Force-Based Algorithm for Motion Planning of Large Agent

Samaneh Hosseini Semnani[ID], Anton H. J. de Ruiter[ID], and Hugh H. T. Liu[ID]

Abstract—This article presents a distributed, efficient, scalable, and real-time motion planning algorithm for a large group of agents moving in 2-D or 3-D spaces. This algorithm enables autonomous agents to generate individual trajectories independently with only the relative position information of neighboring agents. Each agent applies a force-based control that contains two main terms: 1) collision avoidance and 2) navigational feedback. The first term keeps two agents separate with a certain distance, while the second term attracts each agent toward its goal location. Compared with existing collision-avoidance algorithms, the proposed force-based motion planning (FMP) algorithm can find collision-free motions with lower transition time, free from velocity state information of neighboring agents. It leads to less computational overhead. The performance of proposed FMP is examined over several dense and complex 2-D and 3-D benchmark simulation scenarios, with results outperforming existing methods.

Index Terms—Collision avoidance, distributed algorithms, evolutionary computation, flocking algorithm, motion planning, multiagent systems, trajectory optimization.

I. INTRODUCTION

MOTION planning is a significant problem in multiagent systems. This problem: 1) concerns finding a one-to-one assignment between a set of given agents' initial locations and the corresponding destination locations and 2) finding the continuous trajectories connecting each agent's initial position to its goal location. Using motion functions, the agents should be able to reach their goal destinations without colliding with each other or environmental obstacles. Also, the motion planning algorithm should be able to satisfy constraints, such as the maximum velocity of the agents, and the minimum required separation distance between them. These constraints are dependent on the agents' dynamic limitations and may vary from one application to another.

This problem has attracted much interest in the research community, and various motion planners are developed, from discrete planning, motion coordination algorithms to real-time collision-avoidance methods. However, to the best of our knowledge, no existing solution satisfies all requirements of applications simultaneously, such as robustness, scalability, optimality, low computational and communication overhead, being fast, real time, flexible, etc. In this article, we present an algorithm that attempts to balance these goals by taking inspiration from the flocking behavior of birds in nature.

The main contribution of this article is adapting the flocking algorithm, which has recently shown outstanding behavior in the field of mobility control [1], [2], to solve the motion planning problem for a large group of agents while satisfying the above-mentioned multiple requirements of real applications. This adaptation, presented as the force-based motion planning (FMP) algorithm, requires essential modification of the flocking algorithm, including the elimination of velocity matching and flock centering terms, modification of the collision avoidance term to generate collision-free motions with required separation distance as well as the addition of multi-target navigational feedback to guide agents toward their goal locations. Optimal reciprocal collision avoidance (ORCA) is one well-received real-time collision-avoidance algorithm that has recently gained significant attention as a fast and scalable tool. ORCA, while claiming to find an optimal algorithm, struggles with finding a time-optimal solution. Moreover, it requires position and velocity to be exchanged over a communication network among the agents. The presented FMP algorithm is able to find collision-free paths with lower transition time without requiring the velocity information of neighbors and with less computational overhead in comparison with ORCA. Transition time is the time that takes for agents to move from their initial positions to final positions applying the motions produced by the algorithm. The performance of FMP is examined over several dense and complex 2-D and 3-D benchmark simulation scenarios. The results show that FMP computes collision-free and deadlock-free paths for all the scenarios while outperforming ORCA with respect to both transition and execution time. Our approach can compute safe and smooth trajectories for thousands of agents in dense environments with static and dynamic obstacles in a few milliseconds.

Samaneh Hosseini Semnani is with the Department of Electrical and Computer Engineering, Isfahan University of Technology, Isfahan 84156-83111, Iran (e-mail: samaneh.hoseini@iut.ac.ir).

Anton H. J. de Ruiter is with the Aerospace Engineering Department, Ryerson University, Toronto, ON M5B 2K3, Canada (e-mail: aderuiter@ryerson.ca).

Hugh H. T. Liu is with the Faculty of Applied Science and Engineering, University of Toronto, Toronto, ON M3H 5T6, Canada (e-mail: liu@utias.utoronto.ca).

The remainder of this article is given as follows. Section II discusses the main research in the field of agents' motion planning and their drawbacks, in addition to the flocking algorithm and its extensions as the base of the solution proposed in this article. Section III presents FMP, the proposed force-based algorithm for agents' motion planning and the mathematical analyses on its computation complexity, its ability to provide collision-free motions, and guaranteeing a minimum separation distance and the convergence of the algorithm. Section IV demonstrates the simulation results. Finally, concluding remarks and recommendations for future directions are given in Section V.

II. Related Work

A. Motion Planning Algorithms

The multiagent motion planning problem has attracted much interest in the research community, and various motion planners have been developed. Existing works are traditionally classified into two categories: 1) centralized approach and 2) distributed approach; however, some methods cannot fit in these two categories.

Centralized approaches [3]–[6] are suitable for small problems and can benefit from many standard optimization algorithms; however, they are often computationally expensive, especially in large problems, and are not ideal for large teams. A simple centralized approach is motion coordination. Motion coordination refers to controlling the velocity of agents along with their rigid preplanned motions such that none of them pass intersecting points between their movements at the same time. This approach introduces a configuration space and represents agent–agent collisions as obstacles in this space [3]. Efforts have been made to reduce the computational complexity of the calculations [5], but still, for large-scale problems, the computational complexity is much more than the distributed algorithms.

Distributed approaches [7], [8] use local, reactive collision-avoidance algorithms to let each agent plan its motion independently and resolve conflicts in real time when impending collisions are detected. Distributed approaches are highly scalable as each agent makes decisions independently, and their robustness against disturbance is appealing as they are performing real-time computations. However, they do not provide any means to optimize the trajectories from time or energy perspective. An algorithm is robust in a sense that it can perform well when unknown disturbances, such as dynamic obstacles, appear in the environment, or one of the agents is not performing well. ORCA [7] is one well-known algorithm in this group. Given perfect knowledge of neighboring positions and speeds, the ORCA algorithm lets each agent compute its optimal velocity independently by solving an optimization problem at each time step. ORCA requires relative position and velocity to be shared between neighboring agents [9].

Another approach is to solve trajectory planning problems in multiple stages [10]–[12]. The algorithms that use this method often require solving one large optimization problem in one of their steps in which the decision variables define all of the agents' trajectories. As a result, they do not scale well to large teams. Preiss *et al.* [10] decoupled the problem into a discrete planning stage, followed by a refinement stage. In this method, a discrete planning algorithm is applied for both finding an assignment between agents' initial and final locations and for computing a sequence of waypoints. This discrete planner solves the problem by reducing to a maximum-flow problem. Then, continuous refinement methods start working on the produced discrete plan to compute smooth trajectories. However, such an algorithm is unable to avoid dynamic obstacles as it is not a real-time algorithm and plans the trajectories beforehand. It also requires heavy calculations, especially in large size problems, which results in low speed (e.g., on the order of minutes) for a team with 32 agents while using a CPU with ten cores [10].

An alternative approach is to use Voronoi diagrams and the Lloyd algorithm for planning collision-free trajectories. Although Voronoi diagrams were initially used for finding optimal environment coverage and agent placement, some papers [13], [29], and [30] apply this technique to trajectory planning by iteratively calculating the Voronoi cells and planning trajectories to the point in the cell that is closest to the agent's goal position. The trajectories generated are naturally collision free since the points only move within their cell at each iteration. However, in real distributed simulations, especially in 3-D environments, this method still requires the calculation of Voronoi diagrams and the closest points at each iteration, which imposes high computational complexity over the agents, especially when the number of agents and iterations to converge increases. Also, this method does not optimize the agents' motions in terms of time or energy.

Another recent approach is the application of deep reinforcement learning for multiagent motion planning [14], [15]. This approach offloads the expensive online computation (for predicting interacting patterns) by learning policy and a value function by applying reinforcement learning techniques that implicitly encode cooperative behaviors. The learned policy will be used by each agent later in real time to select the best action at each time. Reinforcement learning is an active research topic at its early stage with many unanswered questions. Current development still has drawbacks; for example, the policy cannot be well generalized to new scenarios that do not appear in the entire training period. Also, it is not suitable for large-scale problems due to the enormous computational complexity of the training part.

B. Flocking-Based Algorithms

This section briefly introduces the flocking algorithm [1] and its extensions that are related to agents' motion planning as the basis of the FMP algorithm.

Flocking is a distributed group behavior of a large number of autonomous interacting agents able to self-organize without having a leader or any other central control. This algorithm is based on three Reynolds rules [1]: 1) flock centering; 2) collision avoidance; and 3) velocity matching. Reference [2] provides a solid mathematical background for the three main Reynolds rules in which each agent applies a force vector u_i as a control input unit. The following equations present vehicle
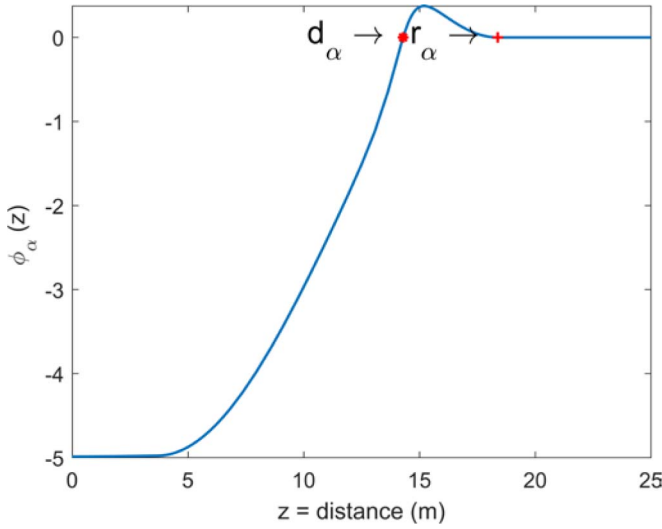
Fig. 1. Attractive/repulsive pairwise function used in the gradient-based term of the flocking algorithm.

Fig. 2. Network of agents created by the flocking algorithm a few seconds after starting in (a) 2-D and (b) 3-D environments [2].

dynamics and u_i force vector for agent i:

$$\dot{p}_i = v_i, \quad \dot{v}_i = u_i, \quad u_i = u_i^{\alpha} + u_i^{\gamma} \tag{1}$$

$$u_i^{\alpha} = f_i^g + f_i^d \tag{2}$$

$$f_i^g = \sum_{j \in N_i} \phi_{\alpha}\left(\left\|p_j - p_i\right\|_{\alpha}\right) n_{ij} \quad \text{Gradient\_based term} \tag{3}$$

$$f_i^d = \sum_{j \in N_i} a_{ij}(q)\left(v_j - v_i\right) \quad \text{Velocity\_Consensus term} \tag{4}$$

$$u_i^{\gamma} = -c_1(p_i - p_r) - c_2(v_i - v_r)c_1, c_2 > 0$$

$$\text{Navigational\_Feedback term} \tag{5}$$

where $p_i, v_i \in \mathbb{R}^3$ denote the position and velocity of agent i, respectively. $p_r, v_r \in \mathbb{R}^3$ are position and velocity of a dynamic/static agent that represents a group objective and can be viewed as a moving target. N_i is the set of neighbors of agent i. $a_{ij}(q)$ is an element of a spatial adjacency matrix $A(q)$. $n_{ij} = (p_j - p_i)/\sqrt{1 + \epsilon\|p_j - p_i\|^2}$ is a vector along the line connecting p_i to p_j and $\epsilon \in (0, 1)$ is a fixed parameter of the σ-norm. c_1 and c_2 are two positive constant values. ϕ_{α} is an attractive/repulsive pairwise function depicted in Fig. 1, in which d_{α} is the preferred distance between the agents and r_{α} is the communication distance between them. f_i^g attempts to keep all the agents' relative distances close to d_{α}.

The term f_i^d is a damping force that tries to keep the velocity of each agent close to its flockmates velocities and u_i^{γ} is the navigational feedback and attempts to attract all the agents toward a group objective. See [2] for more details of (3)–(5). Applying the flocking algorithm over a group of agents results in the creation of a network of the agents moving around a target (p_r, v_r) while trying to keep distance between themselves is equal to d_{α}. Fig. 2 presents the result of the flocking algorithm after a few seconds of its start time [2].

Semiflocking is a recent extension of the main flocking algorithm, which is designed for tracking multiple targets, each being a common objective for a subgroup of agents [16], [17]. Section III shows how we use this characteristic of the semiflocking algorithm for motion planning of large agent
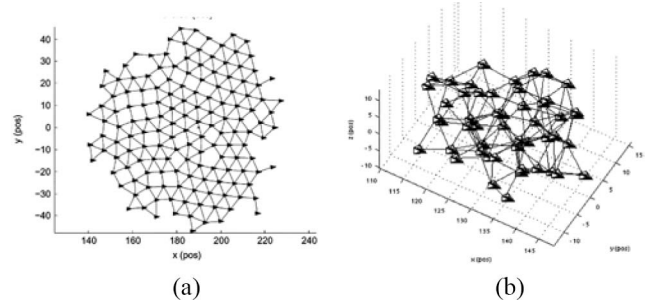
teams. The semiflocking algorithm changes the navigational-feedback term (u_i^{γ}) in the force function u_i as shown in the following equation while keeping the gradient-based and velocity-consensus terms unchanged:

$$u_i^{\gamma} = f_i^{\gamma}\left(p_i, p_{t1}, \ldots, p_{t1}, v_i, v_{t1}, \ldots, v_{tm}\right)$$

$$= \sum_{j=1}^{m} \varphi\left(p_{tj} - p_i\right).\left(c_{1j}\left(p_{tj} - p_i\right) + c_{2j}\left(v_{tj} - v_i\right)\right)/n_{tj} \tag{6}$$

where m represents the number of targets, c_{1j} and c_{2j} are positive constant values, and p_i and v_i are position and velocity of sensor i, respectively. p_{tj} and v_{tj} are position and velocity of target j, respectively. n_{tj} represents the number of sensors currently tracking target tj and φ is a switching function taking 0-1 values. See [16] for more details of (6). The result of applying the semiflocking algorithm on a multitarget problem is small quasi α-lattices of agents around each target while still leaving some agents free in the area for searching new targets (see [16, Fig. 9]).

III. APPROACH

This section presents an FMP algorithm, a fast, real time, and effective solution for 2-D and 3-D motion planning based on the flocking algorithm described in Section II-B. Target tracking is just one of the applications of the flocking algorithm that is introduced in [2]. However, if the flocking algorithm can guide the agents toward a target without any collision, we can exploit that in agents' motion planning. The initial idea is to put the target in the desired goal location and apply an adapted version of the flocking algorithm to find collision-free motions from agents' initial locations to the goal location. Since we do not want to guide all the agents to one goal position, we need to increase the number of targets to be precisely is equal to the number of agents ($m = n$) to have a separate pair of (initial, goal) locations for all the agents. Semiflocking presents a solution for increasing the number of targets from 1 in the flocking algorithm to m ($m < n$) [16]. The FMP algorithm continues this track to have n targets where each is assigned to one agent. Also, FMP makes essential changes in the control input force vector of the flocking framework (u_i), including the elimination of velocity consensus term and modification of the gradient-based and navigational feedback terms.

A. Problem Statement

Given n agents, each with its own target location, the objective is to guide them to their target locations, while maintaining a specified minimum distance between them (as a result of considering the agents volumes and localization errors), avoiding collisions with environmental obstacles, and satisfying communication constraints and maximum agent velocity constraints in horizontal and vertical directions.

B. Assumptions

The following assumptions have been made in this article.
1) We have n agents deployed in a 3-D environment.
2) Each agent is moving independently but in coordination with the other agents' motions. Let p_i, v_i, $u_i \in \mathbb{R}^3$ denote the position, velocity, and acceleration of the agent i, respectively, $p = \mathrm{col}(p_1, \ldots, p_n)$, $v = \mathrm{col}(v_1, \ldots, v_n)$. Agent i obeys the following dynamics:
$$\begin{cases} \dot{p}_i = v_i, \\ \dot{v}_i = u_i, \end{cases} \quad 1 \le i \le n.$$
3) r is the radius of a neighborhood within which the agents must interact.
4) Each agent can measure the relative position of its neighboring agents and the closest point of obstacles.
5) The set of initial and goal locations of the agents are given.
   a) $\mathfrak{I} = \{\mathfrak{I}_1, \mathfrak{I}_2, \ldots, \mathfrak{I}_n\}$: Set of initial positions of agents.
   b) $\mathfrak{F} = \{\mathfrak{F}_1, \mathfrak{F}_2, \ldots, \mathfrak{F}_n\}$: Set of goal positions of agents, where $\mathfrak{I}_i$ and $\mathfrak{F}_i$ represent the initial and final position of agent i, respectively.
6) We have n hypothetical static targets placed at the agents' goal positions. $\mathfrak{T} = \mathfrak{F}$.
7) The separation distance between agents in $\mathfrak{I}$ and $\mathfrak{F}$ is greater than or equal to d, where d is an upper bound for required minimum separation distance between agents.
8) d^*: Required minimum separation distance between agents ($d^* \le d$).
9) v_{max}: Maximum allowable velocity of the agents. v_{max} is a user-specified parameter.

C. Force-Based Motion Planning Algorithm

In contrast to the semiflocking algorithm, which was designed for multiagent target tracking, the FMP algorithm is a motion planning algorithm. Therefore, we need to modify the three main terms of the flocking algorithm: 1) gradient based; 2) velocity consensus; and 3) navigational feedback to fit the requirement of this problem and to satisfy separation and maximum agent velocity constraints.

To reach this point, FMP first generates n hypothetical targets and places them at the given goal locations. Each agent is uniquely assigned to one target. In this article, it is assumed that all targets have been assigned. Hungarian assignment [18] is one possible method in which the global cost function is minimized

$$\min \sum_{i-1}^{n} \sum_{j=1}^{n} \|\mathfrak{T}_j - \mathfrak{I}_i\| \tag{7}$$

where $\|x\|$ is the Euclidean norm of x. A standard solution for this minimization problem is the so-called Hungarian algorithm [18] which presents $\mathcal{O}(N^3)$ computational cost. Giordani *et al.* [19] introduced a distributed version of this algorithm, which makes it suitable for distributed problems as studied in this article. The auction algorithm [20] and its distributed version [21] are lower cost alternatives to the Hungarian algorithm, which produce suboptimal solutions in significantly less time.

After the target–agent assignment, each agent i applies the control function presented in (8) iteratively until it reaches its goal location

$$u_i = \begin{cases} 0, & \text{if } v_i^T \overline{u}_i > 0 \text{ and } \|v_i\| \ge V_{\max} \\ \overline{u}_i = f_i^{\mathcal{R}} + f_i^{\gamma}, & \text{otherwise} \end{cases} \tag{8}$$

where $f_i^{\mathcal{R}}$ is the repulsive force and f_i^{γ} is the navigational feedback that attracts the agents toward the assigned targets. Equations (9) and (10) describe how $f_i^{\mathcal{R}}$ and f_i^{γ} are calculated

$$f_i^{\mathcal{R}} = \sum_{j \in N_i} \left(\phi(\|p_j - p_i\|) \frac{p_j - p_i}{\|p_j - p_i\|} \right) \text{ Repulsive Function} \tag{9}$$

$$f_i^{\gamma} = -c_1(p_i - \mathfrak{T}_i) - c_2(v_i)$$
$$\text{Navigational Feedback} \tag{10}$$

$$\phi(z) = \begin{cases} -\rho \times (z - r)^2, & 0 < z < r \\ 0, & z \ge r \end{cases} \tag{11}$$

$$N_i = \{j \in \mathcal{V} - \{i\} : \|p_j - p_i\| < r\}$$
$$\mathcal{V} = \text{the set of indices of agents} \tag{12}$$

where ϕ is a repulsive function defined in (11) and depicted in Fig. 3(a) and N_i represents the agents within a neighborhood of agent i and is defined in (12). c_1 and c_2 are two positive constant values, ρ is a positive repulsive gradient, and r is the communication radius. Section III-C1 explains how we handled agents' maximum velocity constraint in u_i function.

The control function in FMP, u_i, has three main differences with the flocking control function (1). First, the velocity-consensus term is omitted here. The main functionality of the velocity-consensus term is matching the velocity of each agent with nearby flockmates. However, when the flocking algorithm is going to be used for motion planning of individual agents, each one following a separate target, there is no need to match the velocity of each agent with its neighbors.

Second, the attractive/repulsive function of the flocking algorithm (3) is changed to an only repulsive function presented in (9). As described in Section II-B, the attractive/repulsive function in the gradient term of the flocking algorithm creates a spring model that attempts to keep the distance between two agents is equal to a predefined value d. This function is useful in the flocking algorithm because it needs to create a network of agents placed at a preferred distance from each other. However, in the FMP algorithm, we only need to keep the agents strictly apart from each other, and there is no need to keep them at a specific distance of one another. Therefore, we only need to have a repulsive force between every two agents to repel them from one another when they become close. This function is defined in (11) and
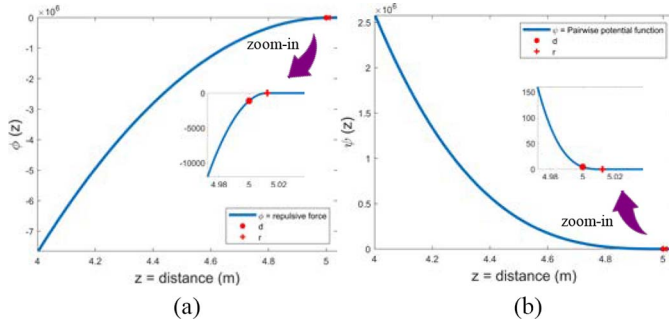
This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

SEMNANI *et al.*: FORCE-BASED ALGORITHM FOR MOTION PLANNING OF LARGE AGENT

5

Fig. 3. (a) Repulsive function and (b) smooth pairwise potential function $\psi(z)$ used in the FMP algorithm for $d = 5$ in $[4, 5.03]$ interval.

Fig. 4. Representation of a spherical obstacle in the neighborhood of an agent.

depicted in Fig. 3(a). The value of the parameter ρ in this function should be selected such that the function be able to provide a high enough repulsive force. In this article, we take $\rho = 7.5$ million.

As represented in Fig. 3(a), two agents start repelling each other when they come within a given communication distance of one another (r). The intensity of the repelling force depends on the distance between the two agents. Section III-C1 describes how the communication distance r is calculated based on the given separation distance d and maximum allowed velocity $V_{\max}$. The following equation shows the smooth potential function between two agents, based on their repulsive force (11). This function is depicted in Fig. 3(b)

$$\psi(z) = \int_0^z \phi(s)ds = \begin{cases} -(\rho/3) \times (z - r)^3 + C, & 0 < z < r \\ C, & z \geq r \end{cases}$$
(13)

where C is a constant value. This repulsive potential function induces a collective potential function in the form

$$V(p) = (1/2) \sum_i \sum_{j \neq i} \psi \left(\| p_j - p_i \| \right).$$
(14)

Comparing Figs. 1 and 3(a) reveal the difference between the attracting/repulsive force in the main flocking algorithm and the only repulsive force in the FMP algorithm. Attractive force in the flocking algorithm in $[d, r]$ is replaced with a repulsive force in the FMP algorithm. It also provides a much higher repulsive force in $[0, d]$ interval which significantly pushes two agents apart when they want to become closer than d and guarantee our required minimum separation distance d^*. Theoretical analysis in Section III-E2 and simulation results presented in Section IV demonstrate this fact.

The third and final difference between the control functions of the flocking and FMP algorithms is about the navigational feedback term. The navigational term of FMP is very similar to the navigational term of the semiflocking that is described in Section II-B, as they both attract the agents toward multiple targets. However, there are still essential differences between them in terms of the number of agents assigned to track each target. Flocking and semiflocking algorithms are both target tracking algorithms that require to have a group of agents around target/targets to provide suitable target coverage. In contrast, the main objective of the FMP algorithm is to create collision-free motions toward the targets.
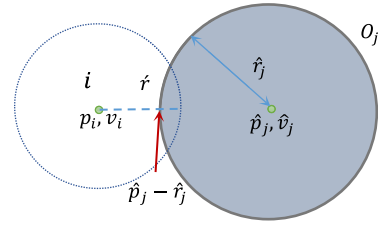
1) Maximum Velocity Constraint: The agents' velocity restrictions are the other constraints in a real motion planning problem, which should be considered in the proposed algorithm. A motion planning algorithm may impose a velocity to the agents, which is beyond their physical ability. This constraint is explicitly considered in the FMP algorithm. The velocity of each agent is influenced by its control input u_i. Therefore, to control the velocity, the FMP algorithm should restrict the two force functions of $\bar{u}_i$ in (8) such that they do not cause over limit velocities. We restrict the value of the repulsive force by selecting a communication distance r that is very close to d. The following equation shows how we calculate r:

$$r = \sqrt[3]{3v_{\max}^2/2\rho} + d.$$
(15)

To keep the velocity always lower than $V_{\max}$ after calculating $\bar{u}_i$, a capping function [cap_velocity(i) in Algorithm 1] is added to the algorithm that limits each agent's velocity to its maximum value. It is equivalent to the restriction condition used in the first term of (8).

Algorithm 1 is a sampled-data implementation of the continuous-time FMP algorithm discussed thus far. As illustrated in this pseudocode, each agent calculates its next velocity and position at each time step in an entirely distributed manner without requiring any central control unit. It only requires the knowledge of the relevant location of its neighbors.

D. FMP With Obstacle Avoidance

In this section, we present an extension of the FMP algorithm, which can avoid multiple dynamic or static obstacles. The main idea is to use an agent-based representation of both dynamic and static obstacles by creating new species of agents called obstacle-agents. This approach is partially motivated by the work of Olfati-Saber [2]. Each obstacle-agent is a virtual agent that is induced by regular agents when they are close to an obstacle. The obstacles are spheres with a defined center and radius ($\hat{p}_j, \hat{r}_j$). Fig. 4 depicts an obstacle O_j which is in the neighborhood of agent i. As illustrated in this figure, each obstacle has three parameters $\hat{p}_j, \hat{v}_j$, and $\hat{r}_j$ which represents the position and velocity of the center of the obstacle and its radius. Also, $\acute{r}$ represents the interaction range of an agent with neighboring obstacle-agents.

The interaction between regular agents and obstacle-agents is described using the notation f_i^{obstacle} in the following equation. This equation is the same as (8) except it has additional repulsive force (f_i^{obstacle}) to avoid obstacles. The modified

Algorithm 1 FMP Algorithm

Input:
- n = number of agents (= number of targets)
- $\mathcal{I}$ = Initial position of agents $\mathcal{F}$ = final position of agents
- d^* = required minimum separation distance between agents
- v_{max}

Output:
- $\forall_{i=0}^{n} p_i(t)$ $p_i(t) : (p_{ix}(t), p_{iy}(t), p_{iz}(t))$ = position of agent i at time t
- $\forall_{i=0}^{n} v_i(t)$ $v_i(t) : (v_{ix}(t), v_{iy}(t), v_{iz}(t))$ = velocity of agent i at time t
  ($0 \leq t$ increases in Δt time intervals)

begin
    $v_i(0) = (0, 0, 0)$ // initialize agent velocity
    $\mathcal{A}$ = agent-target assignment   // e.g., Hungarian
    $\mathcal{T}_i = \mathcal{F}_{\mathcal{A}_i}$ //initialize target position
    Calculate d and r using (36) and (15)
    for $(t = 0; t < MaxSimTime; t+ = \Delta t)$
        for $0 <= i < n$ **do in parallel**
            move_this_agent_to_new_position (i)
        end for
        MaxDis = agents_maximum_distanse_from_ $\mathcal{F}()$
        if ($MaxDis < EndMaxDis$) **do**
            break
        end if
    end for
end

---

move_this_agent_to_new_position (i) {
    $f_i^{\mathcal{R}}$ = repulsive_force (i), f_i^{γ} = navigational_feedback (i)
    $\bar{u}_i = f_i^{\mathcal{R}} + f_i^{\gamma}$
    $v_i(t) = v_i (t - \Delta t) + \bar{u}_i \Delta t$
    cap_velocity (i)   //A hard constraint on velocity
    $p_i(t) = p_i(t - \Delta t) + v_i(t)\Delta t$**}**

---

repulsive_force (i) {
    $f = 0$
    for $0 <= j < n$ **do**
        $dist = \|p_j - p_i\|$
        if $(dist < r$ **and** $j \neq i)$ **then**
            $ForceComponent = -\rho \times (dist - r)^2$
            $f+ = ForceComponent \times (p_j - p_i)/\|p_j - p_i\|$
        end if
    end for
return f**}**

---

navigational_feedback (i) {
    $f = -c_1(p_i - \mathcal{T}_i) - c_2(v_i)$
return f**}**

---

cap_velocity(i) {
    if $(\|v_i\| > v_{max})$ **then**
        $v_i = (v_i/\|v_i\|) \times v_{max}$
    end}

---

control force on agent i is given by

$$u_i = \begin{cases} 0, & \text{if } v_i^T \bar{u}_i > 0 \text{ and} \\ & \|v_i\| \geq V_{max} \\ \bar{u}_i = f_i^{\mathcal{R}} + f_i^{\gamma} + f_i^{\text{obstacle}}, & \text{otherwise} \end{cases} \quad (16)$$

$$f_i^{\text{obstacle}} = \sum_{j \in N_i^{\text{obstacle}}} \left(\phi_{\text{obstacle}}\left(\|\hat{p}_j - p_i\| - \hat{r}_j\right) \times \frac{\hat{p}_j - p_i}{\|\hat{p}_j - p_i\|} \right) \quad (17)$$

$$\phi_{\text{obstacle}}(z) = \begin{cases} -\acute{\rho} \times (z - \hat{r})^2, & 0 < z < \hat{r} \\ 0, & z \geq \hat{r} \end{cases} \quad (18)$$

$$N_i^{\text{obstacle}} = \left\{ j \in \mathcal{V}_{\text{obstacle}} : \|\hat{p}_j - \hat{r}_j - p_i\| < \hat{r} \right\}$$

$$\mathcal{V}_{\text{obstacle}} = \text{the set of indices of obstacles} \quad (19)$$

where f_i^{obstacle} is a repulsive function defined in (17) and $\acute{\rho}$ is its positive repulsive gradient and determines the severity of avoiding the obstacles. Here, we choose $r > \hat{r}$, $\rho = \acute{\rho}$, but in general, they can be chosen independently. N_i^{obstacle} represents the set of neighboring obstacles of agent i and is defined in (19). The other parameters are as described in (8)–(12). Applying (16), agents can avoid obstacles in exactly the same way that they can avoid other agents. When the agents are getting close to an obstacle, they will receive a repulsive force f_i^{obstacle} from the relevant virtual obstacle-agent. This repulsive force is the critical element of the collision-avoidance ability of the FMP algorithm. In terms of sensing requirements, we assume that every agent is equipped with range sensors that allow the agent to measure the relative position between the closest point on an obstacle and itself $(\hat{p}_j - \hat{r}_j)$. Both radars and laser radars can be used as range sensors.

E. Theoretical Analysis

This section presents a theoretical analysis of the computational complexity of the FMP algorithm. Also, theoretical guarantees on collision-free trajectories and the minimum separation distance between agents are given. Finally, the mathematical analysis that proves the convergence of the algorithm is presented.

1) Computational Complexity: In this section, we calculate the computational complexity of each step of the FMP algorithm. Each agent at each step of the algorithm calculates two main forces: 1) repulsive force and 2) navigational force (see Algorithm 1). Calculation of the repulsive force only includes computing the value of the repulsive function for each neighbor. Assuming that each agent has found k neighboring agents in its vicinity, it takes $\mathcal{O}(k)$ time for the FMP algorithm to calculate repulsive forces for each agent. Of course, in practice as the value of r is selected very close to d, the number of each agents neighbors (k) remains zero until it reaches to a distance from others r which, as represented in (15), is a little bit more than the value d. Calculation of navigational feedback force and cap_velocity() function are the two remaining computational steps for each agent at each iteration. The order of complexity of these two steps is $\mathcal{O}(1)$; therefore, it does not increase the total order of computational complexity of the algorithm. As a result, the entire complexity of the algorithm for each agent at each iteration is $\mathcal{O}(k)$, and for all the agents at each iteration is $\mathcal{O}(n.k)$. However, in practice, the execution time of the FMP is much lower than other algorithms with the same computational complexity, for example, ORCA. Simulation results in Section IV confirm this observation. This is because the calculation of the repulsive function for each neighbor is the only effective computational effort of each agent at each iteration in the FMP algorithm. Also, the number of neighbors (k) is smaller in this algorithm. It is also the result of a lower number of steps each agent takes to reach

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

SEMNANI *et al.*: FORCE-BASED ALGORITHM FOR MOTION PLANNING OF LARGE AGENT

7

their goals using the FMP algorithm. It is worth mentioning that as the algorithm is completely distributed, and each agent is independent, the computations can be fully parallelized, which decreases the execution time significantly.

2) Collision-Free Analysis: This section presents an analysis that shows using the FMP algorithm guarantees a minimum separation distance d^* between the agents.

a) Collision-free proof: Before presenting the minimum separation guarantee proof, we need to define the structural dynamics corresponding to the FMP algorithm in addition to the Hamiltonian energy of the system. This analysis is performed with the continuous-time version of the algorithm, and we know that the discretization approximates the continuous-time dynamics. The vehicle dynamics of each agent in the system is in the form

$$\sum : \begin{cases} \dot{p}_i = v_i \\ \dot{v}_i = u_i \end{cases}$$
$$u_i = \begin{cases} 0, & \text{if } v_i^T \bar{u}_i > 0 \text{ and } \|v_i\| \geq V_{\max} \\ \bar{u}_i, & \text{otherwise} \end{cases}$$
$$\bar{u}_i = -\partial U_\lambda(p)/\partial p_i - c_2 v_i \tag{20}$$

where $U_\lambda(p)$ is the aggregate potential function and is defined in the following equation. Solutions of (20) are understood to be in the Filippov sense

$$U_\lambda(p) = V(p) + \lambda J(p) \tag{21}$$

where $V(p)$ is the collective potential function defined in (14), $\lambda = c_1 > 0$ and $J(p)$ is the moment of inertia of all the particles, defined as

$$J(p) = (1/2) \sum_{i=1}^{n} \|p_i - \mathfrak{T}_i\|^2. \tag{22}$$

The structural Hamiltonian of the system $\sum$ is as follows:

$$H_\lambda(p, v) = U_\lambda(p) + K(v) \tag{23}$$

where $K(v) = (1/2) \sum \|v_i\|^2 = (1/2)v^T v$ is the kinetic energy of the particle system.

Lemma 1: If $\|v_i(0)\| \leq v_{\max}$ then $\|v_i(t)\| \leq v_{\max}$ $\forall t \geq 0$.

Proof: Consider the kinetic energy of agent i: $K(v_i) = (1/2)v_i^T v_i$, then, $\dot{K}(v_i) = v_i^T \dot{v}_i = v_i^T u_i$. On the level set $\|v_i\| = v_{\max}$ ($K(v_i) = (1/2)v_{\max}^2$), we have

$$\dot{K}(v_i) = \begin{cases} 0, & \text{if } v_i^T \bar{u}_i > 0 \\ v_i^T \bar{u}_i \leq 0, & \text{otherwise.} \end{cases} \tag{24}$$

To show that $v_i(t)$ cannot leave the set $\|v_i(t)\| \leq v_{\max}$, we only need to evaluate $\dot{K}(v_i)$ on the boundary of that set where $\|v_i(t)\| = v_{\max}$, in which we have $\dot{K} = 0$ if $v_i^T \bar{u}_i > 0$ and $\dot{K} = v_i^T \bar{u}_i$ if $v_i^T \bar{u}_i \leq 0$. In both cases, we have $\dot{K} \leq 0$. Hence, $\dot{K}(v_i) \leq 0$ when $\|v_i\| = v_{\max}$ and any trajectory $v_i(t)$ starting in the sublevel set $X_i = \{v_i : K(v_i) \leq (1/2)v_{\max}^2\}$ cannot leave this set. ∎

Lemma 2: Given a collection of robots, with structural dynamics $\sum$, we have $\dot{H}_\lambda \leq -c_2 v^T v \leq 0$.

Proof:

$$\dot{H}_\lambda(p(t), v(t)) = \sum_i \left((\partial H_\lambda/\partial p_i)\dot{p}_i + (\partial H_\lambda/\partial v_i)\dot{v}_i \right)$$
$$= \sum_i \left((\partial U_\lambda/\partial p_i)^T v_i + v_i^T u_i \right) \tag{25}$$

$$(\partial U_\lambda/\partial p_i)^T v_i + v_i^T u_i$$
$$= \begin{cases} (\partial U_\lambda/\partial p_i)^T v_i, & \text{if } v_i^T \bar{u}_i > 0 \text{ and } \|v_i\| \geq v_{\max} \\ -c_2 v_i^T v_i, & \text{otherwise.} \end{cases} \tag{26}$$

In the first part of (26), we have

$$v_i^T \bar{u}_i > 0 \to -v_i^T \frac{\partial U_\lambda}{\partial p_i} - c_2 v_i^T v_i > 0 \to v_i^T \frac{\partial U_\lambda}{\partial p_i} < -c_2 v_i^T v_i. \tag{27}$$

From (26) and (27), we have

$$\left(\frac{\partial U_\lambda}{\partial p_i} \right)^T v_i + v_i^T u_i \leq -c_2 v_i^T v_i \to \dot{H}_\lambda \leq -\sum_i c_2 v_i^T v_i$$
$$\to \dot{H}_\lambda \leq -c_2 v^T v \leq 0. \tag{28}$$

So, the Hamiltonian energy of the system is nonincreasing for all (p, v). ∎

Theorem 1: Consider a group of agents applying the FMP algorithm with $c_1, c_2 > 0$, and structural dynamics $\sum$. Assume that the initial kinetic energy of the system $K(v(0)) = 0$ and the inertia $J(p(0))$ is finite. Also, assume $H_\lambda(p(0), v(0)) < \psi(d^*)$, then none of the agents can come closer than d^* to each other.

Proof: Suppose $H_\lambda(p(0), v(0)) < \psi(d^*)$ and there is at least one pair of agents that their distance becomes less than d^* at a given time $t_1 \geq 0$. This implies the collective potential of the particle system is at least $\psi(d^*) : H_\lambda(p(t_1), v(t_1)) \geq \psi(d^*)$. However, by Lemma 2, $H_\lambda(p(0), v(0)) \geq H_\lambda(p(t_1), v(t_1))$ so we have

$$H_\lambda(p(0), v(0)) \geq H_\lambda(p(t_1), v(t_1)), H_\lambda(p(t_1), v(t_1))$$
$$\geq \psi(d^*) \to H_\lambda(p(0), v(0)) \geq \psi(d^*).$$

This contradicts with the assumption that $H_\lambda(p(0), v(0)) < \psi(d^*)$. Hence, none of the agents can come closer than d^* at any time $t \geq 0$. ∎

b) Calculating the value of d^*: Having Theorem 1, we are now ready to calculate the guaranteed minimum separation distance d^* by the FMP algorithm. In this theorem, we assume that $H_\lambda(p(0), v(0)) < \psi(d^*)$ and $K(v(0)) = 0$. We also assume that the initial distance between the agents is equal to or greater than d (see Section III-B). We know that in the 2-D environment, each agent can have at most six agents within its distance d (otherwise, the neighbors have to become closer to each other than d). This number grows to 12 agents in the 3-D case. We continue the proof for the 2-D environment. The maximum initial Hamiltonian energy of the system is calculated as

$$H_\lambda(p(0), v(0)) = U_\lambda(p(0)) + K(v(0))$$
$$= V(p(0)) + c_1 J(p(0)$$
$$V^{\max}(p(0)) = (1/2) \sum_i \sum_{j \neq i} \psi(d)$$
$$= (1/2) \sum_i \sum_{j \neq i} \rho/3 \times (r - d)^3$$
$$< 6n \times \rho(r - d)^3/(2 \times 3) = n\rho(r - d)^3 \tag{29}$$

$$J(p_0) = (1/2) \sum_{i=1}^{n} \|p_i - \mathfrak{T}_i\|^2 \leq \xi n/2 \tag{30}$$

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

8

IEEE TRANSACTIONS ON CYBERNETICS

where ξ is the maximum Euclidean distance between the agents and their goal locations. From (29) and (30), we have

$$H_\lambda(p(0), v(0)) < n\rho(r - d)^3 + \xi n/2. \tag{31}$$

As we assumed $H_\lambda(p(0), v(0)) < \psi(d^*) = -\rho/3 \times (d^* - r)^3$ so we have

$$d^* = r - \sqrt[3]{3n(r - d)^3 + (3n\xi)/(2\rho)} . \tag{32}$$

These calculations show if the initial distance between the agents is at least d and their initial velocity is zero then none of them become closer than d^* using the FMP algorithm. Using very similar calculations for the 3-D environment, we have

$$d^* = r - \sqrt[3]{6n(r - d)^3 + (3n\xi)/(2\rho)} . \tag{33}$$

Having the value of d^*, the value of parameter d can be calculated for the 2-D environment as

$$d = r + \sqrt[3]{(d^* - r)^3/(3n) + \xi/(2\rho)}. \tag{34}$$

Applying (15), we have (35) for 2-D and (36) for 3-D

$$d = d^* + \sqrt[3]{((9n - 3) \times v_{\max}^2 + 3n\xi)/(2\rho)} \tag{35}$$

$$d = d^* + \sqrt[3]{((18n - 3) \times v_{\max}^2 + 3n\xi)/(2\rho)}. \tag{36}$$

Equations (15) and (36) are used in the FMP algorithm for calculation of a communication distance r that guarantees given separation distance d^*.

3) Convergence Analysis:

Theorem 2: Using the FMP algorithm, all the agents converge to final positions and zero velocities.

Proof: By the Barbalat lemma, we have $\lim_{t \to \infty} v(t) = 0$. Consequently, there exists a $T > 0$ such that $\|v_i(t)\| < v_{\max}$ $\forall i, t \geq T$. Thus, from T onward, the system satisfies

$$\begin{cases} \dot{p}_i = v_i \\ \dot{v}_i = -\partial U_\lambda(p)/\partial p_i - c_2 v_i \end{cases} \tag{37}$$

which is a continuous time-invariant system. Since from Lemma 2, we have $H(t) \leq H(0)$ $\forall t$, we know all signals are bounded. Hence, LaSalle's theorem applies. We look for the largest invariant set on which $v(t) \equiv 0$. Setting $v(t) \equiv 0$ in the dynamics, we obtain $\partial U_\lambda(p)/\partial p_i = 0$. Hence, $p(t)$ approaches a stationary point of U_λ. ∎

IV. SIMULATION RESULTS

In this section, we present several simulation results of 2-D and 3-D FMP over three main benchmark problems. We compare the results with the most promising existing algorithm ORCA [7]. The Java implementation of the ORCA algorithm is used in these experiments [22]. We have implemented the FMP algorithm in Java, but our simulator is implemented in MATLAB. Both FMP and ORCA algorithms are called from Java in the MATLAB simulator to have a fair comparison. These experiments evaluate the FMP algorithm over scalability issue, obstacle avoidance, deadlock avoidance, optimality of the produced motion, and the ability to solve dense as well as 3-D problems. The algorithms are compared over transition
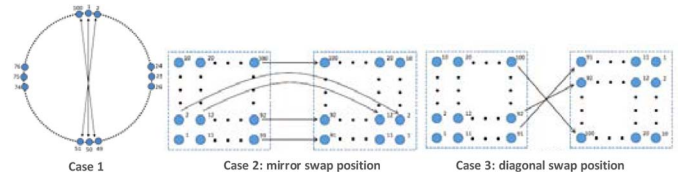

Fig. 5. Three benchmark motion planning problems with 100 agents.

TABLE I
TRANSITION AND EXECUTION TIME FOR THREE BENCHMARK PROBLEMS FOR FMP AND ORCA ALGORITHMS

Example	initial distance (m)	FMP algorithm		ORCA algorithm	
		Transition time (s)	Execution time (ms)	Transition time (s)	Execution time (ms)
circle	-	23.22	292	47.85	784
mirror swap	6	93.50	594	no-result	no-result
	6.5	49.52	364	597.25	11328
	7.5	36.06	273	460.75	7849
	8.5	32.27	273	166.92	2092
	9.5	29.49	272	76.44	1078
diagonal swap	6	38.28	287	no-result	no-result
	6.5	44.4	396	837.63	15956
	7.5	21.77	200	287.24	4883
	8.5	27.19	217	258.52	3808
	9.5	36.84	293	266.84	4317

and execution time parameters. Transition time is the time that takes for agents to move from their initial positions to final positions applying the motions produced by the algorithm and the execution time is the offline execution time of the algorithm (time taken to solve the motion planning problem) which shows the computational complexity of the algorithm. We execute our implementation on a laptop running Windows 10, with an Intel Core i7-7700HQ 2.8-GHz CPU and 16-GB RAM. The step size Δt is set to 0.02 s and EndMaxD = 0.05 m for FMP in all the simulations. The videos accompanying this section are available in Extensions.

A. Dense Benchmark Problems Test

In this section, we compare the performance of the FMP algorithm versus ORCA over three benchmark problems [23] with 100 agents represented in Fig. 5. The first case shows 100 agents whose goal is to move across a circle to the antipodal position. The second and third cases show two different swap positions for 100 agents: mirror and diagonal. For both algorithms: in the circle example $d^* = 3$ m and $V_{\max} = 15$ m/s and in both swap position examples $d^* = 5$ m and $V_{\max} = 15$ m/s. Note that as ORCA does not have any direct input parameter to set required separation distance, we set the input parameter radius = $d^*/2$, which yields the same result. We use the same method in other ORCA tests as well.

Table I compares the transition and execution times of FMP and ORCA algorithms over these three benchmark problems. As this table shows for all the test cases, FMP significantly outperforms ORCA in terms of both transition and execution times.

To show the performance of the algorithms over dense problems, we increase the problems' density on the second and third benchmarks in several steps. Therefore, the initial and final horizontal and vertical distance between neighboring
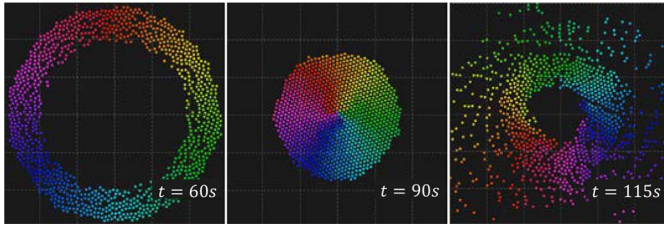
Fig. 6. Simulation of 1000 agents trying to move to antipodal positions at transition times: 60, 90, and 115 s. Agents smoothly move through the congestion that forms in the center. The full video is available as Extension.

points is decreased from 9.5 to 6 m to shrink the problem. The results are compared in Table I.

As the results show, the FMP algorithm has lower transition times, especially in dense problems. ORCA, while claiming to be an optimal algorithm, struggles with finding a time-optimal solution. This is most clearly shown in the original ORCA author's video post of 14 agents position exchange on a circle, which is a highly packed scenario (minute 1:25 of the video [24]). The reason is that ORCA has to reduce the agents' speed to keep its stability, which causes long transition times, especially in dense problems, which is the problem of the focus of many applications. Also, as we expected and as is explained in Section III-E1 although both algorithms have the same order of computational complexity, as calculation of the value of the repulsive function is the only computational effort that should be done for each neighbor at each step of the FMP algorithm, the execution time of the FMP is much lower than ORCA. This difference is more significant in denser problems. It is also the result of a smaller number of steps each agent takes to reach their goals using the FMP algorithm. Note that transition time is a factor of the number of steps and the step size (transitiontime = #steps × Δt). As the step size for both algorithms is equal to ($\Delta t = 0.02$ s), the algorithm that has lower transition time converges to its final solution in a fewer number of steps.

B. Scalability Tests

We test the scalability of the FMP algorithm over the circle benchmark problem depicted in case 1 of Fig. 5. $d^* = 5$ m and $V_{max} = 15$ m/s in this test. We increase the number of agents from 10 to 1000. Table II shows the execution and transition time for this test. As the results show, the algorithm scales well for a large number of agents. Also, as each agent makes independent decisions, the algorithm can be parallelized efficiently by distributing the computations for agents across multiple processors. In these experiments, we do not use the parallelization, but it can be added simply to later extensions. Fig. 6 shows three snapshots of the FMP algorithm for the circle benchmark problem with 1000 agents, which shows that the agents move smoothly through the congestion that forms in the center.

C. Obstacle-Avoidance Tests

This section presents simulation results that show that the obstacle-avoidance version of the FMP algorithm is able to

TABLE II
TRANSITION AND EXECUTION TIME OF FMP OVER CIRCLE BENCHMARK PROBLEM WITH UP TO 1000 AGENTS AND WITHOUT PARALLELIZATION

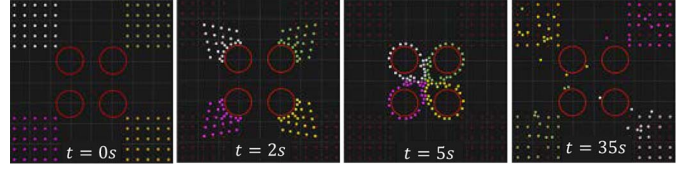| Number of agents	10	100	250	500	1000
Transition time (s)	7.51	28.8	67.86	145.54	270.14
Execution time (ms)	28	245	1949	12402	72343

Fig. 7. Simulation of four groups of 25 agents that moves with opposite directions through a narrow passage formed by the presence of four static obstacles. Red dots represent initial and goal positions—designated times indicating transition time. The full video is available as Extension.

TABLE III
COMPARISON OF FMP AND ORCA OVER 100 RANDOM TEST CASES

	FMP	ORCA
Number of deadlocks	0	9
Overall min separation distance (m)	5.07	4.98
Average Transition time (s)	47.30	422.51
Average Execution time (ms)	99.24	1527.45

avoid dynamic and static obstacles successfully. In these experiments, $\acute{\rho} = \rho = 7.5 \times 10^6$ and $\acute{d^*} = 0.5$ m $< d^*$. The first simulation shows four groups of 25 agents diagonally swapping their positions through a passage formed by the presence of four static obstacles. Fig. 7 shows four snapshots of the FMP algorithm solving this problem. It is numerically verified that no agent ever entered any of the four obstacles. The video that is available as an Extension video shows how the algorithm avoids dynamic obstacles as well as static ones.

D. Random Deadlock-Avoidance Tests

In this simulation, we generated 100 random test cases to compare the average performance of the ORCA and FMP algorithms over transition and execution times and the number of deadlocks occurring in each algorithm. Each test case includes 30 random points generated in a [40 m × 40 m] environment. $d^* = 5$ m and $V_{max} = 3$ m/s. Poisson-disc sampling [25] is used for producing both initial and final random points that are packed but no closer to each other than the minimum distance d. Table III shows the result of this comparison.

As is represented in Table III, ORCA becomes stuck in a deadlock in 9% of the problems. Deadlock is one of the drawbacks of most motion planning algorithms, especially in packed problems. The deadlock problem in ORCA is discussed in [7] and [26]–[28], and some deadlock-avoidance versions have been proposed. Alonso-Mora *et al.* [7] explained the deadlock problem in a nonholonomic version of ORCA in detail. Note that this deadlock issue is inherited from the original ORCA for holonomic agents [28]. ORCA experiences deadlock in packed scenarios in which the distance between goal locations is less than $4r$, with r the radius of the agents [7]. In such scenarios, deadlock for ORCA happens
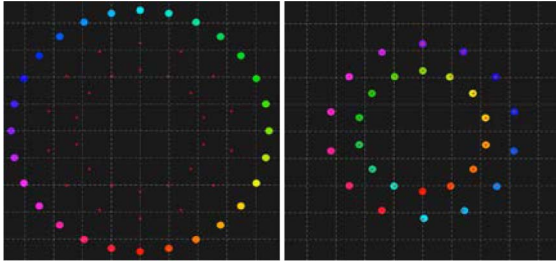
Fig. 8. Snapshots of initial and final agents' formations in the time-optimality test. Agents color shows how initial and final nodes are assigned to each other.

Fig. 9. Comparison of FMP and ORCA produced motions transition times with theoretical lower bound of optimal transition time (Lbt_{opt}).

Fig. 10. Simulation of 100 3-D agents with random initial and goal locations using the 3-D version of FMP as the motion planner. Red dots represent the goal locations. t indicates the transition time. The full video is available as Extension.

Fig. 11. Minimum distance between all agents in [0, 15 s] time interval in the 3-D version of the FMP algorithm.

when all the agents reach their goal locations except a few ones that are not able to reach their goal positions because their motion to the goal is blocked by other agents that are already in their final positions. An extension video shows the performance of FMP on a famous ORCA deadlock scenario [7, Fig. 8]. This article presents a mathematical analysis of the convergence of FMP (Section III-E3), which proves that the algorithm does not have any livelock. Livelock is a condition that occurs when two or more agents continually change their positions in response to change in other agents, and as a result, none of the agents make any progress. As Table III shows, FMP provides lower transition and execution times and better guarantees minimum separation distance d^*. These results confirm the results reported in Section IV-A.

Another impressive result that the data presented in Tables I–III reveal that in all the experiments, the execution time of FMP is always much less than its transition time. This means that FMP is able to do the calculations for each agent while it is transiting toward its goal location.

E. Time-Optimality Test

This section presents simulation results for the evaluation of the time optimality of the motion produced by the FMP algorithm. The following equation presents a theoretical lower bound for the optimal transition time:

$$Lbt_{opt} = \text{lower bound of } t_{opt} = \underset{i=1}{\overset{n}{\text{Max}}}(\|\mathcal{F}_i - \mathcal{I}_i\|/v_{max}).$$

$$(38)$$

Lbt_{opt} shows the time that takes for all the agents to reach their goal locations if they all go straight toward their goal at their maximum speed. This value is a lower bound of the optimal transition time because it does not consider the time agents need to avoid collision with each other. Although Lbt_{opt} is not the optimal transition time; it still can give us a reasonable estimation of how far the FMP transition time is from the optimal value.

In this simulation, agents change their formation from a circle with 28 nodes to a double circle. Fig. 8 shows the snapshots of initial and final configurations. The initial and final distance between nodes decreased from 12 to 1 m to make a range of easy to hard problems. The parameters are set as follows: $d^* = 0.4$ m and $v_{max} = 10$ m/s. Fig. 9 compares the quality of the motions produced by FMP and ORCA with the lower bound of the optimal motion (Lbt_{opt}).
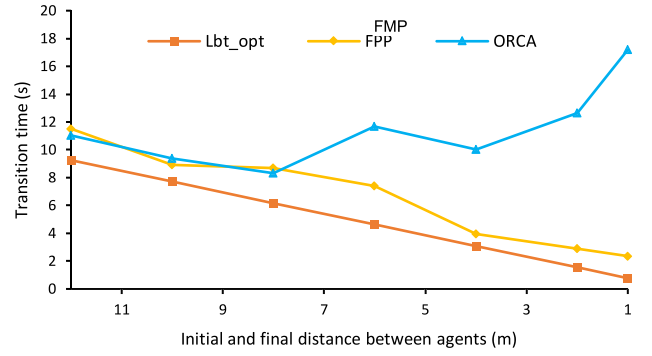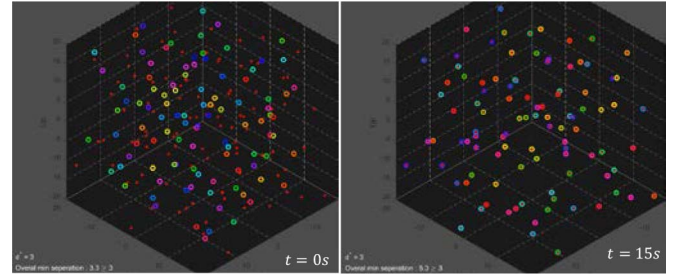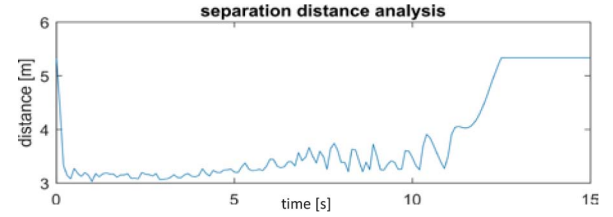
The optimal transition time lies somewhere between the Lbt_{opt} and FMP lines in this figure. As represented in this figure, in all the easy to hard test cases, the transition time of the motion produced by FMP is very close to Lbt_{opt} and is even closer to the optimal transition time consequently. ORCA produces near to optimal transition time for low-density problems, but as the problem becomes harder, its transition time increases sharply.

F. 3-D FMP

The 3-D simulation of the FMP algorithm is presented in this section. Fig. 10 shows the consecutive snapshots of 3-D simulation of FMP for 100 agents. Each agent represents a UAV moving in 3-D. The initial and final state of the agents is chosen at random using Poisson-disc sampling. The parameters are set as follows: $d^* = 3$ m, $v_{max-horizontal} = 9$ m/s, $v_{max-up} = 3$ m/s, and $v_{max-down} = -6$ m/s. As the figure shows, the agents reach their goal locations in a few seconds,

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

SEMNANI *et al.*: FORCE-BASED ALGORITHM FOR MOTION PLANNING OF LARGE AGENT

11

and the minimum separation distance guarantee is numerically verified, as is depicted in Fig. 11. The full video of this experiment and a few more 3-D examples is available as Extension.

V. CONCLUSION

This article presented FMP, a real-time FMP algorithm. We show that the algorithm addresses several practical issues in the motion planning of a multiagent system. First, it is an entirely distributed algorithm in which each agent only requires knowledge of the relative positions of neighboring agents. Second, it has very low computational and communication overhead over the agents. Third, the algorithm scales well to large agent teams involving thousands of agents. Fourth, using this algorithm, agents produce near to time-optimal motions that have much lower transition time in comparison to other famous collision-avoidance algorithms. Fifth, the algorithm enables agents to avoid both dynamic and static obstacles in the environment effectively. Sixth, it can produce deadlock-free motions, even in dense and packed scenarios. Seventh, the algorithm covers the movement of agents in both 2-D and 3-D environments. Eight, it has a solid mathematical background that not only guarantees the production of collision-free motions but also guarantees to keep the minimum required separation distance between the agents. Also, it guarantees the convergence of the algorithm. Finally, it supports user preferences, such as maximum allowed velocity of agents and minimum required separation distance between them. For future work, we are investigating solutions to produce more time-optimal motions leading to lower transition time. Also, expanding this algorithm to take into account agents with more complicated kinematic constraints can be considered as another future work for this article.

APPENDIX

This article has supplementary downloadable material available at https://youtu.be/pruxeMCniDM. This includes an MP4 video showing simulations from Section IV, including scalability, obstacle avoidance, and deadlock-avoidance tests, and the ability to solve dense problems.

REFERENCES

[1] C. W. Reynolds, "Flocks, herds, and schools: A distributed behavioral model," *ACM Comput. Graph.*, vol. 21, no. 4, pp. 25–34, 1987.
[2] R. Olfati-Saber, "Flocking for multi-agent dynamic systems: Algorithms and theory," *IEEE Trans. Autom. Control*, vol. 51, no. 3, pp. 401–420, Mar. 2006.
[3] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006.
[4] R. A. Brooks, "Solving the find-path problem by good representation of free space," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. SMC-13, no. 2, pp. 190–197, Mar./Apr. 1983.
[5] K. S. Barber, T. H. Liu, and S. Ramaswamy, "Conflict detection during plan integration for multi-agent systems," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 31, no. 4, pp. 616–627, Aug. 2001.
[6] B. H. Lee and C. S. G. Lee, "Collision-free motion planning of two robots," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. SMC-17, no. 1, pp. 21–32, Jan. 1987.

[7] J. Alonso-Mora, A. Breitenmoser, M. Rufli, R. Siegwart, and P. A. Beardsley, "Image and animation display with multiple mobile robots," *Int. J. Robot. Res.*, vol. 31, no. 6, pp. 753–773, 2012.
[8] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal N-body collision avoidance," in *Robotics Research*. Berlin, Germany: Springer, 2011, pp. 3–19.
[9] D. Zhou, Z. Wang, S. Bandyopadhyay, and M. Schwager, "Fast, on-line collision avoidance for dynamic vehicles using buffered Voronoi cells," *IEEE Robot. Autom. Lett.*, vol. 2, no. 2, pp. 1047–1054, Apr. 2017.
[10] J. A. Preiss, W. Hönig, N. Ayanian, and G. S. Sukhatme, "Downwash-aware trajectory planning for large quadcopter teams," 2017. [Online]. Available: arXiv:1704.04852.
[11] A. Federico, A. P. Schoellig, and R. D'Andrea. "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach," in *Proc. IEEE/RSJ Int. Conf. IEEE Intell. Robots Syst. (IROS)*, 2012, pp. 1917–1922.
[12] M. Daniel, A. Kushleyev, and V. Kumar, "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2012, pp. 477–483.
[13] S. Bandyopadhyay, S.-J. Chung, and F. Y. Hadaegh, "Probabilistic swarm guidance using optimal transport," in *Proc. IEEE Conf. Control Appl.*, Juan Les Antibes, France, 2014, pp. 498–505.
[14] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2017, pp. 285–292.
[15] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," 2018. [Online]. Available: arXiv:1805.01956.
[16] S. H. Semnani and O. A. Basir, "Semi-flocking algorithm for motion control of mobile sensors in large-scale surveillance systems," *IEEE Trans. Cybern.*, vol. 45, no. 1, pp. 129–137, Jan. 2015.
[17] S. H. Semnani, "Distributed target engagement in large-scale mobile sensor networks," Ph.D. dissertation, Elect. Comput. Eng., Univ. Waterloo, Waterloo, ON, Canada, 2015.
[18] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Res. Logist.*, vol. 2, nos. 1–2, pp. 83–97, 1955.
[19] S. Giordani, L. Marin, and M. Francesco, "A distributed algorithm for the multi-robot task allocation problem," in *Proc. Int. Conf. Ind. Eng. Other Appl. Appl. Intell. Syst.*, 2010, pp. 721–730.
[20] D. P. Bersekas, "The auction algorithm: A distributed relaxation," *Ann. Oper. Res.*, vol. 14, no. 1, pp. 105–123, 1988.
[21] D. P. Bertsekas and D. A. Castanon, "Parallel synchronous and asynchronous implementations of the auction algorithm," *Parallel Comput.*, vol. 17, nos. 6–7, pp. 707–732, 1991.
[22] P. Janovsky. (Mar. 28, 2018). *Java Implementation of ORCA Collision-Avoidance Algorithm*. [Online]. Available: https://github.com/mcapino/orca
[23] D. Zhou, Z. Wang, S. Bandyopadhyay, and M. Schwager, "Fast, on-line collision avoidance for dynamic vehicles using buffered Voronoi cells," *IEEE Robot. Autom. Lett.*, vol. 2, no. 2, pp. 1047–1054, Apr. 2017.
[24] J. Alonso-Mora, A. Breitenmoser, M. Rufli, R. Siegwart and P. Beardsley. (Mar. 28, 2018). *Optimal Reciprocal Collision Avoidance for Multiple Non-Holonomic Robots*. [Online]. Available: https://www.youtube.com/watch?v=s9lvMvFcuCE
[25] R. Bridson, "Fast Poisson disk sampling in arbitrary dimensions," in *Proc. SIGGRAPH Sketches*, 2007, p. 22.
[26] A. Giese, D. Latypov, and N. M. Amato, "Reciprocally-rotating velocity obstacles," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2014, pp. 3234–3241.
[27] S. A. Khan, M. J. Y. Ayaz, S. O. Gillani, M. Naveed, A. H. Qureshi, and K. F. Iqbal, "Collaborative optimal reciprocal collision avoidance for mobile robots," *Int. J. Control Autom.*, vol. 8, no. 8, pp. 203–212, 2015.
[28] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P. Beardsley, and R. Siegwart, "Optimal reciprocal collision avoidance for multiple non-holonomic robots," in *Distributed Autonomous Robotic Systems*. Heidelberg, Germany: Springer, 2013, pp. 203–216.
[29] J. S. Adams, W. Sun, and Y. Chen, "Formations with decentralized centroidal Voronoi tessellation algorithm," *IFAC Proc. Vol.*, vol. 42, no. 22, pp. 13–18, 2009.
[30] W. Sun *et al.*, "A multi-robot target tracking algorithm with centroidal Voronoi tessellation and consensus strategy," in *Proc. IEEE 29th Chinese Control Conf.*, 2010, pp. 4607–4612.

Samaneh Hosseini Semnani received the B.Sc. degree in computer engineering from the University of Kashan, Kashan, Iran, in 2005, the M.Sc. degree from the University of Isfahan, Isfahan, Iran, in 2007, and the Ph.D. degree in computer engineering from the University of Waterloo, Waterloo, ON, Canada, in 2015.

From 2017 to 2019, she was a Postdoctoral Fellow jointly with the University of Toronto, Toronto, ON, Canada, and Ryerson University, Toronto. She is currently an Assistant Professor with the ECE Department, Isfahan University of Technology, Isfahan. Her research interests are in the area of multiagent systems, swarm intelligence, sensor networks, and deep learning.

Anton H. J. de Ruiter received the B.E. degree in mechanical engineering from the University of Canterbury, Christchurch, New Zealand, in 1999, and the M.A.Sc. and Ph.D. degrees in aerospace engineering from the University of Toronto, Toronto, ON, Canada, in 2001 and 2005, respectively.

From 2006 to 2008, he was a Visiting Research Fellow with the Canadian Space Agency, Montreal, QC, Canada, and an Assistant Professor with the Department of Mechanical and Aerospace Engineering, Carleton University, Ottawa, ON, Canada, from 2009 to 2012. He is currently an Associate Professor and a Canada Research Chair (Tier 2) with the Department of Aerospace Engineering, Ryerson University, Toronto. His research interests are in the area of guidance, navigation, and control of aerospace systems.

Dr. Ruiter currently serves as the Editor-in-Chief of the *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, and an Associate Editor for the IEEE TRANSACTIONS ON AEROSPACE AND ELECTRONIC SYSTEMS. He is an Associate Fellow of AIAA.

Hugh H. T. Liu received the bachelor's degree from Shanghai Jiao Tong University, Shanghai, China, in 1991, the master's degree from the Beijing University of Aeronautics and Astronautics, Beijing, China, in 1994, and the Ph.D. degree from the University of Toronto, Toronto, ON, Canada, in 1998.

He is a Professor with the University of Toronto Institute for Aerospace Studies, Toronto, where he has been on the faculty since 2000. He also currently serves as the Director of the Natural Science and Engineering Research Council of Canada Collaborative Research and Training Experience Program on Unmanned Aerial Vehicles and Centre for Aerial Robotics Research and Education. His research interests are in the area of aircraft systems and control include autonomous unmanned systems, cooperative and formation control, fault-tolerant control, active control on advanced aircraft systems, as well as integrated modeling and simulations.

Prof. Liu currently serves as an Associate Editor for the *AIAA Journal of Guidance, Control and Dynamics* and *Canadian Aeronautics and Space Journal*. He is a Fellow of the Engineering Institute of Canada and the Canadian Society of Mechanical Engineers, an Associate Fellow of AIAA, and a registered Professional Engineer in Ontario, Canada.