IFAC Journal of Systems and Control 15 (2021) 100126

Contents lists available at ScienceDirect

IFAC Journal of Systems and Control

journal homepage: www.elsevier.com/locate/ifacsc

Multi-agent motion planning with sporadic communications for collision avoidance

Beth Boardman [a,*], Troy Harden [a], Sonia Martínez [b]

[a] Los Alamos National Laboratory, PO Box 1663, MS J580, Los Alamos, NM 87545, United States of America
[b] Mechanical and Aerospace Engineering, University of California, San Diego, 9500 Gilman Dr, La Jolla, CA 92093-0411, United States of America

ARTICLE INFO

Article history:
Received 6 November 2018
Received in revised form 17 December 2019
Accepted 11 November 2020
Available online 26 November 2020

Keywords:
Motion planning
Collision avoidance
Event-triggered
Multi-agent

ABSTRACT

A novel multi-vehicle motion planning and collision avoidance algorithm is proposed and analyzed in this paper. The algorithm aims to reduce the amount of onboard calculations and inter-agent communications needed for each vehicle to successfully navigate through an environment with static obstacles and reach their goals. To this end, each agent first calculates a path to the goal by means of an asymptotically optimal rapidly-exploring random tree (RRT*) with respect to the static obstacles. Then, other agents are treated as dynamic obstacles and potential collisions are determined by means of collision cones. Collision cones depend on the position and velocity from other agents and are grown conservatively between inter-agent communications. Based on the available information, each agent determines if a deconfliction maneuver is needed, if it can continue along its current path, or if communication is needed to make a decision about a conflict. With probability one, our algorithm guarantees that the agents keep from colliding with each other. Under an assumption on the existence of a solution for a vehicle to its goal, this algorithm also solves the planning problem with probability one. Simulations illustrate a group of agents successfully reaching their goal configurations and examine how the uncertainty affects the communication frequency of the multi-agent system.

1. Introduction

Much work has been devoted to the design and analysis of planning and collision avoidance algorithms for multiple robots in dynamic environments, (Bishop, 2000; Frazzoli, Dahleh, & Feron, 2002; Van Den Berg, Guy, Lin, & Manocha, 2011). With the advent of autonomous vehicles and self-driving cars, this problem has acquired new relevance since these algorithms can ensure the safe co-existence of vehicles and humans, in shared domains. In simple terms, N agents must traverse the environment from their initial configurations to their goal configurations without colliding with static obstacles or the other agents. Individual paths may be at odds with other agents. To remedy this, collision avoidance maneuvers are performed which require globally and continuously available information. Motivated by recent advances in event-based planning, we present in this paper an algorithm that, by exploiting sampling-based motion planners, can guarantee collision avoidance with reduced communications and address the planning objective in obstacle environments.

Literature Review. In static environments, sampling-based planners include Rapidly-exploring Dense Tree algorithms (RDTs, also known as RRTs) (LaValle, 2006) and Sampling-Based Roadmaps (SBRs, including Probabilistic Roadmaps (PRMs) (Kavraki, Svestka, Latombe, & Overmars, 1996)). These planners are probabilistically complete and return a feasible solution quickly. For a slight increase in runtime, the RRT* planner (Karaman & Frazzoli, 2011) returns a path that is asymptotically optimal. The RRT# (Arslan & Tsiotras, 2013) and Fast Marching Tree (FMT*) (Janson, Schmerling, Clark, & Pavone, 2015) are two other sampling-based planners that return an asymptotically optimal path. Our previous paper, Boardman, Harden, and Martínez (2014), introduces the Goal Tree (GT) algorithm for replanning in environments with unexpected static obstacles and maintains an RRT* that is trimmed and extended based on the new obstacle information.

The following selection of works deal with multiple agents in a workspace. A first set of papers solve the multi-agent path planning problem without building a roadmap or considering uncertainty: The algorithm in Biswas, Anavatti, and Garratt (2016) is based on vectorized particle swarm optimization. A real-time algorithm is presented in Chen, Cutler, and How (2015) that uses sequential convex programming. A dynamic programming scheme is used in Luo, Chakraborty, and Sycara (2016) to develop an online coordinated motion planner. While the algorithms in Murano, Perelli, and Rubin (2015) and Sharon, Stern, Goldenberg, and Felner (2013) use roadmaps, they do not use

* Corresponding author.
E-mail addresses: bboardman@lanl.gov (B. Boardman), harden@lanl.gov (T. Harden), soniamd@ucsd.edu (S. Martínez).

https://doi.org/10.1016/j.ifacsc.2020.100126
2468-6018/

sampling-based motion planners. The RRT* is used in the Multi-agent Poli-RRT* in Ragaglia, Prandini, and Bascetta (2016), which uses a hierarchy method to find collision free paths.

Most similar to this paper are Kothari and Postlethwaite (2013), Neto, Macharet, Chaimowicz, and Campos (2013) and Virani and Zhu (2016), which use RRT based planners with uncertainty. More precisely, a motion planning problem with uncertain obstacles is solved using a game theoretic formulation in Virani and Zhu (2016). Here, agents are non-cooperative, and they need to estimate others' dynamics using sensor data. This results in a computationally-involved differential-game formulation whose solution is based on RRT*. Sequential path finding is used in Neto et al. (2013) to develop a real-time algorithm that handles uncertainties and disturbances. The main emphasis of this work is on how to recompute and adapt the non-optimal RRT trees efficiently. Simulations are employed to verify the results. The real-time algorithm for uncertain dynamic obstacles in Kothari and Postlethwaite (2013) uses chance constraints. In particular, Kothari and Postlethwaite (2013) makes use of the chance-constraint framework to guarantee collision avoidance with dynamic obstacles with large probability. As is usual in the literature, Kothari and Postlethwaite (2013) requires agents continuously monitor the other agents.

This paper pertains to multiple agents in the same environment moving to their individual goal configurations. Each of the N agents builds an RRT* that is used to determine the best path to its goal from all other configurations. While executing the current best path, the agent checks for conflict with the other agents in the environment. When a conflict is determined, the agent performs a deconfliction maneuver using collision cones based on the Distributed Reactive Collision Avoidance (DRCA) from Lalish (2009). The deconfliction maneuver updates the agent's velocity so that it is not in conflict with any of the other agents. This new velocity will lead the agent to a nearby node in the graph. The agent then updates its current best path and continues to travel along repeating the above process. The algorithm is first given for perfectly known position and velocity of the other agents. The algorithm is then extended to handle uncertainty in the knowledge of other agents' positions and velocities.

A main advantage to our algorithms are the agents' avoidance of static obstacles in a preprocessing step, thanks to the use of an optimal roadmap or RRT*. The avoidance of static obstacles rarely requires additional computation while the agents are moving. A second main distinction with the previous literature is given by the sporadic communication algorithm, which requires communications among agents only when necessary, and (a) guarantees deconfliction, and (b) does not require detailed dynamic models of other agents or learning them, as they are fully cooperative. Other minor benefits are that conflicts are simple to calculate and, when needed, the algorithm prioritizes agents automatically avoiding deadlocks.

We analyze the algorithm to prove that the agents will never collide with one another or static obstacles. A proof is given for both the certain and uncertain algorithm. We also prove that, under an assumption about the existence of a collision free path to the goal, the agents will eventually reach their goal configurations. The simulations show that no collisions occur. The simulations also examine the amount of uncertainty seen by the agents. Two additional simulations show five agents reaching their goal configurations successfully in a more complex environment.

This paper is organized as follows. First, the problem is formulated with background information on the RRT* and collision cones given in Section 2. Next, Section 3 details our algorithm. The algorithm is analyzed in Section 4 and simulation results are presented in Section 5. The paper concludes with Section 6.

2. Problem formulation and background

The multi-agent path planning problem is formulated below. Then, the relevant background pertaining to the asymptotically optimal Rapidly-exploring Random Tree (RRT*) and collision avoidance using collision cones is outlined.

Let there be N agents, each with the same continuous configuration space, $X_i = X_j = X \subseteq \mathbb{R}^2$, where $i, j \in \{1, \ldots, N\}$. The static obstacles, but not the other agents, are accounted for in the agents' obstacle space, $X_{\text{obs},i} = X_{\text{obs}}$. Then, the free space is, $X_{\text{free}} = X \setminus X_{\text{obs}}$. Agent i has access to its world frame position, $\mathbf{r}_i \in X_{\text{free}}$, and velocity, $\mathbf{v}_i \in \mathbb{R}^2$. The physical volume of each agent is approximated as a two-dimensional Euclidean ball centered at $\mathbf{x}_i \in X_{\text{free}}$ with radius ρ_i, $\mathbb{B}(\mathbf{x}_i, \rho_i)$. The vector of zeros in $\mathbb{R}^2$ is denoted as $\mathbf{0}$. The two-dimensional rotation matrix is defined as $\mathscr{R}(\theta) \subset \mathbb{R}^{2\times 2}$. Finally, the 2-norm is $\| \begin{bmatrix} x & y \end{bmatrix}^\top \| = \sqrt{x^2 + y^2}$.

The N agents are tasked with getting from their initial configurations, $\mathbf{x}_{I,i} \in X_{\text{free}}$, to their goal configurations, $\mathbf{x}_{G,i} \in X_{\text{free}}$. The agents must never collide with other agents or static obstacles. We say that, two agents $i, j \in \{1, \ldots, N\}$ are in *conflict* if, by staying on their current trajectory (heading and speed), they eventually collide. *Deconfliction* is the act of an agent changing its velocity to avoid future collision.

In the next sections, we first provide an initial solution to this problem that requires a synchronous information exchange between all agents by sending and receiving position and velocity information at the same times. Then, we study how to relax the communication requirements by means of an opportunistic approach, which reduces the communication frequency by only requesting the position and velocity from a fellow agent when certain conditions are met. The agents request information at different times. This can be understood as a type of controlled asynchrony: communication only happens when necessary. These conditions ensure that the agents remain collision free.

2.1. The asymptotically optimal rapidly-exploring random tree algorithm

This section briefly describes the RRT* algorithm by Karaman and Frazzoli which is theoretically analyzed in Karaman and Frazzoli (2011). The kinodynamic RRT* is presented in Karaman and Frazzoli (2010).

The RRT*, outlined in Algorithm 1, builds a tree, $\mathscr{T}$ which is dense with probability one in the entire configuration space, X, as the number of samples, n, goes to infinity. The tree is composed of a set of vertices, $v \in \mathscr{T}.V$, and edges, $e \in \mathscr{T}.E$. Each edge is an ordered pair of vertices $e_{1,2} = (v_1, v_2)$, where v_1 is the parent and v_2 is the child. We use Cost as the notation for the cost function being minimized. Each edge added to $\mathscr{T}$ has a cost associated with it, denoted $c_{\text{edge}}(e)$, where $e \in \mathscr{T}.E$. In the original work by Karaman and Frazzoli (2011), the edge cost considered is the *cost-to-go*; that is the cost of $e_{1,2} = (v_1, v_2)$ is the cost of moving from the parent v_1 to the child v_2. Here, we consider the *cost-to-come*, the cost of moving from the child to the parent. Then, the cost-to-come of a vertex, $\text{Cost}(v)$, is the sum of the costs of the edges connecting the root to v. The paths in $\mathscr{T}$ are asymptotically optimal, meaning that as $n \to \infty$ the optimal path from any configuration in X_{free} to the goal configuration, $\mathbf{x}_G \in X_{\text{free}}$, is recovered. The functions involved in the RRT* process are described as follows. With some abuse of notation, we will define a robot configuration as $\mathbf{x}_v$ instead of v.

After initializing $\mathscr{T}$ at $\mathbf{x}_G$, the RRT* begins by using the Sample function to output $\mathbf{x}_{\text{rand}}$, a uniformly sampled random configuration from X_{free}. The Nearest function finds the nearest vertex, $\mathbf{x}_{\text{nearest}} \in \mathscr{T}$, and extends $\mathscr{T}$ a distance ε from $\mathbf{x}_{\text{nearest}}$ to get $\mathbf{x}_{\text{new}}$.

Next, the set of near vertices from $\mathscr{T}$ with respect to $\mathbf{x}_{\text{new}}$ are output as the set X_{near} from the function Near. Vertices that

are farther than $\delta = \min(\varepsilon, \gamma(\log(n_v)/n_v)^{(1/d)})$, where n_v is the number of vertices in $\mathscr{T}$, d is the dimension of the configuration space (in this case 2), and γ is an independent parameter, are omitted from X_{near}. The best parent for $\mathbf{x}_{\text{new}}$, determined in FindBestParent, is the vertex in X_{near} that has a collision-free path from $\mathbf{x}_{\text{new}}$ to $\mathbf{x}_{\text{near}}$, with the lowest Cost($\mathbf{x}_{\text{new}}$). The paths that connect the vertices to each other (determined using Steer), do so according to the system dynamics. Only collision-free edges are added to $\mathscr{T}$. The collision checker, CollisionCheck, returns true if the edge is collision-free. If $\mathbf{x}_{\text{new}}$ is added to $\mathscr{T}$, then Rewire attempts to add the other vertices in X_{near} as children of $\mathbf{x}_{\text{new}}$ based upon a lower cost and collision-free edge.

Algorithm 1 $\mathscr{T} = (V, E) \leftarrow \text{RRT}^*(\mathbf{x}_I, \mathbf{x}_G, \varepsilon)$

```
T ← InitializeTree();
T ← InsertNode(∅, x_G, T);
for i = 1 to i = N do
    x_rand ← Sample(i);
    x_new ← Nearest(T, x_rand, ε);
    X_near ← Near(T, x_new);
    x_parent ← FindBestParent(X_near, x_new);
    if x_parent ≠ NULL then
        T ← InsertNode((x_parent, x_new), x_new, T);
        T ← Rewire(T, X_near, x_new);
    end if
end for
π ← GetPath(T, x_I, x_G);
return T
```

2.2. Collision cones for deconfliction

This section of the paper is based on the collision cones and collision avoidance strategy in Lalish (2009). Collision cones are used to determine and resolve conflict between agents. Recall that each agent is approximated as a circle with radius $\rho_i \in \mathbb{R}_{>0}$ and let $d_{\text{sep},0}$ be the minimum allowable separation distance between any two agents. Then, agents i and j can be approximated as points that are separated by a minimum distance of $d_{\text{sep},ij} = \rho_i + \rho_j + d_{\text{sep},0}$. Denote the positions of agents i and j with respect to a world frame as $\mathbf{r}_i$ and $\mathbf{r}_j$, and similarly, their velocity vectors $\mathbf{v}_i$ and $\mathbf{v}_j$, respectively. The relative position vector between agent i and j is defined as $\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i$ and the relative velocity vector is defined as $\mathbf{v}_{ij} = \mathbf{v}_i - \mathbf{v}_j$. The speed of agent i is denoted as $v_i = \|\mathbf{v}_i\|$.

A depiction of a collision cone is in Fig. 1. The collision cone half angle is $\alpha_{ij} = \arcsin\left(\frac{d_{\text{sep},ij}}{\|\mathbf{r}_{ij}\|}\right)$. The angle between the relative position and relative velocity vectors is $\beta_{ij} = \arccos\left(\frac{\mathbf{r}_{ij}\cdot\mathbf{v}_{ij}}{\|\mathbf{r}_{ij}\|\|\mathbf{v}_{ij}\|}\right)$. Note that α_{ij} and β_{ij} are functions of the $\mathbf{r}_{ij}$ and of $\mathbf{v}_{ij}$, respectively. The two edges of the collision cone between agents i and j are defined using the unit vector $\hat{\mathbf{c}}_{ij}^{\pm} = \mathscr{R}(\pm\alpha_{ij})\frac{\mathbf{r}_{ij}}{\|\mathbf{r}_{ij}\|}$, where $\mathscr{R}(\pm\alpha_{ij})$ are rotations about the cone axis. Note that the half angle $\alpha_{ij} = \alpha_{ji}$ and that each cone contains the circle centered at agent j's position with radius $d_{\text{sep},ij}$. The following proposition characterizes conflicts between a pair of agents (which travel with a constant velocity and direction) in terms of their collision cone.

Proposition 1 (*Lalish, 2009*). *Define a cone between two agents as described above, with parameters* $\beta_{ij} = \angle\mathbf{v}_{ij} - \angle\mathbf{r}_{ij}$, $\alpha_{ij} = \arcsin\left(\frac{d_{\text{sep},ij}}{\|\mathbf{r}_{ij}\|}\right)$. *Let* $\mathbf{r}_{ij}$ *be the relative position vector at the time conflict that is being checked. A necessary and sufficient condition for i and j not to be in conflict is* $|\beta_{ij}| \geq \alpha_{ij}$.

In other words, from Proposition 1, we define a conflict between agents i and j if and only if $|\beta_{ij}| < \alpha_{ij}$.

While moving in the environment, agent i determines the collision cones between itself and all other agents $j \neq i$. When detecting a conflict with at least one other agent, agent i applies a deconfliction maneuver, as discussed in Section 3, which moves agent i out of conflict.

3. Sampling-based collision avoidance

This section combines the RRT* algorithm, Section 2.1, with the collision cone deconfliction strategy of Section 2.2. First, we present an algorithm that requires perfect and continuous communication among all agents. In order to reduce communications among agents, we develop an alternative algorithm in Section 3.2. The sporadic communication introduces uncertainty on the other agents which is handled by dynamically growing the collision cones.

Both algorithms are run independently on each agent, therefore, we describe them from the point of view of agent i. Agent i uses its motion planning tree $\mathscr{T}_i$ to determine and update its path to the goal which avoids the static obstacles. The collision cones are used for collision avoidance between agents. Below, *information* is used in reference to position and velocity knowledge of the other agents.

In an initialization step, all agents are given the maximum velocity and the approximation radius, ρ_i, of all other agents. In a preprocessing step to the algorithm, agent i recovers its best path to the goal from $\mathscr{T}_i$, denoted as $\pi_i = \{\mathbf{x}_i(t_1) = \mathbf{x}_{I,i}, \mathbf{x}_i(t_{i_2}), \ldots, \mathbf{x}_i(t_{i_m}), \ldots, \mathbf{x}_i(t_{i_f}) = \mathbf{x}_{G,i}\}$, where $m \in \mathbb{N}$. Note that each agent may reach its specific goal at different times, t_{i_f}. Once at the goal, the agent sets its velocity to zero, $\mathbf{v}_i = \mathbf{0}$.

To reach $\mathbf{x}_{G,i}$ while avoiding collision, agent i follows a path generated from an RRT*, which minimizes the distance traveled. At each node in the path, the agent checks for conflict with the other agents. If there is no conflict, then the agent continues onto the next edge in the path. When in conflict, the agent updates its velocity as described in Section 2.2, minimizing the distance traveled. The final executed path for agent i, Π_i, is in X_{free} and collision free with respect to the other agents.

3.1. Continuous information case

The pseudo-code for the Sampling-Based Collision Avoidance algorithm with continuously received information is presented in Algorithm 2. First, the agent extracts a path to the goal. While the agent is not at the goal and no conflicts are detected, it moves along its path. At each node in the path, the agent employs the current information received from the other agents. If a conflict is detected, that is, $\beta_{ij} < \alpha_{ij}$, for some other j, then the agent updates its own velocity and obtains a new path to the goal. The details of each step are shown below.

As agent i travels along its path, it checks for conflict with the other agents $j \neq i$. For each node in the path π_i, agent i knows its current time t_{i_m}, position, $\mathbf{r}_i = \mathbf{x}_i(t_{i_m}) \in \pi_i$, and velocity $\mathbf{v}_i = v_i\frac{\mathbf{x}_i(t_{i_{m+1}}) - \mathbf{x}_i(t_{i_m})}{\|\mathbf{x}_i(t_{i_{m+1}}) - \mathbf{x}_i(t_{i_m})\|}$. The time at the next node is $t_{i_{m+1}}$ with $dt = t_{i_{m+1}} - t_{i_m}$. Agent i receives the current position, $\mathbf{x}_j(t_{i_m})$, and velocity, $\mathbf{v}_j(t_{i_m})$, from all agents $j \neq i$.

Once agent i has all the information for t_{i_m}, it calculates the collision cones. When in conflict, a new velocity vector, $\mathbf{v}_i'$, is calculated by solving Problem 1, which determines a deconfliction maneuver that minimizes their cost-to-come Cost from the new position to the goal configuration.

Algorithm 2 $\Pi_i \leftarrow$ Sampling − BasedCollisionAvoidance

$[\mathscr{T}, \pi_i] \leftarrow \text{RRT}^*(\mathbf{x}_{I,i}, \mathbf{x}_{G,i}, \varepsilon)$;
$\mathbf{r}_i = \mathbf{x}_{I,i}$;
Π_i.add($\mathbf{r}_i$);
while $\mathbf{r}_i \neq \mathbf{x}_{G,i}$ **do**
  $\mathbf{r}_i, \mathbf{v}_i \leftarrow$ Move along path π_i to next node
  $\mathbf{r}_j, \mathbf{v}_j \leftarrow$ Communicate with ALL agents
  if Conflict ($\beta_{ij} < \alpha_{ij}$) with agent j **then**
    $\mathbf{v}_i \leftarrow$ Update velocity via maneuvers, Problem 1;
    if $\mathbf{r}'_i \in X_{\text{obs},i}$ **then**
      Add obstacle to agent list
      Restart while iteration
    else
      $\pi_i \leftarrow \text{GetPath}(\mathscr{T}, \mathbf{r}_i, \mathbf{x}_{G,i})$;
    end if
  end if
  Π_i.add($\mathbf{r}_i$);
  Remove obstacle from agent list
end while
$\mathbf{v}_i = \mathbf{0}$;
return Π_i

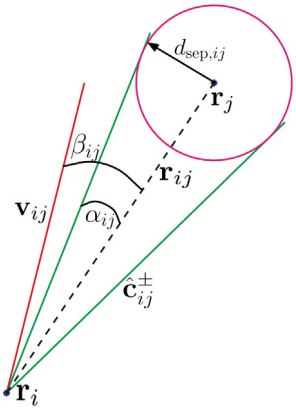

Fig. 1. Depiction of a collision cone.

Problem 1.

$$\begin{aligned} &\min \quad \text{Cost}(\mathbf{r}'_i) \\ \text{s.t.} \quad &\text{Cost}(\mathbf{r}_i) \geq \text{Cost}(\mathbf{r}'_i) \\ &\mathbf{r}'_i \in X_{\text{near}} \triangleq \{\mathbf{x} \in \mathscr{T}.V \mid \frac{\|\mathbf{x} - \mathbf{r}_i\|}{dt} \leq v_{i,\max}\} \\ &\alpha_{ij}(\mathbf{r}'_{ij}) < \beta_{ij}(\mathbf{r}'_{ij}, \mathbf{v}'_{ij}), \qquad \text{for all } j \neq i \end{aligned}$$

where $\mathbf{r}'_{ij} = \mathbf{r}_j - \mathbf{r}'_i$ and $\mathbf{v}'_{ij} = \mathbf{v}'_i - \mathbf{v}_j$, with $\mathbf{v}'_i = \frac{\|\mathbf{r}'_i - \mathbf{r}_i\|}{dt}$

Agent i determines the cost-to-come for each configuration in the finite set X_{near}. If that cost is lower than, or equal to, the current cost, then the possible new position and velocity are checked for conflict. Whichever conflict-free configuration lowers the cost the most is chosen as the next position of agent i. Note that the agent remaining at its current position, $\mathbf{r}'_i = \mathbf{r}_i$, is always a valid solution to Problem 1 since $\mathbf{r}_i$ is in X_{near} and the cost-to-come remains the same, $\text{Cost}(\mathbf{r}_i) = \text{Cost}(\mathbf{r}'_i)$. The agent will remain collision free since the other agents will update their velocities to be conflict free.

To reduce computation time, the collision cones do not account for the static obstacles, therefore the edge to the next position, $\mathbf{r}'_i$, is checked for collision with known obstacles from X_{obs}. If the edge is in collision, then the offending obstacle is treated as an agent with zero velocity, $\mathbf{v}_{\text{obs}} = \mathbf{0}$. Agent i then recalculates an updated velocity vector. If there is not a feasible new velocity vector, agent i stops. Agent i can move again after, at least one other agent has changed its velocity vector and it finds a conflict free velocity vector.

Each conflict updates the path to the goal, π_i. As discussed in Section 4, under some conditions, the deconfliction maneuver ensures that the agents never collide and the underlying RRT* ensures that the agents' reach their respective goal configurations.

3.2. Collision avoidance under opportunistic interactions

Here, we introduce uncertainty into the position and velocity knowledge of the other agents $j \neq i$. It is at the expense of increasing this uncertainty that agents can reduce the needed communications with other agents. Agent i only requests $\mathbf{r}_j$ and $\mathbf{v}_j$ from agent j when agent i thinks it is in "collision" or no solution to $\mathbf{v}'_i$ satisfies all the collision cone conditions. In the first case, agent i only needs to update the $\mathbf{r}_j$ and $\mathbf{v}_j$ for the agent that is causing the "collision". In the second case, information from all agents needs to be updated.

The uncertainty introduces some key differences with respect to Algorithm 2. Algorithm 3 is thepseudo-code for the new algorithm, the differences are highlighted in blue italics. The variables $\mathbf{r}_j$ and $\mathbf{v}_j$, for all $j \neq i$, are not received at each path node. The information for agent j is updated if agent i thinks it is in "collision" with agent j, then the while loop is restarted. The information for all the agents is updated if no feasible $\mathbf{v}'_i$ exists, the while loop then restarts.

Algorithm 3 $\Pi_i \leftarrow$ Opportunistic SB CollisionAvoidance

$[\mathscr{T}, \pi_i] \leftarrow \text{RRT}^*(\mathbf{x}_{I,i}, \mathbf{x}_{G,i}, \varepsilon)$;
$\mathbf{r}_i = \mathbf{x}_{I,i}$;
Π_i.add($\mathbf{r}_i$);
while $\mathbf{r}_i \neq \mathbf{x}_{G,i}$ **do**
  $\mathbf{r}_i, \mathbf{v}_i \leftarrow$ Move along path to next node
  if $\|\mathbf{r}_i - \mathbf{r}_j(t_{ij})\| < \tilde{d}_{\text{sep},ij}$ **then**
    $\mathbf{r}_j, \mathbf{v}_j \leftarrow$ *Communicate with agent j ONLY*
    Restart While Iteration
  else if Conflict ($\tilde{\beta}_{ij} < \tilde{\alpha}_{ij}$) with agent j **then**
    $\mathbf{v}_i \leftarrow$ Update velocity
    if $\mathbf{v}_i$ *Does Not Exist* **then**
      for $j \neq i$ **do**
        $\mathbf{r}_j, \mathbf{v}_j \leftarrow$ *Communicate with ALL agents*
      end for
      Restart While Iteration
    end if
    if $\mathbf{r}'_i \in X_{\text{obs},i}$ **then**
      Add obstacle to agent list
      Restart while iteration
    else
      $\pi_i \leftarrow \text{GetPath}(\mathscr{T}, \mathbf{r}_i, \mathbf{x}_{G,i})$;
    end if
  end if
  Π_i.add($\mathbf{r}_i$);
  Remove obstacle from agent list
end while
$\mathbf{v}_i = \mathbf{0}$;
return Π_i

Let t_{ij} be the time agent i last received $\mathbf{r}_j$ and $\mathbf{v}_j$ from agent j. Every time information is requested at a node $\mathbf{x}_i(t_{i_m})$, we update $t_{ij} = t_{i_m}$, and agent i calculates $\mathbf{v}'_i$ using perfect information from agent j. In a future time $t_{i_m} > t_{ij}$ since the last update, a time increment $\delta t = t_{i_m} - t_{ij}$ has elapsed, and thus agent j is somewhere

within a ball centered at $\mathbf{r}_j$ with radius $v_{j,\max}\delta t$, $\mathbb{B}(\mathbf{r}_j, v_{j,\max}\delta t)$. The uncertain values, $\tilde{\mathbf{r}}_j = \mathbf{r}_j(t_{ij})$ and $\tilde{\mathbf{v}}_j = \mathbf{v}_j(t_{ij})$, are used in place of the certain values when calculating the collision cones and possible velocity-update maneuvers. The worst-case separation distance is now $\tilde{d}_{\text{sep,ij}} = d_{\text{sep,ij}} + v_{j,\max}\delta t$. The uncertain relative position is $\tilde{\mathbf{r}}_{ij} = \tilde{\mathbf{r}}_j - \mathbf{r}_i$ and the uncertain relative velocity is $\tilde{\mathbf{v}}_{ij} = \mathbf{v}_i - \tilde{\mathbf{v}}_j$. This leads to an uncertain half angle, $\tilde{\alpha}_{ij} = \arcsin\left(\frac{\tilde{d}_{\text{sep,ij}}}{\|\tilde{\mathbf{r}}_{ij}\|}\right)$, and an uncertain angle between the relative velocity and position, $\tilde{\beta}_{ij} = \arccos\left(\frac{\tilde{\mathbf{r}}_{ij}\cdot\tilde{\mathbf{v}}_{ij}}{\|\tilde{\mathbf{r}}_{ij}\|\|\tilde{\mathbf{v}}_{ij}\|}\right)$.

3.3. Computational limitations

The above algorithms assume that communication and calculations can be done instantaneously as the agents reach the next node in their path. While fast replanning and evaluation of the collision-check condition can be done in a computationally efficient manner, communications may not be possible instantaneously. At each iteration, agent i may need to communicate with up to $N-1$ other agents simultaneously before finishing their calculations. To account for communication delays, agent i can start their calculations for a future node in its path, $x_i(t_{i_m})$, as soon as they finish the calculations for the previous node to that one, $x_i(t_{i_{m-1}})$. These calculations for node $x_i(t_{i_m})$ cannot, however, be completed until agent i receives all the relevant information from an agent j at a future time t_{i_m}.

There are three cases to consider regarding the type of future information required from an agent j: (i) No information is needed from j: in this case, agent j does not send any information; (ii) the next node for agent j is reached *after* t_{i_m}, then agent j sends its current information; (iii) the next node for agent j is reached *before* t_{i_m}: agent j waits until their calculations are complete, then sends the future information. Once all required agents have sent their current or future information, agent i can complete their calculations for time t_{i_m}.

While agent i is not guaranteed to receive all needed information before they reach $x_i(t_{i_m})$, any extra time is an improvement over doing the calculations instantaneously. Future work will be devoted to more systematically addressing communication delays.

4. Algorithm analysis

This section details the analytical results for the collision avoidance algorithms introduced in the last section. We prove that a system of agents all running the Sampling-Based Collision Avoidance algorithm, or the sporadic communication version, Opportunistic Sampling-Based Collision Avoidance algorithm, will arrive at their goal configurations without collision. This first result, Lemma 1, guarantees that the agents will remain collision free while running the Sampling-Based Collision Avoidance algorithm.

Lemma 1. *Let there be N agents in an environment with static obstacles all running the* Sampling-Based Collision Avoidance *algorithm. If the agents begin collision free then with probability one they will stay collision free for all times.*

Proof. An agent can collide with other agents or static obstacles. First, we guarantee that agents will not collide with one another. Recall, the agents check and update their velocity at each node in their path from the RRT* graph. Because the RRT* is built using random samples, the probability that more than one agent will change their velocity at the same time is zero. Every time an agent is in conflict, see Proposition 1, it changes its velocity to be out of conflict with all other agents. This keeps the agent from colliding with another agent. In the case when there are no feasible velocities, the agent stops and therefore does not collide with any other agent. Static obstacles are avoided by extracting paths from the RRT*. This covers all possible scenarios an agent may encounter, thus keeping the agent collision free. □

There are two types of situations that cause the agents to never reach their goals: deadlock and livelock. A *deadlock* occurs when none of the agents are able to find a feasible velocity and therefore must set their velocities to zero to avoid collision. A *livelock* is when the agents can find a feasible velocity but will never reach their goals. Assumption 1, related to deadlock, says there will always exist a static RRT* solution for at least one agent.

Assumption 1. At all times, there is at least one agent able to treat the other agents as static obstacles and replan using the RRT* to find a collision free path to the goal. This path has a reachable node with a strictly lower cost. •

Remark 1. When a deadlock occurs, all the agents have a zero velocity, they pause the sampling-based collision avoidance algorithm they are running. The agents then build a standard RRT* tree with an X_{obs} that contains the other agents as static obstacles. Let agent i be at node $\mathbf{r}_i$ in the original path π_i and let the path after replanning be π_{new}. Assumption 1 implies that, for some agent i, the new path π_{new} exists and contains a collision-free node x that has a lower cost-to-come compared to the cost-to-come before replanning, $\text{Cost}(\mathbf{r}_i \in \pi_i) > \text{Cost}(\mathbf{x} \in \pi_{\text{new}})$. This assumption is reasonable in uncluttered environments with a small number of agents. If the deadlocked agents are contained in a bounded area, agents do not travel very fast, and the environment is sufficiently large, this condition should be met for at least one agent.

To guarantee all agents reach their goal configuration, we impose Assumption 2 on the problem.

Assumption 2. When an agent reaches its goal configuration, it does not keep any other agent from reaching its goal configuration. •

In Theorem 1, agents are shown to decrease the cost-to-come at each step when subject to Assumption 1.

Theorem 1. *Let there be N agents in an environment with static obstacles all running the* Sampling-Based Collision Avoidance *algorithm 2 with a deconfliction maneuver that solves Problem 1. Let Assumptions 1 and 2 hold. Then, with probability one, all agents will decrease their cost-to-come at each step until they converge to their goal configurations.*

Proof. To prove convergence to the goal configurations, $x_{G,i}$, we use Lyapunov's direct method. The Lyapunov function is defined as,

$$V(\mathbf{r}_1, \ldots, \mathbf{r}_N) \triangleq \sum_{i=1}^{N} \text{Cost}(\mathbf{r}_i).$$

The first condition is that,

$$V(\mathbf{r}_1, \ldots, \mathbf{r}_N) = 0 \iff [\mathbf{r}_1, \ldots, \mathbf{r}_N] = \mathbf{0},$$

which is true. The second condition is,

$$V(\mathbf{r}_1, \ldots, \mathbf{r}_N) > 0 \iff [\mathbf{r}_1, \ldots, \mathbf{r}_N] \neq \mathbf{0},$$

which is true. The final condition is that $V(\mathbf{r}_1, \ldots, \mathbf{r}_N)$ always decreases. We now examine each of the situations to show that the cost always decreases.

According to the Sampling-Based Collision Avoidance algorithm, each agent i will either move to the next node in π_i, perform a deconfliction maneuver, or stop. First, when the agent travels to the next node in π_i, recall that, the RRT* finds, with probability one, an optimal path for each agent with respect to the goal configuration. If the agent is traveling along a path extracted from its RRT* then the cost-to-come decreases,

$$V_i(\mathbf{r}_i) = \text{Cost}(\mathbf{r}_i \in \pi_i) > \text{Cost}(\mathbf{r}_i' \in \pi_i) = V_i(\mathbf{r}_i').$$

Secondly, we consider the deconfliction maneuver case. The maneuver, determined by solving Problem 1, guarantees with probability one that, at each step, the cost-to-come decreases,

$$V_i(\mathbf{r}_i) = \text{Cost}(\mathbf{r}_i \in \pi_i) > \text{Cost}(\mathbf{r}_i' \in \pi_i) = V_i(\mathbf{r}_i').$$

Lastly, we address the situation where the agent stops. The agent could be part of a deadlock or the agent could start moving again because another agent moved.

When the agent stops (not in a deadlock) and is able to start moving again, the cost evolves as,

$$\begin{aligned} V_i(\mathbf{r}_i) =& \text{Cost}(\mathbf{r}_i \in \pi_i) \\ &= \text{Cost}(\mathbf{r}_i \in \pi_i) > \text{Cost}(\mathbf{r}_i' \in \pi_i) = V_i(\mathbf{r}_i'). \end{aligned}$$

All previous cases lead to a strict decrease of a Lyapunov function.

However, assume all agents get into a deadlock. Under Assumption 1, the deadlocked agents replan, treating the other agents as static obstacles, $\mathbf{v}_j = \mathbf{0}$. When agent i stops at node $\mathbf{r}_i$, the cost-to-come stays constant, $\text{Cost}(\mathbf{r}_i \in \pi_i)$, after replanning it may increase, $\text{Cost}(\mathbf{r}_i \in \pi_i) \leq \text{Cost}(\mathbf{r}_i' \in \pi_{\text{i,new}})$. Recall that from Assumption 1, agent i will move to node $\mathbf{r}_i'$, which will decrease the cost-to-come,

$$V_i(\mathbf{r}_i) = \text{Cost}(\mathbf{r}_i \in \pi_i) > \text{Cost}(\mathbf{r}_i' \in \pi_{\text{i,new}}) = V_i(\mathbf{r}_i').$$

Note, the other agents are moving along their paths in a similar fashion and are therefore, decreasing their cost-to-come, $\text{Cost}(\mathbf{r}_j \in \pi_j)$. From the above, we can see

$$V_j(\mathbf{r}_j) > V_j(\mathbf{r}_j'),$$

for all agents j, and therefore,

$$V(\mathbf{r}_1, \ldots, \mathbf{r}_N) > V(\mathbf{r}_1', \ldots, \mathbf{r}_N').$$

Using a standard Lyapunov argument, we have now shown that the agents will asymptotically converge to their goal regions with probability one.

Finally, we prove that the agents will in fact reach their goal configurations in finite time. The sum of the cost-to-come of the agents cannot converge to a strictly positive value. This is due to the fact that, by construction of the RRT* trees, the edges have a cost that is uniformly lower bounded away from zero. The RRT* tree has a finite number nodes and therefore a finite number of edges, which leads to the conclusion. □

4.1. Opportunistic collision-avoidance algorithm analysis

Now, we turn to the Opportunistic Sampling-Based Collision Avoidance algorithm 3.

First, Lemma 2 states that if agent i is more than $\tilde{d}_{\text{sep,ij}}$ distance away from agent j for all time, then agent i and j will never collide.

Lemma 2. *If $\tilde{d}_{\text{sep,ij}} \leq \|\mathbf{r}_i - \tilde{\mathbf{r}}_j\|$ for all time, then $d_{\text{sep,ij}} \leq \|\mathbf{r}_i - \mathbf{r}_j\|$ for all time.*

Proof. By definition of $\tilde{d}_{\text{sep,ij}}$, we know $\tilde{d}_{\text{sep,ij}} \geq d_{\text{sep,ij}}$ and $\|\tilde{\mathbf{r}}_j - \mathbf{r}_j\| \leq v_{j,\max} dt = \tilde{d}_{\text{sep,ij}} - d_{\text{sep,ij}}$. The triangle inequality gives $\|\mathbf{r}_i - \mathbf{r}_j\| + \|\tilde{\mathbf{r}}_j - \mathbf{r}_j\| \geq \|\mathbf{r}_i - \tilde{\mathbf{r}}_j\|$. Putting everything together gives,

$$\begin{aligned} \tilde{d}_{\text{sep,ij}} &\leq \|\mathbf{r}_i - \tilde{\mathbf{r}}_j\| \leq \|\mathbf{r}_i - \mathbf{r}_j\| + \|\tilde{\mathbf{r}}_j - \mathbf{r}_j\|, \\ \tilde{d}_{\text{sep,ij}} &\leq \|\mathbf{r}_i - \mathbf{r}_j\| + \tilde{d}_{\text{sep,ij}} - d_{\text{sep,ij}}, \\ \tilde{d}_{\text{sep,ij}} - \tilde{d}_{\text{sep,ij}} + d_{\text{sep,ij}} &\leq \|\mathbf{r}_i - \mathbf{r}_j\|, \\ d_{\text{sep,ij}} &\leq \|\mathbf{r}_i - \mathbf{r}_j\|. \quad \square \end{aligned}$$

The following Theorem 2 is the parallel convergence result to Theorem 1, but for the Opportunistic Sampling-Based Collision Avoidance algorithm.

Theorem 2. *Let there be N agents in an environment with static obstacles all running the* Opportunistic Sampling-Based Collision Avoidance *algorithm 3. Let Assumption 1 hold and assume that, when an agent has reached its goal configuration, it does not keep any other agent from reaching its goal configuration. Then, with probability one, all agents will eventually reach their goal configurations via a collision free path.* •

The proof for Theorem 2 follows that of Theorem 1, but using the uncertain values. Because of the algorithm implementation, agent i is never within $\tilde{d}_{\text{sep,ij}}$ of any agent. Following the same Lyapunov arguments as in the proof of Theorem 1 each agent will reach its goal configuration.

The uncertain velocity, $\tilde{\mathbf{v}}_j$, does not guarantee that agent i and agent j will be conflict free for all time. This is fine since accounting for the position uncertainty keeps the two agents from colliding. When the two agents, i and j, become close to one another, the algorithm naturally let these agents communicate at each node.

At this point, Lemma 1 holds and the agents are conflict free during this period. Once the agents move away from each other, the communication returns to being sporadic as prescribed by the uncertain collision cones. Then, by Lemma 2, agent i is never within $d_{\text{sep,ij}}$ of any agent j and will remain collision free.

Remark 2. Even though it can be proven that the uncertainty cones contain the true cones, the else-if block of the algorithm alone does not guarantee that the trajectories of agents remain conflict-free. This is because all possible velocities for agents are not considered (which define a cone of velocities). The else-if block leads to an *approximate* conflict-checking condition. Whenever the true velocities and position values are used, the if-else block results in a true condition and deconfliction maneuver. An alternative opportunistic algorithm would consist of using the set of all possible agent velocities and the uncertainty cones. Our algorithm choice avoids the construction and intersection of such sets, and results in an simpler implementation which guarantees collision free trajectories given by Theorem 2.

5. Simulations

The following simulations involve four agents each represented by a different color and shape set: blue stars, red triangles, magenta squares, and green circles. The initial configurations are at the boundary of the space, represented by large colored dots. Each agents' final configuration is at the opposite side of the environment. The underlying RRT* trees are constructed with respect to ten randomly placed static obstacles. The agents are all approximated using the same radius, $\rho_i = \rho_j = 0.5$ for all $i, j \in \{1, \ldots, N\}$. The maximum velocities for each agent are all different.

Fig. 2 depicts the problem with the initial best path in black and the final paths determined by Opportunistic Sampling-Based Collision Avoidance algorithm 3 in color. The separation distance between an agent i (blue stars from Fig. 2) and all other agents j is shown in Fig. 3. The black line is the minimum separation distance, $d_{\text{sep,ij}}$. The three agents get close to agent i but never violate the $d_{\text{sep,ij}}$ condition. The cost-to-come of each

Table 1
Comparison of the results for two different simulations with four agents each. The first simulation has the same setup as Fig. 2. The second simulation has the same initial agent configuration as the other one but there are no static obstacles in the space.

Agents	Iterations	Communication savings		Peak difference: Position		Peak difference: Velocity		Max ρ
		Mean	Percentage	Mean	Maximum	Mean	Maximum	
1	166	49.67	9.97	0.18	0.68	0.15	1.18	8.06
2	81	26.00	10.70	0.10	0.30	0.09	0.52	4.27
3	115	25.00	7.25	0.12	0.36	0.05	0.17	4.81
4	112	18.67	5.56	0.10	0.24	0.09	0.24	4.00
Mean	118.5	29.84	8.37	0.13	0.68	0.10	1.18	5.29
1	107	39.33	12.25	0.32	2.13	0.19	0.96	8.39
2	87	80.00	30.65	0.48	1.19	0.28	0.73	7.36
3	114	53.67	15.69	0.20	1.04	0.24	1.36	4.35
4	82	22.00	8.94	0.21	1.07	0.14	0.73	4.99
Mean	97.50	48.75	16.88	0.26	2.13	0.22	1.36	6.27

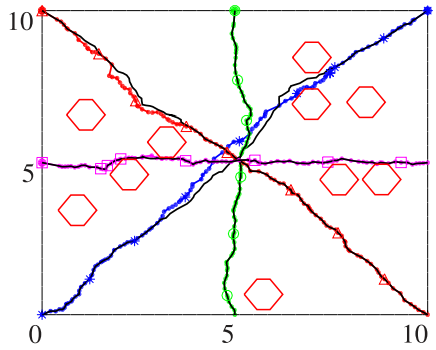

Fig. 2. The initial paths (black) and final paths (colored). The red hexagons are static obstacles that must be avoided.

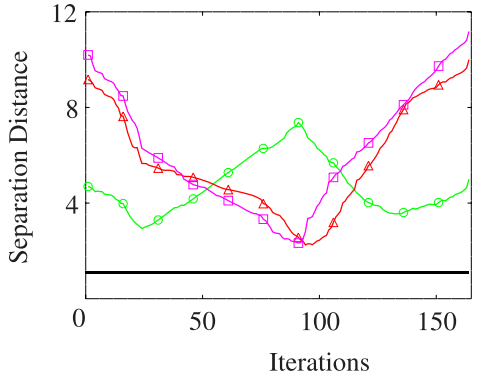

Fig. 3. The distance between agent i (blue stars from Fig. 2) and the other three agents over time.

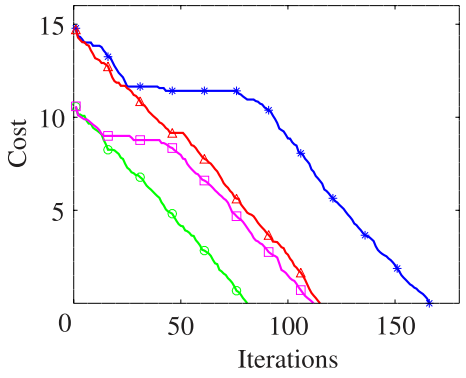

Fig. 4. The cost-to-come for each of the four agents as a function of time.

agent over time is shown in Fig. 4. Note that the cost-to-come never increases. It either stays constant or decreases.

There are two different four agent simulations compared in Table 1. The first simulation is in the same environment depicted in Fig. 2. The second simulation was done in an environment without any static obstacles, but the same initial and final agent configurations as in Fig. 2. The results for each simulation are broken down by agent. Each agent has the potential to communicate with the other three agents during every iteration of their algorithm execution. The mean number of times agent i saves on communicating with a particular agent j is calculated as the total number of communication savings for agent i divided by the number of other agents, in this case three. The percentage of communication savings is the total number of communication savings divided by the total number of possible communications (three times the number of iterations). The mean peak looks at how much the error grows before a communication occurs. The final column is the maximum radius, ρ, value each agent saw during the algorithm. Notice that the error caused by the uncertainty is not very different for each simulation. This is because the obstacles are already accounted for in the underlying graph. The only time an obstacle would affect the algorithm would be if an agent was about to collide with one. An environment dense with obstacles could see an increase in communication compared to the same setup without any, or fewer, obstacles.

The simulation with obstacles needs more communications between agents than the no obstacle simulation. This makes sense because the obstacles limit the set of possible velocities when solving Problem 1.

In the obstacle free environment, the potential velocities are only limited by the other agents. Therefore, there are fewer times when a feasible velocity cannot be found. Thus, reducing the number of communications needed.

Two additional simulations were completed to show a more complex setup. Each simulation has five agents with the initial configurations at the boundary of the space, represented by large colored dots. Fig. 5 has three additional obstacles and one addition agent compared to Fig. 2. The additional obstacles were placed so agents would need to navigate around them to reach their goal configuration. Fig. 6 has three large obstacles that prevent the agents from reaching their goal configurations without going around at least one of the obstacles. Notice that two of the agents have goal configurations that are relatively close to one another, and are still able to successfully reach their goal configurations. During both simulations, the Cost never increases, the minimum separation distance is maintained, and all the agents reach their goal configurations.

The communication savings for each of the simulations in Figs. 5 and 6 are compared in Table 2. The average communication

Table 2
Comparison of the communication savings results for two different simulations with five agents each. The first simulation is Fig. 5. The second simulation is Fig. 6.

Agents	Communication saving for Fig. 5			Communication saving for Fig. 6		
	Iterations	Mean	Percentage	Iterations	Mean	Percentage
1	61	4.0	6.56	45	9.50	21.10
2	51	3.8	7.35	17	9.75	57.35
3	34	0.0	0.0	55	18.0	32.73
4	24	6.0	25.0	21	4.75	22.62
5	64	2.0	3.13	51	30.5	59.80
Mean	46.8	3.15	8.40	37.0	14.5	38.72

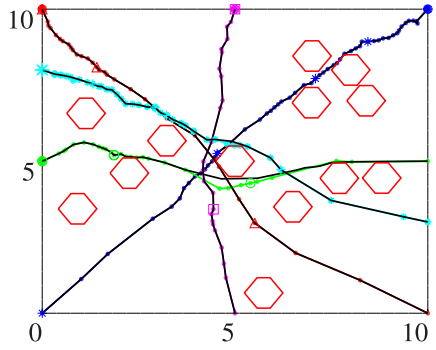

Fig. 5. The initial paths (black) and final paths (colored). The red hexagons are static obstacles that must be avoided.

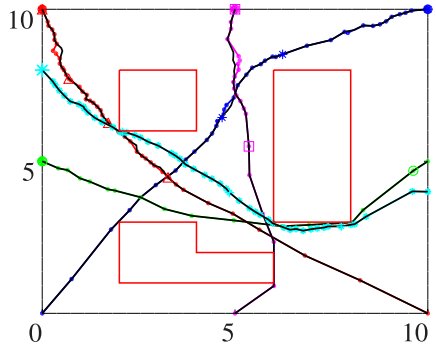

Fig. 6. The initial paths (black) and final paths (colored). The red shapes are static obstacles that must be avoided.

savings percentage for the simulation in Fig. 5, is 8.40%. The average communication savings percentage for the simulation in Fig. 6, is 38.72%. The obstacles in Fig. 6 make the agents travel around them in such a way that they are less likely to interact with the other agents causing a higher communication savings. Whereas with Fig. 5 environment, the agents are all headed for (and pass through) the center of the environment. Without the deconfliction algorithm, the agents in Fig. 5 would likely collide as they all reached center near the same time.

6. Conclusion

This paper presented a multi-agent path planning algorithm for sporadically communicating agents. As a starting point, an uncertainty-free algorithm merges the sampling-based path planner RRT* with collision cones for collision avoidance. Under some conditions this algorithm is proven to avoid collisions between agents and solve the planning problem. In order to reduce communications, controlled uncertainty in the agents' knowledge of the position and velocity of the other agents is introduced. The resulting opportunistic algorithm is then analyzed in a similar manner as the perfect-information case. Simulations show that a collision free solution is found under uncertain knowledge.

CRediT authorship contribution statement

Beth Boardman: Conceptualization, Methodology, Software, Formal analysis, Writing - original draft, Writing - review & editing, Visualization. **Troy Harden:** Supervision, Funding acquisition, Writing - review & editing. **Sonia Martínez:** Supervision, Project administration, Funding acquisition, Conceptualization, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by Los Alamos National Laboratory, USA and is approved for public release under LA-UR-18-29960, Version 2. Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by Triad National Security, LLC for the National Nuclear Security Administration of the U.S. Department of Energy under contract 89233218CNA000001. By approving this article, the publisher recognizes that the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

References

Arslan, O., & Tsiotras, P. (2013). Use of relaxation methods in sampling-based algorithms for optimal motion planning. In *IEEE int. conf. on robotics and automation*, Karlsruhe, Germany, May (pp. 2421–2428).

Bishop, R. (2000). A survey of intelligent vehicle applications worldwide. In *Intelligent vehicles symposium*, Bali, Indonesia, July (pp. 25–30).

Biswas, S., Anavatti, S. G., & Garratt, M. A. (2016). Obstacle avoidance for multi-agent path planning based on vectorized particle swarm optimization. In *The 20th Asia pacific symp. intelligent and evolutionary systems*, Canberra, Australia, November (pp. 61–74).

Boardman, B., Harden, T., & Martínez, S. (2014). Optimal kinodynamic motion planning in environments with unexpected obstacles. In *Allerton conf. on communications, control and computing*, Monticello, IL, USA, October (pp. 1026–1032).

Chen, Y., Cutler, M., & How, J. P. (2015). Decoupled multiagent path planning via incremental sequential convex programming. In *IEEE int. conf. on robotics and automation*, Seattle, WA, USA, May (pp. 5954–5961).

Frazzoli, E., Dahleh, M. A., & Feron, E. (2002). Real-time motion planning for agile autonomous vehicles. *AIAA Journal of Guidance, Control, and Dynamics*, *25*(1), 116–129.

Janson, L., Schmerling, E., Clark, A., & Pavone, M. (2015). Fast marching trees: A fast marching sampling-based method for optimal motion planning in many dimensions. *International Journal of Robotics Research*, *34*, 883–921.

Karaman, S., & Frazzoli, E. (2010). Optimal kinodynamic motion planning using incremental sampling-based methods. In *IEEE int. conf. on decision and control*, Atlanta, GA, USA, February (pp. 7681–7687).

Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, *30*(7), 846–894.

Kavraki, L. E., Svestka, P., Latombe, J.-C., & Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, *12*(4), 566–580.

Kothari, M., & Postlethwaite, I. (2013). A probabilistically robust path planning algorithm for UAVs using rapidly-exploring random trees. *Journal of Intelligent and Robotic Systems*, *71*, 1–23.

Lalish, E. (2009). *Distributed reactive collision avoidance for multivehicle systems* (Ph.D. thesis, Ph.D. thesis), University of Washington.

LaValle, S. M. (2006). *Planning algorithms*. Cambridge University Press.

Luo, W., Chakraborty, N., & Sycara, K. (2016). Distributed dynamic priority assignment and motion planning for multiple mobile robots with kinodynamic constraints. In *American control conference*, Boston, MA, USA, July (pp. 148–154).

Murano, A., Perelli, G., & Rubin, S. (2015). Multi-agent path planning in known dynamic environments. In *Int. conf. on principles and practice of multi-agent systems*, Bertinoro, Italy, October (pp. 218–231).

Neto, A. A., Macharet, D., Chaimowicz, L., & Campos, M. (2013). Path planning with multiple rapidly-exploring random trees for teams of robots. In *Int. conf. on advanced robotics*, Montevideo, Uruguay, March (pp. 1–6).

Ragaglia, M., Prandini, M., & Bascetta, L. (2016). Multi-agent poli-RRT*. In *Int. workshop on modeling and simulation for autonomous systems*, Rome, Italy, June (pp. 261–270).

Sharon, G., Stern, R., Goldenberg, M., & Felner, A. (2013). The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence*, *195*, 470–495.

Van Den Berg, J., Guy, S., Lin, M., & Manocha, D. (2011). Reciprocal n-body collision avoidance. *International Symposium on Robotic Research*, *70*(1), 3–19.

Virani, N., & Zhu, M. (2016). Robust adaptive motion planning in the presence of dynamic obstacles. In *American control conference*, Boston, MA, USA, July (pp. 2104–2109).