Shaping a Swarm Using Wall Friction and a Shared Control Input

Shiva Shahrokhi and Aaron T. Becker

***Abstract*— Micro- and nano-robots hold promise for targeted drug delivery and micro-scale manufacturing. Due to their small size, large numbers of micro-robots are required to deliver sufficient payloads, but their small size makes it difficult to perform onboard computation or contain a power and propulsion source. Nevertheless, these applications require precision control of the shape and position of the robot swarm.**

I. Introduction

Micro- and nano-robots can be manufactured in large numbers. Large numbers of micro robots are required in order to deliver sufficient payloads, but the small size of these robots makes it difficult to perform onboard computation. Instead, these robots are often controlled by a global, broadcast signal. In our previous work we focused on a block-pushing task, where a swarm of robots pushed a larger block through a 2D maze. One surprising result was that humans that only knew the swarm's mean and covariance completed the task faster that humans who knew the position of every robot [**?**]. Inspired by that work, we proved that we can control the mean position of a swarm and that with an obstacle we can control the swarm's position variance (σ_x and σ_y). We then wrote automatic controllers which could complete a block pushing task, but these controllers had some limitations [**?**]. One of the limitations was that we could only compress our swarm along the world x and y axes, and could not navigate workspaces with narrow corridors with other orientations. One solution to these problems would be a controller that regulates the swarm's position covariance, σ_{xy}. For controlling σ_{xy}, we prove that the swarm position covariance σ_{xy} is controllable given boundaries with non-zero friction. We then prove that two orthogonal boundaries with high friction are sufficient to arbitrarily position a swarm of robots. We conclude by designing controllers that efficiently regulate σ_{xy}.

II. Theory

A. Controlling covariance using friction

Controlling covariance needs the swarm to break symmetry when we are using global inputs. We can use an obstacle like a corner, but for having positive and negative covariances, one corner is not enough and for any covariances we should have a special obstacle which is very hard and make us very dependent to the environment. Instead, walls have friction that is sufficient to break the symmetry. The robots which are touching the wall will have negative friction force as shown in Eq. **??**, causing them to slow down in comparison to the free robots. This is enough for breaking symmetry and enables us to control covariance with any horizontal and vertical wall.

S. Shahrokhi and A. Becker are with the Department of Electrical and Computer Engineering, University of Houston, Houston, TX 77204-4005 USA {sshahrokhi2, atbecker}@uh.edu

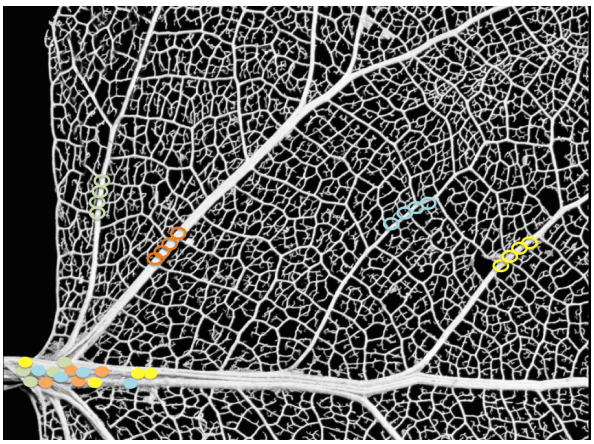

Fig. 1. Vascular networks are common in biology such as the circulatory system and cerebrospinal spaces, as well as in porous media including sponges and pumice stone. Navigating a swarm using global inputs, where each member receives the same control inputs, is challenging due to the many obstacles. This paper demonstrates how friction with walls can be used to change the shape of a swarm.

Let the control input be a vector force $\vec{F}$ with magnitude F and orientation θ. The force of friction F_f is

$$\begin{aligned} N &= F\cos(\theta) \\ F_f &= \begin{cases} \mu_f N, & \mu_f N < F\sin(\theta)) \\ F\sin(\theta)), & \text{else} \end{cases} \\ F_{forward} &= F\sin(\theta) - F_f \end{aligned} \tag{1}$$

Fig. 2 shows forces on two robots when one is touching a wall. As shown, there are different forces for different robots although they are receiving the same input. For ease of analysis, the following algorithms assume μ is infinite and robots touching the wall are prevented from sliding along the wall. This means that if one robot is touching the wall and another robot is free, if the control input is parallel or into the wall, the touching robot will not move. The next section shows how a system with friction model (1) and two walls are sufficient to arbitrarily position two robots.

III. Position control of 2 robots using wall friction

This section focuses on describing an algorithm for positioning two robots in introduces concepts that will be used for multi-robot positioning. Therefore, assume two robots are initialized at s_1 and s_2 with corresponding goal destinations

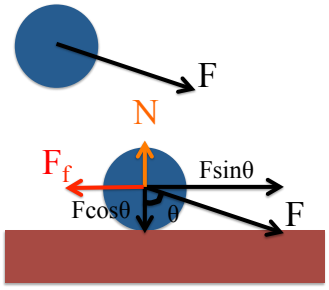

Fig. 2. Wall friction reduces the force for going forward $F_{forward}$ on a robot near a wall, but not for a free robot.

e_1 and e_2. Denote the current positions of the robots r_1 and r_2. Let the subscripts x and y denote the x and y coordinates, i.e., s_{1x} and s_{1y} denote the x and y locations of s_1. The algorithm assigns a global control input at every instance. As a result, our goal is to adjust $\Delta r_x = r_{2x} - r_{1x}$ from $\Delta s_x = s_{2x} - s_{1x}$ to $\Delta e_x = e_{2x} - e_{1x}$ and similarly adjust $\Delta r_y = r_{2y} - r_{1y}$ from $\Delta s_y = s_{2y} - s_{1y}$ to $\Delta e_y = e_{2y} - e_{1y}$ with one global input at every instance. The key to the algorithm is employing the assumption we have made earlier about the walls' friction.

Our algorithm uses a divide and conquer method to solve the positioning problem. It finds the final position of the robots in two steps: (i) First, Δr_x is reduces to zero while Δr_y is kept constant. (ii) Having fixed Δr_x to Δe_x as desired, the algorithm next keeps Δr_x constant and adjusts Δr_y to Δe_y, as desired. Though steps (i) and (ii) are similar from an algorithmic point of view, the following subsections describe the process in detail.

A. Step (i): Fixing Δr_x

- Define $e'_1 = (e_{1x}, s_{1y})$ and $e'_2 = (e_{2x}, s_{2y})$. Our goal for defining e'_1 and e'_2 is to understand the direction to which robots should move in order to adjust Δr_x. Let $e'_{\text{top}} = \arg\max_i e'_{iy}$ and $e'_{\text{bottom}} = \arg\min_i e'_{iy}$. Now if $e'_{\text{top,x}} - e'_{\text{bottom,x}} > 0$, then the global input to both robots would be toward left direction and if $e'_{\text{top,x}} - e'_{\text{bottom,x}} < 0$, then the global input to both robots would be toward right direction. The two robots continue their horizontal path until one of them reaches the ϵ-neighborhood of one of the left or right walls.
- At this step, let $y_{\min} = \min_i r_{iy}$, i.e., $y_{\min}$ is the minimum height of the two robots. We move both robots downward by the amount of $y_{\min}$ such that one of the robots would touch the bottom wall and hence friction force will not let that robot to move left or right.
- The fact that the friction force of the bottom wall would not let the lower robot to move right or left will let the other robot to move to right and left freely to adjust Δr_x according to Δe_x.
- Finally, even if with the free move of the upper robot Δr_x is not set to the Δe_x, we can run the Step (i) (as described in the previous paragraphs) again to adjust the Δr_x. It is easy to show that it is guaranteed that we can adjust Δr_x to Δe_x in only two iterations.

B. Step (ii): Fixing Δr_y

Now that we have adjusted the difference in robots' positions along one axis, we focus to do the same on the other axis as well. Therefore, similar to Section **??**, we employ the following steps:

- Let s'_1 and s'_2 be the points we derived at the end of the steps in Section III-A.
- Define $e''_1 = (s'_{1x}, e_{1y})$ and $e''_2 = (s'_{2x}, e_{2y})$. We define e''_1 and e''_2 to understand the direction to which robots should move in order to adjust Δr_y. Let $e''_{\text{right}} = \arg\max_i e''_{ix}$ and $e''_{\text{left}} = \arg\min_i e'_{ix}$. Now if $e''_{\text{right,y}} - e''_{\text{left,y}} > 0$, then the global input to both robots would be toward down direction and if $e''_{\text{right,y}} - e''_{\text{left,y}} < 0$, then the global input to both robots would be toward up direction. The two robots continue their vertical path until one of them reaches the ϵ-neighborhood of one of the top or bottom walls.
- At this step, let $x_{\min} = \min_i r_{ix}$, i.e., $x_{\min}$ is the minimum distance of the two robots from the origin along the x-axis. We move both robots to the left by the amount of $x_{\min}$ such that one of the robots would touch the left wall and hence friction force will not let that robot to move up or down.
- The fact that the friction force of the left wall would not let one of the robots to move up or down will let the other robot to move to up or down freely to adjust Δr_y according to Δe_y.
- Finally, even if with the free move of the robot which is not touching the wall Δr_y is not set to the Δe_y, we can run the Step (i) (as described in the previous paragraphs) again to adjust the Δr_y. It is easy to show that it is guaranteed that we can adjust Δr_y to Δe_y in only two iterations.

Once Δr_x and Δr_y are set to Δe_x and Δe_y, we can use global input to easily move both robots from r_1 and r_2 toward e_1 and e_2.

Algorithm 1 WallFrictionArrange2Robots(s_1, s_2, e_1, e_2, L)

Require: Knowledge of starting (s_1, s_2) and ending (e_1, e_2) positions of two robots. $(0, 0)$ is bottom corner, s_1 is rightmost robot, L is length of the walls. Current position of the robots are (r_1, r_2).

$\Delta s_x \leftarrow s_{1x} - s_{2x}$
$\Delta s_y \leftarrow s_{1y} - s_{2y}$
$\Delta e_x \leftarrow e_{1x} - e_{2x}$
$\Delta e_y \leftarrow e_{1y} - e_{2y}$
if $\Delta s_x < \Delta s_y$ **then**
    GenerateDesired$x$-spacing(s_1, s_2, e_1, e_2, L)
    GenerateDesired$y$-spacing(s_1, s_2, e_1, e_2, L)
else
    GenerateDesired$y$-spacing(s_1, s_2, e_1, e_2, L)
    GenerateDesired$x$-spacing(s_1, s_2, e_1, e_2, L)
end if

Algorithm 2 GenerateDesired$x$-spacing(s_1, s_2, e_1, e_2, L)

Require: Knowledge of starting (s_1, s_2) and ending (e_1, e_2) positions of two robots. $(0, 0)$ is bottom corner, s_1 is topmost robot, L is length of the walls. Current robot positions are (r_1, r_2).

Ensure: $r_{1y} - r_{2y} \equiv s_{1y} - s_{2y}$

$\epsilon \leftarrow$ small number
$\Delta s_x \leftarrow s_{1x} - s_{2x}$
$\Delta e_x \leftarrow e_{1x} - e_{2x}$
$r_1 \leftarrow s_1$, $r_2 \leftarrow s_2$
if $\Delta e_x < 0$ **then**
    $m \leftarrow (L - \epsilon - \max(r_{1x}, r_{2x}), 0)$ ▷ Move to right wall
else
    $m \leftarrow (\epsilon - \min(r_{1x}, r_{2x}), 0)$ ▷ Move to left wall
end if
$m \leftarrow m + (0, -\min(r_{1y}, r_{2y}))$ ▷ Move to bottom
$r_1 \leftarrow r_1 + m$, $r_2 \leftarrow r_2 + m$ ▷ Apply move
if $\Delta e_x - (r_{1x} - r_{2x}) > 0$ **then**
    $m \leftarrow (\min(\|\Delta e_x - \Delta s_x\|, L - r_{1x}), 0)$ ▷ Move right
else
    $m \leftarrow (-\min(\|\Delta e_x - \Delta s_x\|, r_{1x}), 0)$ ▷ Move left
end if
$m \leftarrow m + (0, \epsilon)$ ▷ Move up
$r_1 \leftarrow r_1 + m$, $r_2 \leftarrow r_2 + m$ ▷ Apply move
$\Delta r_x = r_{1x} - r_{2x}$
if $\Delta r_x \equiv \Delta e_x$ **then**
    $m \leftarrow (e_{1x} - r_{1x}, e_{1y} - r_{1y})$
    $r_1 \leftarrow r_1 + m$, $r_2 \leftarrow r_2 + m$ ▷ Apply move
    return (r_1, r_2)
else
    return GenerateDesired$x$-spacing(r_1, r_2, e_1, e_2, L)
end if

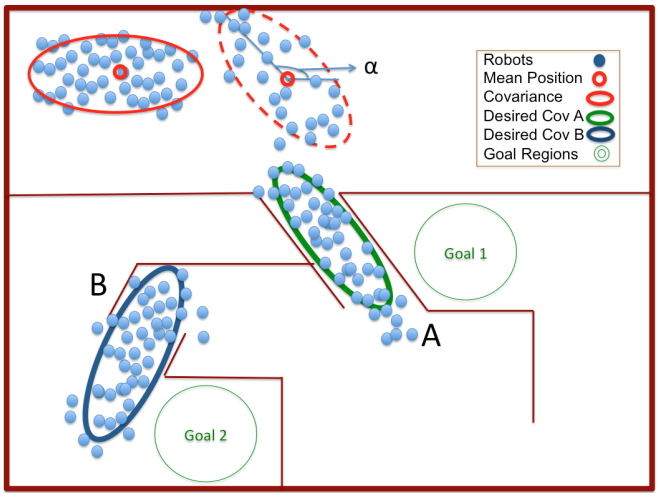

Fig. 3. Maintaining group cohesion while steering a swarm through an arbitrary maze requires covariance control.

Algorithm 3 GenerateDesired$y$-spacing(s_1, s_2, e_1, e_2, L)

Require: Knowledge of starting (s_1, s_2) and ending (e_1, e_2) positions of two robots. $(0, 0)$ is bottom corner, s_1 is rightmost robot, L is length of the walls. Current position of the robots are (r_1, r_2).

Ensure: $r_{1x} - r_{2x} \equiv s_{1x} - s_{2x}$

$\Delta s_y \leftarrow s_{1y} - s_{2y}$
$\Delta e_y \leftarrow e_{1y} - e_{2y}$
$r_1 \leftarrow s_1$, $r_2 \leftarrow s_2$
if $\Delta e_y < 0$ **then**
    $m \leftarrow (L - \max(r_{1y}, r_{2y}), 0)$ ▷ Move to top wall
else
    $m \leftarrow (-\min(r_{1y}, r_{2y}), 0)$ ▷ Move to bottom wall
end if
$m \leftarrow m + (0, -\min(r_{1x}, r_{2x}))$ ▷ Move to left
$r_1 \leftarrow r_1 + m$, $r_2 \leftarrow r_2 + m$ ▷ Apply move
if $\Delta e_y - (r_{1y} - r_{2y}) > 0$ **then**
    $m \leftarrow (\min(\|\Delta e_y - \Delta s_y\|, L - r_{1y}), 0)$ ▷ Move top
else
    $m \leftarrow (-\min(\|\Delta e_y - \Delta s_y\|, r_{1y}), 0)$ ▷ Move bottom
end if
$m \leftarrow m + (0, \epsilon)$ ▷ Move right
$r_1 \leftarrow r_1 + m$, $r_2 \leftarrow r_2 + m$ ▷ Apply move
$\Delta r_y = r_{1y} - r_{2y}$
if $\Delta r_y \equiv \Delta e_y$ **then**
    $m \leftarrow (e_{1x} - r_{1x}, e_{1y} - r_{1y})$
    $r_1 \leftarrow r_1 + m$, $r_2 \leftarrow r_2 + m$ ▷ Apply move
    return (r_1, r_2)
else
    return GenerateDesired$y$-spacing(r_1, r_2, e_1, e_2, L)
end if

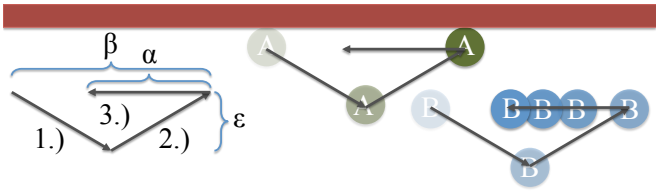

Fig. 4. A DriftMove$(\alpha, \beta, \epsilon)$ consists of repeating a triangular movement sequence $\{(\beta/2, -\epsilon), (\beta/2, \epsilon), (-\alpha, 0)\}$. The robot A touching a top wall will move right β units, while robots not touching the top move right $\beta - \alpha$.

IV. Position Control of n Robots Using Wall Friction

Algorithm 1 can be extended to control the position of n robots using wall friction under several constraints. The solution described here is an iterative procedure with n loops. The kth loop moves the kth robot from staging zone to the desired position in a build zone. At the end the kth loop, robots 1 through k are in their desired final configuration in the build zone, and robots $k+1$ to n are in the staging zone.

Assume an open workspace with four axis-aligned walls with infinite friction.

The axis-aligned build zone of dimension (w_b, h_b) containing the final configuration of n robots must be disjoint

Fig. 5. Illustration of algorithm for position control of n robots using wall friction.

from the axis-aligned staging zone of dimension (w_s, h_s) containing the starting configuration of n robots. Without loss of generality, assume the build zone is above the staging zone . Furthermore, there must be at least ϵ space above the build zone, ϵ below the staging zone , and $\epsilon+2r$ to the left of the build and staging zone, where r is the radius of a robot. The minimum workspace is then $(\epsilon+2r+\max(w_f, w_s), 2\epsilon+h_s, h_f)$.

The n robot position control algorithm relies on a $\mathrm{DriftMove}(\alpha, \beta, \epsilon)$ control input, shown in Fig. 4. A drift move consists of repeating a triangular movement sequence $\{(\beta/2, -\epsilon), (\beta/2, \epsilon), (-\alpha, 0)\}$. The robot A touching a top wall moves right β units, while robots not touching the top move right $\beta - \alpha$.

Let $(0, 0)$ be the lower left corner of the workspace, p_k the x, y position of the kth robot, and f_k the final x, y position of the kth robot. Label the robots in the staging zone from left-to-right and top-to-bottom, and the f_k configurations right-to-left and top-to-bottom as shown in Fig. 5.

Algorithm 4 PositionControl$n$RobotsUsingWallFriction(k)

move($-\epsilon, r - p_{k,y}$)
while $p_{k,x} > r$ **do**
    $\mathrm{DriftMove}(0, \min(p_{k,x} - r, \epsilon), \epsilon)$ left
end while
$m \leftarrow \mathrm{ceil}\ \frac{f_{ky}-r}{\epsilon}$
$\beta \leftarrow \frac{f_{ky}-r}{m}$
$\alpha \leftarrow \beta - \frac{\epsilon}{m}$
for m iterations **do**
    $\mathrm{DriftMove}(\alpha, \beta, \epsilon)$ up
end for
move($r + \epsilon - f_{kx}, 0$)
move($f_{kx} - r, 0$)

Algorithm 4 procedes as follows: First, the robots are moved away from right wall and down so robot k touches bottom. Second, a set of DriftMoves are executed that move the k robot to the left wall with no net movement on the other robots. Third, a set of DriftMoves are executed that move the k robot to it's target height and return the other robots to their initial heights. Fourth, all robots except the k robot are pushed left until the k robot is in the correct relative x position compared to robots 1 to $k-1$. Finally, all robots are moved right until robot k is in the desired target position.

V. Simulation

Algorithms **??**, were implemented in Mathematica using point robots (radius = 0). All code is available at (INSERT). Two figure show...[describe].

VI. Experiment

A. Hardware system

Our experiments are on centimeter-scale hardware systems. It allows us to emulate a variety of dynamics, while enabling a high degree of control over robot function, the environment, and data collection. The kilobot [**?**], [**?**] is a low-cost robot designed for testing collective algorithms with large numbers of robots. It is available commercially or as an open source platform [**?**]. Each robot is approximately 3 cm in diameter, 3 cm tall, and uses two vibration motors to move on a flat surface at speeds up to 1 cm/s. Each robot has one ambient light sensor that is used to implement *phototaxis*, moving towards a light source. In these experiments, we used n=68 kilobots, a 1.2 m×1.5 m whiteboard as the workspace, and four lights arranged at the $\{N, E, S, W\}$ vertices of a 4 m square centered on the workspace. The lights were controlled automatically by using an Arduino Uno board connected to an 8 relay shield board. Also at top of the table, an overhead machine vision system to track the position of the swarm was added.

Experiment: image of testing environment: show table, 101 robots, camera, and 2 lights.

Time-lapse collage of robots changing covariance

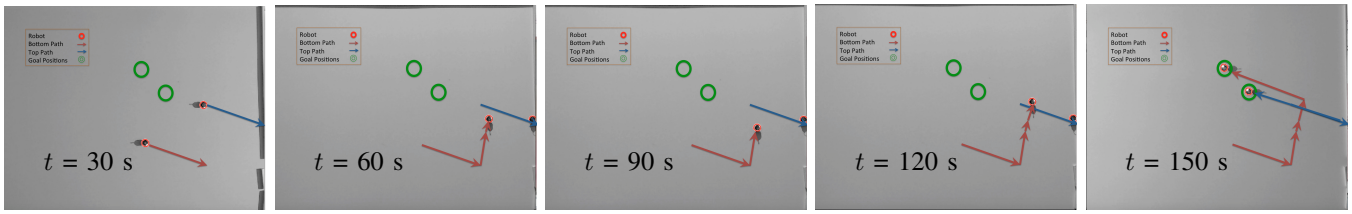

Fig. 6. Two robot positioning with using infinite friction for walls

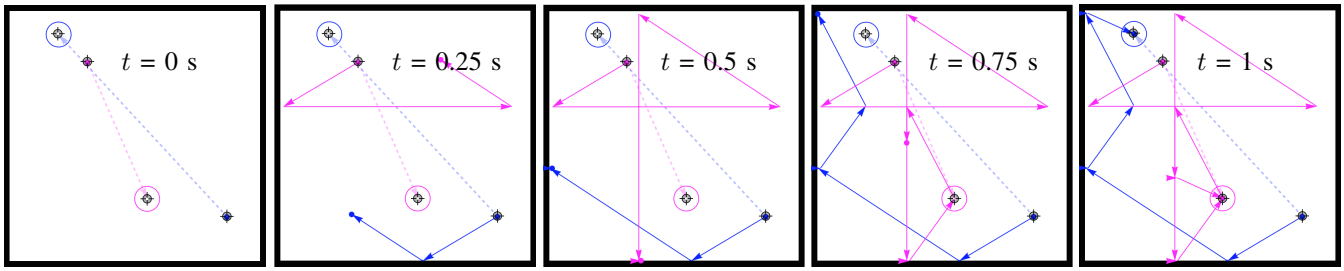

Fig. 7. Two robot positioning with using infinite friction for walls

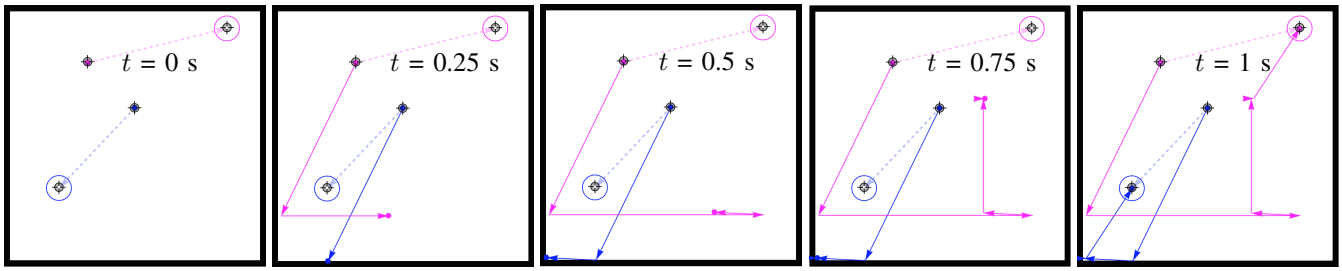

Fig. 8. Two robot positioning with using infinite friction for walls: switching positions. Code available at github address