Analysis of 3D Position Control for a Multi-Agent System of Self-Propelled Agents Steered by a Shared, Global Control Input

Li Huang, Julien Leclerc, and Aaron T. Becker

Abstract—This paper investigates strategies for 3D multi-agent position control using a shared control input and self-propelled agents. The only control inputs allowed are rotation commands that rotate all agents by the same rotation matrix. In the 2D case, only two degrees-of-freedom (DOF) in position are controllable. We review controllability results in 2D, and then show that interesting things happen in 3D. We provide control laws for steering up to nine DOF in position, which can be mapped in various ways, including to control the $x, y, z$ position of three agents, make four agents meet, or reduce the spread of $n$ agents.

I. INTRODUCTION

Interest in swarm robotics in the control and robotics communities has increased. Compared to highly intelligent and advanced robots, each agent in a swarm robot system is inexpensive, easy to manufacture, and suitable to deploy in large populations [1]. At macro scales, swarm robots such as micro aerial vehicles and 2D autonomous ground vehicles have great potential to be applied to sensing, mapping, localization, surveillance, rescue, etc; at micro scale, agents such as ferromagnetic microrobots, magnetotactic bacteria, and catalytic Janus particles are researched for target drug delivery, non-invasive surgery, micro assembly, etc.

In this paper, we consider a swarm of simple robots with limited communication capability such that agents are commanded by a central system, and agent-to-agent information exchange is not applicable. The swarm system might consist of hundreds or thousands of agents, but each agent receives a copy of the same control input. For example, one potential application is steering catalytic Janus particles with uniform magnetic fields. These particles are self-propelled by a reaction between platinum on the particle and the liquid the particle swims in. The particles are also magnetic.

The external magnetic field applies a torque that aligns the magnetic dipole of the particle with the external field. The results in [2] demonstrated steering an individual particle to a desired location and demonstrated that multiple particles could be made to move in different directions when under the control of a single magnetic field.

Consider a swarm with $n$ agents in free space, with their headings randomly initialized. The agents are modeled as self-propelled points that move at a constant velocity along a thrust vector that is fixed in their local coordinate frame. A central system controls agents via a broadcast mechanism. The only control inputs allowed are rotation commands that rotate each agent’s local coordinate frame by the same rotation matrix.

This model is similar to ensemble control systems [3]–[13]. In these problems an ensemble of nearly identical agents that differ only in a set of one or more parameters, are each steered by the same control input. However, the major challenge in this paper is the constrained control input in (28), where $R(t)$ needs to be a rotation matrix.

This paper address the 3D position control problem of a multi-agent system using a shared rotation control input, including (i) with no state perturbations, simultaneously steering up to three agents to arbitrary locations; (ii) with independent rotation perturbations, enabling simultaneously steering many agents ($n > 3$) to arbitrary locations. While boundary interactions could be another method to steer large populations of agents [14], this paper focuses on simple agents in free-space that are affected only by their thrust, local coordinate frames, and the shared control command.

Previous works have addressed control strategies for 3D multi-agent navigation and planning, for example, [15] discussed a distributed control scheme based on Navigation Functions to drive aircraft-like vehicles towards their targets while avoiding colliding into each other or obstacles; [16] proposed an acyclic minimally structural persistent graph based formation control to steer a group of autonomous agents in 3D space; and in [17], a decentralized 3D formation control algorithm was developed for a multi-agent system with unknown dynamics. However, these control schemes rely on independent control of each agent, which is not...
applicable within the scope of this paper.

The key results of the this paper include techniques to make \( n \) agents follow orbits to arbitrary \( x, y, z \) positions, and reducing the variance of the swarm (sections III-D to III-F). Simulation code is available at GitHub [18] and Wolfram Demonstration Projects [19], and the video demo is on Youtube [20].

II. 2D CONTROL OF SELF-PROPELLED AGENTS

We begin our analysis with a 2D version of the broadcast control problem. Figure 1 shows a schematic of a self-propelled point robot in 2D (a disk) and 3D (a sphere).

Bretl [21] and Das et al. [22] have discussed the shared control problem in a 2D plane. Bretl proposed a control law that directs two agents to meet at the same location simultaneously, and with input perturbations. Das demonstrated the possibility of achieving position consensus for large number of agents.

Consider \( n \) agents in a 2D \( xy \) plane: the origin of each local coordinate frame coincides with the individual center of mass, and let the local coordinate frame be initially aligned with the global coordinate frame. The kinematics of the \( i^{th} \) agent is given by

\[
x_i(t + 1) = x_i(t) + R_\theta v_i(t),
\]

with \( x_i(t) \in \mathbb{R}^{2 \times 1} \) the position at time \( t \), \( v_i(t) \in \mathbb{R}^{2 \times 1} \) the thrust vector, and \( R_\theta \) the shared control command that rotates the agent along its local \( z \)-axis:

\[
R_\theta = \begin{bmatrix}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta
\end{bmatrix}
\]

Given two self-propelled agents with any initial positions \( x_1, x_2 \in \mathbb{R}^{2 \times 1} \), and thrust vectors \( v_1, v_2 \in \mathbb{R}^{2 \times 1} \) \( (v_1 \parallel v_2) \), they can meet at only one unique location \( \hat{x}_{1,2} \).

\[
\hat{x}_{1,2} = x_2 + \frac{1}{2} \left[ \tan \frac{1}{\phi_2 - \phi_1} \tan \frac{2}{\phi_2 - \phi_1} \right] (x_1 - x_2)
\]

where \( \phi_i = \arctan(v_i) \). For \( n > 2 \) such agents in a 2D workspace, there are \( n(n-1)/2 \) potential collision locations, as shown in Fig. 2.

In general, a sequence of commands for shared control cannot make large number of agents \( (n > 2) \) to meet simultaneously except for some special initial conditions. If thrust vector perturbations are permitted, Das shows the possibility of point convergence for all agents in [22].

III. 3D CONTROL OF SELF-PROPELLED AGENTS

The effect of any command sequence for 2D position control (without perturbations) of self-propelled agents can be replicated by three commands: an initial rotation along the local \( z \)-axis, a translation, and a final rotation along the local \( z \)-axis. Therefore, the configuration space for these agents’ position is two-dimensional, no matter how many agents are used. This is because the ending position of each agent is the result of the same rigid body transformation modulo an initial rotation. Interestingly, in 3D-space multiple rotation and translation operations can be concatenated to control more than three degrees of freedom.

This work reports on simultaneous 3D position control for multiple self-propelled agents. Without loss of any generality, each agent is initialized with different thrust vectors. The origin of each agent’s coordinate frame is at the agent’s position, and the \( x, y, \) and \( z \) axes are aligned with the global coordinate frame. The agents implement shared rotation commands based on their local coordinate frames. For simplification, the actuation time for rotations is negligible compared to the time of translation.

A. Steer one self-propelled agent

To deliver a single self-propelled agent from its initial position \( x \in \mathbb{R}^{3 \times 1} \) to a goal location \( \hat{x} \), the agent must be rotated so that the thrust vector points toward \( \hat{x} \). Let \( \mathbf{v} \) be the initial thrust vector defined in the global coordinate frame. The desired thrust vector can be described as

\[
\mathbf{\hat{v}} = \frac{\hat{x} - x}{\|\hat{x} - x\|_2}.
\]

First, identify the normal vector \( \mathbf{k} \) of a plane containing both \( \mathbf{v} \) and \( \mathbf{\hat{v}} \).

\[
\mathbf{k} = \mathbf{v} \times \mathbf{\hat{v}}.
\]

Next, rotate \( \theta \) about \( \mathbf{k} \) to align the thrust vector with \( \mathbf{\hat{v}} \), where

\[
\theta = \arccos(\mathbf{v}, \mathbf{\hat{v}}).
\]

B. Station-keeping (orbits) with multiple agents

The following sections provide control laws for steering 3D agents to goal positions. A preliminary challenge is to keep multiple agents at given locations. The solution in 2D is to revolve about the local \( z \)-axis at a constant rate, where the \( z \)-axis is perpendicular to the motion plane. The orbital radius is the thrust velocity divided by the angular frequency: \( r = |\mathbf{v}|/\omega \). The faster we revolve, the tighter the orbit. In three dimensions this technique no longer works.

The position change of the \( i^{th} \) agent under a shared command that rotates \( \theta(t) \) radians about an axis \( k \) is given by

\[
\int_0^t R_{k,e^{\theta(r)}} v_i dr.
\]

It is easy to show that a rotation about the \( x \)-axis does not, in general, return all agents to the initial
location. As shown in Fig. 3, rotating $\theta$ radians at 1 radian per second about the local $x$-axis moves the agents to

$$\begin{bmatrix} \Delta x_i, \Delta y_i \cos \theta - 1 + \Delta y_i \sin \theta, \Delta y_i (1 - \cos \theta) + \Delta z_i \sin \theta \end{bmatrix}^T.$$ 

After one revolution, each agent has been displaced by $[2\pi v_x, 0, 0]$, and any agent with a non-zero $v_x$ has followed a helical trajectory. Self-propelled agents with positive $v_x$ will have moved in the positive $x$ direction, and the others in the negative $x$.

A solution that returns all self-propelled agents to their initial positions is given by eight revolutions that toggle between revolving about the local $x$ and $y$-axes. All revolutions proceed at a constant angular velocity and, as in 2D, the maximum deviation scales linearly with the inverse of the angular velocity:

$$R_{x, \pi}R_{y, -\pi}R_{x, \pi}R_{y, \pi}R_{x, \pi}R_{y, -\pi}R_{x, -\pi}R_{y, \pi} = 1.$$

All $x, y, z$ subscripts for any rotation $R$ refer to the current local $x, y, z$-axes in the following sections. The positions for an agent initially at the origin after each rotation are

$$\begin{bmatrix} \pi v_x \\ -2v_y \\ 2v_y \\ \pi v_x \\ 2v_x + 2v_y \\ -\pi v_y + 2v_x \\ 2v_x + 2v_y \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 4v_y + 4v_x \\ 2v_x + 4v_y \end{bmatrix}.$$ 

(8)

Trajectories of this input sequence are shown in Fig. 4.

C. Simultaneous Position Control

For a system of $n \geq 2$ arbitrary self-propelled agents in a 3D free-space, there are 6n DOF: the position vectors $x_i \in \mathbb{R}^{3 \times 1}$, and the thrust vectors $v_i \in \mathbb{R}^{3 \times 1}$. This work provides both open-loop and closed-loop algorithms to control up to nine DOF in positions with no state perturbations.

For ease of exposition, the shared controller only uses $R_{x, \theta_i}$ and $R_{y, \theta_i}$ as rotation primitives to control the thrust vector heading, because any 3D rotations $R_{k, \theta}$ about a local axis $k$ can be represented by a series of rotations about the current local $x, y,$ and $z$-axis:

$$R_{k, \theta} = R_{x, \theta_1}R_{y, \theta_2}R_{x, \theta_3}.$$ 

(9)

If the shared command steers agents to rotate $\theta_x$ about the current local $x$-axis, the corresponding rotation matrix and resultant thrust orientations in the global coordinate frame are

$$R_{x, \theta_x} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta_x & \sin \theta_x \\ 0 & \sin \theta_x & \cos \theta_x \end{bmatrix},$$

(10)

$$\hat{v}_i = R_{x, \theta_x}v_i,$$

(11)

where the subscripts $x, \theta_x$ in $R_{x, \theta_x}$ denote the local $x$-axis and rotation angle. If the next command is to rotates $\theta_y$ about the current local $y$-axis, the corresponding rotation matrix and thrust orientations in the global coordinate frame are

$$R_{y, \theta_y} = \begin{bmatrix} \cos \theta_y & 0 & -\sin \theta_y \\ 0 & 1 & 0 \\ \sin \theta_y & 0 & \cos \theta_y \end{bmatrix},$$

(12)

$$\hat{v}_i = R_{x, \theta_x}R_{y, \theta_y}v_i.$$ 

(13)

These rotations are performed about the current local $x$ or $y$-axis instead of the fixed global coordinate frame, and both (11) and (13) give a thrust vector $\hat{v}_i$ defined in the global frame.

Each rotation is followed by a translation motion. To simplify the derivation, the time required for any rotation is assumed to be negligible compared to the translation actuation time.

In 3D space multiple self-propelled agents cannot be driven to arbitrary goal locations $\hat{x}_1, \ldots, \hat{x}_n \in \mathbb{R}^{3 \times 1}$ with one rotation and translation. However, concatenating the rotation operations (10) and (12) followed by translations enables controlling additional DOF.

This can be written in matrix form as

$$\begin{bmatrix} \Delta x_1 \\ \vdots \\ \Delta x_n \end{bmatrix} = \begin{bmatrix} R_{1, v_1} & R_{2, v_1} & \cdots & R_{N, v_1} \\ R_{1, v_2} & R_{2, v_2} & \cdots & R_{N, v_2} \\ \vdots & \vdots & \cdots & \vdots \\ R_{1, v_n} & R_{2, v_n} & \cdots & R_{N, v_n} \end{bmatrix} t = R_v t.$$ 

(14)

with $\Delta x_i = \hat{x}_i - x_i$, $R_j = R_{x, \theta_1}R_{y, \theta_2}R_{x, \theta_3}$, $R_{j+1} = R_jR_{y, \theta_{j+1}}$, and $t = [t_1, t_2, \cdots, t_N]^T$. If all $R_j$ and $t$ are
unknown, solving (14) directly is computationally intensive.
Instead, randomly generating the N angles is computationally
cheap and works well in simulation, though methods
shown later can outperform this.

Given the N rotation angles, \( \Delta x_i \), and \( v_i \), where \( i \in \{1, 2, \cdots, n\} \), the goal is to choose \( t \) for (14) that minimizes
\( ||t||_1 \) with \( t_j \geq 0, \forall j \in \{1, N\} \).

Any three independent vectors in \( \mathbb{R}^{3 \times 1} \) forms a basis
for the 3D space. Without loss of generality, assume that \( v_1, v_2, v_3 \) in \( R_v \) (14) are independent and all \( v_i \) are unique.
So any \( v_i \) of \( R_v \) can be expressed as a linear combination of \( v_1, v_2, v_3 \)
\[
v_i = \lambda_{i,1}v_1 + \lambda_{i,2}v_2 + \lambda_{i,3}v_3, \quad (15)
\]
where \( \lambda_{i,1}, \lambda_{i,2}, \lambda_{i,3} \in \mathbb{R} \). Recall that row \( 3i - 2 \) to row \( 3i \in R_v \) has the form
\[
R_v(3i - 2 : 3i, *) = [R_1v_i \quad R_2v_i \cdots R_Nv_i], \quad (16)
\]
where \( v_i \in \mathbb{R}^{3 \times 1} \). Substitute \( v_i \) with (15),
\[
R_v(3i - 2 : 3i, *) = \lambda_{i,1}[R_1v_1 \quad R_2v_1 \cdots R_Nv_1] + \lambda_{i,2}[R_1v_2 \quad R_2v_2 \cdots R_Nv_2] + \lambda_{i,3}[R_1v_3 \quad R_2v_3 \cdots R_Nv_3]
\]
which indicates that any row of \( R_v \) is a linear combination
of its first nine rows. So the row rank of \( R_v \) is at most nine.

Recall that column \( j \) in \( R_v \), has the form
\[
R_v(*, j) = \begin{bmatrix}
R_1v_1 \\
R_2v_2 \\
\vdots \\
R_Nv_N
\end{bmatrix}
\]  
(17)
Assuming there are at most \( k \) independent columns of \( R_v \),
without loss of generality, let \( R_v(*, 1 : k) \) be these columns. Hence, \( R_v(*, j) \) can be expressed as
\[
R_v(*, j) = l_{j,1}R_v(*, 1) + l_{j,2}R_v(*, 2) + \cdots + l_{j,k}R_v(*, k)
\]
\[
\begin{bmatrix}
R_1v_1 \\
R_2v_2 \\
\vdots \\
R_Nv_N
\end{bmatrix} = \begin{bmatrix}
l_{j,1}R_1v_1 + l_{j,2}R_2v_2 + \cdots + l_{j,k}R_kv_k \\
l_{j,1}R_1v_1 + l_{j,2}R_2v_2 + \cdots + l_{j,k}R_kv_k \\
\vdots \\
l_{j,1}R_1v_1 + l_{j,2}R_2v_2 + \cdots + l_{j,k}R_kv_k
\end{bmatrix}
\]
\[
(18)
That is, \( \forall i \in \{1, 2, \cdots, n\} \)
\[
(l_{j,1}R_1v_1 + l_{j,2}R_2v_2 + \cdots + l_{j,k}R_kv_k - R_j)v_i = 0_{3 \times 1}
\]  
(19)
Therefore
\[
l_{j,1}R_1v_1 + l_{j,2}R_2v_2 + \cdots + l_{j,k}R_kv_k = R_j \quad (19)
\]
We write \( R_k \) as
\[
R_k = \begin{bmatrix}
r_{k,11} & r_{k,12} & r_{k,13} \\
r_{k,21} & r_{k,22} & r_{k,23} \\
r_{k,31} & r_{k,32} & r_{k,33}
\end{bmatrix}, \quad (20)
\]
and let \( r_k = [r_{k,11}, r_{k,12}, r_{k,13}, \cdots, r_{k,31}, r_{k,32}, r_{k,33}]^T \) ,
\( l_j = [l_{j,1}, l_{j,2}, \cdots, l_{j,k}]^T \). Then (19) can be written as
\[
[r_1 \quad r_2 \cdots \quad r_k]l_j = r_j. \quad (21)
\]
To have a unique solution to \( l_j, [r_1, r_2, \cdots, r_k] \) needs to be
invertible, that is, \( k = 9 \). We can conclude that the column
rank of \( R_v \) is at most nine. Because the maximum rank of \( R_v \) is nine, \( n \leq 3 \) self-propelled agents can be steered to
arbitrary goal locations, and only if these agents have linearly
independent initial thrust vectors.

Only nine of \( 3n \) DOF can be manipulated if \( n > 3 \). A
special case is to make four agents to meet simultaneously.
There is a unique position \( \hat{x} \) where four such agents can
meet.
\[
\hat{x} = x_1 + \sum_{i=1}^{N} R_i v_1 t_i = x_2 + \sum_{i=1}^{N} R_i v_2 t_i = x_3 + \sum_{i=1}^{N} R_i v_3 t_i = x_4 + \sum_{i=1}^{N} R_i v_4 t_i \quad (22)
\]
Let \( R_t = \sum_{i=1}^{N} R_i t_i, \)
\[
x_1 - x_2 = R_t(v_2 - v_1) \quad (23)
x_2 - x_3 = R_t(v_3 - v_2) \quad (23)
x_3 - x_4 = R_t(v_4 - v_3)
\]
Flatten \( R_t \) as a vector \( r_t \). Rewrite the right side of (23) as
\[
\begin{bmatrix}
x_1 - x_2 \\
x_2 - x_3 \\
x_3 - x_4
\end{bmatrix} = \begin{bmatrix}
R_\Lambda(v_2, v_1) \\
R_\Lambda(v_3, v_2) \\
R_\Lambda(v_4, v_3)
\end{bmatrix} r_t = R_\Lambda v_t. \quad (24)
\]
where
\[
R_\Lambda(v_i, v_j) = \begin{bmatrix}
v_i^T - v_j^T & 0_{1 \times 3} & 0_{1 \times 3} \\
0_{1 \times 3} & v_i^T - v_j^T & 0_{1 \times 3} \\
0_{1 \times 3} & 0_{1 \times 3} & v_i^T - v_j^T
\end{bmatrix} \quad (25)
\]
So \( r_t \) (i.e., \( R_t \)) has a unique solution if and only if \( R_\Lambda \) is
invertible. Hence we can derive the meeting point
\[
\hat{x} = x_1 + R_t v_1. \quad (26)
\]

D. Open-loop control using linear programming

One way to solve (14) is via linear programming. The
objective is to find a vector \( t \) such that the total control time
\( ||t||_1 \) is minimized subject to (14) with \( n = 3 \).
\[
\min \sum_{j=1}^{N} t_j, \quad \text{s. t.} \quad t_j \geq 0, j = 1, \ldots, N \text{ and }
\begin{bmatrix}
R_1v_1 & R_2v_1 & \cdots & R_Nv_1 \\
R_1v_2 & R_2v_2 & \cdots & R_Nv_2 \\
R_1v_3 & R_2v_3 & \cdots & R_Nv_3
\end{bmatrix} t = \begin{bmatrix}
\Delta x_1 \\
\Delta x_2 \\
\Delta x_3
\end{bmatrix} \quad (27)
\]
where the initial thrust vectors are linearly independent.
Because the \( t_j \) must be nonnegative, the number of rotation
matrices (\( N \)) should be much greater than nine, as shown in
Fig. 7a.

E. Feedback control

Let the state space representation of \( n \) such agents be
\[
[\begin{bmatrix}
x_1 & \cdots & x_n
\end{bmatrix}] = R(t) [\begin{bmatrix}
v_1 & \cdots & v_n
\end{bmatrix}], \quad (28)
\]
where \( x_i \) denotes position of the \( i \)th agents, \( v_i \) describes
the initial thrust vector, and \( R(t) \) is the shared control input. We
Fig. 5. Open-loop control simulations using linear programming. The left figure has two self-propelled agents, and the right figure has three agents. The goal locations are indicated by green orbits. In each figure, all spheres move the same total distance and reach the goal location at the same time. A colored line describes the trajectory of each sphere. Black arrows indicate the local coordinate frame $z$-axis for the subsequent move.

We construct an objective function based on the sum of squared Euclidean distance error

$$V(t) = \frac{1}{2} \sum_{i=1}^{n} (\hat{x}_i - x_i(t))^T (\hat{x}_i - x_i(t)), \quad (29)$$

where $\hat{x}_i$ is the $i^{th}$ goal location. We hold $R(t)$ constant for time interval $[t_k, t_k + \tau_k]$, such that $\tau_k$ minimizes the convex objective function. Consider the first-order necessary condition:

$$\dot{V}(t_k + \tau_k) = \frac{d}{dt}V(t_k + \tau_k) = 0,$$

$$- \sum_{i=1}^{n} (\hat{x}_i - x_i(t_k) - \tau_k R(t_k) v_i)^T R(t_k) v_i = 0,$$  

and thus

$$\tau_k = \frac{\sum_{i=1}^{n} \|R(t_k) v_i\|^2}{\sum_{i=1}^{n} \|R(t_k) v_i\|^2} \cdot (31)$$

Note that the time interval $\tau_k$ must be non-negative, so if there exists an $R(t_k)$ such that $\tau_k > 0$, then $\int_{t_k}^{t_k + \tau_k} V(t) dt < 0$, and the total distance error decreases monotonically with time; otherwise, the objective function has reached the minimum. Thus the system kinematics can also be written as

$$x_i(t_{k+1}) = x_i(t_k) + R(t_k) v_i \tau_k. \quad (32)$$

F. Greedy optimal control

We discussed the optimal actuation time to minimize the objective function $V(t)$ with a constant rotation matrix $R(t)$ within time interval $[t, t + \tau_k)$. By carefully choosing this $R(t)$, the objective function follows the steepest gradient during each time interval:

$$R(t_k) = \arg \min_{\alpha_k, \beta_k, \gamma_k} \dot{V}(t_k), \quad (33)$$

with $R(t_k)$ a function of rotation angles $\alpha_k, \beta_k$, and $\gamma_k$

$$R(t_k) = R(t_{k-1}) R_x, \alpha_k R_y \beta_k R_x \gamma_k. \quad (34)$$

This indicates that at $t_k$, each agent implements the shared rotation control $R(t_k)$ which is equivalent to rotating $\alpha_k$ about the current local $x$-axis, then $\beta_k$ about the current $y$-axis, and finally rotating $\gamma_k$ about the current $x$-axis.

Fig. 6. Simulations of self-propelled agents using greedy optimal control. (a) Three agents reach arbitrary goal locations simultaneously. (b) The corresponding objective function plot with simulation time. (c) Four agents move towards their mean position till they meet. (d) The corresponding objective function plot with simulation time.

G. Control with state perturbations

Under ideal conditions, agent motions are assumed to be perfectly implemented without errors. Up to four agents can be steered to meet simultaneously with a shared control input. In general, for $n > 4$ agents, the objective function reaches a local minimum, and the shared control cannot bring all agents to their target locations simultaneously.

In practice, agent rotations and translations may not be precise due to process noise and measurement noise. Inspired by the 2D results in [22], it is possible to escape local minima in 3D, if we relax previous assumptions and assume agents are subjected to independent random disturbances on their thrust vectors after each translation. In the following we show that there always exist disturbances that enable steering the agents closer to their goal. In simulation, we show that randomly perturbing each agent’s thrust vector enables convergence.

Section III-E shows that $\tau_k \leq 0$ if the objective function reaches a local minimum, and according to $31$,

$$\sum_{i=1}^{n} (\hat{x}_i - x_i(t_k))^T v_i(t_k) \leq 0. \quad (35)$$

where $R(t_k) v_i$ is replaced by $v_i(t_k)$. Let $\delta_i(t_k) \in \mathbb{R}^3$ be a perturbation of the $i^{th}$ thrust vector, such that the perturbed thrust vector $v'_i(t_k) \parallel (\hat{x}_i - x_i(t_k))^T$, where

$$v'_i(t_k) = \frac{v_i(t_k) + \delta_i(t_k)}{\|v_i(t_k) + \delta_i(t_k)\|} . \quad (36)$$

Therefore

$$\sum_{i=1}^{n} (\hat{x}_i - x_i(t_k))^T v'_i(t_k) > 0. \quad (37)$$
except for the case that the total position error is 0.

This example proves the existence of a perturbation that makes the objective function monotonically decreasing, even if the system is in a local minimum. In this paper’s simulations, the thrust vector perturbations are sampled from a zero-mean normal distribution.

IV. CONTROL SIMULATIONS OF SELF-PROPELLED SPHERES

In this section, we show the controllability for self-propelled agents in 3D free-space, including steering up to three spheres to arbitrary goals with a shared open-loop control (linear programming) and a shared closed-loop control, and control of four agents to meet. With perturbations on thrust vectors, we show the capability of driving ten agents to the origin.

In Figure 5, open-loop control with linear programming is implemented with up to three self-propelled agents and drives them to arbitrary goals. According to section III-D, $N$ angles are randomly generated to provide a large number of rotation matrix candidates, but linear programming selects $N_k$ among $N$ ($N_k \ll N$) matrices to apply actuation and steer the spheres to goal locations. In the simulation, a rotation matrix could be generated by $R(t) = R_{x,\alpha} R_{y,\beta} R_{x,\gamma}$, where $\alpha, \beta, \gamma \in [0, 2\pi]$. $N$ is set to 200, and usually $N_k = 9$. Figure 7a shows that $N$ can affect the performance of linear programming: if $N$ is small, the provided paths to goals might be much longer than the optimal solution. The total path length decreases monotonically with $N$. The total path length has little change when $N \geq 200$.

Section III-E introduced a closed-loop control law such that a rotation matrix $R(t)$ is held constant for each actuation time interval $\tau_k$. Aply choosing each $\tau_k$, the cost function decreases monotonically along the trajectory. In simulation, $R(t)$ is generated with the following methods: (i) random angle generation, (ii) using the $N_k$ rotation matrices selected by linear programming, and (iii) minimizing $V(t)$ in (30) with respect to $\alpha_k, \beta_k, \gamma_k$. Figure 6 shows the trajectories of greedy optimal control with 3 and 4 agents using method (iii). Not all $\alpha_k, \beta_k, \gamma_k$ must be non-zero values. For example, we could let $\alpha_k, \beta_k = 0$, or $\alpha_k, \gamma_k = 0$, or $\gamma_k = 0$ for greedy optimal control. For more simulation results, please refer to the repository on GitHub [18].

A performance comparison of the above three methods is shown in Figure 7b, which indicates that methods (ii) and (iii) have competitive performance, while method (i) takes about twice as long to converge.

Section III-C showed we can control up to nine DOF with no state perturbations. Figure 6 gives two examples of controlling up to four self-propelled agents: steering three agents to predefined goal locations, and moving four spheres to their mean positions, as shown in (26).

When thrust vector perturbations are considered, the shared control is capable of moving out of local minima and bringing many agents ($n > 4$) to arbitrary locations, as shown in Figure 7d. In addition, the influence of position noises are compared in Figure 7c.

V. CONCLUSIONS

This paper proved limitations on control for self-propelled agents that all receive the same rotation commands, but extended the existing literature which focused on two-dimensional results to show that nine degrees-of-freedom of position can be controlled. In 2D only one agent can be steered to an arbitrary position, and two agents have only one possible meeting point. In 3D up to three agents may be steered to arbitrary positions, and four agents have only one possible meeting point.

There are many avenues for future work. These include optimal control results and analytical solutions to the optimal rotations for the controllers in section III. In particular, we would like to calculate the meeting location for two spheres that requires the shortest control sequence. This problem is trivial in 2D, but potentially hard in 3D.

These controllers have potential insights for real-world systems that are self-propelled and can be steered by the orientation of a global field. The size of micro-scale robots makes it difficult to include onboard computation, so they are often steered by external fields. Examples include steering magnetized single-celled organisms [23]–[26], magnetotactic bacteria [27]–[30] and catalytic Janus particles with magnetic cores [2], [31]–[33].
REFERENCES


