To determine whether this is the case, we extend a line segment upward in C_{ij} from the next position along the path for robot i, and check whether this line segment intersects with o_k^{ij}. If so, then robot i should stop and wait for robot j to make some progress, ensuring that the robots' path passes above o_k^{ij} in C_{ij}. Naturally, when robot i has completed its path, that is, when $x_i(t) = K_i$, it should stop. If neither of these two stopping conditions holds, then robot i chooses $a_i(t) = 1$, attempting to make progress toward its goal.

V. Algorithm Description

This section describes our approach. The essential motivation can be seen in the top of Figure 4. In this example, robot j has experienced a lengthy disturbance, whereas robot i has been able to progress through its path steadily. Notice that the original, offline trajectory planner formed a global plan in which robot j should cross the collision region before robot i. This decision was reasonable in the absence of disturbances, but the reader will of course recall that the disturbance probabilities across the environment are unknown when the plan is generated. Consequently, as it executes the RMTRACK control law (Equation 3), robot i will reach the start of the collision region within its path, and then wait until robot j overcomes its disturbances to pass first.

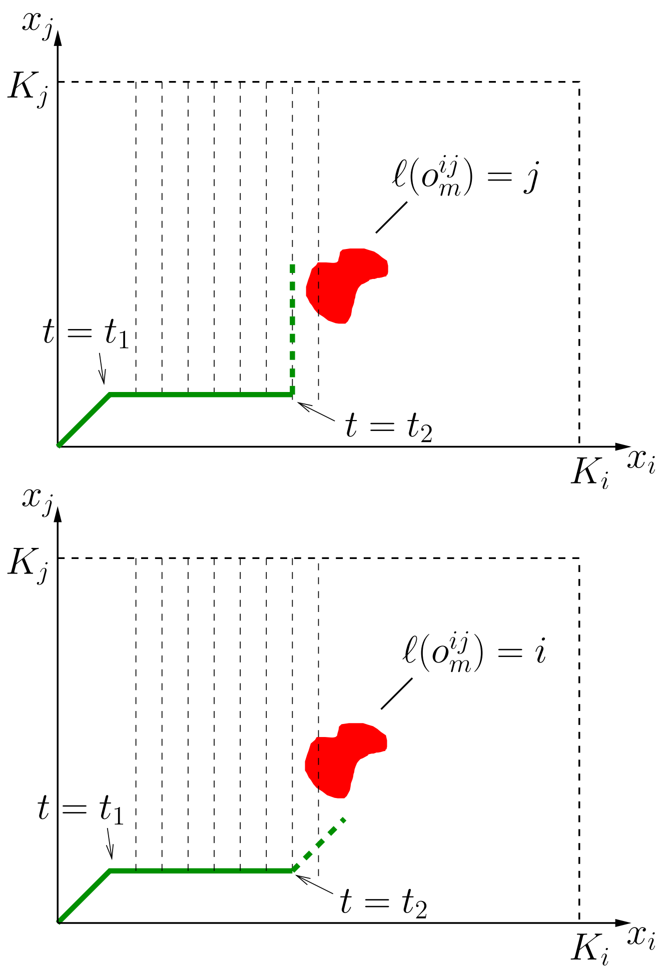

Fig. 4. An illustration of the behavior of RMTRACK. Robot i and robot j share a collision region in the coordination space C_{ij}. In this example, robot j begins to experience a lengthy disturbance starting at time t_1. The path through this coordination space until time t_2 is shown in green; the dotted green lines show possible future trajectories for the robots. The key question is: What should robot i do at time t_2? [top] If the obstacle o_k^{ij} has label $l(o_k^{ij}) = j$, then robot j is planned to pass through this collision region first. Equivalently, the coordination space path should travel over o_k^{ij}. Robot i must wait, choosing $a_i(t_2) = 0$. This continues, realized in C_{ij} as upward vertical movement, until robot j has advanced far enough to clear o_k^{ij}. [bottom] If the obstacle o_k^{ij} has label $l(o_k^{ij}) = j$, then robot i is planned to pass through this collision region first; the coordination space path should travel under o_k^{ij}. Robot i can continue immediately, choosing $a_i(t) = 1$, without regard to the progress of robot j.

One readily notices, however, that if robot j's progress has been slowed much more than that of robot i, then *robot i might attempt pass this collision region immediately*, thereby 'flipping' the coordination space obstacle from $l(o_k^{ij}) = j$ to $l(o_k^{ij}) = i$. The bottom part of Figure 4 illustrates the result of this change: Robot i continues to use RMTRACK to govern its movements, but because of the altered obstacle label, robot i can proceed immediately. By the time robot j finally reaches this region, it is likely that robot i will be safely out of the way.

The essence of our approach is to detect when opportunities for these kinds of on-the-fly changes to the coordination space obstacle labels may be beneficial to the overall performance of the system. We note that the alternative of simply re-executing the global trajectory planner in such situations is not generally a feasible option, since that sort of joint planning scales, as a general rule, quite poorly as the number of robots increases.

There are three essential elements to the approach: First, in Section V-A we consider the conditions under which the system should even consider an obstacle flip. Second, in Section V-B, we describe how the robots can estimate the disturbance probabilities based on their own observations of the disturbances they have experienced. Finally, in Section V-C we propose two algorithms for determining whether to actual execute the obstacle flip.

A. When to Check for Obstacle Flips

Before we address the question of how to determine *if* flipping an obstacle might be helpful, we first consider *when* during their execution the robots might reasonably consider this kind of change. Recall that the advantage of an obstacle flip derives from enabling a robot whose progress might have been delayed because of the first case in Equation 3 to proceed immediately instead of waiting for the other robot to pass a certain collision region. Thus, robot i performs a flip check at most once for each obstacle o_k^{ij}, specifically the first time that obstacle triggers the first condition in Equation 3.

B. Estimating the Disturbance Probabilities

Our goal is to change the label of a coordination space obstacle, allowing a robot to pass through without waiting, only when doing so is unlikely to delay the other robot. To make such a decision requires an estimate of the disturbance probability function p, at positions along each of the two robots' paths, from their current positions through to the end of the collision region. We write $\widehat{p} : W \to [0, 1]$ to denote this estimate of the disturbance probability.

The robots use their own observations of the actual disturbances, realized during the current execution, to compute $\widehat{p}$. Each robot keeps track of its last s time steps, in which s is a tunable parameter, and tracks both its position π_i and whether it experienced a disturbance δ_i, in those time steps. Based on those observations, robot i can compute a position-probability pair, which estimates the probability of a disturbance at the centroid of its positions across the last s time steps:

$$\left(\frac{1}{s} \sum_{i=0}^{s-1} \pi_i(x_i(t-i)), \frac{1}{s} \sum_{i=0}^{s-1} \delta_i(t-i) \right) \tag{4}$$

The system then uses these estimates of p at various places within W, to form its global estimate $\widehat{p}$, using a Gaussian Process regression model. We also use a k-means clustering approach to reduce the size of the observation set, to ensure that the Gaussian Process learning is completed efficiently. Figure 5 illustrates an example of this process.

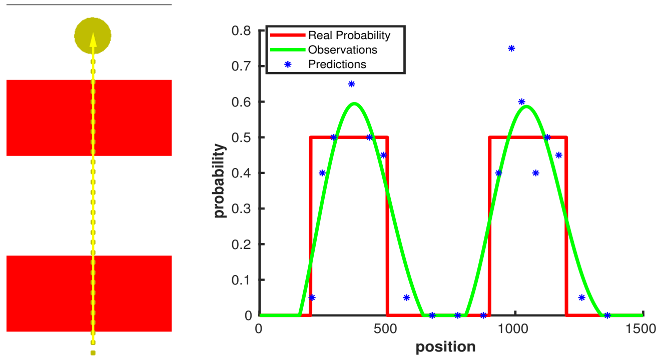

Fig. 5. [left] A robot moves through two regions, shown in red, in which the probability of disturbance is elevated. The robot does not know of these regions beforehand, and must estimate the disturbance probability based on its own experience of disturbances. [right] Results of the process of estimating the disturbance probability. Blue points mark the observations, computed via Equation 4. The green curve shows $\widehat{p}$, as computing via Gaussian Process regression over these observations. For comparison, the actual disturbance probability p is plotted in red. Note that this illustration shows only a one-dimensional slice of the estimated disturbance probability function $\widehat{p}$, along the robot's actual path. Our approach computes $\widehat{p}$ across the full 2-dimensional domain.

C. *Testing Whether Flipping an Obstacle Is Helpful*

Finally, we can establish conditions under which we expect the average travel time for the robots to benefit from changing the label of one of the obstacles on-the-fly. We propose two methods for this: The first method, TESTFLIPFAST (Section V-C.1) is very efficient, but overly conservative for some types of coordination space obstacles; the second, TESTFLIPAGGRESSIVE is more computationally expensive, but can identify flipping opportunities overlooked by the first method. Throughout this section, we consider the case in which robot i has begun to wait for robot j because of obstacle o_k^{ij} with label $l(o_k^{ij}) = j$; the question is whether to change this label to i.

1) TestFlipFast: Our first method decides to flip a coordination space obstacle if the expected time for robot i to clear o_k^{ij} —that is, to reach its right boundary in the coordination space— is less than the expected time for robot j to arrive at o_k^{ij}. The idea to making this approach efficient is to consider only the axis-aligned bounding box of o_k^{ij}, rather than its precise shape. This simplifying assumption means that we can consider the movements of robot i independently of those of robot j.

Specifically, we compute the expected time for robot i to clear o_k^{ij} using dynamic programming over the recurrence

$$E_i(k_i) \leftarrow \sum_{k_i' \in \{k_i, k_{i+1}\}} p(k_i') E_i(k_i'), \tag{5}$$

in which $E(k_i)$ denotes the expected time for robot i to clear the obstacle, starting from state k_i, amd $p(k_i')$ is the probability of robot i successfully moving forward in the next time step at that location, computed using the learned estimate $\widehat{p}$. We can compute values for E using the standard value iteration [10] algorithm. We then flip obstacle o_k^{ij} if $E_i(a) < E_j(b)$, in which a is the path step at which robot i clears the obstacles, and b is the path step at which robot j reaches the obstacle.

2) TestFlipAggressive: We also consider an alternative to TESTFLIPFAST, which considers the interactions between robot i and robot j as they travel near the obstacle. These kinds of interactions are important if, for example, the obstacle is long, narrow, and diagonal in the coordination space, as would occur if the paths for robot i and robot j travel in parallel for some distance. Using TESTFLIPFAST would be unlikely to flip such an obstacle, since the time at which robot i fully clears the obstacle will be far in the future.

To account for those kinds of interactions, we propose TESTFLIPAGGRESSIVE as an alternative. The core recurrence of TESTFLIPAGGESSIVE is

$$E_{ij}(k_i, k_j) \leftarrow \sum_{(k_i', k_j' \in S} p(k_i', k_j') E_{ij}(k_i', k_j'). \tag{6}$$

In the recurrence, $E_{ij}(k_i, k_j)$ represents the expected time for robot i to clear the obstacle, accounting for the fact that its motion by be delayed, according to Equation 3, waiting for robot j; S is the set of possible next states,

$$S = \{(k_i, k_j), (k_i+1, k_j), (k_i, k_j+1), (k_i+1k_j+1)\};$$

and the next state probabilities $p(k_i', k_j')$ are computed based on both estimated disturbance probability $\widehat{p}$, as well as the action variables a_i. That is, if robot i would choose $a_i(t) = 0$ at this situation, then $p(k_{i+1}', k_j') = p(k_{i+1}', k_{j+1}') = 0$. As with TESTFLIPFAST, we use value iteration to compute these values. However, because of the additional states (since we consider joint positions of both robot i and robot j, rather than robot i individually) and because of the time needed to evaluate Equation 3 at each point, this approach can be significantly slower than the expected time computation in FLIPCHECKFAST.

Finally, we can use these sorts of expected time computations to decide whether to flip obstacle o_k^{ij}. Because we want to consider the effects of this obstacle's label, we compute four different expected times:

- $E_{ij}(k_i, k_j)$ — the expected time for robot i to clear o_k^{ij}, using the current label.
- $\reflectbox{E}_{ij}(k_i, k_j)$ — the expected time for robot i to clear o_k^{ij}, using the opposite label, which it would acquire if it were flipped.
- $E_{ji}(k_j, k_i)$ — the expected time for robot j to clear o_k^{ij}, using the current label.
- $\reflectbox{E}_{ji}(k_j, k_i)$ — the expected time for robot j to clear o_k^{ij}, using the opposite label, which it would acquire if it were flipped.

Using these estimates of the consequences of an obstacle flip, we choose to carry out that flip if the anticipated benefit (that is, the anticipated reduction in travel time) for robot i outweighs the anticipated cost to robot j. That is, if

$$E_{ij}(k_i, k_j) - \reflectbox{E}_{ij}(k_i, k_j) < E_{ji}(k_j, k_i) - \reflectbox{E}_{ji}(k_j, k_i) \tag{7}$$

then we change the label of o_k^{ij}.

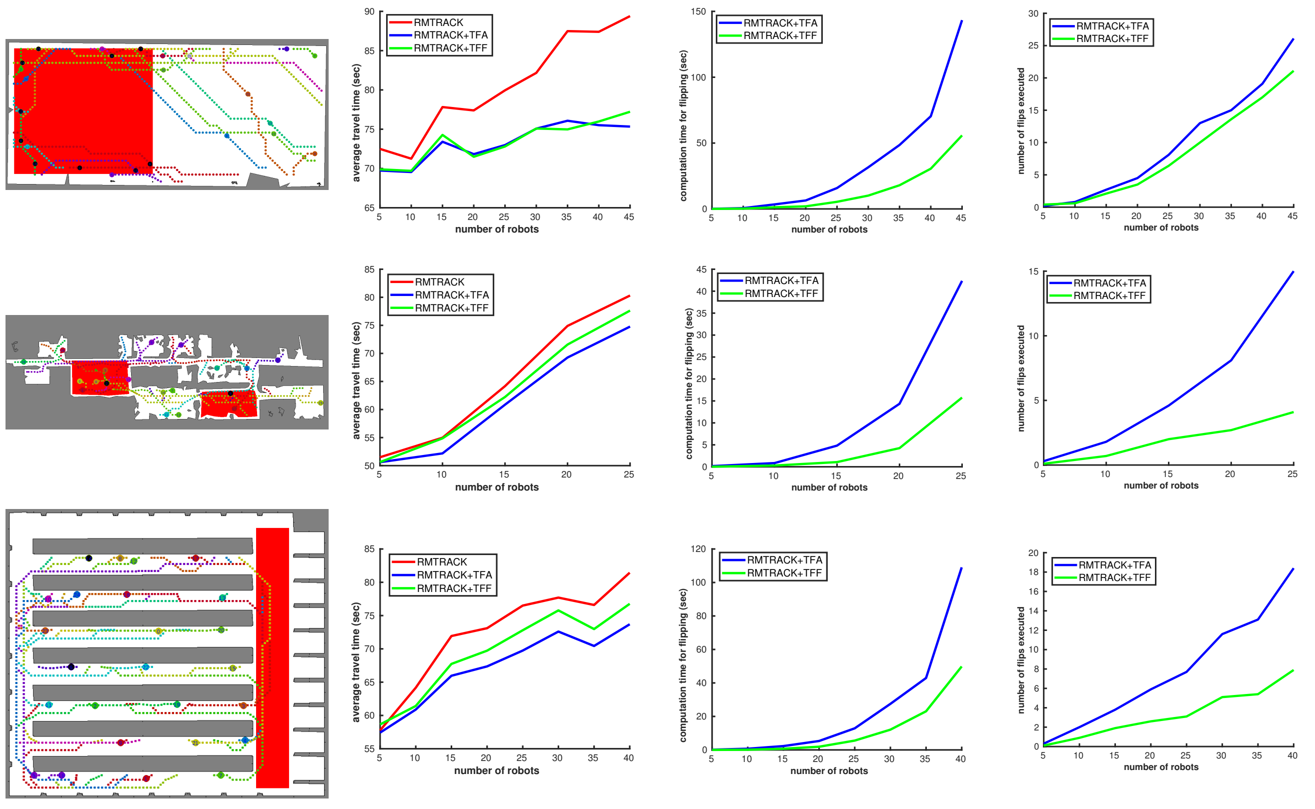

Fig. 6. Experimental Results

That completes our discussion of the approach. In summary, as the robots execute RMTRACK, the system attempts to identify times at which it can opportunistically modify the label of an obstacle, to repair the initial trajectory, to recover from large, unexpected disturbances with full replanning.

VI. Experimental Results

We have implemented these algorithms in Java, building upon the original RMTRACK implementation.[1] For Gaussian Process regression, we use the Statistical Machine Intelligence and Learning Engine (SMILE).[2] The experiments were conducted on an Ubuntu 16.04 computer with a 2.8GHz processor. We conducted experiments in three distinct environments, each with certain large regions designated as high-disturbance zones. The environments shown in the left column of Figure 6; the high disturbance zones are shown in red. The disturbance probability in these red zones are 0.8, 0.7, and 0.8 for each row, respectively. Outside the red zones, the disturbance probability is 0.05.

For each environment, we varied the number n of robots in increments of 5 and selected random starting and goal positions for each robot. For each n, we conducted ten trials, executing three algorithms for each configuration of state and goal positions: (1) vanilla RMTRACK, (2) RMTRACK with obstacle flipping via TestFlipFast (RMTRACK+TFF), and (3) RMTRACK with obstacle flipping via TestFlipAggressive (RMTRACK+TFA). For the latter two, we used $s = 20$ when generating observations for Gaussian process regression.

Under each algorithm, the robots were able to complete each trial successfully. For each algorithm, we measured the average completion time. These results appear in the second column of Figure 6. We observe that, for these problem instances, both obstacle flipping approaches can generate sizable decreases to the average travel time for the robots. For the obstacle flipping algorithms, we also measured the amount of computation time consumed by the TestFlip algorithms. From these results, which are in the third column of Figure 6, we conclude that this computation is nearly negligible, in comparison to the savings in robot travel time. Finally, we counted the number of flips executed by each approach, as shown in the right column of Figure 6. Those results confirm that the extra computation time invested in TestFlipAggressive does indeed identify larger numbers of opportunities to flip the obstacle labels.

VII. Conclusion

This paper presented a technique for online repair of multiple-robot coordination plans. The idea is to start from a planned joint trajectory for the robots, but to adjust that path by 'flipping' the order in which pairs of robots should pass through their shared collision regions. These decisions are made on-the-fly, without full replanning, in response to unexpected disturbances in the execution speed of some of the robots along their paths. In future work, we plan to consider a decentralized version of this problem, in which robots have limited information about the progress made by the other robots must nevertheless decide how to proceed.

[1]`https://github.com/mcapino/rmtrack`
[2]`http://haifengl.github.io/smile/`

References

[1] Javier Alonso-Mora, Pascal Gohl, Scott Watson, Roland Siegwart, and Paul Beardsley. Shared control of autonomous vehicles based on velocity space optimization. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 1639–1645. IEEE, 2014.

[2] Michal Čáp, Jean Gregoire, and Emilio Frazzoli. Provably safe and deadlock-free execution of multi-robot plans under delaying disturbances. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 5113–5118. IEEE, 2016.

[3] Michal Čáp, Peter Novák, Alexander Kleiner, and Martin Seleckỳ. Prioritized planning algorithms for trajectory coordination of multiple mobile robots. *IEEE transactions on automation science and engineering*, 12(3):835–849, 2015.

[4] Hamid Chitsaz, Steven M. LaValle, and Jason M. O'Kane. Exact Pareto-optimal coordination for two translating polygonal robots on a cyclic roadmap. In *Proc. Canadian Conference on Computational Geometry*, 2008.

[5] Hamid Chitsaz, Jason M. O'Kane, and Steven M. LaValle. Exact Pareto-optimal coordination for two translating polygonal robots on an acyclic roadmap. In *Proc. IEEE International Conference on Robotics and Automation*, pages 3981–3986, 2004.

[6] Christopher M Clark, Stephen M Rock, and J-C Latombe. Motion planning for multiple mobile robots using dynamic networks. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 3, pages 4222–4227. IEEE, 2003.

[7] Michael Erdmann and Tomas Lozano-Perez. On multiple moving objects. *Algorithmica*, 2(1-4):477, 1987.

[8] Robert Ghrist, Jason M. O'Kane, and Steven M. LaValle. Computing pareto optimal coordinations on roadmaps. *International Journal of Robotics Research*, 24(11):997–1010, November 2005.

[9] Andrew Howard and Nicholas Roy. The robotics data set repository (radish), 2003. *URL http://radish. sourceforge. net*, 30, 2015.

[10] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[11] Stephane Leroy, Jean-Paul Laumond, and Thierry Siméon. Multiple path coordination for mobile robots: A geometric algorithm. In *IJCAI*, volume 99, pages 1118–1123, 1999.

[12] Tsai-Yen Li and Hsu-Chi Chou. Motion planning for a crowd of robots. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 3, pages 4215–4221. IEEE, 2003.

[13] Tomas Lozano-Perez et al. Deadlock-free and collision-free coordination of two robot manipulators. In *1989 IEEE International Conference on Robotics and Automation*, pages 484–489. IEEE, 1989.

[14] Mike Peasgood, Christopher Michael Clark, and John McPhee. A complete and scalable strategy for coordinating multiple robots within roadmaps. *IEEE Transactions on Robotics*, 24(2):283–292, 2008.

[15] Jamie Snape, Stephen J Guy, Jur Van Den Berg, and Dinesh Manocha. Smooth coordination and navigation for multiple differential-drive robots. In *Experimental Robotics*, pages 601–613. Springer, 2014.

[16] Jur Van Den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics research*, pages 3–19. Springer, 2011.

[17] Jur Van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 1928–1935. IEEE, 2008.