Algorithms For Shaping a Particle Swarm With a Shared Control Input Using Boundary Interaction*

Shiva Shahrokhi, Arun Mahadev, and Aaron T. Becker

***Abstract*— There are driving applications for large populations of tiny robots in robotics, biology, and chemistry. These robots often lack onboard computation, actuation, and communication. Instead, these "robots" are particles carrying some payload and the particle swarm is controlled by a shared, global control input. In previous work, we showed that the 2D position of each particle in such a swarm is controllable if the workspace contains a single obstacle the size of one particle.**

Requiring a small, rigid obstacle suspended in the middle of the workspace is a strong constraint, especially in 3D. This paper relaxes this constraint, and provides position control algorithms that only require interactions with the boundaries. Both in vivo and artificial environments often have boundaries. We assume that particles in contact with the boundaries have zero velocity if the global control input pushes the particle into the wall. This paper provides an optimal shortest-path algorithm for a swarm with two particles, and a generalization to n-particle swarms. Results are validated with simulations and a hardware demonstration.

I. Introduction

Particle swarms propelled by a global field are common in applied mathematics, biology, and computer graphics. As a current example, micro- and nano-robots can be manufactured in large numbers, see [**? ? ? ? ? ? ?**]. Someday large swarms of robots will be remotely guided to assemble structures in parallel and through the human body to cure disease, heal tissue, and prevent infection. For each task, large numbers of micro robots are required 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. They require techniques to shape the swarm that can reliably exploit large populations despite severe underactuation ($2 \ll 2n$).

Swarm shape control is necessary for navigation through narrow passages, for self-assembly, and for manipulating objects.

The paper is arranged as follows. After a review of recent related work in Sec. II, Sec. III-A introduces general models for boundary interaction. We introduce algorithms with two orthogonal boundaries with high friction sufficient to arbitrarily position two robots in Sec. IV-A, and Sec. IV-B extends this to prove a rectangular workspace with high-friction boundaries can position a swarm of n robots arbitrarily within a subset of the workspace. Sec. V describes implementations of the algorithms in simulation and Sec. VI

*This work was supported by the National Science Foundation under Grant No. [IIS-1553063] and [IIS-1619278].

Authors are with the Department of Electrical and Computer Engineering, University of Houston, Houston, TX 77204 USA {sshahrokhi2, aviswanathanmahadev, atbecker}@uh.edu

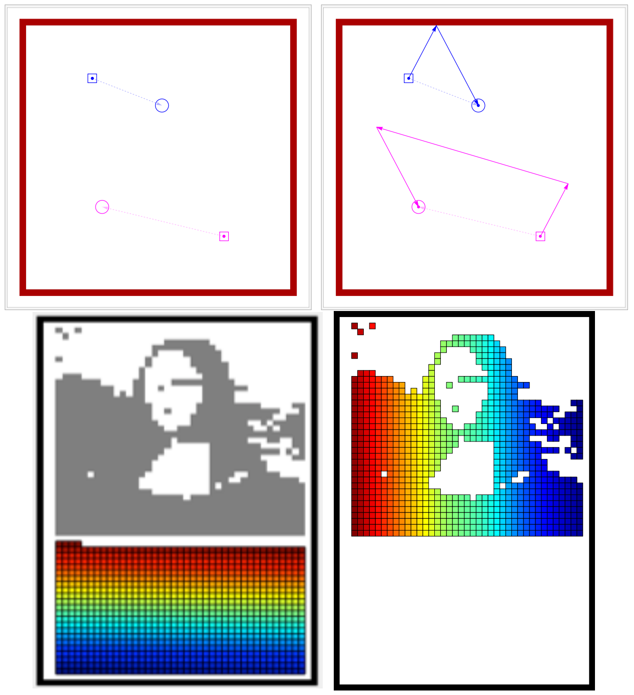

Fig. 1. All the robots get shared control input. Here in the left picture the wall has infinite friction. Therefore, when the first robot is touching the wall, the second robot can move to anywhere in the shaded area. We introduce a dynamic algorithm to find the shortest path the robots can move to get to their goal positions. Consider the reflection of the goal position to the wall that is aimed to be touched by one of the robots. In the right picture, it is shown that the shortest path with three moves is the straight line between the starting position of the robot and the reflection of the goal.

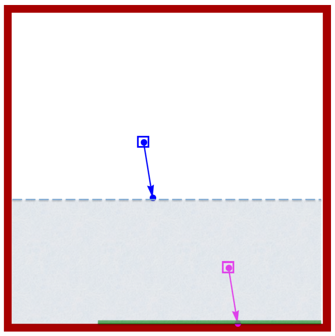

Fig. 2. All the robots get shared control input. Here in the left picture the wall has infinite friction. Therefore, when the first robot is touching the wall, the second robot can move to anywhere in the shaded area. We introduce a dynamic algorithm to find the shortest path the robots can move to get to their goal positions. Consider the reflection of the goal position to the wall that is aimed to be touched by one of the robots. In the right picture, it is shown that the shortest path with three moves is the straight line between the starting position of the robot and the reflection of the goal.

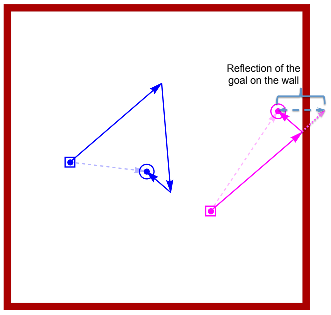

Fig. 3. All the robots get shared control input. Here in the left picture the wall has infinite friction. Therefore, when the first robot is touching the wall, the second robot can move to anywhere in the shaded area. We introduce a dynamic algorithm to find the shortest path the robots can move to get to their goal positions. Consider the reflection of the goal position to the wall that is aimed to be touched by one of the robots. In the right picture, it is shown that the shortest path with three moves is the straight line between the starting position of the robot and the reflection of the goal.

describes hardware experiments, as shown in Fig. 1. We end with directions for future research in Sec. VII.

II. Related Work

Controlling the *shape*, or relative positions, of a swarm of robots is a key ability for a range of applications. Correspondingly, it has been studied from a control-theoretic perspective in both centralized and decentralized approaches. For examples of each, see the centralized virtual leaders in [**?**], and the gradient-based decentralized controllers using control-Lyapunov functions in [**?**]. However, these approaches assume a level of intelligence and autonomy in individual robots that exceeds the capabilities of many systems, including current micro- and nano-robots. Current micro- and nano-robots, such as those in [**? ? ?**] lack onboard computation.

Instead, this paper focuses on centralized techniques that apply the same control input to each member of the swarm. Precision control requires breaking the symmetry caused by the global input. Symmetry can be broken using agents that respond differently to the global control, either through agent-agent reactions, see work modeling biological swarms [**?**], or engineered inhomogeneity [**? ? ?**]. This work assumes a uniform control (**??**) with homogenous agents, as in [**?**]. The techniques in this paper are inspired by fluid-flow techniques and artificial force-fields.

Fluid-flow: Fluid flow along boundaries generates a shear force that pushes different parts of a body in opposing directions. Most introductory fluid dynamics textbooks provide models [**?**]. Similarly, a swarm of robots under global control pushed along a boundary will experience shear forces. This is a position-dependent force, and so can be exploited to control the configuration or shape of the swarm. [**?**] used these forces to disperse a swarm's spatial position for coverage for physics-based swarm simulations.

Artificial Force-fields: Much research has focused on generating non-uniform artificial force-fields that can be used to rearrange passive components. Applications have included techniques to design shear forces for sensorless manipulation of a single object by [**?**]. [**? ?**] demonstrated a collection of 2D force fields generated by 6DOF vibration inputs to a rigid plate. These force fields, including shear forces, could be used as a set of primitives for motion control to steer the formation of multiple objects. However unlike the uniform control model in this paper, theirs was multi-modal and position-dependent.

III. Theory

A. Using Boundaries: Friction and Boundary Layers

Global inputs move a swarm uniformly. Shape control requires breaking this uniform symmetry. A swarm inside an axis-aligned rectangular workspace can reduce variance normal to a wall by simply pushing the swarm into the boundary. If the swarm can flow around each other, pushing the swarm into a boundary produces the limited set of configurations presented in Sec. **??**. Instead of pushing our robots directly into a wall, the following sections examine an oblique approach using boundaries that generate friction with the robots. These frictional forces are sufficient to break the symmetry caused by uniform inputs. Robots touching a wall have a friction force that opposes movement along the boundary. This causes robots along the boundary to move more slowly than robots in free-space.

Let the control input be a vector force $\vec{F}$ with magnitude F and orientation θ with respect to a line perpendicular to and into the nearest boundary. N is the normal or perpendicular force between the robot and the boundary. The force of friction F_f is nonzero if the robot is in contact with the boundary and $\sin(\theta) < 0$. The resulting net force on the robot, $F_{forward}$, is aligned with the wall and given by

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

Fig. 4 shows the resultant forces on two robots when one is touching a wall. Though each receives the same inputs, they experience different net forces. For ease of analysis, the following algorithms assume μ_f 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, the touching robot will not move when the control input is parallel or into the wall. There are many alternate models of friction that also break control symmetry. Fig. 4c shows fluid flow along a boundary. Fluid in the free-flow region moves uniformly, but flow decreases to zero in the boundary layer [**?**].

$$F_{forward}(y) = F - F_f \begin{cases} \frac{h-y}{h}, & y < h \\ 0, & \text{else} \end{cases} \quad (2)$$

The next section shows how a system in a rectangularly bounded workspace with friction model (1) can arbitrarily position two robots.

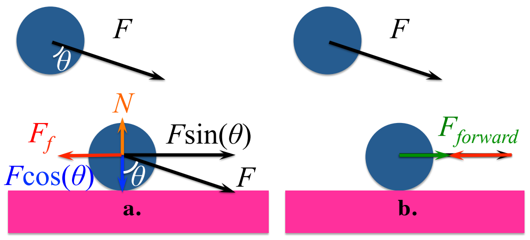

Fig. 4. (a,b) Wall friction reduces the force for going forward $F_{forward}$ on a robot near a wall, but not for a free robot. (c) velocity of a fluid reduces to zero at the boundary.

Algorithm 1 PoseControl2Robots(s_1, s_2, g_1, g_2, L)

```
Require: knowledge of starting $(s_1, s_2)$ and goal $(g_1, g_2)$ positions of two robots.
    $(0,0)$ is bottom corner, $L$ is length of the walls. Current robot positions are
    $(r_1, r_2)$. Assume $s_1.x < s_2.x$ and $s_1.y \geq s_2.y$. If not, rotate workspace
    coordinates counterclockwise $90°$. $\epsilon$ is a small, nonzero, user-specified value.
Ensure: $(g_1, g_2), (s_1, s_2)$ all at least $\epsilon$ distance from walls.
 1: $\Delta g_y = e_2.y - e_1.y$
 2: $\Delta g_x = e_2.x - e_1.x$
 3: Move $(-r_1.x, -\min(r_2.y, r_1.y + \Delta e_y) + \epsilon)$    ▷ Touch left wall
 4: Move $(r_2.x, |\text{sgn}(\Delta e_y)| \min(r_1.y - r_2.y + \Delta e_y - \epsilon, L - r_2.y - \epsilon) + \epsilon)$
                                                                       ▷ Adjust $y$
 5: Move $(\max(\epsilon, -\text{sgn}(r_2.y - r_1.y)\Delta e_x), -\min(r_1.y, r_2.y))$
                                                                       ▷ Touch bottom wall
 6: Move $(\text{sgn}(r_2.y - r_1.y)\Delta e_x, (|\text{sgn}(\Delta e_y)| - 1)|r_2.y - r_1.y|)$ ▷ Adjust $x$
 7: if $\Delta e_y \neq r_2.y - r_1.y$ then
 8:     if $\Delta e_x \neq 0$ then
 9:         Move $(\text{sgn}(r_2.y - r_1.y)\Delta e_x, |\text{sgn}(\Delta e_y) - 1||r_1.y - r_2.y|)$ ▷
    Touch right wall
10:     else
11:         Move $(|\text{sgn}(\Delta e_y - r_2.y + r_1.y)|\epsilon, |\text{sgn}(\Delta e_y) - 1||r_1.y - r_2.y|)$
    ▷ Touch right wall
12:     end if
13:     if $\Delta e_x < 0$ then
14:         Move $(L - \max(r_1.x, r_2.x), \epsilon)$
15:     else
16:         Move $(L - \max(r_1.x, r_2.x), \Delta e_y - \max(r_1.y, r_2.y)$
                                                                       ▷ Adjust $y$
17:     end if
18: end if
19: Move $(e_2.x - r_2.x, e_2.y - r_2.y)$                              ▷ Go to goal
```

IV. Algorithms

A. Position Control of Two Robots Using Wall Friction

Alg. 1 uses wall-friction to arbitrarily position two robots in a rectangular workspace. This algorithm introduces concepts that will be used for multi-robot positioning. Fig. 6 shows a Mathematica implementation of the algorithm, and is useful as a visual reference for the following description.

Assume two robots are initialized at s_1 and s_2 with corresponding goal destinations g_1 and g_2. We can exploit symmetry in the solution by labeling the leftmost (or, if they have the same x coordinate, the topmost) robot s_1. If s_1 is not also the topmost robot, we rotate the coordinate frame by $90°$ counterclockwise. Denote the current positions of the robots r_1 and r_2. Values $.x$ and $.y$ denote the x and y coordinates, i.e., $s_1.x$ and $s_1.y$ denote the x and y locations of s_1. Define the sign function as:

$$\text{sgn}(x) = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases} \quad (3)$$

The algorithm assigns a global control input at every instance. The goal is to adjust $\Delta r_x = r_2.x - r_1.x$ from $\Delta s_x = s_2.x - s_1.x$ to $\Delta e_x = e_2.x - e_1.x$ and adjust $\Delta r_y = r_2.y - r_1.y$ from $\Delta s_y = s_2.y - s_1.y$ to $\Delta e_y = e_2.y - e_1.y$ using a shared global control input.

Our algorithm solves the positioning problem in four steps: First, it adjusts $\Delta r_y, \Delta r_x$ as much as possible with the left wall. Second, $\Delta r_x - \Delta e_x$ is reduced to zero with the bottom wall. Third, if the robots were not correctly positioned relative to each other, $\Delta r_y - \Delta e_y$ is reduced to zero with the right wall. Lastly, the robots, now correctly positioned relative to each other, are moved to their goal locations.

In the worse case, adjusting both Δr_x and Δr_y needs all four steps. The worst case path length is $2(\sqrt{2}+1)L$.

B. Position Control of n Robots Using Wall Friction

Alg. 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 a *staging zone* to the desired position in a *build zone*. All robots move according to the global input, but due to wall friction, 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. See Fig. 8 for a schematic of the build and staging zones.

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 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 $(\epsilon + 2r + \max(w_b, w_s), 2\epsilon + h_s, h_b)$.

The n robot position control algorithm relies on a DriftMove$(\alpha, \beta, \epsilon)$ control input, shown in Fig. 11. A drift move consists of repeating a triangular movement sequence $\{(\beta/2, -\epsilon), (\beta/2, \epsilon), (-\alpha, 0)\}$. The robot 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 bottom-to-top, and the f_k configurations top-to-bottom and right-to-left as shown in Fig. 8.

Alg. 2 proceeds as follows: First, the robots are moved left away from the right wall, and down so robot k touches the bottom wall. Second, a set of DriftMoves are executed that move robot k to the left wall with no net movement of the other robots. Third, a set of DriftMoves are executed that move robot k to its target height and return the other robots to their initial heights. Fourth, all robots except robot k are pushed left until robot k 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. Running

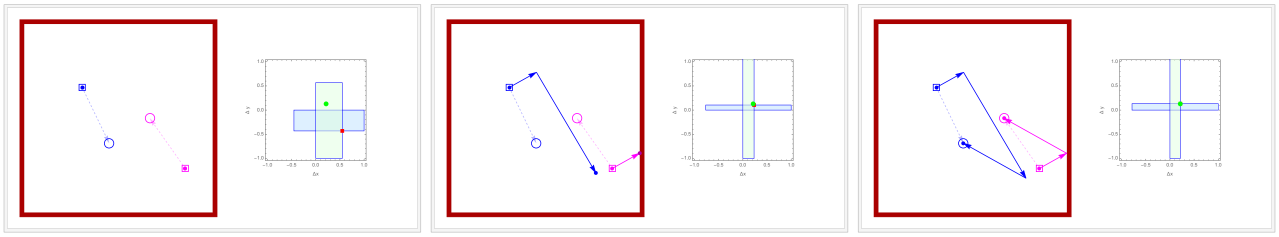

Fig. 5. Two rectangles representing the reachable regions for the current start and goal positions. The red square represents the starting Δx and Δy and the green circle represents the goal Δx and Δy. The green rectangle illustrates one move reachable Δx and Δy by horizontal walls and the blue rectangle illustrates the vertical walls reachable region.

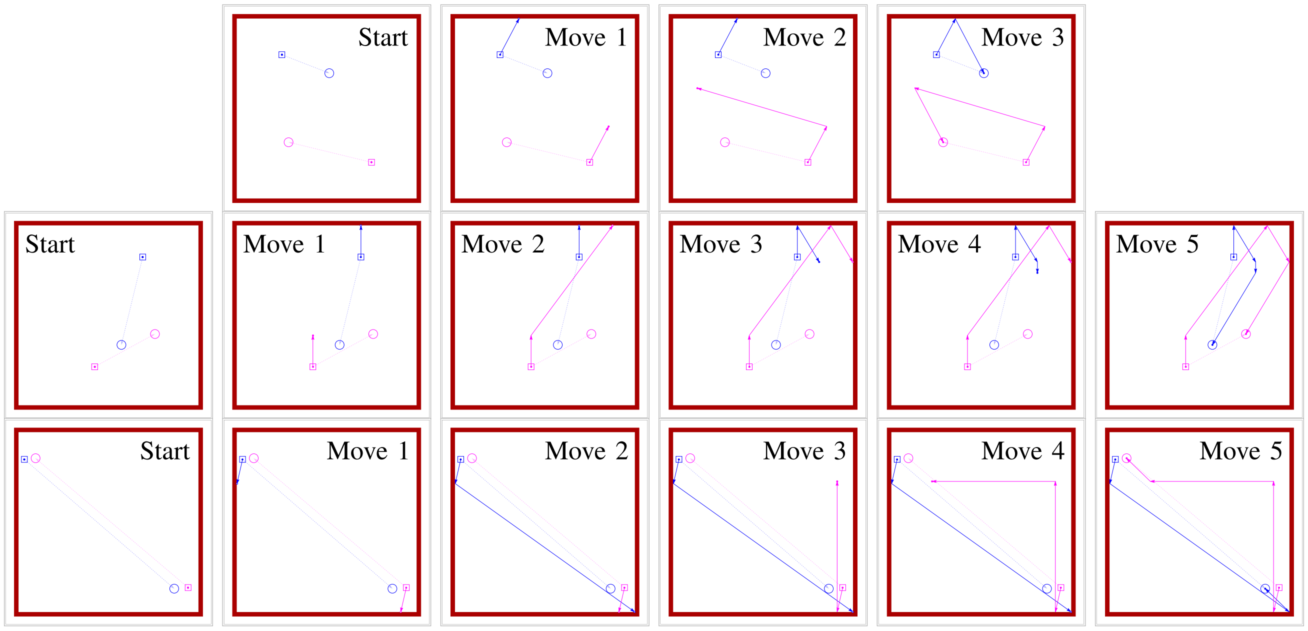

Fig. 6. Frames from an implementation of Alg. 1: two robot positioning using walls with infinite friction. Robot start positions are shown by a square, and goal positions by a circle. Dashed lines show the shortest route if robots could be controlled independently. Solid arrows show path given by Alg. 1. Online demonstration and source code at [?].

Algorithm 2 PositionControl$n$RobotsUsingWallFriction(k)

```
1:  Move( $-\epsilon, r - p_{ky}$)
2:  while  $p_{kx} > r$ do
3:      DriftMove($\epsilon, \min(p_{kx} - r, \epsilon), \epsilon, left$)
4:  end while
5:  $m \leftarrow$ ceil($\frac{f_{ky}-r}{\epsilon}$)
6:  $\beta \leftarrow \frac{f_{ky}-r}{m}$
7:  $\alpha \leftarrow \beta - \frac{r-p_{ky}-\epsilon}{m}$
8:  for  $m$ iterations do
9:      DriftMove($\alpha, \beta, \epsilon, up$)
10: end for
11: Move ($r + \epsilon - f_{kx}, 0$)
12: Move ($f_{kx} - r, 0$)
```

Algorithm 3 DRIFTMOVE($\alpha, \beta, \epsilon, direction$)

```
1: Move( $-\epsilon, r - p_{ky}$)
2: Move ($r + \epsilon - f_{kx}, 0$)
3: Move ($f_{kx} - r, 0$)
```

time is $O(n(w + h))$.

V. SIMULATION

Two simulations were implemented using wall-friction for position control. The first controls the position of two robots, the second controls the position of n robots.

Two additional simulations were performed using wall-friction to control variance and covariance. The first is an open-loop algorithm that demonstrates the effect of varying friction levels. The second uses a closed-loop controller to achieve desired variance and covariance values.

A. *Position Control of Two Robots*

Algorithm 1 was implemented in Mathematica using point robots (radius = 0). An online interactive demonstration and source code of the algorithm are available at [?]. Fig. 6 shows an implementation of this algorithm wiht robot initial positions shown by hollow squares and final positions by circles. Dashed lines show the shortest route if robots could be controlled independently, solid lines the generated path.

B. *Position Control of n Robots*

Alg. 2 was simulated in MATLAB using square block robots with unity width. Code is available at [?]. Simulation

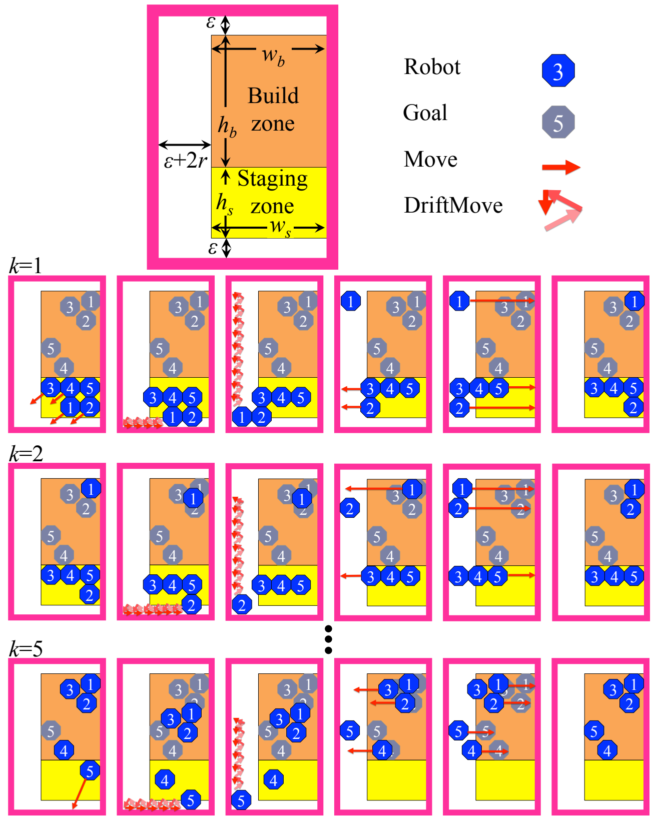

Fig. 7. Illustration of Alg. 2, n robot position control using wall friction.

results are shown in Fig. 12 for arrangements with an increasing number of robots, n= [8, 46, 130, 390, 862]. The distance moved grows quadratically with the number of robots n. A best-fit line $210n^2 + 1200n - 10,000$ is overlaid by the data.

In Fig. 12, the amount of clearance is $\epsilon = 1$. Control performance is sensitive to the desired clearance. As ϵ increases, the total distance decreases asymptotically, as shown in Fig. 13, because the robots have more room to maneuver and fewer DriftMoves are required.

VI. Experiment

A. Hardware Experiment: Position Control of n Robots

Kilobots have too much stochasticity to implement Alg. 2. Instead, a hardware setup with a bounded platform, magnetic sliders, and a magnetic guide board was used. Designs for each are available at [?]. The pink boundary is toothed with a white free space, as shown in Fig 8. Only discrete, 1 cm moves in the x and y directions are used. The goal configuration highlighted in the top right corner represents a ‘U’ made of seven sliders. The dark red configuration is the current position of the sliders. Due to the discretized movements allowed by the boundary, drift moves follow a 1 cm square. Free robots return to their start positions but robots on the boundary to move laterally, generating a net sliding motion.

Fig. 8 follows the motion of the sliders through iterations k=1, 2, and 7. All robots receive the same control inputs, but

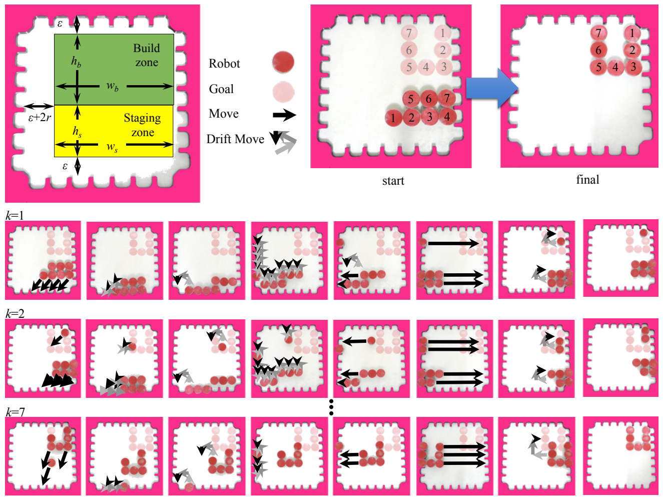

Fig. 8. Illustration of Alg. 2, n robot position control using wall friction.

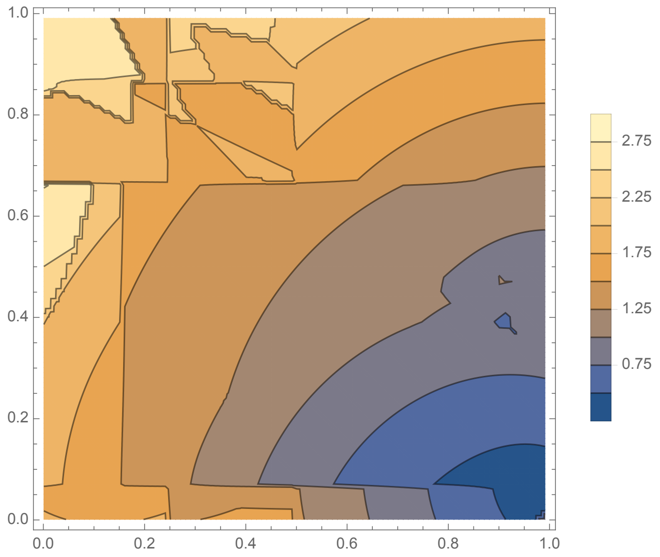

Fig. 9. The contour plot keeps 3 points steady.

boundary interactions break the control symmetry. Robots reach their goal positions in a first-in, first-out arrangement beginning with the bottom-left robot from the staging zone occupying the top-right position of the build zone.

VII. Conclusion and Future Work

This paper presented techniques for controlling the shape of a swarm of robots using uniform global inputs and interaction with boundary friction forces. The paper provided algorithms for precise position control, as well as robust and efficient covariance control. Extending algorithms 1 and 2 to 3D is straightforward but increases the complexity. Future efforts should be directed toward improving the technology and tailoring it to specific robot applications.

With regard to technological advances, this includes designing controllers that efficiently regulate σ_{xy}, perhaps using Lyapunov-inspired controllers as in [?]. Additionally, this paper assumed nearly infinite wall friction. The

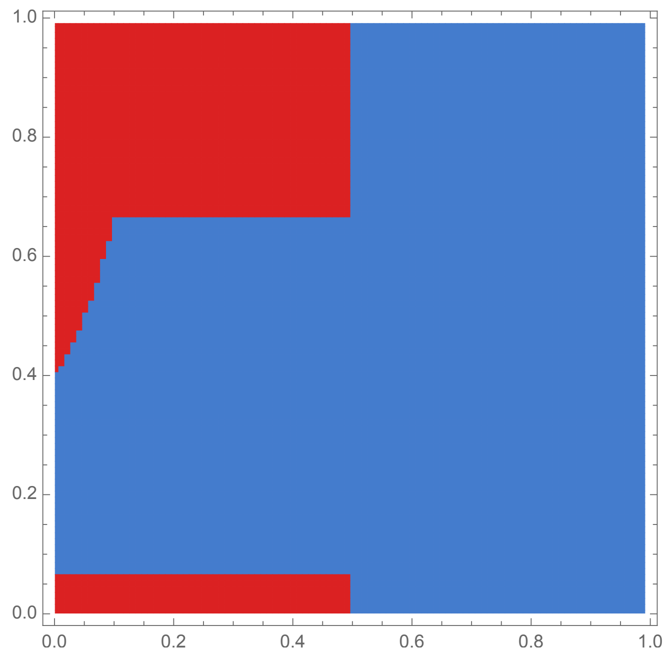

Fig. 10. Number of moves needed.

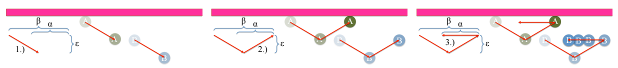

Fig. 11. A DriftMove$(\alpha, \beta, \epsilon)$ to the right repeats a triangular movement sequence $\{(\beta/2, -\epsilon), (\beta/2, \epsilon), (-\alpha, 0)\}$. Robot A touching a top wall moves right β units, while robots not touching the top move right $\beta - \alpha$.

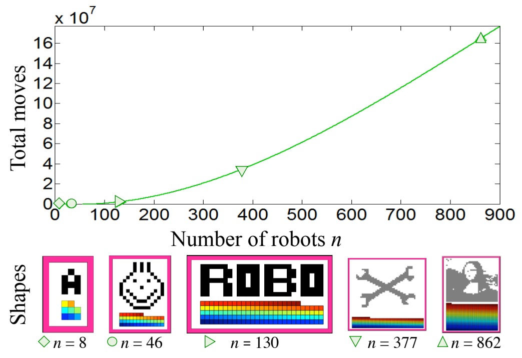

Fig. 12. The required number of moves under Alg. 2 using wall-friction to rearrange n square-shaped robots. See hardware implementation and simulation at [**?**].

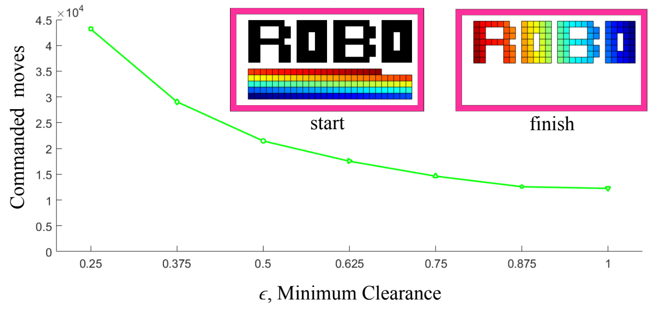

Fig. 13. Control performance is sensitive to the desired clearance ϵ. As ϵ increases, the total distance decreases asymptotically.

algorithms require retooling to handle small μ_f friction coefficients.