Diffusion-Based Control of Robotic Swarms

Brendon Anderson
Mechanical and Aerospace Engineering
University of California, Los Angeles
Los Angeles, California 90095
Email: bganderson@ucla.edu

Swagata Biswas
Department of Mathematics
University of California, Los Angeles
Los Angeles, California 90095
Email: swagata_biswas@yahoo.com

Marissa Gee
Department of Mathematics
Harvey Mudd College
Claremont, California 91711
Email: mgee@hmc.edu

Eva Loeser
Department of Mathematics
Brown University
Providence, Rhode Island 02912
Email: eva_loeser@brown.edu

Fei Ren
Department of Mathematics
University of California, Los Angeles
Los Angeles, California 90095
Email: feiren1995@g.ucla.edu

Ge Zhang
Control Science and Engineering
Harbin Institute of Technology
Harbin, Heilongjiang, China 150001
Email: zhangge19951114@gmail.com

***Abstract*—In this report, we analyze a decentralized stochastic control law used to transport a robotic swarm toward a desired distribution. We first review relevant background information, including the error metric used to evaluate the swarm's performance. We then develop a definition of "steady state" for this stochastic system using an exponential decay model. Next, we compute the optimal configuration of robots with respect to the desired distribution to develop a reference point for our error metric. We further study the error metric, proving that for randomly chosen robot configurations drawn from the desired distribution, its probability density function converges to a normal distribution. We show that for robots distributed according to the desired distribution, the error metric approaches zero as the number of robots goes to infinity and their size goes to zero. Furthermore, we show that if the desired distribution is uniform, the control law preserves and tends towards uniformity. By using a simple "bounceback" boundary behavior, convergence toward the desired distribution is achieved, aside from slight warping in corner regions of the domain. Finally, we explore the use of a deterministic control law, and its accuracy in approximating a macroscopic diffusion model is demonstrated using a one-dimensional simulation.**

I. Introduction

Swarm robotics is the study of large groups of robots that cooperate to accomplish a goal that a single robot could not. To be manufactured on a large scale, it is preferable that these robots are relatively simple and inexpensive. However, this limits their computational power and communication capabilities, and can make it infeasible to issue individualized instructions to each agent. With these restrictions in mind, this report explores a stochastic swarm control law that provides simple rules of motion for each agent, and does not rely on communication or localization. Instead, the proposed control law relies only on scalar measurements of each robot's local environment to achieve the desired group behavior. Such a control law has applications in commercial pollination, surveillance, and search and rescue operations.

The primary theoretical result underpinning this project is presented in [3]. In this paper, the authors present a stochastic control law for a robotic swarm, which ensures that the swarm converges to a distribution proportional to the scalar field being measured. The scalar field can be any property of the task environment that can be measured and understood in terms of a scalar value. In [3], the authors model the control law with both a stochastic differential equation and a corresponding partial differential equation. It shows that the solution to the PDE, and thus the corresponding stochastic process, converges to the scalar field for their control law.

Motivated by these theoretical results, last summer's swarm robotics team created simulations and ran physical experiments that implement this control law [1], [8]. In their results, they demonstrate that the swarm converges in simulation and in experiment to the desired distribution. Additionally, the team explores different error metrics and how well they characterize how far a swarm is from the desired distribution. Finally, they investigate the rate at which the swarm converges, and show that in experiment the rate of convergence appears to grow linearly with $\frac{1}{\sqrt{N}}$, where N is the number of robots in our swarm.

This summer our team has worked to answer lingering questions from the previous summer's research, and has looked into methods that extend or modify the control law. The report is organized into the following sections, based on the topics we have studied this summer. First, Section II outlines useful definitions for our report. Section III seeks to understand how the error evolves in time, and to quantify when the swarm has reached its steady state distribution. Section IV investigates numerical methods for computing the optimal error value, and how this value depends on properties of our swarm. In Section V, we further contextualize the error metric by characterizing its distribution in steady state. Section VI examines different methods of controlling the robots at the boundary of our domain, how these methods affect the steady state swarm distribution, and how we can account for boundary collisions for a circular domain, which could also represent collisions with obstacles. Section VII explores how the distribution of robots evolves over a single time step, and how this evolution is affected by a robot's proximity to the boundary. Next, in Section VIII we consider applications of the control to three additional scenarios: the case that collisions between robots are significant, the case of a three-dimensional scalar field, and the case of a time-dependent scalar field. Section IX summarizes the continued work in running physical experiments for our control law. Finally, in Section X we examine a new method for solving PDEs of the same type as our model and discuss the possible extension of this method to the development of a deterministic control law for a robotic swarm.

II. Preliminaries

In this paper, our robots are moving in a finite rectangular domain $\Omega \in \mathbb{R}^2$. On this domain, we define a scalar field, F. Our control law is designed with the goal that the robot positions converge to a distribution that is proportional to this scalar field. In our physical experiments the scalar field is represented by the color of the floor underneath the robot. The black represents a high value of the scalar field. The white represents a low value of the scalar field. The red represents the boundary. There are two non-uniform scalar fields used in this analysis. The first is a ring scalar field, in which the scalar field is 36 on an annulus and 1 outside the annulus. The second is a row scalar field, in which there are three vertical bars in the domain in which the scalar field has a value of 36, and 4 vertical bars in the domain in which the scalar field has a value of 1.

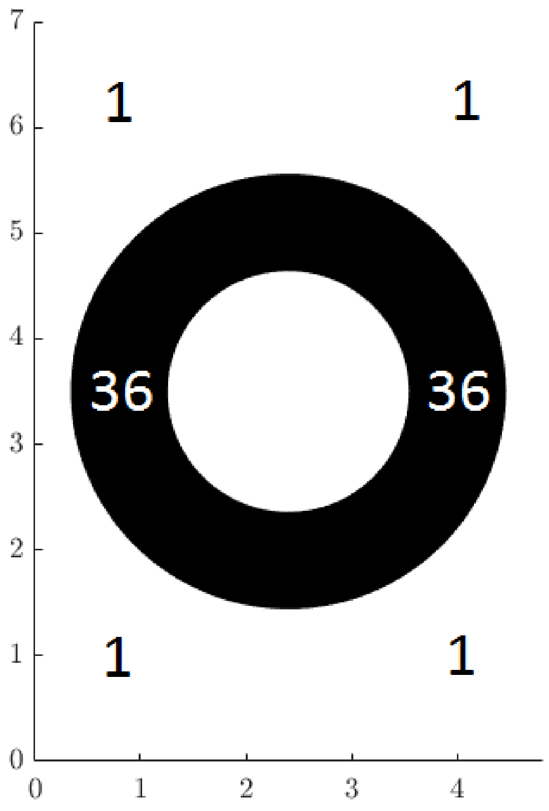
Our robots are moving according to a control law in which each robot follows a random walk with speed inversely correlated to the square root of the scalar field at the robot's position. To be precise, if a robot is at position $\mathbf{x}(t)$, its next move will be governed by

$$d\mathbf{x}(t) = D(\mathbf{x}(t))dW + d\Psi(t),$$

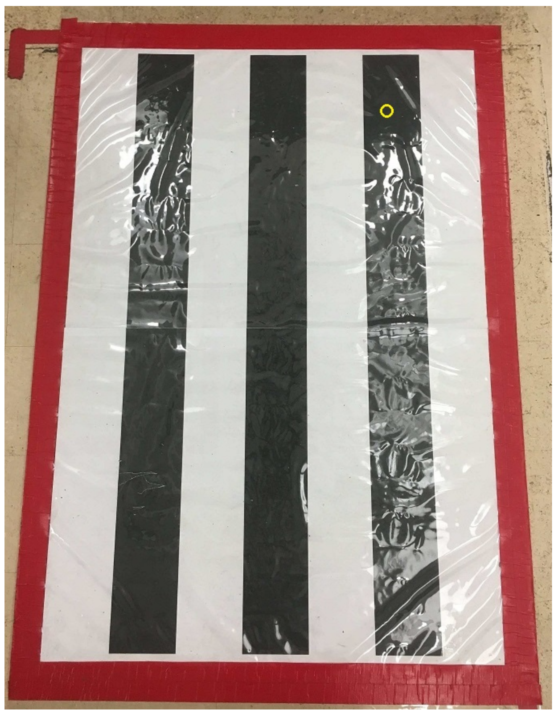
where

$$D(\mathbf{x}(t)) = \frac{1}{\sqrt{F(\mathbf{x}(t))}},$$

and dW represents the standard Weiner process. The function, $\Psi(t)$, cannot be explicitly defined that reverses robot velocity when the robot hits the boundary. Fur-



(a)



(b)

Fig. 1: (a) The scalar field with the ring pattern used in simulations. (b) The scalar field with the row pattern used in both simulations and physical experiments. The yellow circle on the top right marks the initial position for the robot.

thermore, as stated in [3], this stochastic model can be understood via the following partial differential equation:

$$\begin{aligned} \frac{\partial u}{\partial t} - \Delta(D(\mathbf{x})^2 u) = 0 \qquad & \text{in } \Omega \times [0,T], \\ \mathbf{n} \cdot \nabla(D(\mathbf{x})^2 u) = 0 \qquad & \text{on } \partial\Omega \times [0,T], \\ u(\mathbf{x},0) = u_0(\mathbf{x}) \qquad & \text{in } \Omega. \end{aligned}$$

We think of each robot as patrolling a space of radius δ around its position, and represent it by the so-called Gaussian blob function

$$G(x_1, x_2) = \frac{1}{2\pi\delta^2} e^{-\frac{{x_1}^2 + {x_2}^2}{2\delta^2}}.$$

Where $\mathbf{x} = (x_1, x_2)$. Namely, if the robots have a configuration $X = (X_1, ..., X_N)$ we define the blob function to be

$$G(\mathbf{X}) = \frac{1}{N} \sum_{i=1}^{N} G\left(\mathbf{x} - X_i\right)$$

We define the error, $e_N^\delta(X)$, of a configuration $X = (X_1, ..., X_N)$ of N robots that we think of as patrolling a space of radius δ around their position as

$$e_N^\delta(X) = \int_\Omega \left| \frac{1}{N} \sum_{i=1}^{N} G\left(\mathbf{x} - X_i\right) - F \right| d\mathbf{x}$$

We often set this function to 0 outside a ball of radius r, where r is positive real number of our choosing.

III. Steady State Measurement

A. Motivation

Since we are interested in studying the average error metric in steady state and determining how long the physical experiments should be run, we need a way to measure when steady state of a system of the robots with a given scalar field starts. Conceptually, we define the steady state of the simulations and experiments as the state of the robots in which the robots finish spreading out on the scalar field. At the steady state, the robots' distribution is similar to the target scalar field even though they are still constantly moving. Even though we can gauge how well the robots spread out on the domain in the case of uniform scalar field with naked eyes, it is very hard for us to tell whether the steady state is reached in case of non-uniform scalar fields, especially the ones we use which are not continuous, from pure observation. The reason is that we cannot compare the ratio of the density of the robots in the flower region to density of robots outside the flower region with the ratio prescribed by the scalar field in real time while the robots are moving. Therefore, we need a more quantitative measurement for steady state for arbitrary scalar fields. Since the error metric measures how well the robots spread out according to the target scalar fields, which matches with our conception of differences between transient and steady states, we study the time convergence of the error metric to measure steady state.

B. Methods

We want to define the steady state to be the state in which certain properties of the error metric of robots with respect to the target scalar field are constant. In this case, one interpretation would be the steady state starts when $\frac{de}{dt} = 0$ where e is the error metric. Since the calculation of derivative is a set of floating number operations, the number can never reach 0 exactly. Instead, we use 10^{-4} as a threshold for 0. Namely, if the derivative is smaller than this threshold, then it is considered as $\frac{de}{dt} = 0$. However, like Figure 2 shows, the error metric signal always has random noise. Thus, we need to smooth the error metric signal first.

We have tried two signal processing methods to smooth the error metric. The first method is wrapping the signal with two envelope functions by applying the `envelope` function in MATLAB. The two envelope functions, high and low, are specified by the peak separation parameter (np). We use $np = 31$ which is determined by observation. After we obtain the two envelope functions, we can calculate the mean line of the two functions and use this new function, which is much more smoother than the original error metric curve, to approximate the error metric. If we check the derivative of the mean line with the threshold, we can use the first data point which has a derivative less than the threshold as the starting point for steady state. However, the simulation results show that the starting points indicated by this method are often unstable stationary points which come before the starting points of the real steady state measured with the underlying exponential decay model of the noisy error metric curve (Figure 3).

The other method to smooth the error metric curve is taking the cumulative average of all of existing error metric values. This method is motivated by the moving average filter. The moving average filer method requires at least two parameters: width of the moving window and the threshold for approximation of 0, in this case, chosen to be 10^{-4}. These two parameters are inherently arbitrary so it is hard to justify which numerical values to use. Compared to the moving average filter method, the cumulative average method has one less arbitrary constant, the width of the window. Like its name suggests, the cumulative average method is just calculating cumulative average of the error metric. The advantage of this method is that the smoothed curve after applying this method is monotonously decreasing instead of oscillating. However, due to the nature of this method,

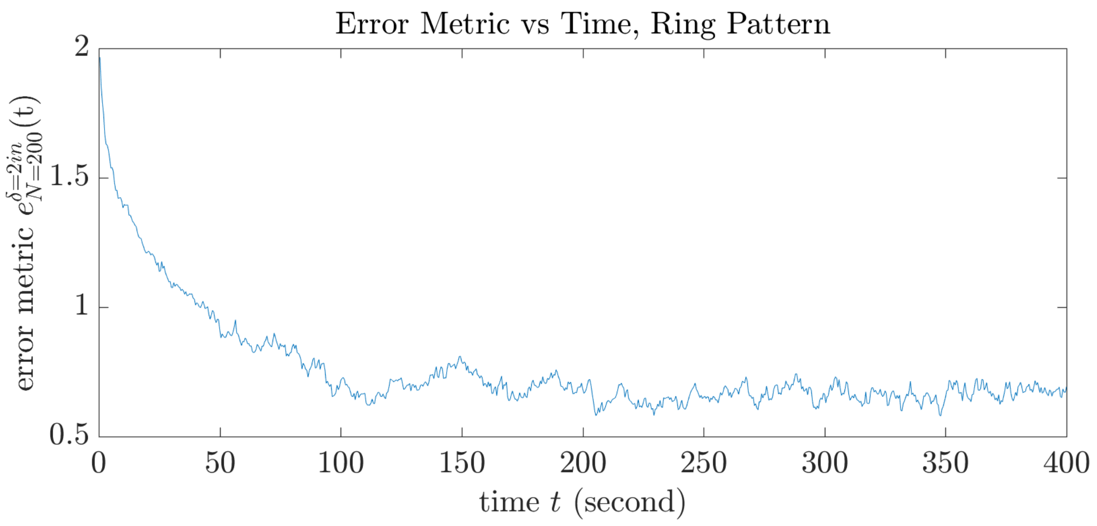


Fig. 2: Simulation with the ring pattern scalar field. This plot shows that the error metric always has random noise. The shape of the error metric curve suggests we should consider fitting a decaying exponential model to it.

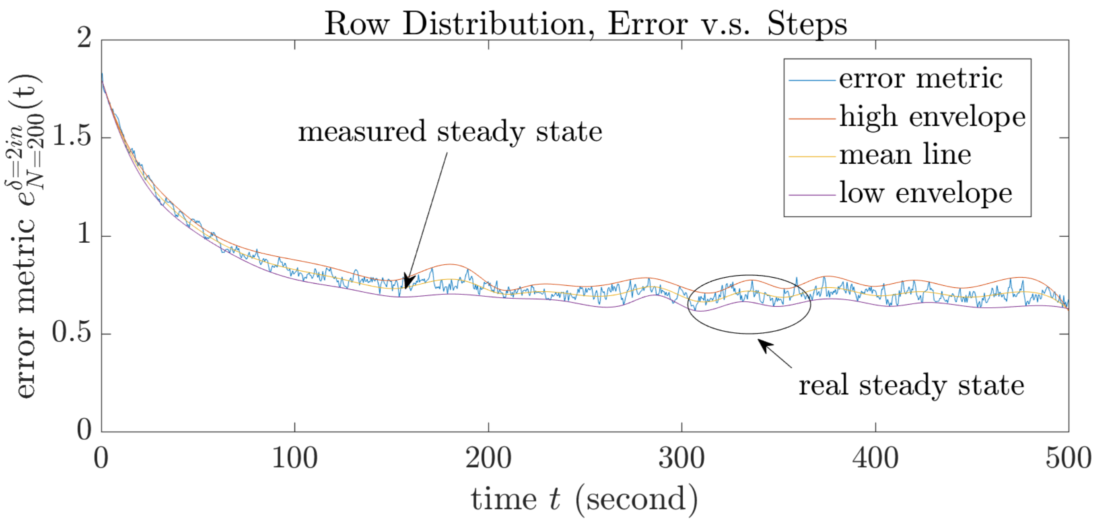


Fig. 3: Measurement of the steady state with the envelope method with target scalar field as the row pattern scalar field. The yellow mean line has low-frequency noise. The starting point of the measured steady state sometimes precedes that of the real steady state.

the measured steady state always comes long after the where the real steady state starts (Figure 4).

Lastly, we decide to fit exponential models to the error metric curve since Figure 2 looks like an exponential decay model with noise. Later in Section III-D, **Theorem 1**, we mention that the time convergence of our PDE model is indeed bounded by decaying exponential. We use `fit` function in the Curve Fitting Toolbox of MATLAB to fit the following exponential model:

$$y = ae^{-bt} + c. \tag{1}$$

This is achieved by using `exp2` as the parameter for the model and setting specific limits on the unwanted parameters. Then we apply the engineering convention of defining steady state [12]. Since we know the model (f_{fit}) we fit, we can calculate the asymptote of the exponential curve ($\lim_{t\to\infty} f_{\text{fit}}(t)$) and then define the threshold value (T) as follows:

$$T(f_{\text{fit}}) = \lim_{t\to\infty} 0.02(f_{\text{fit}}(0) - f_{\text{fit}}(t)) + f_{\text{fit}}(t). \tag{2}$$

The steady state starts when the error metric value first drops below the line defined by $y = T$.

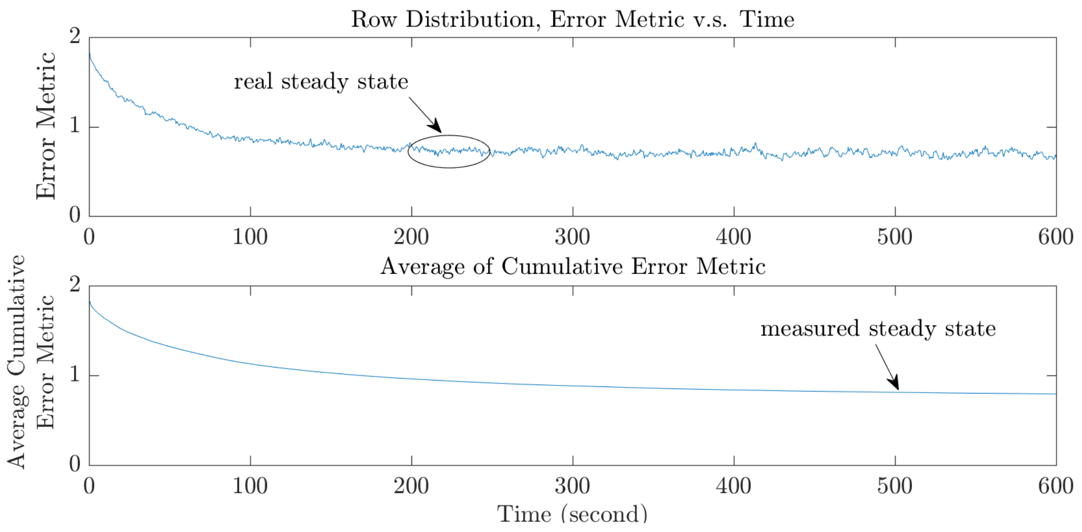


Fig. 4: Error metric plots with the target scalar field as the row scalar field. The top plot is the error metric curve. The bottom plot is the cumulative average of the error metric. This pair of plots shows that the starting point of the measured steady state with the cumulative averaging method comes long after the real steady state starts.

This method is not flawless, either. Even though the measured steady state is very close to the steady state determined by naked eye, it can still sometimes come before the real steady state. To make sure the measured steady state almost never comes before the real steady state, we pick an exponential decay model which looks like the error metric curve and add noise, which is drawn randomly from the shifted steady state error distribution, to it. We fit the exponential model according to equation (1) with various constants including 0.02 to define thresholds. We have tried constants ranging from 0.02 to 0.01. We decide to use 0.016 since it is the largest value with which the measured steady state only comes before the real steady state of the original exponential curve less than $10^{-3}\%$ of the time.

C. Results

Figure 5 shows one example of how to measure steady state with the error metric for the ring scalar field.

D. Theoretical Comparisons

After fitting the rate of convergence of the error metric to an exponential function, we decided to investigate the relationship between the convergence of our error metric and the convergence of the PDE model to steady state. In [3], the authors showed that with our governing control law and a zero-flux boundary condition, the solution to the PDE model converges to the scalar field in steady state. Furthermore, they showed that the rate of this convergence was bounded by a decaying exponential. However, they do not present expressions for the coefficient of the exponential term. Thus, we decided to begin our investigation by determining what this coefficient was in our case. We begin by exploring the simplified case of a uniform scalar field.

For the case of a uniform distribution, the rate of convergence is determined by the Poincaré constant of our domain. We can show this using standard arguments from analysis. For the following, recall that Ω, our domain, is a bounded subset of $\mathbb{R}^2$ with Lipschitz boundary. Let $u : \Omega \times [0, T] \to \mathbb{R}$ satisfy the following boundary value problem:

$$\begin{aligned} \frac{\partial u}{\partial t} - D^2 \Delta u &= 0 && \text{in } \Omega \times [0, T], \\ \mathbf{n} \cdot D^2 \nabla u &= 0 && \text{on } \partial\Omega \times [0, T], \\ u(\mathbf{x}, 0) &= u_0(\mathbf{x}) && \text{in } \Omega, \end{aligned} \tag{3}$$

where D is a constant because the scalar field is uniform. We now establish that $u(\mathbf{x}, t)$ converges as $t \to \infty$ to u_Ω, where

$$u_\Omega = \frac{1}{|\Omega|} \int_\Omega u_0(\mathbf{x}) d\mathbf{x}.$$

Note that $|\Omega|$ is the Lebesgue measure of our domain.

Theorem 1. For Ω, u, and η defined above, and $t \in [0, T]$, the following inequality holds:

$$\int_\Omega (u(\mathbf{x}, t) - u_\Omega)^2 d\mathbf{x} \le e^{-\frac{2}{C_\Omega} t} \int_\Omega (u_0(\mathbf{x}) - u_\Omega)^2 d\mathbf{x},$$

where C_Ω is the Poincaré constant of our domain.

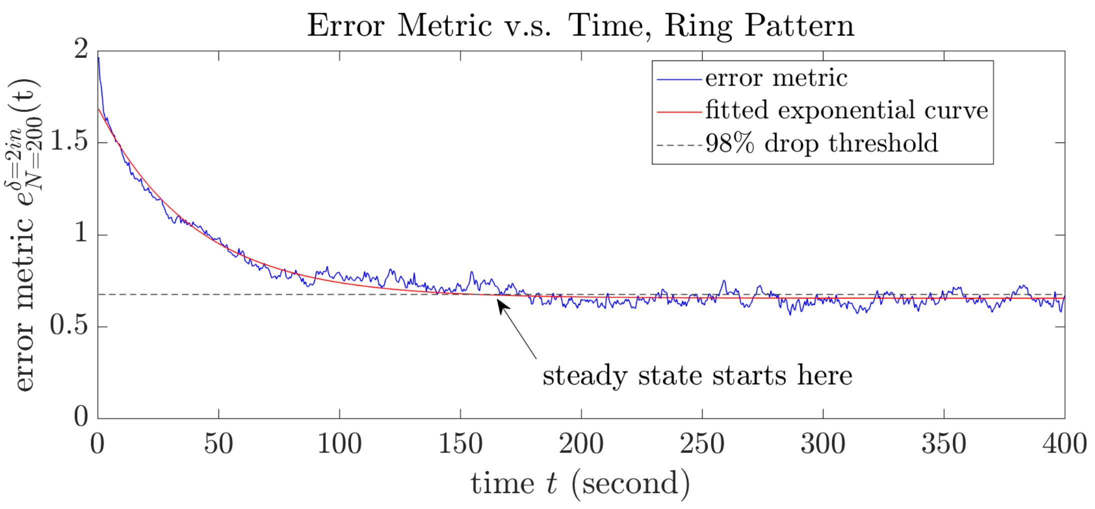


Fig. 5: An example of the steady state measurement with the exponential curve fitting method.

Proof: We begin by noting that for all $t > 0$, we have

$$\int_\Omega u(\mathbf{x},t)d\mathbf{x} = \int_\Omega u_0(\mathbf{x})d\mathbf{x}. \tag{4}$$

This is obtained by taking the time derivative of the left-hand side, and using equation (3). Next, we consider the quantity

$$\eta(t) = \frac{1}{2}\int_\Omega (u(\mathbf{x},t) - u_\Omega)^2 d\mathbf{x}.$$

Observe that

$$\begin{aligned}\frac{d\eta}{dt} &= \frac{d}{dt}\frac{1}{2}\int_\Omega (u(\mathbf{x},t) - u_\Omega)^2 d\mathbf{x}\\ &= \int_\Omega (u(\mathbf{x},t) - u_\Omega)u_t(\mathbf{x},t)d\mathbf{x}\\ &= \int_\Omega (u(\mathbf{x},t) - u_\Omega)\Delta u(\mathbf{x},t)d\mathbf{x},\end{aligned}$$

where the final substitution follows from the fact that $u(\mathbf{x},t)$ satisfies equation (3). Integrating this quantity by parts we obtain

$$\begin{aligned}&\int_\Omega (u(\mathbf{x},t) - u_\Omega)\Delta u(\mathbf{x},t)d\mathbf{x}\\ &= \int_{\partial\Omega} u(\mathbf{x},t)\nabla u(\mathbf{x},t)\cdot d\mathbf{S}\\ &- \int_\Omega \nabla u(\mathbf{x},t)\cdot\nabla u(\mathbf{x},t)d\mathbf{x}\\ &= -\int_\Omega (\nabla u(\mathbf{x},t))^2 d\mathbf{x}. \end{aligned}\tag{5}$$

Additionally, note that by equation (4) the function $(\mathbf{x},t) \to (u(\mathbf{x},t) - u_\Omega)$ has integral 0 on Ω. Therefore, we can apply the Poincaré inequality in order to obtain

$$\int_\Omega (u(\mathbf{x},t) - u_\Omega)^2 \leq C_\Omega \int_\Omega (\nabla u(\mathbf{x},t))^2,$$

where C_Ω is the Poincaré constant for our domain. This tells us that

$$\begin{aligned}\frac{d\eta}{dt} &= -\int_\Omega (\nabla u(\mathbf{x},t))^2 d\mathbf{x}\\ &\leq -\frac{1}{C_\Omega}\int_\Omega (u(\mathbf{x},t) - u_\Omega)^2\\ &= -\frac{2}{C_\Omega}\eta(t).\end{aligned}\tag{6}$$

Combining this with equation (5) we obtain

$$\eta(t) \leq e^{-\frac{2}{C_\Omega}t}\eta(0).$$

Applying the definition of $\eta(t)$ we obtain

$$\int_\Omega (u(\mathbf{x},t) - u_\Omega)^2 d\mathbf{x} \leq e^{-\frac{2}{C_\Omega}t}\int_\Omega (u_0(\mathbf{x}) - u_\Omega)^2 d\mathbf{x},$$

as desired. ■

With these results in mind, we were interested in determining the exact value of the Poincaré constant for our domain, in this case the rectangle $[0, 4.8] \times [0, 7]$. Unfortunately, the exact value of the Poincaré constant exactly has not been determined for a region of this type. However, it is not difficult to find an upper bound for the constant. For the case of the L^2 norm, according to [9], the Poincaré constant for a bounded, convex domain with Lipschitz boundary has an upper bound of $\frac{d}{\pi}$ where d is the diameter of the region of interest. Therefore, this gives the lower bound $\frac{2}{C} \geq 0.74$, where $\frac{2}{C}$ is the value of the coefficient in the exponential.

Having obtained an approximate value of C, simulations were run to compare the coefficient found in the exponential curve fit with the bound found analytically. Note, however, that the coefficient in the exponential model could depend on N, the number of robots, and δ, the size of the blob around each robot, both of which are variables not found in the PDE model. Thus, we were additionally interested in investigating the relationship between N, δ, and b, the coefficient in the exponential model. Preliminary simulation results have not supported the conjecture of the relationship between b and the Poincaré constant. In general, the values of b found in simulation were lower than the values predicted analytically. However, due to time constraints we were not able to run the simulation for very small values of δ. Thus, this topic requires further investigation. It is possible that in considering smaller values of δ and larger values of N, as well as non-uniform distributions, a relationship between b and the Poincaré constant could be found.

IV. Error Metric Extrema

In order to better understand whether error metric values represent good or poor swarm coverage, it is necessary to develop reference points on which comparisons can be based. It is known that the error metric falls between 0 and 2, but these values are not appropriate references since these bounds are only approached in limiting cases of N and δ [1]. The minimum and maximum values of the error metric for the desired swarm size, robot radius, and scalar field are natural choices for reference points. If the error found at steady state is 10% larger than the error metric minimum, it seems reasonably justified to classify the swarm as well distributed. In contrast, an error 10% smaller than the error metric maximum represents a poor robot configuration. The computation of these extrema is carried out numerically in Section IV-A, and the dependence of the error metric minimum on swarm characteristics is studied further in Section IV-B.

A. *Nonlinear Programming Problem*

To compute the error metric extrema, we first define the nonlinear optimization problem of interest. Let X represent a vector of N robot coordinates. The optimization problem is,

$$\begin{aligned} &\text{minimize } e_N^\delta(X), \\ &\text{subject to } X_i \in \Omega \text{ for } i \in \{1, 2, \ldots, N\}, \\ &X = (X_1, X_2, \ldots, X_N). \end{aligned} \tag{7}$$

Thus, the only constraints imposed on the problem are bounds keeping each robot coordinate within the domain. Note that to find the maximum of the error metric, the same problem structure can be used by minimizing the negative of e_N^δ. Problems of this form are readily solved in MATLAB using the built-in function `fmincon`. However, one limitation of `fmincon` is its restriction to finding local minima. Therefore, by running the optimization many times, the lowest local minimum can be taken as an estimate of the global minimum. Using a 4.8×7.0 ring scalar field with a swarm of $N = 200$ robots with radius $\delta = 0.2$, 50 local minima were computed with a mean of 0.2968 and a standard deviation of 0.0081. Therefore, using the smallest local minimum to represent the global minimum appears reasonable. These results are shown in Figures 6 and 7. Note that, as intuition predicts, the minimum of the error metric is higher than zero for a finite number of robots of nonzero radius.

We also tried another popular large-scalar nonlinear programming solver - IPOPT. Unlike `fmincon`, the MATLAB interface of IPOPT requires the user to specify the gradient of the objective function for this optimization problem. Since the objective function is complicated for a large number of robots, finding the symbolic expression for the gradient of the objective function is impractical. Therefore, IPOPT is not as applicable to our optimization problem as `fmincon`.

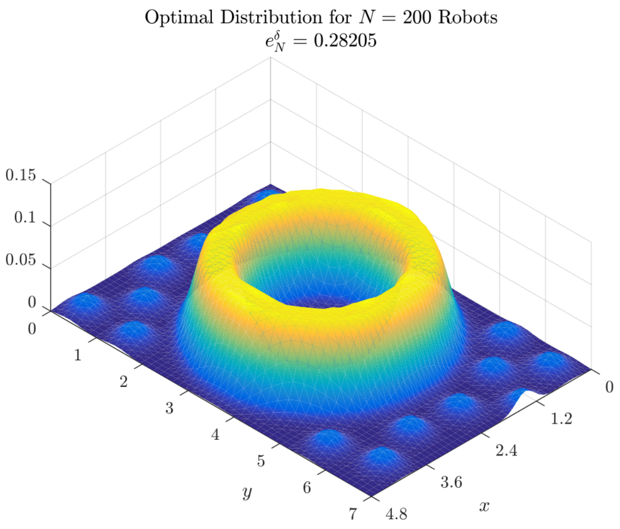


Fig. 6: Robot distribution over ring scalar field yielding minimum value of error metric.

B. *Dependence on Swarm Size*

Further insight into the error metric can be gained by studying the dependence of the metric extrema on the number of robots N and their radius δ. This is done by slightly modifying the optimization problem (7) to include δ as a decision variable; $X = (\delta, X_1, X_2, \ldots, X_N)$. Thus, `fmincon` not only computes the optimal coordinate for each robot X_i, but it also determines which radius δ yields the lowest error for a given swarm size. Figure 8 shows the error metric minimum versus the number of robots in the swarm. Note that each data point has a different radius δ. As one would expect, the error decreases as N increases. Furthermore, the error roughly scales linearly

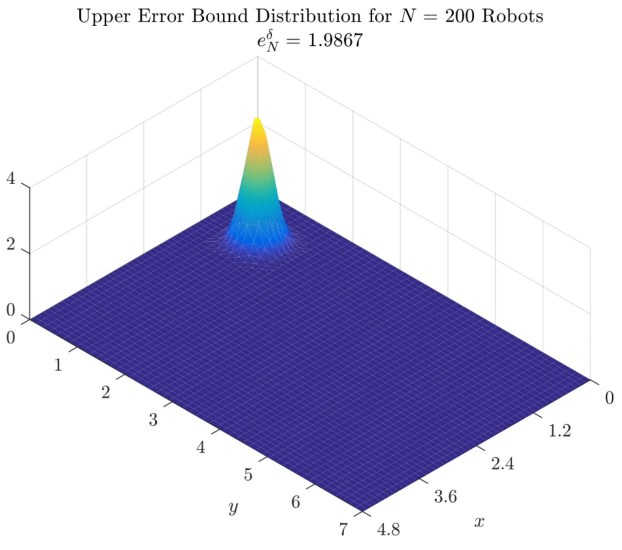


Fig. 7: Maximum error metric value arises when all robots coincide outside the ring.

with $1/\sqrt{N}$, which agrees with the claim made from the 2016 REU Robots group. Interestingly, Figure 8 shows a slight oscillation about the trendline, with an increasing frequency as N increases. We suspect that this may be an artifact of the numerical solver, but further investigation should be carried out to validate this suspicion.

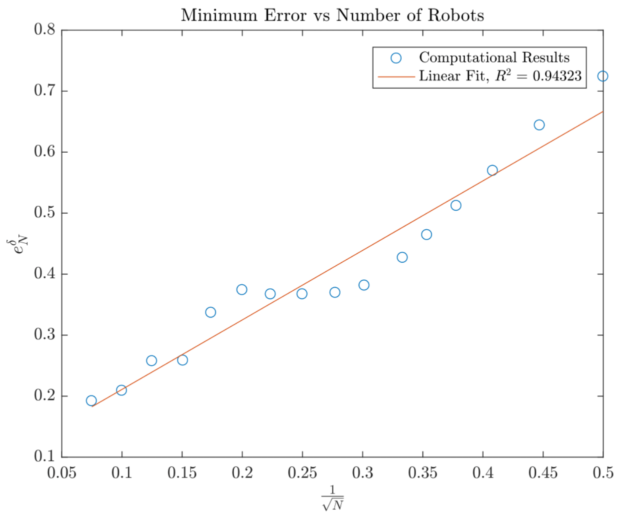


Fig. 8: Minimum value of error metric scales linearly with $1/\sqrt{N}$.

V. Error Probability Density Function

The goal of our error metric is to quantify the distance a given robot configuration is from being proportional to our scalar field. Thus, one way to contextualize error metric values is to compare them to values that occur when our robot positions are pulled from a distribution proportional to the scalar field (ie., the goal is met). Let f be this distribution. Define random variable

$$\tilde{e}_N^\delta = e_N^\delta(X),$$

where X is a vector $(X_1, .., X_N)$ and X_is are independent, identically distributed random variables, each taking values in $\mathbb{R}^2$, with density function f. In order to contextualize values of the error metric, we want to find the distribution of this random variable. Thus, the following section focuses on finding the PDF of $\tilde{e}_N^\delta$. We will show, both numerically and analytically, that as N gets large and δ gets small (with special conditions on the rates) the error of $\tilde{e}_N^\delta$ approaches a normal random variable whose mean and variance also decrease in N and δ. Further, we will show that $\tilde{e}_N^\delta$ converges in the sense of distributions to 0 under these conditions, proving that X_i converges in the sense of distributions to f implies convergence to the target distribution with respect to this error metric. The following is a formula for the derived CDF of a random variable $\tilde{e}_N^\delta$

$$\mathbb{P}(e(X) \leq y) = \int_{\{x|e(x)\leq y\}} g_X(x)dx,$$

where g_X is the pdf of $X = (X_1, ..., X_N)$. In our circumstance, this gives

$$\mathbb{P}(e(X) \leq y) = \int_\Omega \mathbb{1}_{\{x|e_N^\delta(x)\leq y\}} \prod_{i=1}^N f(x_i)dx. \tag{8}$$

A. Numerical Integration for Derived Distribution

For a swarm of N robots, the integral defining the cumulative density function of the error metric is of dimension $2N + 1$, and therefore finding analytical representations for the derived distribution becomes unfeasible for large swarms. Therefore, we approximate the solution to (8) using Monte Carlo integration. Monte Carlo integration is a numerical scheme well-suited for high-dimensional integration [10]. The result for a ring scalar field with 200 robots using $M = 1000$ Monte Carlo evaluation points is shown in Figure 9. The numerical solution appears to closely match an $\mathrm{erf}(\cdot)$ function, and therefore an analytical curve is fit to the data using MATLAB's `lsqcurvefit` least squares curve fitting function. To obtain the probability density function of the error metric, the analytical expression found for the CDF is differentiated, and the result is shown in Figure 10.

An important property of the numerical integration scheme used is that it relies on random variables; Monte Carlo method yields a different value for the CDF each time the MATLAB code is run. However, due to its $1/\sqrt{M}$ convergence, any noticeable discrepancies vanish as the number of evaluation points increases

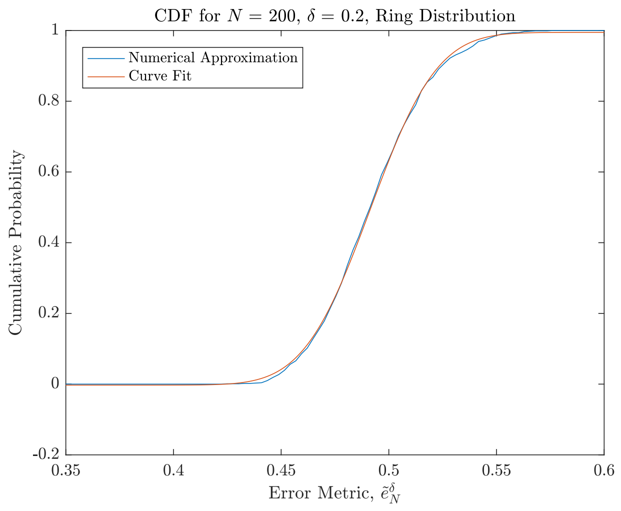


Fig. 9: Cumulative density function of error metric obtained from Monte Carlo integration. The use of an $\mathrm{erf}(\cdot)$ curve fit matches very closely.

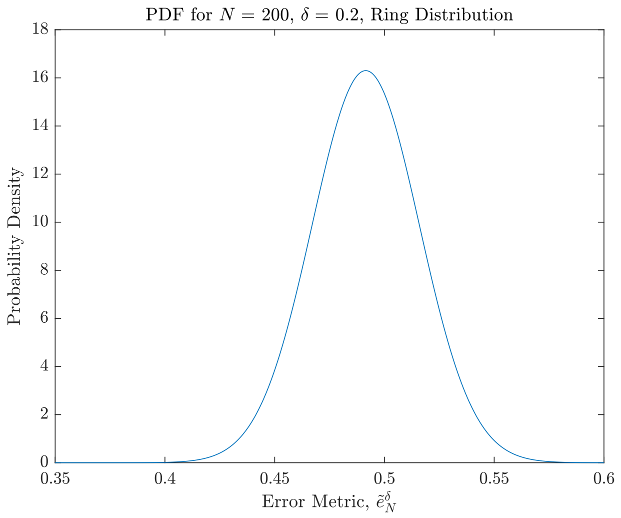


Fig. 10: Probability density function of error metric. The $\mathrm{erf}(\cdot)$ CDF curve fit yields a Gaussian PDF.

[10]. Deterministic numerical schemes similar to Monte Carlo method do exist for high-dimensional integration, namely, quasi-Monte Carlo methods. These schemes utilize the same general form as Monte Carlo method, but with evaluation points chosen from a low-discrepancy sequence. Quasi-Monte Carlo integration have convergence rates between $1/\sqrt{M}$ and $1/M$, where the rate of convergence tends towards that of standard Monte Carlo as the dimensionality of the problem increases and the smoothness of the integrand deteriorates. Furthermore, in order to use quasi-Monte Carlo for non-uniform scalar fields, a non-uniform low-discrepancy sequence must be generated. This is typically done using the Hlawka-Mück method whereby an existing low-discrepancy sequence is transformed according to the desired distribution. Unfortunately, this method is computationally expensive with a complexity of $\mathcal{O}(M^2)$ and no explicit algorithm for distributions of dependent variables [6]. Rejection sampling methods for obtaining non-uniform low-discrepancy sequences do exist, but the quasi-Monte Carlo efficiency is lost.

B. Convergence to Normal

In this section we present some theoretical results from Horváth [7] that support these numerical findings. Because N and δ are related in this analysis, we assign one δ value to each N, and refer to it as δ_N. We will refer to $\tilde{e}_N^{\delta_N}$ simply as $\tilde{e}_N$ for this reason. Our work fits into the theoey of kernel density estimators. Indeed, our blob function defined II is exactly analogous to the kernel density estimator associated to f which is denoted by f_n in [7]. Before proceeding with our analysis, we will define some expressions: Let

$$D = \int_{\mathbb{R}^2} K^2(u)du.$$

Let

$$\phi(u) = (2\pi)^{\frac{-1}{2}} e^{\frac{-u^2}{2}}.$$

We take

$$f_{(N)} = \frac{1}{\delta_N^2} \int_{\mathbb{R}^2} K\left(\frac{x-y}{\delta_N}\right) f(y)dy,$$

where K is a kernel, which, as explained in Section II, we take to be Gaussian type. Specifically, we define

$$K = \frac{1}{2\pi} e^{\frac{-x^2-y^2}{2}} \times \mathbb{1}_{\{\mathbb{B}_r\}} \frac{1}{C},$$

where C and r are chosen such that K integrates to 1. Let

$$m_n(x) = N^{\frac{1}{2}} \delta_N (f_{(N)} - f).$$

We take the asymptotic expected value, v_N, to be

$$v_N = \int_{-\infty}^{\infty} \int_{\mathbb{R}^2} |f^{\frac{1}{2}} D^{\frac{1}{2}} u + m_N(x)| \mathbb{1}_{\{\Omega\}} \phi(u) d\mathbf{x} du$$

and let $\sigma(N)$ be defined as in [7]. For our purposes, it is important to note only that $0 < \sigma(N) < C$ for some constant C independent of N.

Theorem 2.

$$\frac{1}{\sigma(N)\delta_N}(N^{\frac{1}{2}} \tilde{e}_N - v_N) \to_{\mathscr{D}} \mathcal{N}(0,1)$$

when

- $\delta_N = \mathcal{O}(N^{\frac{-1}{6}})$ as $N \to \infty$
- $\delta_N N^{\frac{1}{4}} \to \infty$ as $N \to \infty$

This result is a direct application of the theorem proved in Horváth's paper. This theorem applies if we smooth our scalar field to be C^2 because our kernel satisfies hypotheses (C1)-(C6) from the paper.

An important corollary to this theorem is the following:

Corollary 3. $\tilde{e}_N \to_{\mathscr{D}} 0$

Proof: Using symmetry of the kernel and the Taylor approximation of f [5], we see that $v_N = \mathcal{O}(N^{\frac{1}{2}}\delta_N^3)$. Because $\sigma(N)$ is finite, we get

$$\begin{aligned}
&\frac{1}{\sigma(N)\delta_N}(N^{\frac{1}{2}}\delta_N\tilde{e}_N - v_N) \to_{\mathscr{D}} N(0,1)\\
&\Rightarrow \frac{1}{\sigma(N)\delta_N}(N^{\frac{1}{2}}\delta_N\tilde{e}_N - CN^{\frac{1}{2}}\delta_N^2) \to_{\mathscr{D}} N(0,1)\\
&\Rightarrow \frac{N^{\frac{1}{2}}}{\sigma(N)}(\tilde{e}_N - C\delta_N^2) \to_{\mathscr{D}} N(0,1)\\
&\Rightarrow \frac{N^{\frac{1}{2}}}{\sigma(N)}(\tilde{e}_N - C\delta_N^2) = N(0,1) + o_N(1)\\
&\Rightarrow \tilde{e}_N = N(C\delta_N^2, \frac{\sigma^2(N)}{n}) + o_N(1)\\
&\Rightarrow \tilde{e}_N \to_{\mathscr{D}} 0
\end{aligned}$$

The goal of our error metric is to measure how far a given configuration of robots is from being proportional to our scalar field. Thus Corollary 3 is of importance to our analysis because it shows that when our robot positions are pulled from a distribution proportional to our scalar field, the error does in fact converge to 0.

Further, we conjecture that smoothing our scalar field is unnecessary. Giné et. al. [4] showed in 2003 that the result did not rely on the continuity of the target distribution for univariate scalar fields, and alluded to a multivariate counterpart that would apply in our case. The author further lessens constraints on the relationship between δ_n and n, giving the constraints

- $\delta_n \to 0$ as $n \to \infty$
- $\sqrt{n}\delta_n \to \infty$ as $n \to \infty$

instead of those given in Theorem 2.

VI. Boundary Control Law Simulations

A. Introduction

In this section, we study various boundary control laws and how they affect the distribution of robots. The boundary control laws we had available at the beginning of the project were:

1) Specular reflection [11]: The robot reaches the boundary and reflects across a line normal to the boundary. The incident angle is equal to the angle of reflection.
2) Bounceback: This boundary control law was developed by the summer 2016 swarm robotics team. The robot hits the boundary and then reverses its direction for the remaining time.
3) Single rejection [11]: If the calculated step takes the robot out of bounds then the robot does not move for the given time step.
4) Multiple rejection [11]: The robot continues to calculate new steps until it calculates a step within the boundary.
5) Interruption [11]: The robot reaches the boundary and calculates a new step for the remaining time.

We also developed a new boundary control law called implementable rejection, which is discussed later in this section. Although specular reflection is commonly used to simulate a zero-flux boundary condition and has been demonstrated to not warp the boundary distribution [11], this boundary control law is not implementable since our robots do not have localization capabilities. Therefore, following the summer 2016 swarm robotics team, we continue using bounceback as our preferred boundary control law.

In order to investigate the robustness of our bounceback boundary control law and compare it with other boundary control laws, we simulate $N = 200$ robots with $\delta = 0.2021$ moving on a uniform distribution. These simulations are adapted from the summer 2016 swarm robotics team's MATLAB code and modified to be compatible with C++. In order to increase the efficiency of the code, we use a stamping method to calculate the Gaussian blob function of each robot. Additionally, we introduce wrap-around to the Gaussian blob function calculations to increase accuracy. In these simulations, the domain Ω of the robots' movement is discretized into a 200 by 200 grid space. Further explanations in this section refer to discrete elements of this space as bins. We refer to a grid with side lengths equal to those of the original domain as a square.

B. Implementation

1) Wrap-Around: In the simulations, each side of a square consists of 200 bins. Previous calculations of the error metric had a deficiency in the Gaussian blob distribution along the boundaries because as the position of a robot approached any of the boundaries of the domain, an increasingly large portion of the robot's Gaussian blob was lost because it exceeded the limits of the domain. Our solution is to increase the size of the domain over which the Gaussian blob is calculated to a triple grid that has sides three times as long as the original grid. In other words, the triple grid is composed of nine squares. Our original domain corresponds to the middle square of the triple grid. If we put this on a coordinate axis, then since the width of the original domain ranges from 0 to 4.8 and the height ranges from 0 to 7, the width of the triple grid ranges from -4.8 to 9.6 and the height ranges from -7 to 14. After calculating the Gaussian blobs of the robots over this triple grid, we match the bottom left corner of each square and sum the Gaussian blob values for each bin, thus collapsing the triple grid into a single grid with the dimensions of the original grid.

2) Stamping: Previous simulations calculated the Gaussian blobs of each robot for each time step. To decrease the number of calculations, we create a stamp of the Gaussian blob for a single robot located at the point $(0, 0)$, i.e. the bottom left corner of the original domain. This blob is calculated on a "double grid" twice as large as the actual domain. The double grid consists of four squares, with the original domain corresponding to the top left square of the double grid. Instead of recalculating the Gaussian blob for each robot, we shift the blob stamp to be centered at each robot's position and add the stamp to the triple grid.

C. Results

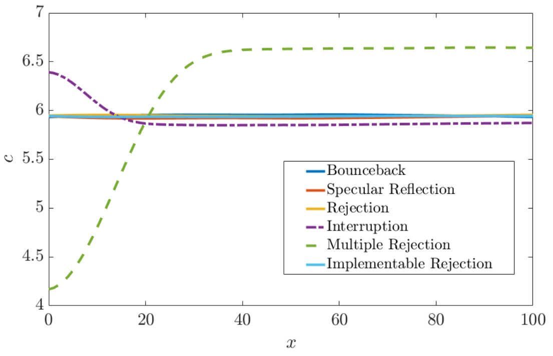


Fig. 11: Concentration profiles of boundary control laws

Figure 16 demonstrates that bounceback maintains a uniform distribution in the middle of the domain and the density in the corners is higher than that of the rest of the domain. However, the corner warping does not result in distortion of the distribution along the rest of the boundary. Neither multiple rejection nor interruption maintains a uniform distribution. Bounceback, specular reflection, single rejection, and implementable rejection maintain similar concentration profiles while multiple rejection and interruption do not (Figure 11).

1) Corner Warping: The warping we see in the corners from the bounceback boundary control law is caused by double bounce: a robot in a corner takes a step large enough to cause it to hit the boundary twice, effectively getting stuck in the corner. To study the warping in the corner region, we examine the normalized density surplus of the corner region. This quantity represents how much more dense a region is than the average density of the whole domain. The lower bound of the color axis in Figure 13 is 0 because we are interested in studying the region in which there is a density surplus.

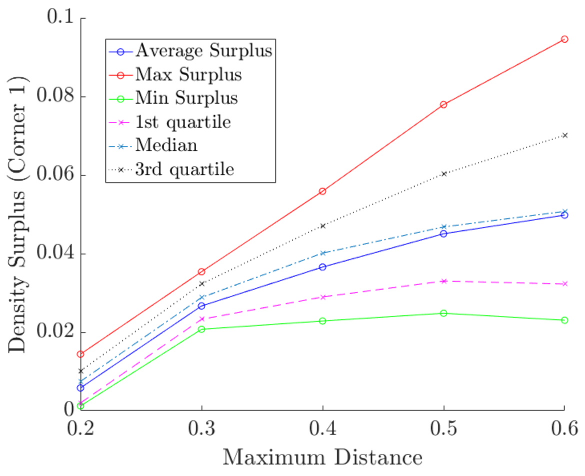


Fig. 12: Due to double bounce, the average density in the corner increases as d_{max} increases

The amount of warping in the corners decreases as the maximum distance d_{max} decreases (Figure 12) because as the maximum distance the robot can travel decreases, the area in which the robot can take a step large enough to cause it to hit the boundary twice decreases. We expect the corner region to take the shape of a quadrant. However, the results do not reflect this as d_{max} increases. The reason for this warping is a possible area of further study. Additionally, the theoretical region in which double bounce is possible does not entirely encompass the region in which a density surplus occurs. An exact analytic expression for the density surplus region remains to be determined.

2) Implementable Rejection: Single rejection does not cause warping at the corners, but is also not implementable because the robots lack localization capabilities. Let us consider an implementable version of single rejection that is similar to bounceback: after hitting a boundary, the robot reverses its direction and attempts to return to its original position; if there is not enough time for it to return then it goes as far back as possible. Like single rejection, implementable rejection does not cause warping at the corners. However, we expect the convergence time to be higher for implementable rejection. Additionally, the error metric for implementable rejection on a uniform distribution is not significantly lower than that for bounceback.

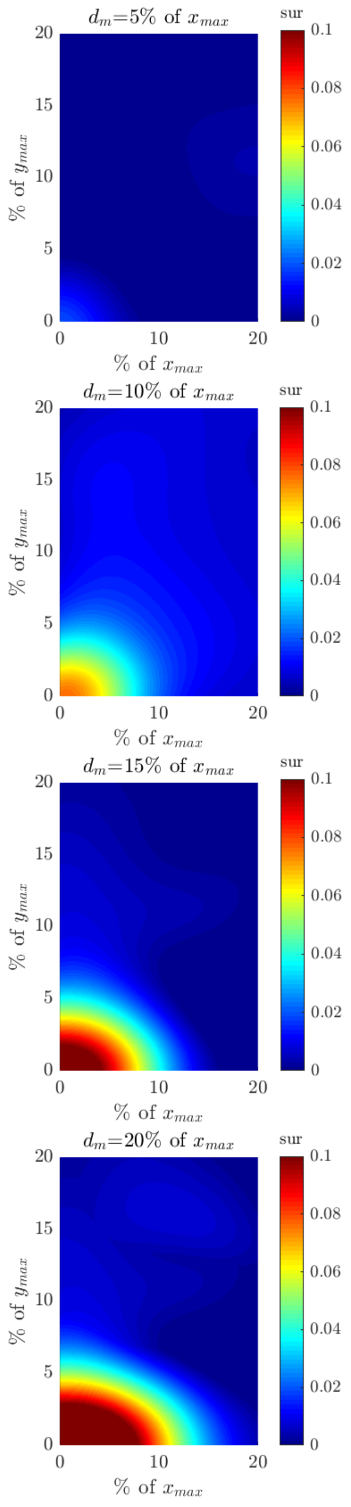


Fig. 13: Density of a corner as a ratio of the average density of the domain. The size and value of the surplus varies with d_{max}.

D. Further Exploration

We also wish to examine the behavior of robots moving on a circular domain.

Figures 14 and 15 compares the results of single rejection and bounceback on a circular domain. Without the problem of double bouncing in a corner, the two distributions appear approximately equally uniform. Note that the deficiency at the boundaries probably occurs because we did not implement wrap-around on the circular domain.

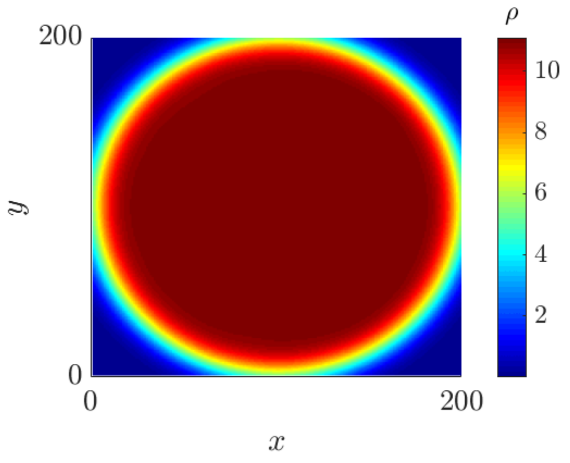


Fig. 14: Single rejection on circular domain.

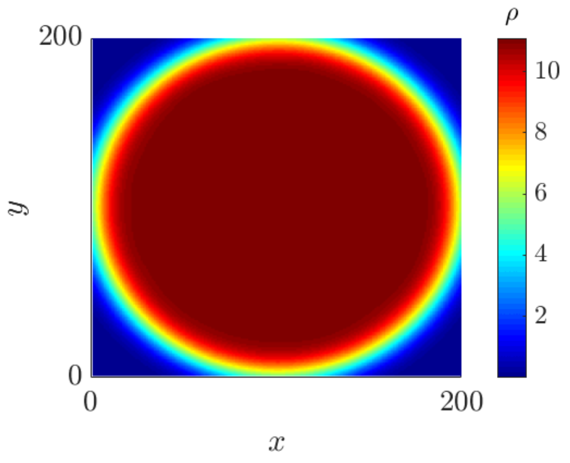
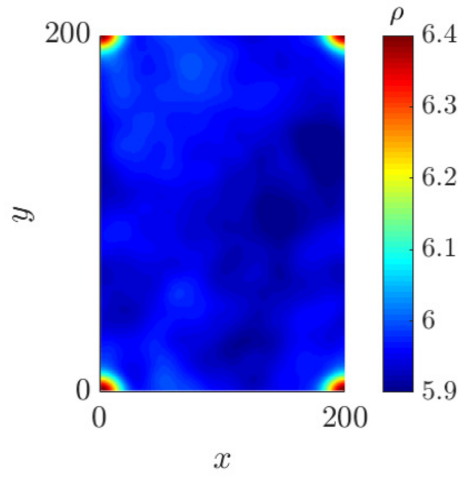
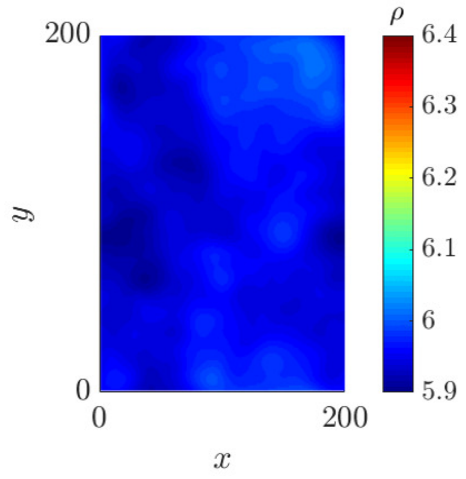


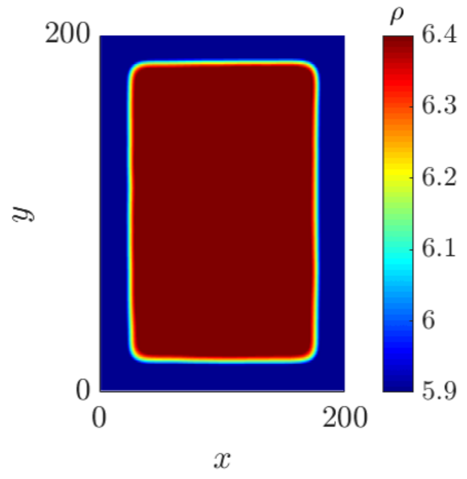
Fig. 15: Bounceback on circular domain.



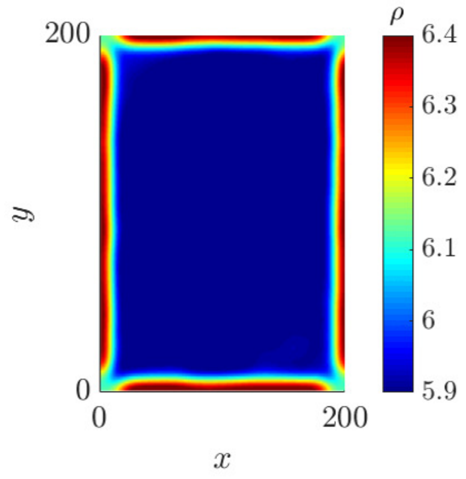
(a) Bounceback



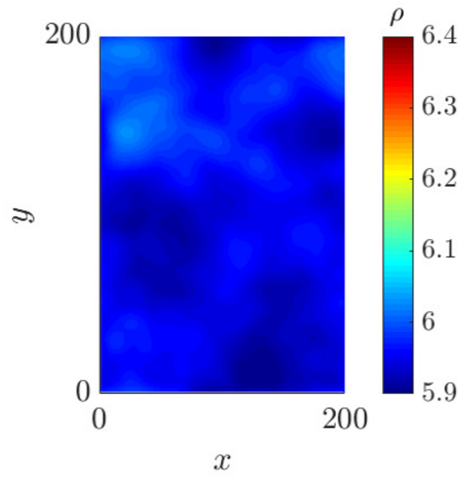
(b) Specular Reflection



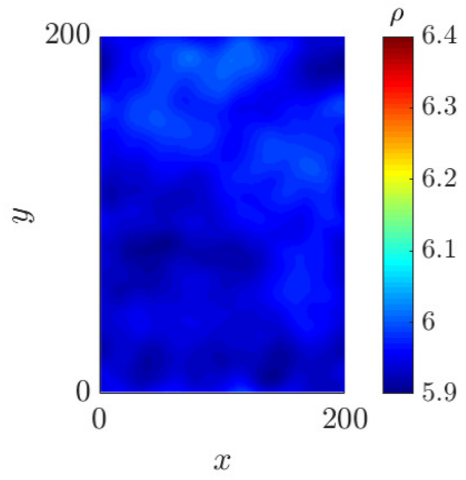
(c) Multiple Rejection



(d) Interruption



(e) Single Rejection



(f) Implementable Rejection

Fig. 16: Gaussian blob distributions for various boundary control laws. Following the previously described discretization method, the height and width of the domain have been divided into 200 bins.

VII. Robot Flux

A. Introduction

In order to better understand how the distribution of the swarm evolves, we developed an analytical framework for understanding how the density of agents changes over one time step. To this end, we developed an idea of robot flux, or, the likelihood that a robot at a given position would enter or exit a target region in a single time step. We developed this idea in order to understand how the density of robots changes over time. We showed that for a scalar field, if agents are distributed according to the scalar field, then near no boundaries and near a single boundary, the robots tend to maintain their uniform distribution. Furthermore, the argument breaks down for the case when a robot is near a corner. Finally, we were also able to show that for a uniform scalar field, if the robots are not distributed uniformly, their distribution tends toward uniform. We will begin by rigorously defining robot flux. We begin with preliminary definitions that help us describe the movement of the robots over a single time step, and how this relates to robot flux:

B. Preliminaries

For the following section, assume the scalar field F is uniform. Let $\Omega = [0,1] \times [0,1]$. We take

$$H_1 = \left\{(x_1, x_2) : \left((|x_1 - a| < \frac{\alpha_{max}}{2}) \vee (|x_1 - b| < \frac{\alpha_{max}}{2})\right)\right\}$$

and

$$H_2 = \left\{(x_1, x_2) : \left((|x_2 - c| < \frac{\alpha_{max}}{2}) \vee (|x_2 - d| < \frac{\alpha_{max}}{2})\right)\right\}.$$

Define $H = H_1 \cap H_2$. Let $\alpha_{max} \in \mathbb{R}^+$ be the maximum distance a robot can travel in one time step. Assume $\alpha_{max} < \frac{1}{4}$. Define $\boldsymbol{f_\alpha}$, the probability density function of α, to be a function such that its support is a subset of $[0, \alpha_{max}]$. We take α be a random variable with distribution f_α. Let β be a random variable pulled uniformly at random from $[0, 2\pi]$. Let $\rho(x_1, x_2)$ be a PDF with domain Ω that represents the current distribution of robots.

We introduce some notation. The **displacement function** gives the position after one time step of a robot located at $\mathbf{x} = (x_1, x_2)$, given that the robot chooses a direction β and a speed α at which to travel. Let $\tilde{s}_{\alpha,\beta}(x_1, x_2) = (x_1, x_2) + (\alpha\cos(\beta), \alpha\sin(\beta))$. Define the displacement function, $s : \Omega \setminus H \to \Omega$, to be

$$s_{\alpha,\beta}(x_1, x_2) = \begin{cases} \tilde{s}_{\alpha,\beta}(x_1, x_2) & \tilde{s}_{\alpha,\beta}(x_1, x_2) \in \Omega \\ \tilde{s}_{\alpha_0,\beta}(x_1, x_2) + \tilde{s}_{\alpha_0-\alpha,\beta}(x_1, x_2) & \tilde{s}(x, y, \alpha, \beta) \notin \Omega, \end{cases}$$

where α_0 is the unique value such that $\tilde{s}(x, y, \alpha_0, \beta) \in \partial\Omega$. We define the **entrance function** on a region $S \subset \Omega \setminus H$, $N_S : \Omega \setminus S \to \mathbb{R}$ such that $N_S(x_1, x_2)$ is the probability that $s_{\alpha,\beta}(x_1, x_2) \in S$. We define the **exit function** on a region $S \subset \Omega \setminus H$, $T_S : S \to \mathbb{R}$ such that $T_S(x_1, x_2)$ is the probability that $s_{\alpha,\beta}(x_1, x_2) \notin S$. The **influx** of a region $S \subset \Omega \setminus H$, I_S is

$$I_S = \int_{\Omega \setminus S} N_S(x_1, x_2)\rho(x_1, x_2)d\mu.$$

The **outflux** of a region $S \subset \Omega \setminus H$, O_S is

$$O_S = \int_S T_S(x_1, x_2)\rho(x_1, x_2)d\mu.$$

We will define the **flux** of a region S,

$$\Phi_S = I_S - O_S.$$

C. Main Result

Theorem 4. For any $S \subset \Omega \setminus H$, $\Phi_S = 0$ when $\rho(x_1, x_2)$ is the uniform distribution.

Proof.
Part 1: Center Region
Assume for all $x \in S$, $d(x, \partial\Omega) > \alpha_{max}$. If we think of α and β as fixed quantities, in this region $s_{\alpha,\beta-\pi}(s_{\alpha,\beta}(x_1, x_2)) = (x_1, x_2)$, so $s_{\alpha,\beta-\pi} = s^{-1}_{\alpha,\beta}$. This inverse, $s_{\alpha,\beta-\pi}$, is differentiable because it is a translation. Thus we will use it as a change of variables function in the following proof. Because $\rho(x_1, x_2)$ is a uniform distribution on the unit square, the $\rho(x_1, x_2)$ term is identically 1 everywhere in Ω, so we are left with influx of

$$\begin{aligned} I_S &= \int_{\Omega \setminus S} N_S d\mu \\ &= \int_{\Omega \setminus S} \int_0^{\alpha_{max}} \int_0^{2\pi} \mathbb{1}_{\{s_{\alpha,\beta}(x_1,x_2) \in S\}} f_\alpha \frac{1}{2\pi} d\beta d\alpha d\mu \\ &= \int_0^{\alpha_{max}} f_\alpha \frac{1}{2\pi} \int_0^{2\pi} \int_{\Omega \setminus S} \mathbb{1}_{\{s_{\alpha,\beta}(x_1,x_2) \in S\}} d\mu d\beta d\alpha. \end{aligned}$$

Then we make the change of variables in (x_1, x_2)

$$(x_1, x_2) = s_{\alpha,\beta-\pi}(x_1', x_2').$$

In the center region,

$$s_{\alpha,\beta-\pi}(x_1', x_2') = (x_1', x_2') + (\alpha\cos(\beta-\pi), \alpha\sin(\beta-\pi))$$

thus this change of variables has a Jacobian of 1. Since this is a simple change of variables in x_1 and x_2, we compute only the inner integral, viewing α and β as fixed quantities.

$$\int_{\Omega \setminus S} \mathbb{1}_{\{s_{\alpha,\beta}(x_1,x_2) \in S\}} d\mu$$

$$= \int_{s_{\alpha,\beta}(\Omega\setminus S)} \mathbb{1}_{\{s_{\alpha,\beta}(s_{\alpha,\beta-\pi}(x_1',x_2'))\in S\}} d\mu$$
$$= \int_{s_{\alpha,\beta}(\Omega\setminus S)} \mathbb{1}_{\{(x_1',x_2')\in S\}} d\mu$$
$$= \int_{\Omega} \mathbb{1}_{\{s_{\alpha,\beta-\pi}(x_1',x_2')\}\notin S\}} \mathbb{1}_{\{(x_1',x_2')\in S\}} d\mu$$
$$= \int_{S} \mathbb{1}_{\{s_{\alpha,\beta-\pi}(x_1',x_2')\}\notin S\}} d\mu.$$

Thus we have

$$I_S = \int_0^{\alpha_{max}} f_\alpha \frac{1}{2\pi} \int_0^{2\pi} \int_S \mathbb{1}_{\{s_{\alpha,\beta-\pi}(x_1',x_2')\}\notin S\}} d\mu d\beta d\alpha$$
$$= \int_0^{\alpha_{max}} f_\alpha \frac{1}{2\pi} \int_0^{2\pi} \int_S \mathbb{1}_{\{s_{\alpha,\beta}(x_1',x_2')\}\notin S\}} d\mu d\beta d\alpha$$
$$= \int_S \int_0^{\alpha_{max}} \int_0^{2\pi} \mathbb{1}_{\{s_{\alpha,\beta}(x_1',x_2')\}\notin S\}} f_\alpha \frac{1}{2\pi} d\beta d\alpha d\mu$$
$$= \int_S T_S d\mu$$
$$= O_S.$$

Thus $\Phi_S = I_S - O_S = 0$. ■

Part 2: Near One Edge

Now assume for all $(x_1, x_2) \in S, x \in [0, \alpha_{max}], y \in [\alpha_{max}, 1 - \alpha_{max}]$.

$$I_S = \int_{\Omega\setminus S} N_S d\mu$$
$$= \int_{\Omega\setminus S} \int_0^{\alpha_{max}} \int_0^{2\pi} \mathbb{1}_{\{s_{\alpha,\beta}(x_1,x_2)\in S\}} f_\alpha \frac{1}{2\pi} d\beta d\alpha d\mu$$
$$= \int_0^{\alpha_{max}} f_\alpha \frac{1}{2\pi} \int_0^{2\pi} \int_{\Omega\setminus S} \mathbb{1}_{\{s_{\alpha,\beta}(x_1,x_2)\in S\}} d\mu d\beta d\alpha.$$

Now we make a similar change of variables, but this time we must split it into two cases.

Case 1 $x > \alpha\cos(\beta)$

Here, it is exactly the same as the center region case.

Case 2 $x < \alpha\cos(\beta)$

In this case, $s_{\alpha,\beta}$ is its own inverse. So we use $(x_1, x_2) = s_{\alpha,\beta}(x_1', x_2')$ as our change of variables.

In this case we have

$$s_{\alpha,\beta} = (-\alpha\cos(\beta) - x_1, x_2 - 2x_1\tan(\beta) - \alpha\sin(\beta)).$$

Thus the Jaccobian is

$$J = \left|\det\left(\begin{bmatrix} -1 & -\tan(\beta) \\ 0 & 1 \end{bmatrix}\right)\right| = 1$$

in this case as well. Once again, we only calculate the inner integral. We use the notation

$$s_{\alpha,\beta}(x_1, x_2) = (s_{\alpha,\beta}(x_1,x_2)_1, s_{\alpha,\beta}(x_1,x_2)_2).$$

$$\int_{\Omega\setminus S} \mathbb{1}_{\{s_{\alpha,\beta}(x_1,x_2)\in S\}} d\mu$$
$$= \int_{\Omega\setminus S\cap\{x>\alpha\cos(\beta)\}} \mathbb{1}_{\{s_{\alpha,\beta}(x_1,x_2)\in S\}} d\mu$$
$$+ \int_{\Omega\setminus S\cap\{x>\alpha\cos(\beta)\}} \mathbb{1}_{\{s_{\alpha,\beta}(x_1,x_2)\in S\}} d\mu$$
$$= \int_{\Omega} \mathbb{1}_{\{(x_1,x_2)\in\Omega\setminus S\}} \mathbb{1}_{\{x>\alpha\cos(\beta)\}} \mathbb{1}_{\{s_{\alpha,\beta}(x_1,x_2)\in S\}} d\mu$$
$$+ \int_{\Omega} \mathbb{1}_{\{(x_1,x_2)\in\Omega\setminus S\}} \mathbb{1}_{\{x<\alpha\cos(\beta)\}} \mathbb{1}_{\{s_{\alpha,\beta}(x_1,x_2)\in S\}} d\mu$$
$$= \int_{s_{\alpha,\beta}(\Omega)} \mathbb{1}_{\{s_{\alpha,\beta-\pi}(x_1',x_2')\in\Omega\setminus S\}} \mathbb{1}_{\{s_{\alpha,\beta-\pi}(x_1',x_2')_1>\alpha\cos(\beta)\}}$$
$$\times \mathbb{1}_{\{s_{\alpha,\beta-\pi}(s_{\alpha,\beta}(x_1',x_2'))\in S\}} d\mu$$
$$+ \int_{s_{\alpha,\beta}(\Omega)} \mathbb{1}_{\{s_{\alpha,\beta}(x_1',x_2')\in\Omega\setminus S\}} \mathbb{1}_{\{s_{\alpha,\beta}(x_1',x_2')_1<\alpha\cos(\beta)\}}$$
$$\times \mathbb{1}_{\{s_{\alpha,\beta}(s_{\alpha,\beta}(x_1',x_2'))\in S\}} d\mu$$
$$= \int_{\Omega} \mathbb{1}_{\{s_{\alpha,\beta-\pi}(x_1',x_2')\in\Omega\setminus S\}} \mathbb{1}_{\{s_{\alpha,\beta-\pi}(x_1',x_2')_1>\alpha\cos(\beta)\}}$$
$$\times \mathbb{1}_{\{(x_1',x_2')\in S\}} d\mu$$
$$+ \int_{\Omega} \mathbb{1}_{\{s_{\alpha,\beta}(x_1',x_2')\in\Omega\setminus S\}} \mathbb{1}_{\{s_{\alpha,\beta}(x_1',x_2')_1<\alpha\cos(\beta)\}}$$
$$\times \mathbb{1}_{\{(x_1',x_2')\in S\}} d\mu$$
$$= \int_S \mathbb{1}_{\{s_{\alpha,\beta-\pi}(x_1',x_2')\notin S\}} \mathbb{1}_{\{s_{\alpha,\beta-\pi}(x_1',x_2')_1>\alpha\cos(\beta)\}} d\mu$$
$$+ \int_S \mathbb{1}_{\{s_{\alpha,\beta}(x_1',x_2')\notin S\}} \mathbb{1}_{\{s_{\alpha,\beta}(x_1',x_2')_1<\alpha\cos(\beta)\}} d\mu.$$

So then

$$I_S = \int_0^{\alpha_{max}} f_\alpha(\alpha) \frac{1}{2\pi} \int_0^{2\pi} \int_S \mathbb{1}_{\{s_{\alpha,\beta-\pi}(x_1',x_2')\notin S\}}$$
$$\times \mathbb{1}_{\{s_{\alpha,\beta-\pi}(x_1',x_2')_1>\alpha\cos(\beta)\}} d\mu$$
$$+ \int_S \mathbb{1}_{\{s_{\alpha,\beta}(x_1',x_2')\notin S\}} \mathbb{1}_{\{s_{\alpha,\beta}(x_1',x_2')_1<\alpha\cos(\beta)\}} d\mu d\beta d\alpha.$$

But since we are integrating β from 0 to 2π, replacing $\beta - \pi$ with β does not change the value of the integral in any way. Thus we have:

$$= \int_0^{\alpha_{max}} f_\alpha(\alpha) \frac{1}{2\pi} \int_0^{2\pi} \int_S \mathbb{1}_{\{s_{\alpha,\beta}(x_1',x_2')\notin S\}}$$
$$\times \mathbb{1}_{\{s_{\alpha,\beta}(x_1',x_2')_1>\alpha\cos(\beta)\}} d\mu$$
$$+ \int_S \mathbb{1}_{\{s_{\alpha,\beta}(x_1',x_2')\notin S\}} \mathbb{1}_{\{s_{\alpha,\beta}(x_1',x_2')_1<\alpha\cos(\beta)\}} d\mu d\beta d\alpha$$
$$= \int_0^{\alpha_{max}} f_\alpha(\alpha) \frac{1}{2\pi} \int_0^{2\pi} \int_S \mathbb{1}_{\{s_{\alpha,\beta}(x_1',x_2')\notin S\}} d\mu d\beta d\alpha$$
$$= \int_S \int_0^{\alpha_{max}} \int_0^{2\pi} \mathbb{1}_{\{s_{\alpha,\beta}(x_1',x_2')\notin S\}} d\beta d\alpha d\mu$$
$$= \int_S T_S d\mu$$
$$= O_S.$$

Thus $\Phi_S = I_S - O_S = 0$. Since flux is unchanged by rotating the region, this is true near any single, straight boundary. ■

D. Flux When Robots are Not Uniformly Distributed

Theorem 5. Let $S \subset \Omega \setminus H$. If there exists C such that $\rho(x_1, x_2) < C$ in S and $\rho(x_1, x_2) \geq C$ in $\Omega \setminus S$. Then $\Phi_S < 0$.

Proof

$$\begin{aligned} I_S &= \int_S N_S(x_1, x_2)\rho(x_1, x_2) d\mu \\ &< \int_S N_S(x_1, x_2) C d\mu \\ &< C \int_S N_S(x_1, x_2) d\mu \\ &< C \int_{\Omega \setminus S} T_S(x_1, x_2) d\mu \\ &< \int_{\Omega \setminus S} T_S(x_1, x_2)\rho(x_1, x_2) d\mu \\ &< O_S. \end{aligned}$$

Thus $\Phi_S < 0$. By the same logic, we have $\Phi_S > 0$ when the inequalities with respect to C and $\rho(x_1, x_2)$ are flipped. This shows that robots tend towards the target distribution because if there are any areas that are less populated than the rest of the region, we expect more robots to enter than to exit.

E. Conclusion

In conclusion, when the scalar field is uniform and we are examining regions far from the corners, Theorem 4 shows that our control law preserves the target distribution, and Theorem 5 shows that our control law tends towards the target distribution. Further, this analysis breaks down in the corners because $s_{\alpha,\beta}$ no longer has a piecewise differentiable inverse, which may be a clue as to why we see warping of the distribution in the corners. For any region S for which there exists $n < \infty$ such that robots with initial position in S cannot bounce more than n times in the next step, $s_{\alpha,\beta}$ has a piecewise differentiable inverse. Because a region arbitrarily close to the corner that does not include the corner has this property, the authors conjecture that there is a singularity at the corner points that causes the negative flux near the corners that we see in simulation. Future work includes performing analysis of flux on the corner regions and of flux over arbitrary scalar fields. One method to generalize this result to an arbitrary scalar field would be to use change of variables to prove that flux is 0 on any open ball. Using the analysis in this section, one can check that flux is a finitely additive set function with bounded total variation whose absolute value is bounded above by the $\max_\Omega\{\rho(x_1, x_2)\}\mu$, where μ is the Lebesque measure. Thus it is a signed measure, so proving 0-flux on an open ball does prove 0-flux everywhere.

VIII. Control Law Extensions

A. Collisions between Robots

1) Motivation: Last summer, the team assumed that the robots are small relative to the environment or that collisions between robots are unimportant and would not affect the distribution of the swarm, given its stochastic nature. They also considered the case in which collisions need to be avoided to prevent damage to the robots. According to their work in [1], if each robot were equipped with a rangefinder to sense the relative locations of other robots, additional repulsive force can be applied to robots in order to prevent collisions between neighboring robots without altering the diffusive nature of the swarm. However, they have found in [1] through simulation that the speed of convergence to the desired distribution will slow down when applying such a control law that prevents collisions, and the steady state error will increase, especially in regions where high density is expected.

Since our robots do not have localization capability, it is not implementable to prevent collision. Therefore, we need to find a control law for when robots collide with each other.

2) Method: We design our bounceback control law to deal with the situation of collisions between robots. A robot that collides with another robot will bounce back in the same way as they would when they hit the boundary, which is shown in Section VI. That is, both the robots reverse their direction and move opposite to the direction from which they were originally moving. First, we calculate the next positions of the robots and examine whether any robots are going to overlap with each other. If this is the case, we then apply our bounceback control law. Previously, each robot chose a random direction with the speed inversely proportional to the square root of the scalar field. During one time step, robots kept moving in one direction. After applying our bounceback control law, each robot uses this previous method to choose a direction and a speed. However, during one time step, robots spend part of time moving towards each other and spend the rest of time bouncing back, which means that the velocity vectors of robots are reversed after collision. The distance that a robot bounces back will not exceed that of moving towards each other in order to prevent hitting the boundary. We want to find whether the application of our bounceback control between robots will affect the diffusive nature of the swarm, which is measured by the error metric. In the simulation where the radius of robots is considered inconsequential, the initial positions of all the robots are the same, which is not implementable when taking the radius of robots into consideration. Also, the initial positions of the robots cannot scatter randomly in the scalar field since it contradicts the purpose of employing swarm robots, so we restrict the initial positions of robots to the domain ranging from 3.9 to 4.8 and the height ranging from 5.9 to 6.8. The time duration of simulation is 500 s.

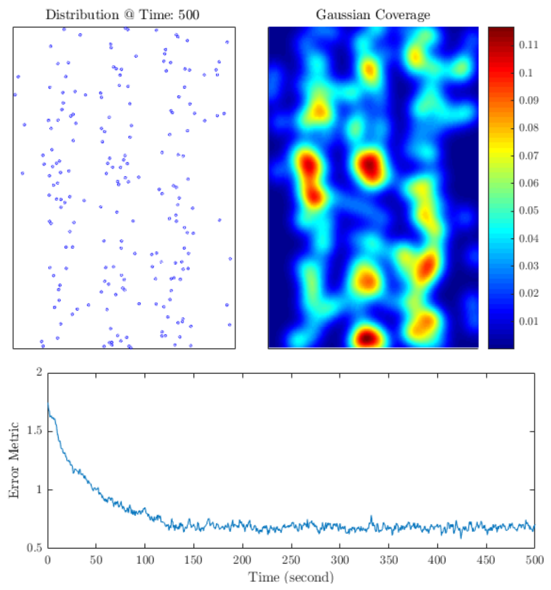


Fig. 17: Sample simulation result of the distribution of robots applying bounceback control law between robots

3) Simulation Result: Figure 17 shows the simulation result for rows pattern of the distribution of the number of robots $N = 200$ with the radius of robots $R = 0.025$, the size of Gaussian blob $\delta = 0.2$, time step $dt = 0.5$. Our simulation reaches steady state at $t = 349$ s. The maximum of the error metric e_N^δ is 1.7456 and the minimum of it reaches 0.5842.

We conjecture that the speed of convergence to the desired distribution will slow down when the radius of the robots R gets larger since applying the bounceback control law between robots will slow down the speed of movement in the area where the density of robots is high and robots are more likely to collide with each other when their radius becomes larger. We need simulations to further verify the assumption.

B. 3D Simulation

1) Motivation: Since we want to apply the control laws to crop pollination, it is sensible to consider scalar fields in 3D. We set up simulations with the target scalar field as the 3D row scalar field (see Figure 18) to test whether our control laws work in 3D.

2) Results: We set the scalar field to be 100 for flower region and 1 for the empty region. We ran 5 simulations and measured the steady state for each simulation. The average of number of steps to reach the steady state with 3D row scalar field is 380.2 with standard deviation of 43.69. We did similar simulation with a 2D scalar field, too. Specifically, we kept the 2D scalar field the same as the dissection of the 3D scalar field at $z = 0$. The robots start at the same position in 2D and 3D simulations from the top view. The average of number of steps to reach the steady state with 2D row scalar field is 621.8 with standard deviation of 87.84.

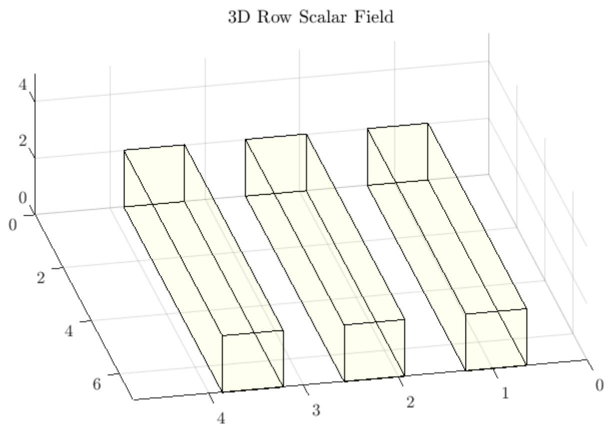


Fig. 18: The yellow semi-transparent surfaces outline the flower regions. The dimensions of this domain are $7 \times 4.8 \times 5$ and the dimensions of each flower row are $7 \times \frac{4.5}{7} \times 2$.

We conjecture that this discrepancy of how the time convergence of the error metric originates from difference of the percentages of the area/volume of the flower regions. For the 2D row scalar field, flower rows take up 40.2% of the total area. For 3D row scalar fields, flower rows take up only 16.1% of the domain. Since robots start at empty region for both cases of simulations, the larger the empty region is relative to the domain, the faster the robots can spread out according to the target scalar fields. This conjecture can explain the results from these five pairs of simulations but more simulations are needed to further verify it.

C. Time-Varying Scalar Field

1) Motivation: Another application for swarm robotics is surveillance or searching operations where robots follow a moving target. We simulate this with a time varying scalar field. In this case, the target scalar field is a bar that spans the height of the domain and moves sinusoidally across the domain.

2) Results: We observe that the distribution of robots follows the motion of the scalar field (Figure 19). Further work on calculating the error metric of a time-varying scalar field needs to be done.

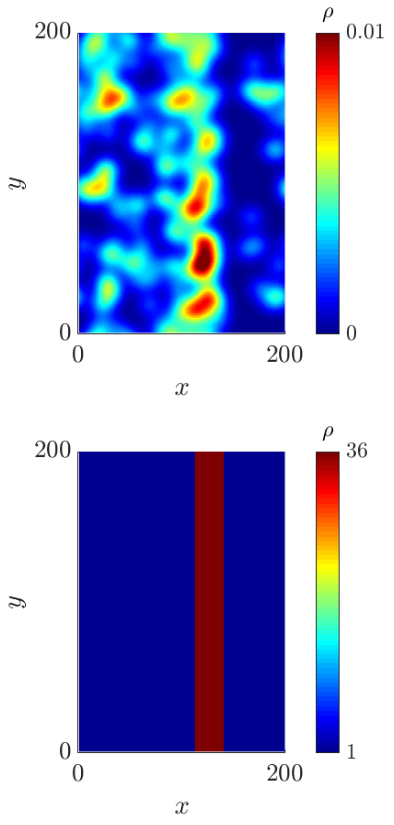


Fig. 19: Comparison of the Gaussian blob distribution (top) with the position of a time-varying scalar field (bottom) at a single time step.

IX. Physical Experiment

A. Introduction

This section focuses on the physical experiments we did. The reasons why we conducted experiments are that we try to verify the analytic and simulation results mentioned in Sections IV, VII, and VI as well as the robustness of our control laws. In this section, we will first introduce the camera and the robot we used and how we setup our experiments. Then we will discuss the challenges we encountered when we conducted the experiments. Lastly, we will analyze the experimental results.

B. Camera

We use a Microsoft Life Camera which has 720p resolution for videos. We fix it to the overhead pole over the test bed.

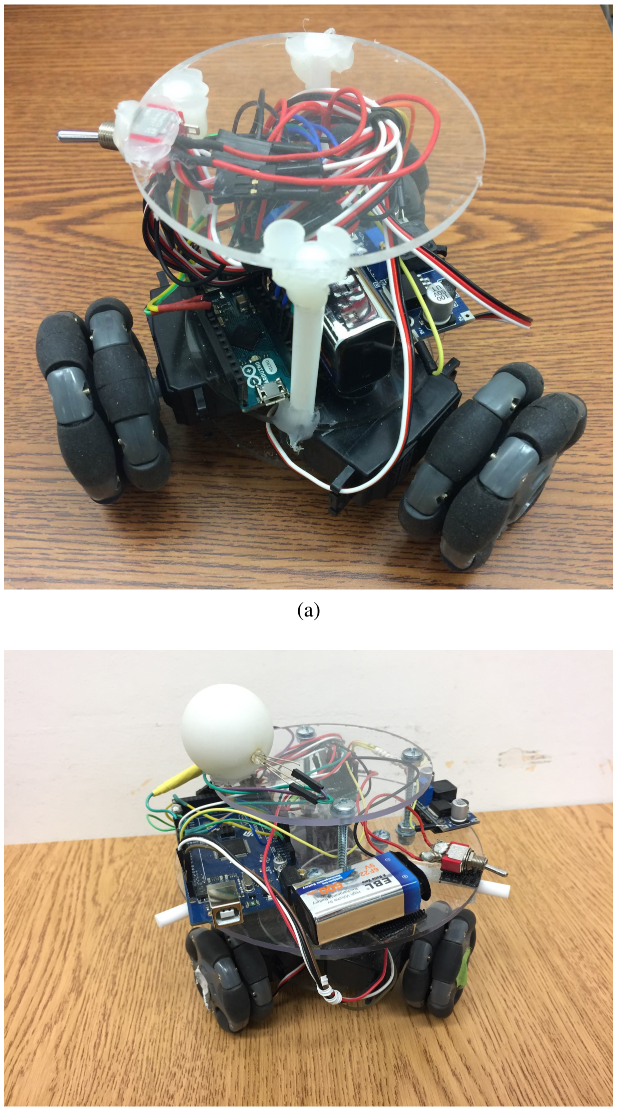


Fig. 20: (a) The Kiwi-drive robot used in Summer 2016 REU [1]. (b) The same robot with modifications used in summer 2017.

C. Robot

We use the Kiwi-drive robot (Figures 20a and 20b) made by students from last summer and made some changes to it. Students from last summer discussed wheels, the color sensor, the processor, the battery, and the overall structure of the robot in detail in their final report “Decentralized Stochastic Control of Robotic Swarm Density” Section 4.3 “Robots” [1].

Based on the design of the robot made from last year, John Cutone, a student researcher who is not in this summer’s REU program, added another transparent plastic panel to the robot to better stabilize everything on board. Students who continued this robotics research

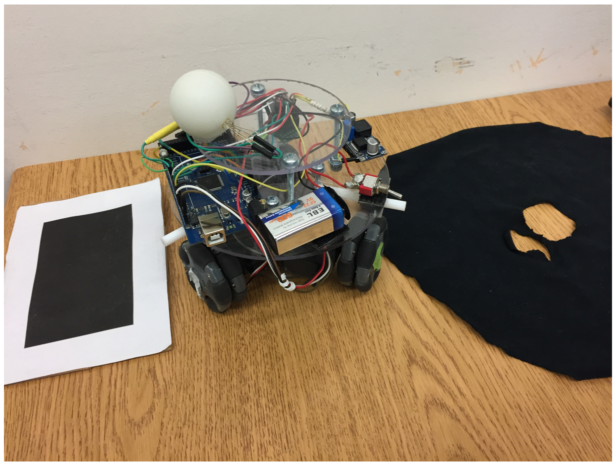


Fig. 21: The robot with the tag and the black skirt.

in 2016-17 academic year installed a white ping-pong ball on the top plastic panel and put a blue LED light inside the ping-pong ball. For our experiments, this LED light serves as a means for the robot to communicate with the camera (see details in Subsection IX-E). Since part of the battery and the microprocessor board is blue, it confuses the camera when the camera tries to capture the blue light from the LED. Therefore, John added a small black covering made with cloth (referred as "the black skirt" for the rest of the report) with holes (see Figure 21) in the center to the top of the robot to shade everything except the ping-pong ball. He also added three white rods, which are projected from the center of the robot beneath the bottom plastic panel, to expand and hold the black skirt. Since the camera needs to see the tag on top of the robot and the LED light inevitably blocks some part of the tag due to the space limit of the top panel, we raised the tag above the ping-pong ball by fixing a pillar on the top panel and fix the tag on top of the pillar.

D. *Experiment Setup*

We use row pattern printed by students from last summer as the test bed (Figure 1b). The dimension of the test bed is 70.1875 inches $\times$ 48.125 inches with each darker strip of the size 67.875 inches $\times$ 6.75 inches. (Note: the white margins on the two sides of the test bed do not have the same width and the width of one of the margin is larger than that of any of the black stripes.) We fix the test bed to the floor and overlay a transparent plastic sheet with red tape on the boundary on it. The initial position of the robot (the yellow circle on Figure 1b) is on the corner close to the shelf and away from the door facing the slurry group's experiment equipment. The coordinate is 8 inches away from the longer side of the test bed and 9.4 inches away from the shorter side to match the initial position of the simulations. We decide to run each experiment for 568 steps. We choose this number by running computer simulations with the same set of parameters and found out that, on average, it takes 200 robots 394 steps for their error metric to converge. We decide to let the robot run for another 174 steps after the error metric reaches the steady state. By some arguments involving Central Limit Theorem, 174 is the smallest number with which we can get a steady state error metric value very close to the true steady state error metric value by taking the average of the error metric for all steady state steps.

E. *Communication between the Robot and the Camera*

1) Tag Tracking: We adopt the method students from last used, which is mentioned in Section 4.2 of last year's final report [1], to track the tag and record the x,y coordinates of the robot. Besides the outputting the x, y, time data, we also generate videos of each experimental run and output it. We automate the process of starting and ending the recording. After the robot stands still for more than 200 consecutive frames, the recording on the camera/desktop computer end stops itself.

2) LED Light Capture: The robot keeps the blue light on when it is moving, i.e. from its first step to the last step. The camera starts recording when it detects the blue light from the LED light. Since the tag is above the LED light, the LED light might be shaded and it cannot be detected by the camera depending of the relative position between the robot and the camera. Therefore, before every experiment run, we put the robot on its initial position with the LED light facing the camera.

F. *Challenges and Solutions*

This subsection includes challenges we encountered when we ran physical experiments. We will also discuss the analysis and solutions, if available.

1) Recording before the Robot Starts Moving: Like we mentioned in the Subsection IX-E, we try to start the recording as soon as the robot begins its first step. However, we noticed that the camera sometimes picks up more blue pixels than the existing pixels in the designated region in the first few milliseconds after the camera is turned on. We tried to delay the camera's detection of the blue light but this solution only solved this problem partially.

2) Robot Stops Prematurely: We noticed that the robot stopped prematurely before it finished all of steps sometime. We confirmed that this problem is not caused by the test bed, the color sensor, servos, and any mechanical failures. In the end, we tweaked the end of wires on the robot to tighten the connection between the wires and

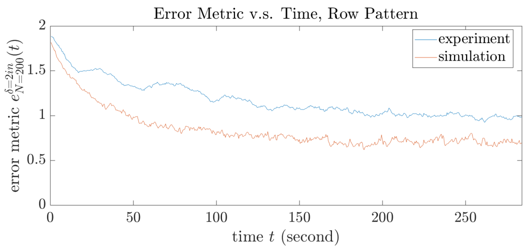


Fig. 22: The error metric e_N^δ of a simulation and experiments over time. The error metric corresponding to the experiments does not reach steady state whereas that corresponding to the simulations reaches steady state.

the female connector and the problem has not happened again since we made the fix.

3) Out of Bounds: The robot sometimes ran out of the boundary marked by the red tape instead of bouncing back as it should do. We believe that this problem is related to the fact that the robot rotates when it is moving. If the robot's previous position is very close to one of the boundaries and its new direction forms a very small angle with the closest the boundary, the robot might end up moving in a direction parallel to the boundary when it is on the boundary region (red tape). Then, after the robot bounces back, it moves to the opposite direction but the direction is still parallel. Depending on how the robot rotates while it is bouncing back, it could either go out of bound or come back to the test bed region. We have tried to find out why the robot rotates even though it is not programmed to do so but we could not figure out the causes. If the robot goes out of the boundary for any experiment run, we terminate the experiment and delete all of files related to that run.

G. Result

We finished 228 experiment runs. Ignoring some runs with too much invalid data, we have 200 runs of data. We overlap 200 runs' data as if 200 robots were used in one experiment run, calculate the error metric, and get the error metric curve in Figure 22. The error metric calculated with the experimental data is significantly greater than the one calculated with the simulation with the same set of parameters. Also, after applying way to measure the steady state discussed in Section III, we noticed that the experiment error metric did not converge.

When we ran the experiments, we noticed that the robot almost never went to the bottom left corner of the test bed (Figure 1b). This might partially explain why the experiment error metric is higher than the simulation error metric. We also suspect that the robot did not follow the control law and the boundary control law exactly. We have started investigating the experimental data to check whether the directions the robot travelled are randomly drawn from uniform distribution. From our preliminary analysis of the experimental data from 25 experiment runs, we found out that the directions are not uniformly distributed. Further analysis of the experimental data may include investigating the speed of the robots to verify that the speeds are normally distributed. It also remains to be verified that the speed of the robot scales correctly according to the target scalar field. That is, the average speed in the white regions of the test bed should be 6 times as much as the average speed in the black regions.

X. Deterministic Control Law

The diffusion of our robotic swarm can be modeled using the partial differential equation

$$u_t - \Delta\left(\frac{u}{F(\mathbf{x})}\right) = 0 \text{ in } \Omega \times (0,\infty), \tag{9}$$

$$\mathbf{n}\cdot\nabla\left(\frac{u}{F(\mathbf{x})}\right) = 0 \text{ on } \partial\Omega \times (0,\infty), \tag{10}$$

where $F(\mathbf{x})$ is the desired distribution. One method for approximating the solution to this PDE is to convolve a distribution of Dirac masses with a smooth mollifier function, ϕ_ε [2]. This gives

$$\rho_\varepsilon(\mathbf{x},t) = \phi_\varepsilon * \sum_{i=1}^N \delta(\mathbf{x}_i(t)-\mathbf{x})m_i.$$

Through the sifting property, this becomes

$$\rho_\varepsilon(\mathbf{x},t) = \sum_{i=1}^N \phi_\varepsilon(\mathbf{x}-\mathbf{x}_i(t))m_i, \tag{11}$$

which represents the superposition of N blobs as an approximation to u. In order to specify the trajectory of each blob, we let the blob position follow the ODE

$$\dot{\mathbf{x}}_i(t) = \mathbf{v}(\mathbf{x}_i(t),t), \tag{12}$$

where the velocity $\mathbf{v}$ comes from the gradient flow structure of the PDE. For (9) this becomes

$$v_k(\mathbf{x},t) = -a(\mathbf{x})\sum_{i=1}^N \frac{\partial_{x_k}\phi_\varepsilon(\mathbf{x}-\mathbf{x}_i(t))m_i}{\rho_\varepsilon(\mathbf{x}_i(t))} - a(\mathbf{x})\frac{\partial_{x_k}\rho_\varepsilon(\mathbf{x})}{\rho_\varepsilon(\mathbf{x})} - a_{x_k}(\mathbf{x}), \tag{13}$$

where $a(\mathbf{x}) = 1/F(\mathbf{x})$ and $\mathbf{v} = (v_1, v_2, \ldots, v_d)$. Hence, by representing each robot by one of the blobs, the swarm diffusion can be deterministically controlled according to these ODEs. Advantages to this method are natural adaptation to non-uniform scalar fields and high spatial dimensionality, as well as a suspected error convergence of $\mathcal{O}(N^{-1})$, as opposed to the slower

$\mathcal{O}(N^{-1/2})$ of the stochastic control law. However, the trajectory of each robot is now dependent on not only the local scalar field, but also its relative distance from the other $N-1$ robots. This requires the robots to have localization and communication capabilities. For large swarms, the velocity of a given robot may be dominated by only its nearest neighboring robots. Therefore, advanced implementations of this control law may enforce sparsity patterns where the influence between two robots is low, effectively lowering computation time.

A. *Implementing Boundary Conditions*

Although the solution defined by (11), (12), and (13) is claimed to converge to u as $\varepsilon \to 0$ and $N \to \infty$, there is no explicit implementation of the no-flux boundary condition (10). To approximate this boundary condition, the scalar field can be modified such that $F(\mathbf{x}) \approx 0$ for all $\mathbf{x} \notin \Omega$. For a strictly positive F, this results in a large potential around the boundary $\partial\Omega$ which prevents the blobs from exiting the domain. In the one-dimensional case, this modification is as simple as multiplying the scalar field by steep sigmoid functions at the boundaries.

B. *One-Dimensional Simulations*

To test the capabilities of this control law, we ran simulations for $N = 200$ robots over a bounded domain $x \in (-100, 100)$ with a non-uniform scalar field. The initial distribution is taken to be

$$u(x,0) = \begin{cases} 1 - \left(\frac{x}{50}\right)^2, & x \in (-50, 50), \\ 0, & x \notin (-50, 50). \end{cases}$$

The desired distribution is

$$F(x) = S(x)\left[\cos\left(\frac{x}{10}\right) + 3\right],$$

where S is the sigmoid function

$$S(x) = \frac{1}{1 + e^{-10(x+100)}} - \frac{1}{1 + e^{-10(x-100)}}.$$

Our choice of mollifier is a standard Gaussian;

$$\phi_\varepsilon(x) = \frac{1}{\varepsilon\sqrt{2\pi}} e^{-(x/\varepsilon)^2/2}.$$

The mollifier width ε is analogous to the robot radius δ used for the Gaussian blobs in the stochastic control law. To choose the mollifier width ε and the initial configuration of blobs, we resort to minimizing the L^1 norm of the difference between the initial condition and approximate solution;

$$e = \int_\Omega \mid u(x,0) - \rho_\varepsilon(x,0) \mid dx.$$

This minimization is carried out using MATLAB's `fmincon` optimization function. The resulting approximation is shown in Figure 23 with an optimal mollifier width of $\varepsilon = 2.2944$.

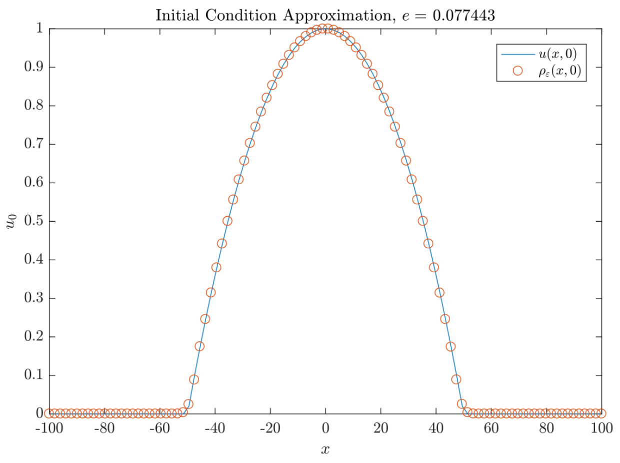


Fig. 23: Approximation of initial condition using `fmincon`.

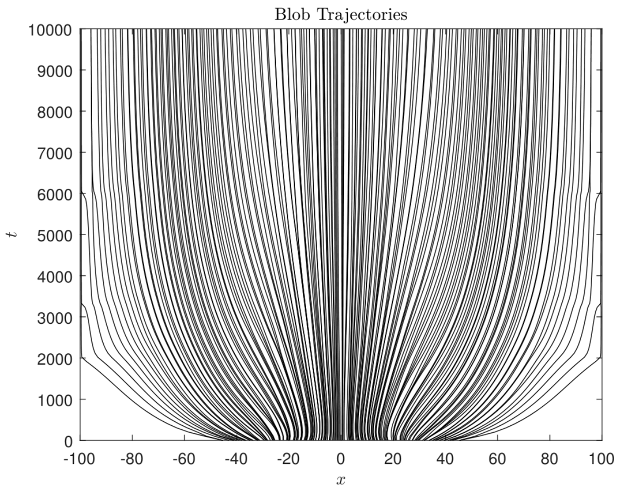


Fig. 24: Robot trajectories for $N = 200$.

Using the initial distribution of Figure 23, the simulation is carried out with constant mollifier width, and the robot trajectories and resulting distribution evolution are shown in Figures 24 and 25 respectively. To validate this method of control, the robot diffusion governed by (9) and (10) is also solved using the built-in MATLAB function `pdepe` and shown in Figure 26. The two PDE solutions show very close agreement, indicating the accuracy of this control law in representing diffusion at the macroscopic level.

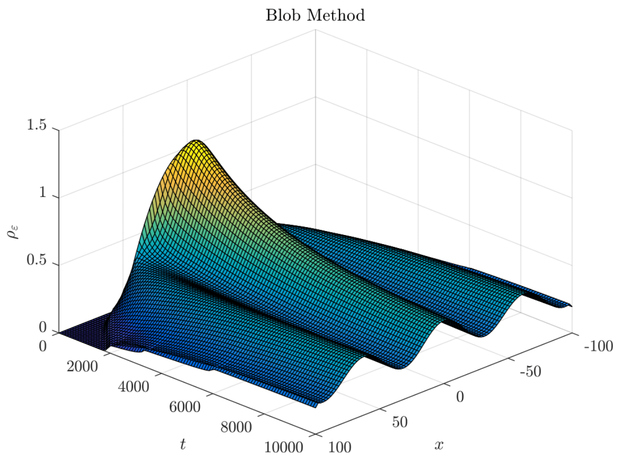


Fig. 25: Robot diffusion through time. Robot distribution converges toward the desired sinusoidal distribution.

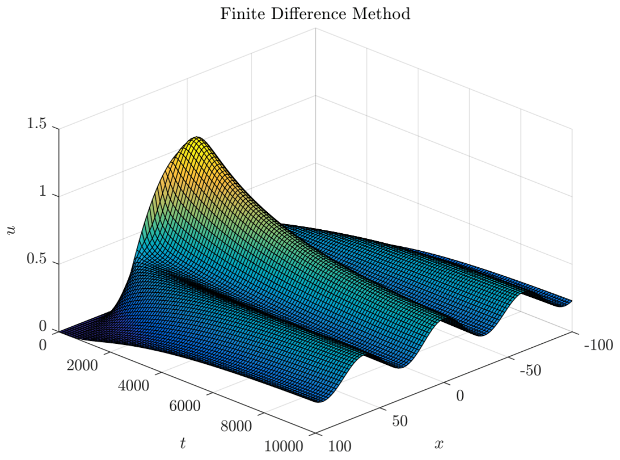


Fig. 26: Finite difference solution to robot diffusion equation.

XI. Conclusion

In conclusion, we have presented a robotic swarm algorithm that takes a swarm of robots with no communication or localization capabilities and distributes them according to a scalar field. While we found some warping of the target distribution in the corners of our regions, we have verified the robustness of our bouceback boundary control law and stochastic control law with simulation data for non-uniform 2D, 3D, and time-varying scalar fields, and compared our bounceback boundary control law to other possible control laws. Further, we showed that for regions far from corners, our control law preserves and tends towards the target distribution when the target distribution is uniform. We have found a reliable way to measure steady state on arbitrary scalar fields, and contextualized our steady state error metric values by approximating lower bounds on error. We also found the PDF error values when the robots are distributed according to the target distribution. Furthermore, we investigated the relationship between convergence of our error metric and convergence of the PDE solution to steady state. Lastly, we adapted Bertozzi and Craig's blob method [2] for solving our robot diffusion PDE, and developed a method for implementing no flux boundary conditions for this approach.

Acknowledgment

The authors would like to thank our incredibly supportive advisors, Olga Turanova and Matt Haberland. Thank you two for teaching us how to do research, creating a collaborative and productive work environment, and for your infinite patience. We also would like to thank Andrea Bertozzi, who created this team and guided us with her expertise. We would like to thank Hao Li for fruitful conversation and great paper suggestions. Lastly, we would like NSF Grant 1659676 for funding this project.

References

[1] S. Berman *et al.*, "Decentralized Stochastic Control of Robotic Swarm Density," UCLA, Los Angeles, CA, Aug. 2016.
[2] K. Craig, and A. Bertozzi. "A blob method for the aggregation equation," *Mathematics of Computation*, vol. 85, no. 300, pp. 16811717, 2016.
[3] K. Elamvazhuthi *et al.* "Coverage and Field Estimation on Bounded Domains by Diffusive Swarms," In *IEEE Conference on Decision and Control (CDC),* pp. 6300-6307. IEEE. 2016.
[4] E., Giné, *et al.* "The L_1 density estimator process," *The Annals of Probability,* vol. 31 pp. 719-768. 2003.
[5] B, Hansen. *Lecture Notes on Nonparametrics*. 2009. [Online]
[6] J. Hartinger and R. Kainhofer. "Non-Uniform Low-Discrepancy Sequence Generation and Integration of Singular Integrands," *Monte Carlo and Quasi-Monte Carlo Methods*, pp. 163-179, 2004.
[7] L. Horváth, "On L_p norms of multivariate density estimators," *Ann. Statist.* vol. 19, pp. 1933-1949. 1991.
[8] H., Li *et al.* "Decentralized Stochastic Control of Robotic Swarm Density: Theory, Simulation, and Experiment," *Proc. of International Conference on Intelligent Robots and Systems*, 2017.
[9] L.E. Payne and H. F. Weinberger. "An Optimal Poincaré Inequality for Convex Domains," *Archive for Rational Mechanics and Analysis*. **Vol. 5**, pp. 286-292, 1960.
[10] I. Sloan. "Integration and Approximation in High Dimensions," *Uncertainty Quantification*, Edinburgh, May 2010.
[11] P. Szymczak and A. J. C. Ladd, "Boundary conditions for stochastic solutions of the convection-diffusion equation," *Physical Review E,* 68(3), 2003.
[12] T. T. Tay *et al.*, "Off-line Controller Design," in *High performance control*, Birkhuser, ch. 4, pp. 93, 2012.