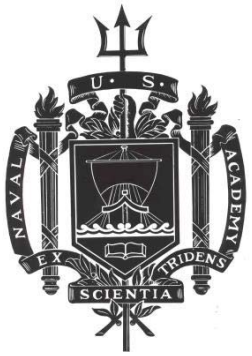
A TRIDENT SCHOLAR PROJECT REPORT

NO. 471

Stability of Nonlinear Swarms on Flat and Curved Surfaces

by

Midshipman 1/C Carl C. Kolon, USN



UNITED STATES NAVAL ACADEMY
ANNAPOLIS, MARYLAND

USNA-1531-2

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
05-21-18		

4. TITLE AND SUBTITLE
Stability of Nonlinear Swarms on Flat and Curved Surfaces

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Kolon, Carl C.

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Naval Academy
Annapolis, MD 21402

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
Trident Scholar Report no. 471 (2018)

12. DISTRIBUTION / AVAILABILITY STATEMENT

This document has been approved for public release; its distribution is UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Swarming is a near-universal phenomenon in nature. Many mathematical models of swarms exist, both to model natural processes and to control robotic agents. We study a swarm of agents with spring-like at-traction and nonlinear self-propulsion. Swarms of this type have been studied numerically, but to our knowledge, no proofs of stability yet exist. We are motivated by a desire to understand the system from a mathematical point of view. Previous numerical experiments have shown that the system either converges to a rotating circular limit cycle with a fixed center of mass, or the agents clump together and move along a straight line. We show that this is not always the case, and the behavior is sometimes more nuanced. Our specific goal is to investigate stability of the system's circular rotating state. The system is translation-invariant, and when the center of mass comes to a halt, the agents decouple from each other. We apply methods from the stability theory of dynamical systems, including Lienard's Theorem, Lasalle's Invariance Principle, and Lyapunov's direct and indirect methods, to globally characterize the behavior of these decoupled systems, and to locally characterize the desired behavior of the entire swarm. We confirm our theoretical findings with numerical experiments. So far, swarm models like this have only been studied in Euclidean, or flat, space. We extend a class of swarm models to curved geometries, or Riemannian Manifolds, using concepts from differential geometry. Through numerical simulation, we find that their behavior mimics the behavior of the same swarms in at space. We use this in two ways. We modify swarms to fit on embedded surfaces, creating swarms which move on spheres and in hyperbolic space. We also use it to modify the limit cycles of swarms in at space into desired shapes, like ovals. Finally, we use Gazebo, a high-fidelity robotics simulator, to simulate a robotic swarm following the same swarm model. We show that robotic agents display the behavior which we predict mathematically, indicating that it is feasible to control robot swarms using this method.

15. SUBJECT TERMS
swarming, differential geometry, dynamical systems, coupled differential equations

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**		53	**19b. TELEPHONE NUMBER** *(include area code)*

U.S.N.A. --- Trident Scholar project report; no. 471 (2018)

Stability of Nonlinear Swarms on Flat and Curved Surfaces

by

Midshipman 1/C Carl C. Kolon
United States Naval Academy
Annapolis, Maryland

_____
(signature)

Certification of Adviser(s) Approval

Associate Professor Kostya Medynets
Mathematics Department

_____
(signature)

_____
(date)

Professor William N. Traves
Mathematics Department

_____
(signature)

_____
(date)

Acceptance for the Trident Scholar Committee

Professor Maria J. Schroeder
Associate Director of Midshipman Research

_____
(signature)

_____
(date)

USNA-1531-2

Abstract

Swarming is a near-universal phenomenon in nature. Many mathematical models of swarms exist, both to model natural processes and to control robotic agents. We study a swarm of agents with spring-like attraction and nonlinear self-propulsion. Swarms of this type have been studied numerically, but to our knowledge, no proofs of stability yet exist. We are motivated by a desire to understand the system from a mathematical point of view. Previous numerical experiments have shown that the system either converges to a rotating circular limit cycle with a fixed center of mass, or the agents clump together and move along a straight line. We show that this is not always the case, and the behavior is sometimes more nuanced. Our specific goal is to investigate stability of the system's circular rotating state. The system is translation-invariant, and when the center of mass comes to a halt, the agents decouple from each other.

We apply methods from the stability theory of dynamical systems, including Liénard's Theorem, Lasalle's Invariance Principle, and Lyapunov's direct and indirect methods, to globally characterize the behavior of these decoupled systems, and to locally characterize the desired behavior of the entire swarm. We confirm our theoretical findings with numerical experiments.

So far, swarm models like this have only been studied in Euclidean, or flat, space. We extend a class of swarm models to curved geometries, or Riemannian Manifolds, using concepts from differential geometry. Through numerical simulation, we find that their behavior mimics the behavior of the same swarms in flat space. We use this in two ways. We modify swarms to fit on embedded surfaces, creating swarms which move on spheres and in hyperbolic space. We also use it to modify the limit cycles of swarms in flat space into desired shapes, like ovals.

Finally, we use Gazebo, a high-fidelity robotics simulator, to simulate a robotic swarm following the same swarm model. We show that robotic agents display the behavior which we predict mathematically, indicating that it is feasible to control robot swarms using this method.

Key words: Swarming, differential geometry, dynamical systems, coupled differential equations.

Contents

Acknowledgments

I would like to thank my adviser, Prof. Kostya Medynets, for the immense work that he has devoted to this project. I would also like to thank Prof. Irina Popovici, Prof. Alexis Alevras, and Prof. Geoffrey Price for their fruitful discussion and attention to this work. Finally, I would like to thank Dr. Ira Schwartz of Naval Research Laboratory for his guidance on the problem of interacting swarms.

1 Introduction

Swarming is a phenomenon commonly seen in nature [1, 2, 3]. Many models have been proposed for swarms, both Eulerian models (using continuous dynamics) and Lagrangian models (predicting the behavior of individual agents) [4]. More recently, mathematical models of swarm dynamics have been used to control robotic agents, especially by the United States military [5, 6]. With this application in mind, we choose to study Lagrangian models as a means to control each agent individually. Among Lagrangian models, the most commonly studied swarms are known as potential or gradient swarms: swarms which attempt to minimize a potential function by moving into a desirable configuration [7].

Our work is primarily concerned with the nonlinear parabolic potential model for a swarm, with self-propulsion. In the model, N agents with position vectors $\mathbf{r}_i$ obey the following equation of motion:

$$\ddot{\mathbf{r}}_i = (1 - \dot{\mathbf{r}}_i \cdot \dot{\mathbf{r}}_i)\dot{\mathbf{r}}_i - (\mathbf{r}_i - \mathbf{R}) \tag{1}$$

Where $\mathbf{R}$ represents the center of mass of the system.

$$\mathbf{R} = \frac{1}{N}\sum_{j=1}^{N} \mathbf{r}_j \tag{2}$$

This model has been studied numerically before [8, 9], but to our knowledge there does not yet exist a rigorous mathematical characterization of its limit behavior for any number of agents. The commonly observed limit behavior is pictured in Figure 1. We are motivated to characterize its behavior analytically, because we want to understand the system from a mathematical point of view.

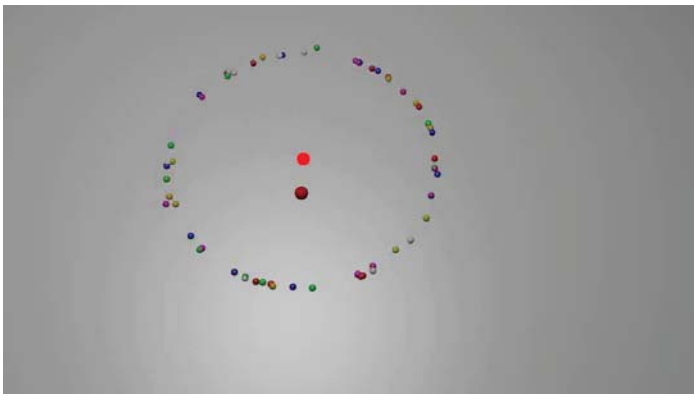


Figure 1: Limit behavior of the parabolic potential model, rendered in Blender. Each small colored dot represents an agent. The dull red dot is the original center of mass. The bright red dot is the current center of mass. The agents rotate in a circle of radius 1, at speed 1, around the center of mass. We believe that this behavior is locally attracting.

We note that Haraux and Jendoubi [10] investigated second-order systems

of the form:

$$\ddot{\mathbf{r}}_i = \mathbf{f}(\dot{\mathbf{r}}_i) - \frac{1}{N}\sum_{j=1}^{N} \nabla_{\mathbf{r}_i} U(\mathbf{r}_i, \mathbf{r}_j),$$

where $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ is a vector-valued function of the agent's velocity and $U : \mathbb{R}^n \to \mathbb{R}$ is a potential function which is smooth and radially unbounded.

A function f is called *negative-definite* if $f(x) < 0$ for $x \neq 0$, and $f(0) = 0$. If $\mathbf{f}(\dot{\mathbf{r}}_i) \cdot \dot{\mathbf{r}}_i$ is negative-definite, the system will converge to a limit configuration which minimizes U locally. We would like to stress that this negative-definiteness condition does not hold for Equation (1).

Previous researchers have observed in simulation that the system either converges to a rotating circular limit cycle with a fixed center of mass, or the agents clump together and move along a straight line [11]. We will show that the behavior is sometimes more nuanced (see Section 3.4). Our goal is to investigate stability of the system's circular rotating state. It is straightforward to show that the system is translation-invariant, and that, when the center of mass is fixed, the system decouples into N independent equations.

To understand the decoupled state, we first investigate the behavior of a simpler system, which we call the *decoupled parabolic potential model*. It contains one agent with position vector $\mathbf{r}$, which is attracted to the origin as though it is the center of mass. This yields the following equation of motion:

$$\ddot{\mathbf{r}} = (1 - \dot{\mathbf{r}} \cdot \dot{\mathbf{r}})\dot{\mathbf{r}} - \mathbf{r} \tag{3}$$

We prove the following about the simplified parabolic potential model:

Theorem 1. *Suppose $\mathbf{r} : \mathbb{R} \to \mathbb{R}^2$ is a solution of Equation (3).*

(i) If $\mathbf{r}$ and $\dot{\mathbf{r}}$ are parallel at some instant, they remain parallel and follow a unique limit cycle (Theorem 6).

(ii) The origin of the phase space is the unique equilibrium point of the system, and it is unstable (Theorem 7).

(iii) If $\mathbf{r}$ and $\dot{\mathbf{r}}$ are not parallel at some instant, then the system converges to a circular limit cycle of radius 1 with speed 1 centered on the origin (Theorem 8).

Theorem 1 globally characterizes the behavior of the decoupled model.

In the next result, we do local stability analysis for the original system with multiple agents.

Theorem 2. *Let $N = 2$ or 3. Circular limit cycles satisfying Equation (1) with stationary center of mass are locally stable up to translation (Theorem 9).*

We conjecture that this result holds for any number of agents, and discuss methods to prove it in Section 3.5.

Conjecture 1. *For any $N > 0$, circular limit cycles satisfying Equation (1) with stationary center of mass are locally stable up to translation.*

The other focus of our project is an extension of the parabolic potential model to Riemannian manifolds. The goal was to find a swarm model which would exhibit similar behavior to the parabolic model on the manifold: the limit

cycles would be circular with respect to the Riemannian metric, the speed of translation would be constant, and the agents would move along geodesics if there were no interaction. As far as we know, this has not been done before. In Section 5.1, we present such a model for any gradient swarm. Our observations through numerical simulation confirm that the behavior of a swarm in Euclidean space and the modified swarm on a manifold tend to be similar.

The structure of the paper is as follows. In Section 3, we define gradient swarms in Euclidean space, and prove some results on these swarms. Specifically, in Section 3.3, we characterize the global behavior of a simplified, decoupled system with one agent by constructing an *explicit Lyapunov function* for the system and using Lasalle's invariance principle. In Section 3.4, we prove that the "typical" limit behavior of a rotating state is locally stable for a system of two or three agents using Lyapunov's indirect (or linearization) method. In Section 4, we numerically investigate stability of more complicated Morse swarms when subject to collisions and delay. In Section 5, we use ideas from differential geometry to generate gradient swarms on Riemannian manifolds. We present a swarm model whose limit cycles are precisely the circles (the set of points equidistant from their common center) on the manifold. Finally, in Section 6, we discuss our technical implementation of our ideas, in mathematics programs and robotic simulation.

2 Background

For our purposes, a *dynamical system* is a system in which functions describe the time-dependent location of agents in space. Dynamical systems are used to study mathematical models of moving objects. As such, they are often used to describe swarms. We study continuous dynamical systems, in which the agents are represented as moving dots on a plane.

In particular, we study *first-order* and *second-order* systems. In a first-order system, a function controls the velocity of an agent. In a second-order system, a function controls the acceleration of the agent. Many systems in nature are second-order, since force (and therefore acceleration) is often controlled. An example is the motion of the planets, which have acceleration proportional to the gravitational force on them. Under some conditions, second-order systems can be approximated by first-order ones, which are often much simpler [10, 4]. For an agent with position vector x, we represent the velocity as $\dot{x}$ and the acceleration as $\ddot{x}$; each dot represents a derivative with respect to time.

The systems we study are also *autonomous*. This means that the functions which control the system depend only on x and its derivatives, and not on time itself. In a first-order autonomous system, $\dot{x}$ is purely a function of the agent's position. In a second-order autonomous system, $\ddot{x}$ is a function of x and $\dot{x}$. These systems are also known as *time-invariant*, because the velocity and acceleration of the agents do not depend on time. Finally, the systems are *homogeneous*: all agents have the same equation of motion.

We define a *swarm* as a dynamical system of coupled agents. Agents are *coupled* if their equations of motion depend on the positions of other agents. Agents in a swarm each have a position vector $\mathbf{r}_i$, and some equation of motion. Since we study homogeneous swarms, we can represent the entire system by giving the equation of motion of one agent. Our representation of these systems

can therefore be written as follows:

$$\dot{\mathbf{r}}_i = f(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N) \qquad \text{(a first-order system)}$$
$$\ddot{\mathbf{r}}_i = f(\dot{\mathbf{r}}_1, \dot{\mathbf{r}}_2, \ldots, \dot{\mathbf{r}}_N, \mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N) \qquad \text{(a second-order system)}$$

We define an *equilibrium configuration* of a swarm of agents with positions $\mathbf{r}_i$ as a list of position vectors $(\bar{\mathbf{r}}_1, \bar{\mathbf{r}}_2, \ldots, \bar{\mathbf{r}}_N)$ such that if. at time t, $\mathbf{r}_i = \bar{\mathbf{r}}_i$ and $\dot{\mathbf{r}}_i = 0$ for all i, the agents will remain at these positions for all time.

2.1 Manifolds and Curved Space

We are interested in extensions of swarm models to curved spaces for several reasons. In nature, animals may find themselves constrained to certain surfaces. For example, microorganisms living near hydrothermal vents may need to stay on the surface of a sphere, to remain both safe from heat and close enough for photosynthesis [12]. Man-made robot swarms may also find themselves constrained to surfaces. Swarms of robots are being considered for applications to space [13], in which they may need to move on the surface of a sphere (for example, orbits of equal height). In a military context, swarms of robots may need to avoid anti-aircraft fire, which may be easier if they are constrained to a surface of negative curvature (like a psuedosphere, which exhibits hyperbolic geometry).

Mathematically, the concept of a surface is formalized as a *Riemannian manifold*. A Riemannian manifold is a surface that is locally flat. Informally, a resident of a Riemannian manifold might believe that he lived on a plane, if he was small enough. The earth is an example of a Riemannian manifold, since at first glance it appears that we live on a plane.

Riemannian manifolds allow us to perform vector operations on their surface, and therefore calculate distances and angles, using an operation called the *inner product*. We may also define coordinate systems on Riemannian manifolds, called *curvilinear coordinates*, which allow us to calculate these quantities.

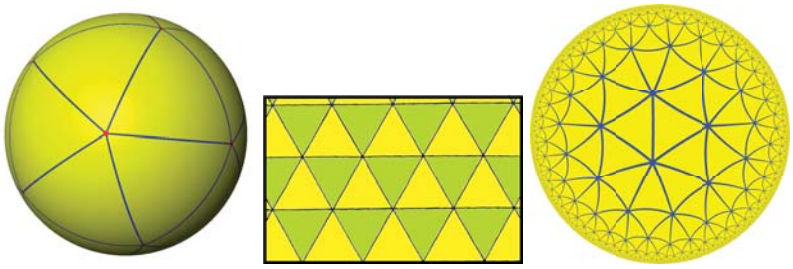


Figure 2: Examples of Riemannian manifolds, each tiled by triangles. From left to right, the sphere, the Euclidean plane, and the Poincaré disk representation of the hyperbolic plane (images from Wikipedia).

On Riemannian manifolds, the shortest distance between two points is not necessarily a straight line in the curvilinear coordinates. Instead, the concept of a straight line is generalized as a *geodesic*: a locally length-minimizing curve. For any two points on a Riemannian manifold M, the shortest distance between them is a geodesic. On the earth, a geodesic is a great circle. Great circles are

Figure 3: An example of a geodesic on the earth, showing a flight path from Beijing to Baltimore.

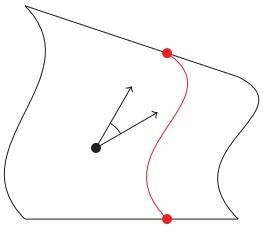


Figure 4: Two vectors on a Riemannian manifold. Their angle and lengths can be calculated using the inner product. A geodesic is shown in red.

used for navigation, since traveling along them yields the shortest path between two points (see Figure 3).

2.2 Lyapunov Stability Theory

Much of our work relies on Lyapunov stability theory. Lyapunov stability theory was developed in the late 19th Century, in order to characterize the behavior of systems with no exact solution [14, 15, 16]. In particular, we use Lasalle's invariance principle to characterize the behavior of the single-agent system, and Lyapunov's indirect (linearization) method to characterize the local behavior of multi-agent systems.

Lyapunov stability theory uses the concept of a *Lyapunov function*, analogous to the energy of a system. If we can show that the system has decreasing energy, and that this energy approaches a certain value, Lyapunov stability theory tells us that the entire system will approach the corresponding behavior.

Lyapunov functions can be used to show stability of points in nonlinear systems. An example is a damped swinging pendulum. The angle of the pendulum

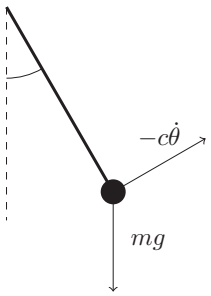


Figure 5: Diagram of a damped pendulum.

to the vertical, θ, can be represented by the dynamical system:

$$\ddot{\theta} = -c\dot{\theta} - \frac{g}{l}\sin\theta$$

Where g is the acceleration of gravity, l is the length of the pendulum's rod, and c is a positive damping coefficient. We can use a Lyapunov function gained from the energy of the system:

$$L = \frac{1}{2}l\dot{\theta}^2 + g(1 - \cos\theta)$$

Differentiating L with respect to t, we have:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}L &= l\ddot{\theta}\dot{\theta} + g\dot{\theta}\sin\theta \\ &= l\dot{\theta}(-c\dot{\theta} - \frac{g}{l}\sin\theta) + g\dot{\theta}\sin\theta \\ &= l\dot{\theta}(-c\dot{\theta}) = -cl\dot{\theta}^2\end{aligned}$$

So the derivative of L is negative-definite, meaning that L composes to a minimum. L is minimized at the point $\theta = 0, \dot{\theta} = 0$, so the system will converge to be motionless at the bottom of its swing.

The theorems that follow are all variations on this concept.

Theorem 3 (Lasalle)**.** *Let an autonomous dynamical system be defined on a region $D \subset \mathbb{R}^n$, with the equation of motion:*

$$\dot{\mathbf{x}} = f(\mathbf{x}) \tag{4}$$

Where $f : D \to \mathbb{R}^n$ is a locally Lipschitz map. Let $\Omega \in D$ be a compact set that is positively invariant with respect to (4) - in other words, a solution $\mathbf{x}(t)$ of (4) with $\mathbf{x}(0) \in \Omega$ satisfies $\mathbf{x}(t) \in \Omega$ for all $t > 0$. Let $V : D \to \mathbb{R}$ be a continuously differentiable function such that $\dot{V}(\mathbf{x}) \leq 0$ for all trajectories in Ω. Let E be the set of all points in Ω where $\dot{V}(\mathbf{x}) = 0$. Let M be the largest invariant set in E. Then every solution starting in Ω approaches M as $t \to \infty$.

Proof. This proof is adapted from a proof published in *Nonlinear Systems*, by Khalil [16].

Let $\mathbf{x}(t)$ be a solution of (4) starting in Ω. $\dot{V}(\mathbf{x}) \leq 0$ in Ω, and trajectories beginning in Ω will remain within Ω for all time, since Ω is invariant. Therefore,

$V(\mathbf{x}(t))$ is a decreasing function of t. $V(\mathbf{x})$ is continuous on the compact set Ω, so it is bounded from below on Ω. Therefore, $V(\mathbf{x}(t))$ has a limit a as $t \to \infty$. Ω is compact, so it is closed, so the positive limit set L^+ is in Ω. For any $\mathbf{p}$ in L^+, there is a sequence t_n such that $t_n \to \infty$ and $\mathbf{x}(t_n) \to \mathbf{p}$ as $n \to \infty$. By continuity of $V(\mathbf{x})$, $V(\mathbf{p}) = \lim_{n\to\infty} V(\mathbf{x}(t_n)) = a$. Therefore, $V(\mathbf{x}) = a$ on L^+. Thus:

$$L^+ \subset M \subset E \subset \Omega$$

Since $\mathbf{x}(t)$ is bounded, $\mathbf{x}(t)$ approaches (becomes arbitrarily close to a member of) L^+ as $t \to \infty$. Therefore, $\mathbf{x}(t)$ approaches M as $t \to \infty$. □

The idea behind Theorem 3 is that, if there exists a function V which decreases (although not strictly) along trajectories of the system, and V is bounded below with bounded sublevel sets, eventually V must converge, and so the system must converge to an invariant state where $\dot{V} = 0$. Informally, we may think of V as a sort of "energy function" of the system. Indeed, energy is often used as this function in damped physical systems.

We also rely on the following theorems about linearized systems:

Theorem 4. *For the linear system $\dot{x} = Ax$, the equilibrium point $x = 0$ is stable if and only if all eigenvalues of A satisfy $\mathrm{Re}(\lambda_i) \leq 0$ and for every eigenvalue with $\mathrm{Re}(\lambda_i) = 0$ and algebraic multiplicity $q_i \geq 2$, $\mathrm{rank}(A - \lambda_i I) = n - q_i$, where n is the dimension of x. The equilibrium point $x = 0$ is (globally) asymptotically stable if and only if all eigenvalues of A satisfy $\mathrm{Re}(\lambda_i) < 0$.*

For a proof, see Khalil [16], Theorem 4.5.

Theorem 5 (Lyapunov's indirect (linearization) method)**.** *Let $x = 0$ be an equilibrium point of the nonlinear system*

$$\dot{x} = f(x)$$

where $f : D \to \mathbb{R}^n$ is continuously differentiable and D is a neighborhood of the origin. Let

$$A = \left.\frac{\partial f}{\partial x}(x)\right|_{x=0}$$

Then:

1. *The origin is asymptotically stable if $\mathrm{Re}(\lambda_i) < 0$ for all eigenvalues of A.*
2. *The origin is unstable if $\mathrm{Re}(\lambda_i) > 0$ for one or more eigenvalues of A.*

For a proof, see Khalil [16], Theorem 4.7.

Theorems 4 and 5 allow us to automatically find an energy function for linear and nonlinear systems respectively, but in nonlinear systems, this only works close to equilibrium points. These theorems are useful because they allow us to automatically characterize the behavior of a nonlinear system near a certain configuration, but they are limited because they tell us nothing about the global behavior of a nonlinear system.

3 Gradient Swarm Models

We study a class of swarm models called *gradient swarm models*, which have been studied extensively [4, 7, 8, 17, 18, 19]. This section lays out the theory behind these models, and the advances that we have made in the parabolic potential model, which is itself a gradient swarm model.

For some functions $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ and $U : \mathbb{R}^n \to \mathbb{R}$, Haraux and Jendoubi [10] define first- and second-order gradient systems respectively as follows:

$$\dot{\mathbf{x}} = -\nabla U(\mathbf{x}) \tag{5}$$

$$\ddot{\mathbf{x}} = -\mathbf{f}(\mathbf{x}) - \nabla U(\mathbf{x}) \tag{6}$$

Let a swarm of N agents in $\mathbb{R}^n$ be given with position vectors $\mathbf{r}_i$. We define a potential function is a function $U : \mathbb{R}^{2n} \to \mathbb{R}$. Examples of commonly-studied potential functions are:

$$U(\mathbf{r}_i, \mathbf{r}_j) = \|\mathbf{r}_i - \mathbf{r}_j\|^2 \qquad \text{(parabolic potential)}$$

$$U(\mathbf{r}_i, \mathbf{r}_j) = \frac{10}{9} \exp\left(-\frac{4\|\mathbf{r}_i - \mathbf{r}_j\|}{3}\right) - \exp(-\|\mathbf{r}_i - \mathbf{r}_j\|) \qquad \text{(Morse potential)}$$

In a gradient swarm, the parabolic potential will produce attraction to the center of mass. The Morse potential will produce pairwise attraction at long distances, and repulsion at short distances.

We work with potential functions satisfying several criteria:

1. U is smooth (infinitely differentiable).
2. U is bounded below.
3. All sublevel sets of U are bounded (sublevel sets are sets of the form $\{x : U(x) < k\}$ for some $k \in \mathbb{R}$).

We define the swarm potential, S, as the sum of the potential functions for every pair of agents:

$$S(\mathbf{r}_1, \mathbf{r}_2, \ldots \mathbf{r}_N) = \sum_{\substack{i=1 \\ j=1 \\ i \neq j}}^{N} U(\mathbf{r}_i, \mathbf{r}_j)$$

3.1 First-Order Gradient Swarms

A first-order gradient swarm is a dynamical system of N agents in n dimensions satisfying the following equation of motion:

$$\dot{\mathbf{r}}_i = -\frac{1}{N} \sum_{j=1}^{N} \nabla_{\mathbf{r}_i} U(\mathbf{r}_i, \mathbf{r}_j)$$

Where $\nabla_{\mathbf{r}_i}$ signifies taking the gradient with respect to $\mathbf{r}_i$.

Mogilner et al. [4] showed that one-dimensional first-order gradient swarms always converge to an equilibrium configuration if U satisfies the properties of a potential function, and produced a way to prove this in arbitrarily many

dimensions. Alternatively, we may view the positions of all agents as one large vector $\mathbf{R}$, and the swarm as a large gradient system of the form:

$$\dot{\mathbf{R}} = -\nabla S(\mathbf{R})$$

It is well-known that systems like this converge to equilibrium points if S is bounded below, and all sublevel sets of S are bounded.

3.2 Second-Order Gradient Swarms

A second-order gradient swarm is a dynamical system of N agents in n dimensions satisfying the following equation of motion:

$$\ddot{\mathbf{r}}_i = \mathbf{f}(\dot{\mathbf{r}}_i) - \frac{1}{N}\sum_{j=1}^{N} \nabla_{\mathbf{r}_i} U(\mathbf{r}_i, \mathbf{r}_j)$$

Where $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ is a vector-valued function of the agent's velocity.

In the case where $\mathbf{f}(\dot{\mathbf{r}}_i)\cdot\dot{\mathbf{r}}_i$ is negative-definite, a second-order gradient swarm will converge to a limit configuration. Furthermore, the set of limit configurations is equal to the set of limit configurations of the first-order system with the same potential function.

Mogilner et al. [4] argued that this is true because such second-order gradient systems are similar to first-order systems at low speeds. We found that these swarms are high-dimensional cases of the second-order gradient systems studied by Haraux and Jendoubi [10]. Their work gives us a Lyapunov function for the system:

$$L = S + \sum_{i=1}^{N} \frac{1}{2}(\dot{\mathbf{r}}_i \cdot \dot{\mathbf{r}}_i)$$

This is minimized when S is minimized and the agents are motionless, therefore the system will converge to minima of S.

If $\mathbf{f}(\dot{\mathbf{r}}_i) \cdot \dot{\mathbf{r}}_i$ is not negative-definite, second-order gradient swarms do not necessarily converge to equilibrium configurations. An example of a gradient swarm which does not necessarily converge is the nonlinear parabolic potential model:

$$\ddot{\mathbf{r}}_i = (1 - \dot{\mathbf{r}}_i \cdot \dot{\mathbf{r}}_i)\dot{\mathbf{r}}_i - \frac{1}{N}\sum_{j=1}^{N} \nabla_{\mathbf{r}_i} \|\mathbf{r}_i - \mathbf{r}_j\|^2 \tag{7}$$

This system is equal to the parabolic potential model in Equation (1), and is our main topic of study.

We have observed through numerical simulation that typically (though not always) the center of mass of the system converges to an equilibrium point, and all agents converge to circular limit cycles around the point (pictured in Figure 7). While the system has been examined (e.g. by Ebeling and Schweitzer [8]), to our knowledge, there exist no proofs of stability concerning this limit behavior.

In order to show that this limit behavior is stable, we first note that often the system consists of agents rotating in both directions: clockwise and counterclockwise. These systems can be split in two: the agents rotating one way can be decoupled from the agents rotating the opposite way. Because of this, we attempt to show that rotating behavior with all agents rotating in one direction is locally stable.

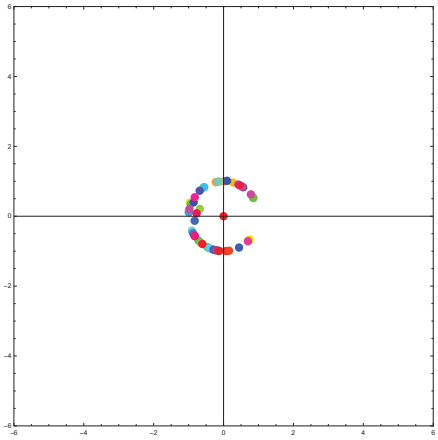


Figure 6: Limit behavior of the parabolic potential model. Agents rotate both ways, but we can decouple the system into all agents rotating in one direction, and all agents rotating in the opposite direction. Both groups should have a stationary center of mass.

3.3 Simplified Parabolic Model

In order to understand the swarm, we construct a simplified model with one agent centered on the origin. The system's behavior is given by the following equation of motion:

$$\ddot{\mathbf{r}} = (1 - \dot{\mathbf{r}} \cdot \dot{\mathbf{r}})\dot{\mathbf{r}} - \mathbf{r} \tag{8}$$

Theorem 6 (Original)**.** *If* $\mathbf{r}$ *and* $\dot{\mathbf{r}}$ *are parallel at some instant, then they remain parallel and follow a unique limit cycle.*

Proof. First, suppose that $\mathbf{r}$ and $\dot{\mathbf{r}}$ are parallel, i.e. for some unit vector $\hat{\mathbf{r}}$, we have:

$$\mathbf{r} = a\hat{\mathbf{r}}$$
$$\dot{\mathbf{r}} = b\hat{\mathbf{r}}$$

Then (8) yields:

$$\ddot{\mathbf{r}} = (b - b^3 - a)\hat{\mathbf{r}}$$

The acceleration is parallel to $\hat{\mathbf{r}}$, so the entire system is constrained to the line $k\hat{\mathbf{r}}$. We can rewrite differential equations for a and b:

$$\begin{aligned} \dot{a} &= b \\ \dot{b} &= b - b^3 - a \end{aligned} \tag{9}$$

Liénard theory has been used before to investigate coupled systems [20], so we were motivated to use it to understand this one. We may rewrite this system as a Liénard system, using the substitution:

$$x = b$$
$$y = -a$$
$$f(x) = x^3 - x$$

Then (9) becomes:

$$\begin{aligned} \dot{x} &= y - f(x) \\ \dot{y} &= -x \end{aligned} \tag{10}$$

Liénard [21] investigated this type of system. This system is a special case studied by Lins, Melo, and Pugh [22]. It has a unique stable limit cycle if:

1. f is continuous,
2. f is odd,
3. f has a unique positive root at $x = k$, and
4. f is monotone increasing for $x > k$.

f satisfies all four properties with $k = 1$, so a unique stable limit cycle exists if $\mathbf{r}$ and $\dot{\mathbf{r}}$ are parallel. □

Theorem 7 (Original)**.** *$a = 0$, $b = 0$ is the unique equilibrium point of the system in Equation (refonedimension), and it is unstable.*

Proof. Suppose $\dot{a} = \dot{b} = 0$. Then $b = 0$, and $b - b^3 - a = -a = 0$. So $(0, 0)$ is the unique equilibrium point.

To see that it is unstable, use Lyapunov's indirect method [14]. The Jacobian of the system at $(0, 0)$ is:

$$\mathbf{J} = \begin{pmatrix} 0 & 1 \\ -1 & 1 \end{pmatrix}$$

Which has eigenvalues $\frac{1}{2} \pm \frac{\sqrt{3}}{2}i$. Since the real parts of all eigenvalues are positive, the equilibrium point is unstable. □

Theorem 8 (Original)**.** *Suppose that $\mathbf{r}$ and $\dot{\mathbf{r}}$ are not parallel. Then the system converges to a circular limit cycle of radius 1 with speed 1 centered on the origin.*

Proof. We define scalars u, v and w using the following substitution:

$$\begin{aligned} u &= \mathbf{r} \cdot \mathbf{r} \\ v &= \dot{\mathbf{r}} \cdot \dot{\mathbf{r}} \\ w &= \dot{\mathbf{r}} \cdot \mathbf{r} \end{aligned} \tag{11}$$

Differentiating each scalar, we have:

$$\begin{aligned} \dot{u} &= 2\dot{\mathbf{r}} \cdot \mathbf{r} & &= 2w \\ \dot{v} &= 2\ddot{\mathbf{r}} \cdot \dot{\mathbf{r}} = 2(1 - \dot{\mathbf{r}} \cdot \dot{\mathbf{r}})\dot{\mathbf{r}} \cdot \dot{\mathbf{r}} - 2\mathbf{r} \cdot \dot{\mathbf{r}} & &= 2v(1 - v) - 2w \\ \dot{w} &= \ddot{\mathbf{r}} \cdot \mathbf{r} + \dot{\mathbf{r}} \cdot \dot{\mathbf{r}} = (1 - \dot{\mathbf{r}} \cdot \dot{\mathbf{r}})\dot{\mathbf{r}} \cdot \mathbf{r} - \mathbf{r} \cdot \mathbf{r} + \dot{\mathbf{r}} \cdot \dot{\mathbf{r}} & &= w(1 - v) - u + v \end{aligned} \tag{12}$$

This system is also subject to the following constraints:

$$\begin{aligned} u &\geq 0 \quad \text{because it is the squared norm of a vector.} \\ v &\geq 0 \quad \text{because it is the squared norm of a vector.} \\ w^2 &\leq uv \quad \text{by the Cauchy-Schwarz Inequality.} \end{aligned} \tag{13}$$

In the case where $w^2 = uv$, we have $(\dot{\mathbf{r}} \cdot \mathbf{r})^2 = (\mathbf{r} \cdot \mathbf{r})(\dot{\mathbf{r}} \cdot \dot{\mathbf{r}})$, so the two vectors are parallel. In this case, the system reduces to the one-dimensional case. If $u = 0$ or $v = 0$, it must be the case that $w^2 = uv$, so we also have the one-dimensional case. If we have $u = v = 0$, we have the unstable equilibrium point in the one-dimensional case. Therefore, we are only interested in the behavior within the following region, which we call Ω:

$$\begin{aligned} u &> 0 \\ v &> 0 \\ w^2 &< uv \end{aligned} \tag{14}$$

Strogatz [15] remarks on the difficulty of finding Lyapunov functions:

> Unfortunately, there is no systematic way to construct Liapunov functions. Divine inspiration is usually required, although sometimes one can work backwards.

Despite this, we were able to find a function which proves stability of the decoupled system. We define this function $L : \Omega \to \mathbb{R}$ as:

$$L = u + v - \log(uv - w^2)$$

Since $uv - w^2 > 0$ in Ω, L is defined throughout Ω. It is worth noting that L does not correspond to any actual physical quantity. It is merely a function that has the properties we require. We will show that L satisfies all requirements of Lasalle's Theorem (3).

L is radially unbounded, i.e. $L \to \infty$ as $|(u, v, w)| \to \infty$. To see this, note that:

$$L = u + v - \log(uv - w^2) \geq (u - \log(u)) + (v - \log(v))$$

Since $u - \log(u)$ and $v - \log(v)$ are unbounded above, L is radially unbounded. Furthermore, L approaches infinity on the boundaries of Ω. For some fixed values of u and v, as $w^2 \to uv$ from below, $\log(uv - w^2) \to -\infty$, so $L \to \infty$. Together, this means that all sublevel sets of L are bounded.

For some constant k, we denote the sublevel set $\{(u, v, w) : L(u, v, w) \leq k\}$ by $\Omega(k)$. Since L is continuous, and all sets $L \leq k$ are closed, all $\Omega(k)$ are closed. Therefore, all $\Omega(k)$ are compact.

L has one stationary point in Ω, at $(1, 1, 0)$. To show this, take the gradient, and set it equal to zero:

$$\nabla L = \begin{pmatrix} 1 - \frac{v}{uv - w^2} \\ 1 - \frac{u}{uv - w^2} \\ \frac{2w}{uv - w^2} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

This yields:

$$\begin{aligned} v &= uv - w^2 & & & (1 - u)v &= 0 \\ u &= uv - w^2 & \Rightarrow & & (1 - v)u &= 0 \\ w &= 0 & & & w &= 0 \end{aligned} \tag{15}$$

Since u and v are positive in Ω, the only solution is $(u, v, w) = (1, 1, 0)$. In addition, $(1, 1, 0)$ is a minimum. To show this, we compute the Hessian of L:

$$\mathbf{H} = \begin{pmatrix} \frac{v^2}{(uv-w^2)^2} & \frac{uv}{(uv-w^2)^2} - \frac{1}{uv-w^2} & -\frac{2vw}{(uv-w^2)^2} \\ \frac{uv}{(uv-w^2)^2} - \frac{1}{uv-w^2} & \frac{u^2}{(uv-w^2)^2} & -\frac{2uw}{(uv-w^2)^2} \\ -\frac{2vw}{(uv-w^2)^2} & -\frac{2uw}{(uv-w^2)^2} & \frac{4w^2}{(uv-w^2)^2} + \frac{2}{uv-w^2} \end{pmatrix}$$

At the point $(1, 1, 0)$, this becomes:

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

Since this is positive-definite, (1,1,0) is a minimum. Since it is the only stationary point in Ω, and $L \to \infty$ on the boundary of Ω, it is a global minimum.

For all trajectories beginning in Ω:

$$\begin{aligned} \frac{\mathrm{d}L}{\mathrm{d}t} &= \nabla L \cdot \begin{pmatrix} u'(t) \\ v'(t) \\ w'(t) \end{pmatrix} \\ &= \begin{pmatrix} 1 - \frac{v}{uv-w^2} \\ 1 - \frac{u}{uv-w^2} \\ \frac{2w}{uv-w^2} \end{pmatrix} \cdot \begin{pmatrix} 2w \\ 2v(1-v) - 2w \\ w(1-v) - u + v \end{pmatrix} \\ &= \left(2w - \frac{2vw}{uv-w^2}\right) + \left(2v(1-v) - 2w - \frac{2uv(1-v)}{uv-w^2} + \frac{2uw}{uv-w^2}\right) \\ &\quad + \left(\frac{2w^2(1-v)}{uv-w^2} - \frac{2uw}{uv-w^2} + \frac{2vw}{uv-w^2}\right) \\ &= 2v(1-v) - \frac{2(uv-w^2)(1-v)}{uv-w^2} \\ &= -2(1-v)^2 \end{aligned} \tag{16}$$

So L is decreasing on any trajectory beginning in Ω. Therefore, L is indeed a Lyapunov function, and all sublevel sets of L are invariant [14].

We have, for any $\Omega(k)$, $\Omega(k)$ is closed, bounded, and invariant. Given any trajectory $(u(t), v(t), w(t)$, we may apply Theorem 3 on $\Omega(L(u(0), v(0), w(0)))$. Theorem 3 guarantees that (u, v, w) approaches the largest invariant set inside which $\dot{L}(u, v, w) = 0$. All that remains is to show that this set is equal to the point $(1, 1, 0)$.

Since $(1, 1, 0)$ is a global minimum of L, it is in every nonempty sublevel set of L. Suppose that $\dot{L} = 0$ for some trajectory $(u(t), v(t), w(t))$. Then:

$$-2(1 - v(t))^2 = 0 \Rightarrow v(t) = 1 \quad \text{for all } t > 0$$

So by equation (12):

$$\dot{v}(t) = 0 \Rightarrow 2v(t)(1 - v(t)) - 2w(t) = 0 \Rightarrow w(t) = 0 \quad \text{for all } t > 0$$

So:

$$\dot{w}(t) = 0 \Rightarrow w(t)(1 - v(t)) - u(t) + v(t) = 0 \Rightarrow u(t) = 1 \quad \text{for all } t > 0$$

So the only invariant set in any $\Omega(k)$ with $\dot{V}(u(t), v(t), w(t)) = 0$ is the point $(1, 1, 0)$.

Therefore, by Theorem 3, every trajectory beginning in Ω converges to the point $(1, 1, 0)$.

So $\mathbf{r} \cdot \mathbf{r} \to 1$, $\dot{\mathbf{r}} \cdot \dot{\mathbf{r}} \to 1$, and $\dot{\mathbf{r}} \cdot \mathbf{r} \to 0$.

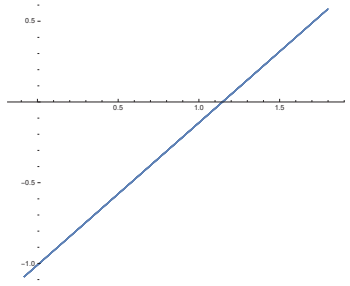
So if $\mathbf{r}(0)$ and $\dot{\mathbf{r}}(0)$ are not parallel, the system converges to a circular limit cycle of radius 1 with speed 1 centered on the origin. □

3.4 Two-Agent Model

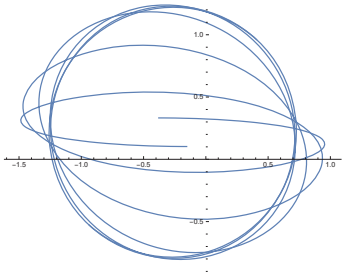
We next examine the behavior of two coupled agents, with position vectors $\mathbf{r}_1$ and $\mathbf{r}_2$. The agents have equations of motion:

$$\begin{aligned} \ddot{\mathbf{r}}_1 &= (1 - \dot{\mathbf{r}}_1 \cdot \dot{\mathbf{r}}_1)\dot{\mathbf{r}}_1 - \frac{1}{2}\mathbf{r}_1 + \frac{1}{2}\mathbf{r}_2 \\ \ddot{\mathbf{r}}_2 &= (1 - \dot{\mathbf{r}}_2 \cdot \dot{\mathbf{r}}_2)\dot{\mathbf{r}}_2 - \frac{1}{2}\mathbf{r}_2 + \frac{1}{2}\mathbf{r}_1 \end{aligned} \tag{17}$$

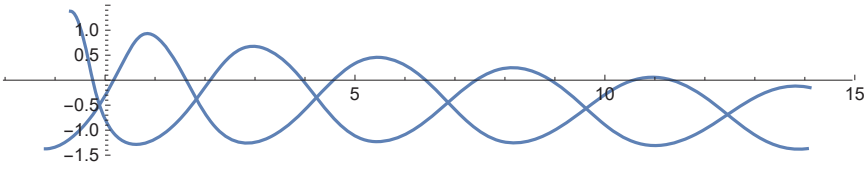
We have not characterized the behavior of the two-agent system globally. There exist at least three behaviors which we have seen in simulation: a one-dimensional oscillation, a rotating behavior, and a translating behavior, pictured in Figure 7.



(a) One-dimenstional oscillating state



(b) Rotating state



(c) Translating state

Figure 7: Different behaviors of a swarm of two agents. Different initial conditions produce these behaviors. The plotted lines are the trajectories of two agents on the plane, with varying initial conditions.

We have shown that the rotating behavior is locally stable by rewriting the system and using Lyapunov's indirect (linearization) method.

Theorem 9 (Original)**.** *Rotating behavior is locally stable for a parabolic potential model of two agents.*

Proof. We begin by reformulating the system using the following substitution:

$$\begin{aligned} u &= |\mathbf{r}_1 - \mathbf{r}_2| \\ v_1 &= |\dot{\mathbf{r}}_1| \\ v_2 &= |\dot{\mathbf{r}}_2| \\ \theta_1 &= \text{angle between } \dot{\mathbf{r}}_1 \text{ and } (\mathbf{r}_1 - \mathbf{r}_2) \\ \theta_2 &= \text{angle between } \dot{\mathbf{r}}_2 \text{ and } (\mathbf{r}_2 - \mathbf{r}_1) \end{aligned} \tag{18}$$

We now differentiate each term.

$$\begin{aligned} \dot{u} &= \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|}(\dot{\mathbf{r}}_1 + \dot{\mathbf{r}}_2) \cdot (\mathbf{r}_2 - \mathbf{r}_1) \\ &= \frac{1}{u}(uv_1 \cos\theta_1 + uv_2 \cos\theta_2) \\ &= v_1 \cos\theta_1 + v_2 \cos\theta_2 \end{aligned} \tag{19}$$

$$\begin{aligned} \dot{v}_1 &= \frac{1}{|\dot{\mathbf{r}}_1|}(\ddot{\mathbf{r}}_1 \cdot \dot{\mathbf{r}}_1) \\ &= \frac{1}{v_1}((1 - \dot{\mathbf{r}}_1 \cdot \dot{\mathbf{r}}_1)\dot{\mathbf{r}}_1 \cdot \dot{\mathbf{r}}_1 - \frac{1}{2}(\mathbf{r}_1 - \mathbf{r}_2) \cdot \dot{\mathbf{r}}_1) \\ &= \frac{1}{v_1}((1 - v_1^2)v_1^2 - \frac{1}{2}uv_1 \cos\theta_1) \\ &= (1 - v_1^2)v_1 - \frac{1}{2}v_1 \cos\theta_1 \end{aligned} \tag{20}$$

A similar argument can be used to show that:

$$\dot{v}_2 = (1 - v_2^2)v_2 - \frac{1}{2}v_2 \cos\theta_2 \tag{21}$$

The final two derivatives can be calculated implicitly. First, denote the angle between $\dot{\mathbf{r}}_1$ and $\dot{\mathbf{r}}_2$ as ϕ. We can see from Figure 8 that $\phi = \pi - \theta_1 - \theta_2$. Then we have:

$$\cos\phi = \cos(\pi - \theta_1 - \theta_2) = -\cos(\theta_1 + \theta_2) = \sin\theta_1 \sin\theta_2 - \cos\theta_1 \cos\theta_2 \tag{22}$$

We can now calculate the final two derivatives implicitly. First, note that:

$$uv_1 \cos\theta_1 = \dot{\mathbf{r}}_1 \cdot (\mathbf{r}_1 - \mathbf{r}_2) \tag{23}$$

Differentiating the left hand side yields:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}(uv_1 \cos\theta_1) &= (\dot{u}v_1 + u\dot{v}_1)\cos\theta_1 - uv_1\dot{\theta}_1 \sin\theta_1 \\ &= v_1^2 \cos^2\theta_1 + v_1 v_2 \cos\theta_1 \cos\theta_2 + uv_1(1 - v_1^2)\cos\theta_1 - \frac{1}{2}u^2 \cos^2\theta_1 - uv_1\dot{\theta}_1 \sin\theta_1 \end{aligned} \tag{24}$$

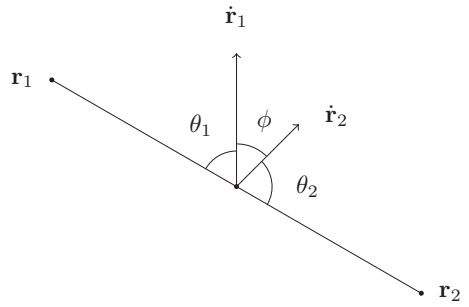


Figure 8: Calculating the angle between $\dot{\mathbf{r}}_1$ and $\dot{\mathbf{r}}_2$.

Differentiating the right hand side:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}(\dot{\mathbf{r}}_1 \cdot (\mathbf{r}_1 - \mathbf{r}_2)) &= \ddot{\mathbf{r}}_1 \cdot (\mathbf{r}_1 - \mathbf{r}_2) + \mathbf{r}_1 \cdot \mathbf{r}_1 - \mathbf{r}_1 \cdot \mathbf{r}_2 \\ &= (1 - \dot{\mathbf{r}}_1 \cdot \dot{\mathbf{r}}_1)\dot{\mathbf{r}}_1 \cdot (\mathbf{r}_1 - \mathbf{r}_2) - \frac{1}{2}(\mathbf{r}_1 - \mathbf{r}_2) \cdot (\mathbf{r}_1 - \mathbf{r}_2) + \mathbf{r}_1 \cdot \mathbf{r}_1 - \mathbf{r}_1 \cdot \mathbf{r}_2 \\ &= (1 - v_1^2)uv_1 \cos\theta_1 - \frac{1}{2}u^2 + v_1^2 + v_1 v_2 \cos\theta_1 \cos\theta_2 - v_1 v_2 \sin\theta_1 \sin\theta_2 \end{aligned} \tag{25}$$

Equating (24) and (25) and canceling:

$$v_1^2 \cos^2\theta_1 - \frac{1}{2}u^2 \cos^2\theta_1 - uv_1\dot{\theta}_1 \sin\theta_1 = -\frac{1}{2}u^2 + v_1^2 - v_1 v_2 \sin\theta_1 \sin\theta_2 \tag{26}$$

Rearranging for $\dot{\theta}_1$:

$$\begin{aligned}\dot{\theta}_1 &= \frac{\frac{1}{2}u^2 - \frac{1}{2}u^2\cos^2\theta_1 - v_1^2 + v_1^2\cos^2\theta_1 + v_1 v_2 \sin\theta_1 \sin\theta_2}{uv_1 \sin\theta_1} \\ &= \frac{\frac{1}{2}u^2 \sin^2\theta_1 - v_1^2 \sin^2\theta_1 + v_1 v_2 \sin\theta_1 \sin\theta_2}{uv_1 \sin\theta_1} \\ &= \left(\frac{u}{2v_1} - \frac{v_1}{u}\right)\sin\theta_1 + \frac{v_2}{u}\sin\theta_2 \end{aligned} \tag{27}$$

A similar argument shows that:

$$\dot{\theta}_2 = \left(\frac{u}{2v_2} - \frac{v_2}{u}\right)\sin\theta_2 + \frac{v_1}{u}\sin\theta_1 \tag{28}$$

Equations (19), (20), (21), (27), and (28) give the following system of differ-

ential equations:

$$\begin{aligned}
\dot{u} &= v_1 \cos\theta_1 + v_2 \cos\theta_2 \\
\dot{v}_1 &= (1 - v_1^2)v_1 - \frac{1}{2} v_1 \cos\theta_1 \\
\dot{v}_2 &= (1 - v_2^2)v_2 - \frac{1}{2} v_2 \cos\theta_1 \\
\dot{\theta}_1 &= \left(\frac{u}{2v_1} - \frac{v_1}{u}\right) \sin\theta_1 + \frac{v_2}{u} \sin\theta_2 \\
\dot{\theta}_2 &= \left(\frac{u}{2v_2} - \frac{v_2}{u}\right) \sin\theta_2 + \frac{v_1}{u} \sin\theta_1
\end{aligned} \tag{29}$$

The rotating behavior corresponds to the case when the distance between particles and the center of mass is 1, and therefore $u = 2$. The speed of both agents v_1 and v_2 will be 1. We also expect that the direction of rotation will be orthogonal to the vector between the two agents, and in opposite directions, giving us that θ_1 and θ_2 are both $\frac{\pi}{2}$ or both $-\frac{\pi}{2}$. This gives us the following two points: $(2, 1, 1, \pi/2, \pi/2)$, and $(2, 1, 1, -\pi/2, -\pi/2)$. At these two points, we linearize the system by calculating the Jacobian matrices:

$$\begin{aligned}
\mathbf{J}(2, 1, 1, \pi/2, \pi/2) &= \begin{pmatrix} 0 & 0 & 0 & -1 & -1 \\ 0 & -2 & 0 & 1 & 0 \\ 0 & 0 & -2 & 0 & 1 \\ \frac{1}{2} & -\frac{3}{2} & \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & -\frac{3}{2} & 0 & 0 \end{pmatrix} \\
\mathbf{J}(2, 1, 1, -\pi/2, -\pi/2) &= \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & -2 & 0 & -1 & 0 \\ 0 & 0 & -2 & 0 & -1 \\ -\frac{1}{2} & \frac{3}{2} & -\frac{1}{2} & 0 & 0 \\ -\frac{1}{2} & -\frac{1}{2} & \frac{3}{2} & 0 & 0 \end{pmatrix}
\end{aligned} \tag{30}$$

We calculated the values of the eigenvalues of these matrices. For both matrices, the eigenvalues are $(-1.544, -1 \pm i, -0.228 \pm 1.115i)$. Since all these eigenvalues are in the negative complex half-plane, local stability follows from Lyapunov's indirect method [14]. □

3.5 Systems with More Agents

We are currently investigating stability of systems with more than two agents. For a system of N agents, we use the following change of coordinates to describe the position of the ith agent in system (1):

$$\begin{aligned}
u_i &= \text{Distance between the } i\text{th agent and the center of mass} \\
v_i &= \text{Speed of the } i\text{th agent} \\
\theta_i &= \text{Angle between the } i\text{th agent and the positive } x\text{-axis} \\
\phi_i &= \text{Angle between the vector } (\mathbf{r}_i - R) \text{ and the vector } \dot{r}_i
\end{aligned} \tag{31}$$

Where $\mathbf{r}_i$ is the position vector of the ith agent, and R is the center of mass of the system. This leads to the following system of differential equations:

$$\begin{aligned}
\dot{u}_i &= v_i \cos\phi_i - \frac{1}{N}\sum_{j=1}^{N} v_j \cos(\phi_j + \theta_j - \theta_i) \\
\dot{v}_i &= (1 - v_i^2)v_i - u_i \cos\phi_i \\
\dot{\theta}_i &= \frac{v_i}{u_i}\sin\phi_i - \frac{1}{u_i N}\sum_{j=1}^{N} v_j \sin(\phi_j + \theta_j - \theta_i) \\
\dot{\phi}_i &= (\frac{u_i}{v_i} - \frac{v_i}{u_i})\sin\phi_i + \frac{1}{u_i N}\sum_{j=1}^{N} v_j \sin(\phi_j + \theta_j - \theta_i)
\end{aligned} \tag{32}$$

This system shifts the center of mass onto the origin. It is therefore necessary to include the constraint that $\sum_{i=1}^{N}(u_i \cos\theta_i, u_i \sin\theta_i) = (0,0)$. As might be expected, the manifold created by this constraint is invariant, so we may restrict ourselves to studying the behavior of the system on this manifold.

We believe that the following conjecture is true:

Conjecture 1. *For any $N > 0$, circular limit cycles satisfying Equation (1) with stationary center of mass are locally stable up to translation.*

We have shown this using a linearization for 3 agents, but with higher numbers of agents, the system has dimensions of neutral stability. For example, there are many possible configurations in a swarm of five agents: we may perturb all agents along the unit circle in a way which maintains the center of mass. This type of perturbation is pictured in Figure 9.

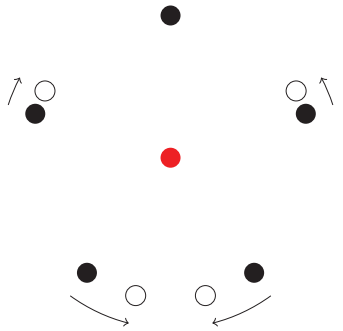


Figure 9: A perturbation which keeps the center of mass constant in a five-agent system.

MIDN Carl Kolon, Prof. Kostya Medynets, and Prof. Irina Popovici, all at USNA, are currently attempting to use center manifold analysis to prove that this conjecture is indeed true for any number of agents.

4 Swarm Collisions

Robot swarms can be controlled with the same potential models analyzed in the literature [23, 24]. Typically, a translating or flocking state is desired, in

which all agents have similar speed and are spaced fairly regularly. It is also often desirable for two flocks to merge and continue as a larger flock.

4.1 Morse Swarms

While the parabolic potential model admits more analytical study, other models have been studied which produce results that more closely mimic the behavior of real organisms. A commonly studied example is the attractive-repulsive model proposed by D'Orsogna et al. [7]. The model is a system of N agents in n-dimensional space with position vectors x_i, acting under the following equation of motion:

$$\ddot{x}_i = (\alpha - \beta|\dot{x}_i|^2)\dot{x}_i - \frac{\lambda}{N}\sum_{j-1,i\neq j}^{N} \nabla_{x_i} U(x_i, x_j) \tag{33}$$

Where α, β, λ are constants, $U : \mathbb{R}^{2n} \to \mathbb{R}$ is a potential function of the two agents' position, and ∇_{x_i} represents taking the gradient with respect to x_i. Many potential functions can be used, but a typical choice is the scaled Morse potential:

$$U(x_i, x_j) = C\exp(-|x_i - x_j|/l) - \exp(-|x_i - x_j|) \tag{34}$$

When the Morse potential is used, we call the resulting swarm a *Morse swarm*.

4.2 Without Delay

The Morse swarm's behavior is too complicated to characterize analytically, but we are still interested in its stability properties. In particular, we are interested in the stability of colliding swarms.

When $\nabla_{x_i} U(x_i, x_j) = 0$ for all $i, j \leq N$ with $i \neq j$, the configuration of agents is called a *flock* [18]. In a Morse swarm, a flock undergoes translating motion at a constant speed $\sqrt{\alpha/\beta}$, and can withstand sufficiently small perturbations [19]. We are interested in flock collisions. Two separate flocks are initialized and pointed towards each other with some incident angle, which we call θ. We are interested in the behavior after collision, and how it varies for varying λ, θ, and τ.

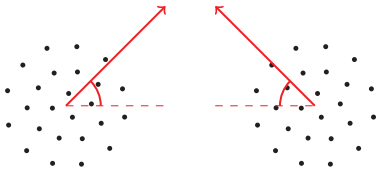
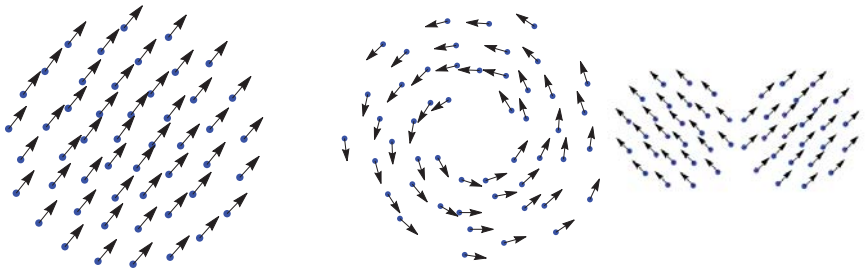


Figure 10: The incident angle of the collision, θ, which we vary as a parameter.

Flocking is not the only behavior that Morse swarms display. They also can perform milling behavior, where the agents rotate around a stationary center of mass. To differentiate between the two behaviors, Armbruster, Martin, and Thatcher [17] calculated the polarization of the swarm: the directed sum of velocities. For a system of N agents with position vectors x_i, the polarization P of the flock is:

$$P(x_1, x_2, \ldots, x_N) = \left| \frac{\sum_{i=1}^{N} \dot{x}_i}{\sum_{i=1}^{N} |\dot{x}_i|} \right|$$

If the velocities are coherent, we expect a value of P close to 1. If the velocities point in all directions, we expect a value close to 0. Therefore, polarization is a good measure of whether the system displays flocking ($P \approx 1$), or milling ($P \approx 0$). Additionally, two colliding swarms can scatter, in which they do not form a coherent group which either flocks or mills. In this case, we expect P to be neither close to 0 nor close to 1.



(a) Flocking. (b) Milling. (c) Scattering.

Figure 11: Flocking, milling, and scattering behavior in a Morse swarm.

Using Wolfram Mathematica 11.2, we simulated two flocks of 25 agents each colliding. We varied the values of λ, the coupling strength, and θ, the incident angle of the collision.

We first reproduced the results of Armbruster, Martin, and Thatcher [17] by performing the experiment with $\tau = 0$, corresponding to a system with instantaneous interactions. We simulated 400 collisions and recorded the polarization after 100 time units. We generated the plot in figure 12.

This is similar to previous results. For low values of λ, scattering is likely unless the swarm motion is almost parallel to begin with. For intermediate values of λ, flocking is likely. For high values of λ, milling dominates. A region of metastability exists between the flocking and milling states.

4.3 With Delay

In any robotic swarm there must be some delay between sensing and actuation. We investigated the result of applying delay to the coupling terms of a swarm model. As a result, every agent behaves as though the other agents are at the positions they were at τ time units before. We found that the flocking state is unlikely following collisions of delay-coupled swarms, even for fairly small values of τ.

We modified the model to include the time delay τ, creating the following delay differential equation from equation (33).

$$\ddot{x}_i(t) = (\alpha - \beta|\dot{x}_i(t)|^2)\dot{x}_i(t) - \frac{\lambda}{N}\sum_{j-1,i\neq j}^{N} \nabla_{x_i} U(x_i(t), x_j(t-\tau)) \qquad (35)$$

With U defined in equation (34). Since this is a delay differential equation, it requires a history function, for which we used translating motion of the two flocks along linear trajectories.

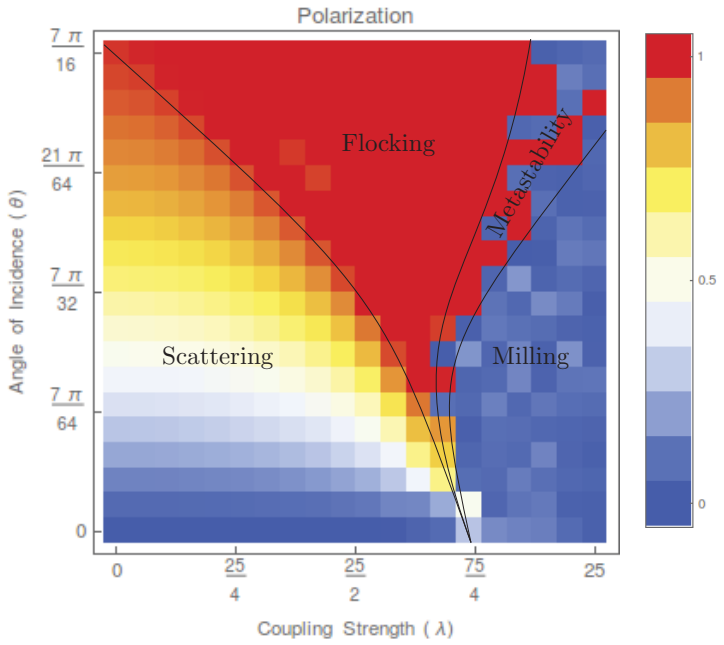


Figure 12: Polarization as a function of λ and θ for an instantaneously-interacting swarm. Note the distinct regions of flocking, milling, scattering, and metastability.

Initially, we tried three distinct values of τ: 0, 0.1, and 0.2. We generated the plots in figure 13.

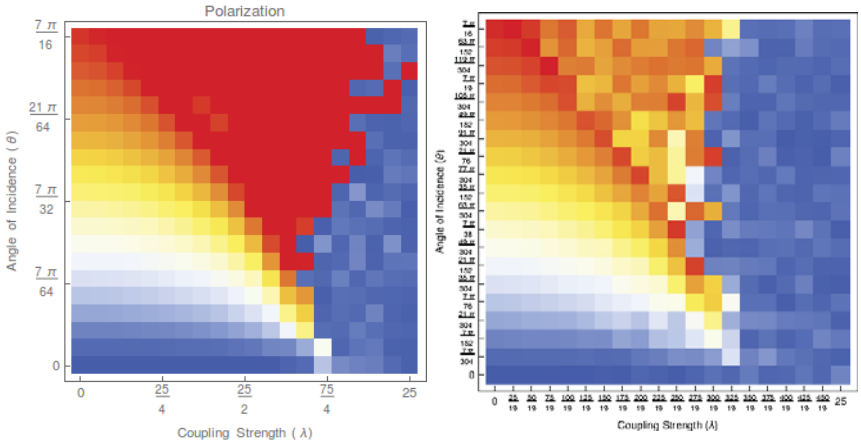
We can note several things about the introduction of delay. Even at the fairly small τ value of 0.2, flocking behavior nearly vanishes. Instead, scattering behavior becomes more likely for low λ, and milling behavior becomes more likely for high λ.

Next, we fixed θ at $\frac{\pi}{4}$ and varied τ finely, to observe polarization as a function of τ and λ. We generated the plot in figure 14. This confirms that as delay increases steadily, milling behavior largely replaces flocking behavior. A complete writeup of our results is available on the ArXiv [25].

5 Motion on Riemannian Manifolds

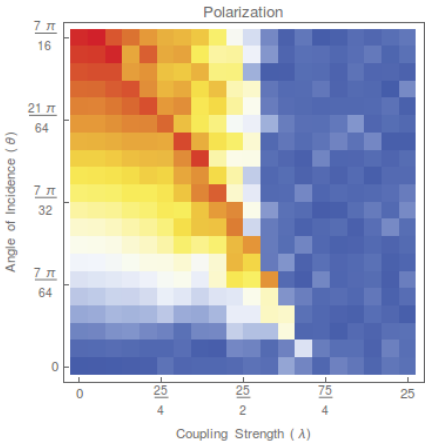
We investigate the motion of swarms on curved surfaces, or Riemannian manifolds, in order to create a new class of swarms which can operate when constrained to surfaces. In this section, we generalize the concept of a gradient swarm to Riemannian manifolds. To our knowledge, this has never been done. So far, we have only observed the behavior of the system numerically, but the behavior of gradient swarms on manifolds appears to mimic their behavior in Euclidean space. As is common in differential geometry, we use Einstein summation notation for our equations.

A Riemannian manifold is a smooth manifold M equipped with an inner



(a) $\tau = 0$

(b) $\tau = 0.1$



(c) $\tau = 0.2$

Figure 13: Polarization as a function of λ and θ for 3 distinct values of τ.

product $\langle \cdot, \cdot \rangle_p$ (the metric tensor) on the tangent space T_pM at every point p. The inner product varies smoothly along any vector field on M. On a chart of a Riemannian manifold with coordinates x_i, the metric tensor is often represented as g, such that for two tangent vectors $\mathbf{u}$ and $\mathbf{v}$, $\langle \mathbf{u}, \mathbf{v} \rangle = g_{ij} u^i v^j$.

A geodesic is a locally length-minimizing curve. A geodesic γ satisfies the geodesic equation [26]:

$$\ddot{\gamma}^a + \Gamma^a_{ij} \dot{\gamma}^i \dot{\gamma}^j = 0$$

in all coordinate directions a, where Γ is the Christoffel symbol of the second kind, defined as:

$$\Gamma^a_{ij} = \frac{1}{2} g^{ak} \left(\frac{\partial g_{ik}}{\partial x^j} + \frac{\partial g_{jk}}{\partial x^i} - \frac{\partial g_{ij}}{\partial x^k} \right)$$

An agent on the manifold with position vector $\mathbf{x} = (x^1, x^2, \ldots, x^a)$ moving with no external acceleration will follow geodesic paths, so such an agent will

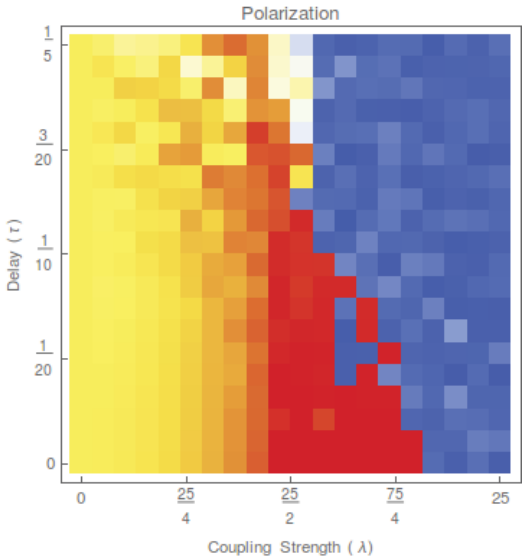


Figure 14: A plot of polarization as a function of λ and τ for fixed θ. Note that flocking behavior all but disappears above $\tau \approx 0.1$.

satisfy the following equation of motion:

$$\ddot{x}^a + \Gamma^a_{ij}\dot{x}^i\dot{x}^j = 0$$

The concept of the gradient is different on a Riemannian manifold as well. For a function $f : M \to \mathbb{R}$, and some vector field V on M, we desire that the gradient have the following property:

$$(\nabla f)^T g V = df^T(V)$$

Where df is the directional derivative of f, and T represents the transpose. Since this is an identity for any V:

$$(\nabla f)^T g = df^T$$

Which gives:

$$g^T \nabla f = df$$

And since inner products are symmetric:

$$\nabla f = g^{-1} df$$

This gives the following form for the gradient on a Riemannian manifold:

$$\nabla f = \sum_k g^{ik} \frac{\partial f}{\partial x^k} \mathbf{e}_i \tag{36}$$

5.1 Swarming on Riemannian Manifolds

The original swarm in Euclidean space has equation of motion:

$$\ddot{\mathbf{r}}_i = \mathbf{f}(\dot{\mathbf{r}}) - \frac{1}{N}\sum_{j=1}^{N} \nabla_{\mathbf{r}_i} U(\mathbf{r}_i, \mathbf{r}_j) \tag{37}$$

To make this swarm suitable for manifolds, we must do two things:

1. We must change the flat representation of the gradient to the Riemannian manifold representation.
2. We must ensure that, if $\mathbf{f}$ and U are everywhere zero, the agent moves along a geodesic.

To do this, we define a gradient swarm of N agents with positions $\mathbf{r}_i = (r_i^1, \ldots, r_i^n)$ (in curvilinear coordinates) by the following equation of motion:

$$\ddot{r}_i^a = f^a(\dot{\mathbf{r}}_i) - \Gamma^a_{jk}\dot{r}_i^j\dot{r}_i^k - \frac{1}{N}\sum_{j=1}^{N}\sum_{k=1}^{n} g^{ak}\frac{\partial}{\partial r_i^k} U(\mathbf{r}_i, \mathbf{r}_j) \tag{38}$$

Our numerical simulations of this system show that it inherits many familiar properties of gradient swarms. To replicate the parabolic potential model, we used the following values of $\mathbf{f}$ and U:

$$\begin{aligned} f^a(\dot{\mathbf{r}}) &= (1 - g_{ij}\dot{r}^i\dot{r}^j)r^a \\ U(\mathbf{r}_i, \mathbf{r}_j) &= \text{the squared length of the shortest path between } \mathbf{r}_i \text{ and } \mathbf{r}_j \end{aligned} \tag{39}$$

On some Riemannian manifolds (e.g. the sphere and hyperbolic half-plane), there are closed-form expressions for both $\mathbf{f}$ and U. We numerically simulated the parabolic potential model on these surfaces, and found that the swarms still formed circular limit cycles (see Figure 15). Our current methods for simulating the system when there are not closed-form expressions for $\mathbf{f}$ and U are too computationally intensive to use, but we are working on methods which use less computing power.

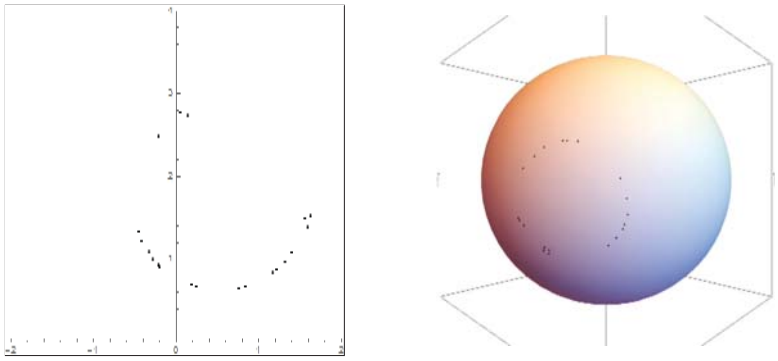
5.2 Modifying Limit Cycles

We used the concepts developed with the generalization of gradient swarms to modify the limit cycles of the nonlinear parabolic potential model. Other methods to achieve this were considered by Medynets and Schwartz [9]. By using a constant (flat) matrix as our metric tensor, we can skew the motion of the agents. In this case, there is also a closed form for the distance between agents. This allows us to numerically simulate the modified swarms, which have oval limit cycles rather than circles.

5.3 Generalization of other Gradient Swarms

We have used Equation (38) to generalize other types of gradient swarms to Riemannian manifolds. For example, we used the new potential function:

$$U(\mathbf{r}_i, \mathbf{r}_j) = C\exp(-d(x_i, x_j)/l) - \exp(-d(x_i, x_j))$$



(a) Swarm in hyperbolic half-plane

(b) Swarm on sphere

Figure 15: Generalized parabolic potential swarms simulated on two different manifolds.

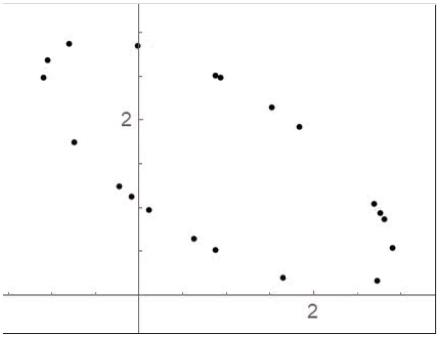


Figure 16: Modifying the parabolic potential model to produce a swarm with oval limit cycles.

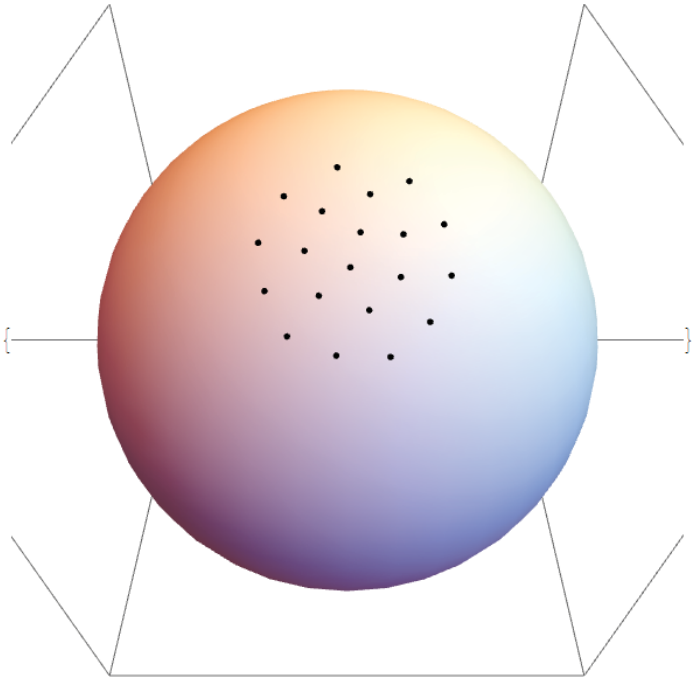


Figure 17: A Morse swarm on a sphere

To generalize the Morse potential. Our results on a sphere are pictured in Figure 17. We observed behavior similar to the Morse swarm in flat space. Our results can be repeated by modifying the potential function in the Mathematica code in Appendix A.

6 Technical Implementation

We are interested in the applications of this research to the motion of swarms of unmanned vehicles. We have implemented swarm models like the nonlinear parabolic potential model in Wolfram Mathematica (see Appendix A). We then designed a package in ROS, the Robot Operating System, to communicate the desired velocity (generated by numerically solving the differential equation in Mathematica) to UAV models in Gazebo, a high-fidelity physics simulator. Our control structure is pictured in Figure 18.

ROS and Mathematica communicated through CSV files. Our work was based on the work of Gilner [27], who controlled a Parrot ArDrone 2.0 using Mathematica and ROS.

Our method follows:

6.1 ROS and Gazebo Setup

We used ROS Kinetic on a machine running Ubuntu 16.04.

1. To install ROS on Ubuntu, follow the instructions at `http://wiki.ros.org/kinetic/Installation/U`

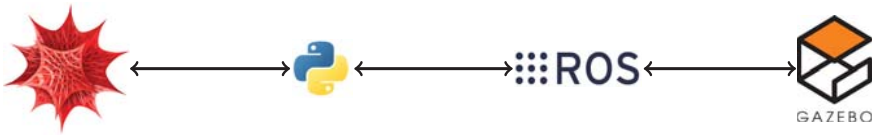


Figure 18: A diagram of the control structure. Differential equations are solved in Wolfram Mathematica using Euler's method. ROS receives desired data using Python and transmits it to simulated robots in Gazebo. Feedback is relayed back to Mathematica through ROS using Python.

2. We must create a workspace to hold our packages. To create a workspace, run the following terminal commands:

```
$ mkdir -p ~/my_ros_pkgs/src
$ cd ~/my_ros_pkgs
$ catkin_make
$ echo "source ~/my_ros_pkgs/devel/setup.bash" >> ~/.bashrc
```

   The final command allows the terminal to directly run ROS programs.

6.2 Installing the Hector Quadrotor Software

1. Hector seems to be the most documented and supported quadrotor simulation software for ROS/Gazebo. Unfortunately, the packages aren't all available in the linux installation database for ROS Kinetic. We used an online script to make it run anyway.
2. Download the file `installhectorquadrotor.bash`, and save it.
3. Move the file into the `src` directory of your catkin workspace (e.g., if your workspace is `~/my_ros_pkgs`, then copy the file into `~/my_ros_pkgs/src`)
4. From the `src` directory, run the following commands in the terminal:

```
$ chmod +x hectorquadrotorinstallation.bash
$ ./hectorquadrotorinstallation.bash
```

5. Enter your password and/or type Y when prompted.
6. Once the process is finished, run the following:

```
$ cd ..
$ catkin_make
$ source devel/setup.bash
$ sudo apt-get install ros-kinetic-teleop-twist-keyboard
$ roslaunch hector_quadrotor_gazebo quadrotor_empty_world.launch
```

   A quadrotor in an empty world should spawn.

7. Now in a new terminal:

```
$ source devel/setup.bash
$ rosservice call \enable_motors true
$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

8. Use the keyboard to fly the drone around!

6.3 Running Multiple Hector Quadrotors

1. Create a new package in your catkin workspace (I called it `multi_hector`).
2. Copy the launch folder and the urdf folder from `hector_quadrotor/hector_quadrotor_gazebo` into the `multi_hector` folder.
3. Download `createhectorlaunchfile.py` (Appendix B) and put it in the `multi_hector/launch` folder.
4. Download `multihectorListener.py` and `multihectorPublisher.py` (Appendix B) and put them in the `multi_hector/src` folder.
5. Download `enablemotors.bash` (Appendix B) and put it in your main catkin workspace folder.
6. Go to a terminal and run the following:

```
$ cd ~/my_ros_pkgs
$ catkin_make
$ source devel/setup.bash
$ rosrun multi_hector createhectorlaunchfile.py 10 (for 10 robots,
    adjust as necessary)
$ roslaunch multi_hector multi_quadrotor_world10.launch
```

7. Now in a new terminal:

```
$ cd ~/my_ros_pkgs
$ source devel/setup.bash
$ ./enablemotors.bash
... wait while motors are enabled ...
$ rosrun multi_hector multihectorListener.py 10 (for 10 robots,
    adjust as necessary)
```

8. And in another new terminal:

```
$ cd ~/my_ros_pkgs
$ source devel/setup.bash
$ rosrun multi_hector multihectorPublisher.py 10 (for 10 robots,
    adjust as necessary)
```

9. Now the robots are listening for commands, which you can issue by running the Mathematica file `multi_hector_control.nb`, printed in Appendix A.

Figure 19 shows the simulated robotic swarm in action.

References

[1] M. Ciszak et al. "Swarming Behavior in Plant Roots". In: *PLoS ONE* 7 (2012), p. 1. DOI: 10.1371/journal.pone.0029759.

[2] D. Kearns. "A field guide to bacterial swarming motility". In: *Nature Reviews Microbiology* 8 (2010), pp. 634–644.

[3] C. Reynolds. "Flocks, herds and schools: A distributed behavioral model." In: *Computer Graphics* 21.(4) (1987), pp. 25–34.

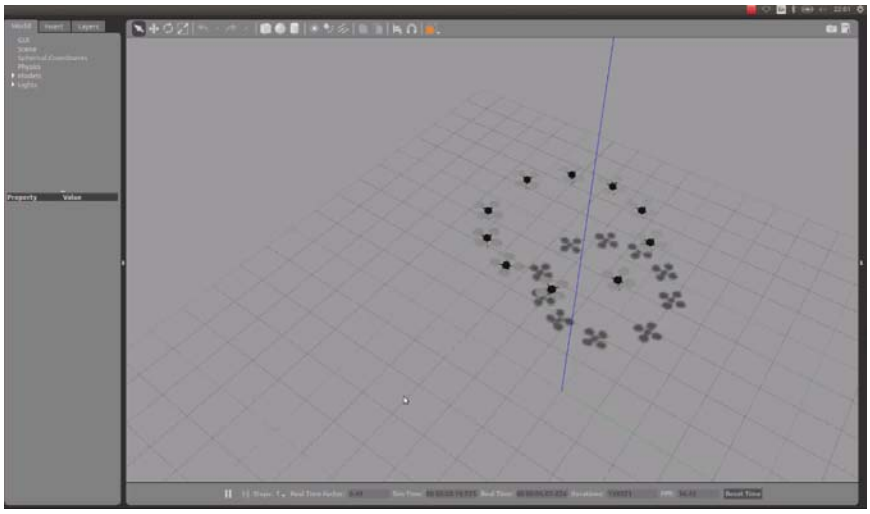


Figure 19: Quadrotors following the limit cycle of the nonlinear parabolic potential swarm in Gazebo. The agents are controlled by ROS, which is feeds velocity and position data between Gazebo and Mathematica. Note the similarity to the limit behavior pictured in Figure 6.

[4] A. Mogilner et al. "Mutual interactions, potentials, and individual distance in a social aggregation". In: *Journal of Mathematical Biology* 47 (2003), pp. 353–389.

[5] Office of Naval Research. *LOCUST, a Low Cost UAV Swarm*. 2015.

[6] *Unmanned Systems Integrated Roadmap*. Department of Defense. 2013.

[7] M. R. D'Orsogna et al. "Self-Propelled Particles with Soft-Core Interactions: Patterns, Stability, and Collapse". In: *Phys. Rev. Lett.* 96 (10 Mar. 2006), p. 104302. DOI: 10.1103/PhysRevLett.96.104302.

[8] W. Ebeling and F. Schweitzer. "Swarms of Particle Agents with Harmonic Interactions". In: *Theory in Biosciences* 120/3-4 (2001), pp. 207–224.

[9] K. Medynets and I. Schwartz. "Modelling Swarm Convergence with Arbitrarily Shaped Limit Cycles". In preparation. 2016.

[10] A. Haraux and M. A. Jendoubi. "Convergence of Solutions of Second-Order Gradient-Like Systems with Analytic Nonlinearities". In: *Journal of Differential Equations* 144.DE973393 (1998), pp. 313–320.

[11] Klementyna Szwaykowska et al. "Collective motion patterns of swarms with delay coupling: Theory and experiment". In: *Physical Review E* 93.3 (Mar. 2016). DOI: 10.1103/physreve.93.032307.

[12] J. T. Beatty et al. "An obligately photosynthetic bacterial anaerobe from a deep-sea hydrothermal vent". In: *Proceedings of the National Academy of Sciences* 102.26 (June 2005), pp. 9306–9310. DOI: 10.1073/pnas.0503674102.

[13] Louise Knapp. "Look, Up in the Sky: Robofly". In: *Wired* (2000).

[14] A. M. Lyapunov. "The general problem of the stability of motion (originally published in Russian in 1893)". In: *International Journal of Control* 55.3 (Mar. 1992), pp. 531–534. DOI: 10.1080/00207179208934253.

[15] Steven H. Strogatz. *Nonlinear Dynamics And Chaos: With Applications To Physics, Biology, Chemistry, And Engineering (Studies in Nonlinearity)*. CRC Press, 2000. ISBN: 0738204536.

[16] H.K. Khalil. *Nonlinear Systems*. Pearson Education. Prentice Hall, 2002. ISBN: 9780130673893.

[17] D. Armbruster, S. Martin, and A. Thatcher. "Elastic and inelastic collisions of swarms". In: *Physica D: Nonlinear Phenomena* 344.Supplement C (2017), pp. 45–57. ISSN: 0167-2789. DOI: https://doi.org/10.1016/j.physd.2016.11.008.

[18] Herbert Levine, Wouter-Jan Rappel, and Inon Cohen. "Self-organization in systems of self-propelled particles". In: *Physical Review E* 63.1 (Dec. 2000). DOI: 10.1103/physreve.63.017101.

[19] J.A. Carrillo, Y. Huang, and S. Martin. "Nonlinear stability of flock solutions in second-order swarming models". In: *Nonlinear Analysis: Real World Applications* 17 (June 2014), pp. 332–343. DOI: 10.1016/j.nonrwa.2013.12.008.

[20] R. Skoog and J. Aggarwal. "Some observations concerning the boundedness of solutions of coupled Lienard's equations". In: *IEEE Transactions on Circuit Theory* 19.6 (Nov. 1972), pp. 625–626. ISSN: 0018-9324. DOI: 10.1109/TCT.1972.1083562.

[21] A. Liénard. *Etude des oscillations entretenues*. Revue générale de l'électricité, 1928.

[22] A. Lins, W. de Melo, and C. C. Pugh. "On Liénard's equation". In: *Lecture Notes in Mathematics*. Springer Berlin Heidelberg, 1977, pp. 335–357. DOI: 10.1007/bfb0085364.

[23] John H Reif and Hongyan Wang. "Social potential fields: A distributed behavioral control for autonomous robots". In: *Robotics and Autonomous Systems* 27.3 (1999), pp. 171–194.

[24] Dong Hun Kim, Hua Wang, and Seiichi Shin. "Decentralized control of autonomous swarm systems using artificial potential functions: Analytical design guidelines". In: *Journal of Intelligent and Robotic Systems* 45.4 (2006), pp. 369–394.

[25] C. Kolon and I. B. Schwartz. "The Dynamics of Interacting Swarms". In: *ArXiv e-prints* (Mar. 2018).

[26] S. Kobayashi and K. Nomizu. *Foundations of Differential Geometry*. A Wiley Publication in Applied Statistics v. 1. Wiley, 1996. ISBN: 9780471157335.

[27] Loris Gilner. *Connecting ROS to the Wolfram Language*. Tech. rep. http://community.wolfram.com/grou/m/t/615905: Wolfram Community, 2016.

Appendix A Mathematica Code

We have made extensive use of Wolfram Mathematica to visualize our results. Attached are several pieces of code which we used to simulate swarms in flat

space and curved space.

Spherical Swarming

This code generates a parabolic potential swarm on a sphere, using the methods described in the paper.

First, we initialize all necessary parameters.

In[1]:=
```
ClearAll["Global`*"] (* Clear all variables and definitions *)
numofbots = 20; (* Number of agents *)
dim = 2; (* Dimension *)
totaltime = 100; (* Total time to simulate for *)
initpos = RandomReal[{1, 1.5}, {numofbots, dim}];
(* Initial positions, generated randomly *)
initvel = RandomReal[{-.1, 0.1}, {numofbots, dim}];
(* Initial velocities, generated randomly *)
sphereRadius = 3;
(* Distance function - depends on the metric tensor,
and difficult to calculate in general*)
dist[pos1_, pos2_] := sphereRadius * ArcCos[Sin[pos1[[2]]] * Sin[pos2[[2]]] +
    Cos[pos1[[2]]] * Cos[pos2[[2]]] * Cos[pos1[[1]] - pos2[[1]]]]
u[pos1_, pos2_] := dist[pos1, pos2]^2; (* Potential function,
based on distance function*)
g = {{sphereRadius^2 * Cos[x[2]]^2, 0}, {0, sphereRadius^2}};
(* Metric tensor for spherical surface *)
ginv = Inverse[g]; (* Inverse of the metric tensor *)
christoffel[a_, i_, j_] := 1 / 2 * Sum[Inverse[g][[a]][[k]] *
     (D[g[[i, k]], x[j]] + D[g[[j, k]], x[i]] - D[g[[i, j]], x[k]]), {k, 2}];
(* Christoffel symbols for the space *)
```

Next, we use these parameters to generate the equations of motion.

In[13]:=
```
eqns = Table[With[{i = i}, {Table[r[i][j]'[t] == v[i][j][t], {j, dim}],
     Table[v[i][j]'[t] == (1 - Sum[v[i][k][t] * v[i][l][t] * (g[[k, l]] /. Table[
                 x[ii] → r[i][ii][t], {ii, dim}]), {k, dim}, {l, dim}]) * v[i][j][t] -
        1 / numofbots * Sum[Sum[(ginv[[j, jj]] /. Table[x[ii] → r[i][ii][t], {ii, dim}]) *
            D[u[Table[r[i][k][t], {k, dim}], Table[r[ii][k][t], {k, dim}]],
             r[i][jj][t]], {jj, dim}], {ii, numofbots}] -
        Sum[(christoffel[j, k, l] /. Table[x[jj] -> r[i][jj][t], {jj, dim}]) *
          v[i][k][t] * v[i][l][t], {k, dim}, {l, dim}],
      {j, dim}], Table[r[i][j][0] == initpos[[i]][[j]], {j, dim}],
     Table[v[i][j][0] == initvel[[i]][[j]], {j, dim}]
    }], {i, numofbots}]; (* Equations for the model *)
unknownFun = Table[r[i][j], {i, numofbots}, {j, dim}];
(* unknowns for the model: positions of each agent *)
```

We solve the resulting second-order differential equation.

In[15]:=
```
sol = First@NDSolve[eqns, unknownFun, {t, 0, totaltime},
     "Method" → {"EquationSimplification" → {Automatic, "TimeConstraint" → 10}}];
(* Solve the diff eq *)
swarmpos = unknownFun /. sol; (* calculate the unknowns given the solution *)
```

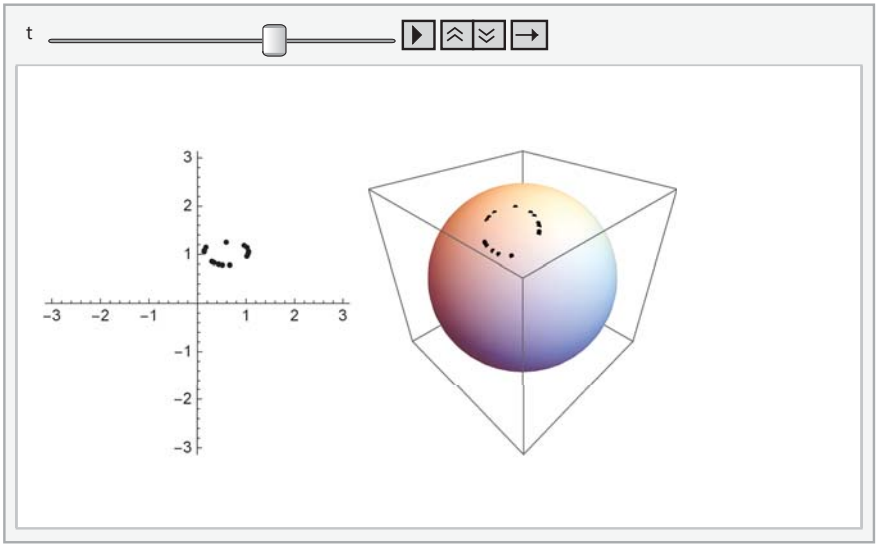
We create parameters for our animation.

In[17]:=
```
project[vec_] :=
  {Cos[vec[[2]]] * Cos[vec[[1]]], Cos[vec[[2]]] * Sin[vec[[1]]], Sin[vec[[2]]]};
(* Define projection onto spherical surface *)
funddom[t_] :=
  Show[{Graphics[Table[Point[Table[swarmpos[[i]][[j]][t], {j, dim}]], {i, numofbots}],
     PlotRange → {{-π, π}, {-π, π}}, Axes → True]}];
(* Define animation on fundamental domain *)
manif[t_] := Show[{Graphics3D[{Sphere[],
     Table[Point[project[Table[swarmpos[[i]][[j]][t], {j, dim}]]], {i, numofbots}]},
    PlotRange → {{-1, 1}, {-1, 1}, {-1, 1}}, ViewPoint → {1, 1, 1}, ImageSize → Large]}]
 (* Define animation on spherical surface *)
```

Finally, we display our animation.

In[20]:=
```
Animate[GraphicsGrid[{{funddom[t], manif[t]}}], {t, 0, totaltime}]
(* Display both animations side-by-side *)
```

Out[20]=



Hyperbolic Swarming

This code generates a parabolic potential swarm on the hyperbolic plane, using the methods described in the paper.

First, we initialize all necessary parameters.

In[39]:=
```
ClearAll["Global`*"] (* Clear all variables and definitions *)
numofbots = 20; (* Number of agents *)
dim = 2; (* Dimension *)
totaltime = 30; (* Total time to simulate for *)
initpos = RandomReal[{0.2, 1}, {numofbots, dim}] - Table[{.4, 0}, {i, numofbots}];
(* Initial positions *)
initvel = RandomReal[{0, 0}, {numofbots, dim}]; (* Initial velocities *)
(* Distance function - depends on the metric tensor,
and difficult to calculate in general*)
dist[pos1_, pos2_] :=
  2 * Log[(Sqrt[(pos2[[1]] - pos1[[1]])^2 + (pos2[[2]] - pos1[[2]])^2] + Sqrt[
        (pos2[[1]] - pos1[[1]])^2 + (pos2[[2]] + pos1[[2]])^2]) /
     (2 * Sqrt[pos1[[2]] * pos2[[2]]])];
u[pos1_, pos2_] := (dist[pos1, pos2])^2; (* Potential function,
based on distance function*)
g = {{1 / x[2]^2, 0}, {0, 1 / x[2]^2}}; (* Metric tensor for hyperbolic space *)
ginv = Inverse[g]; (* Inverse of the metric tensor *)
christoffel[a_, i_, j_] := 1 / 2 * Sum[Inverse[g][[a]][[k]] *
     (D[g[[i, k]], x[j]] + D[g[[j, k]], x[i]] - D[g[[i, j]], x[k]]), {k, 2}];
(* Christoffel symbols for the space *)
```

Next, we use these parameters to generate the equations of motion.

In[50]:=
```
eqns = Table[With[{i = i}, {Table[r[i][j]'[t] == v[i][j][t], {j, dim}],
     Table[v[i][j]'[t] == (1 - Sum[v[i][k][t] * v[i][l][t] * (g[[k, l]] /. Table[
                x[ii] → r[i][ii][t], {ii, dim}])], {k, dim}, {l, dim}]) * v[i][j][t] -
        1 / numofbots * Sum[Sum[(ginv[[j, jj]] /. Table[x[ii] → r[i][ii][t], {ii, dim}]) *
            D[u[Table[r[i][k][t], {k, dim}], Table[r[ii][k][t], {k, dim}]],
             r[i][jj][t]], {jj, dim}], {ii, numofbots}] -
        Sum[(christoffel[j, k, l] /. Table[x[jj] -> r[i][jj][t], {jj, dim}]) *
          v[i][k][t] * v[i][l][t], {k, dim}, {l, dim}],
      {j, dim}], Table[r[i][j][0] == initpos[[i]][[j]], {j, dim}],
     Table[v[i][j][0] == initvel[[i]][[j]], {j, dim}]
    }], {i, numofbots}]; (* Equations for the model *)
unknownFun = Table[r[i][j], {i, numofbots}, {j, dim}];
(* unknowns for the model: positions of each agent *)
```

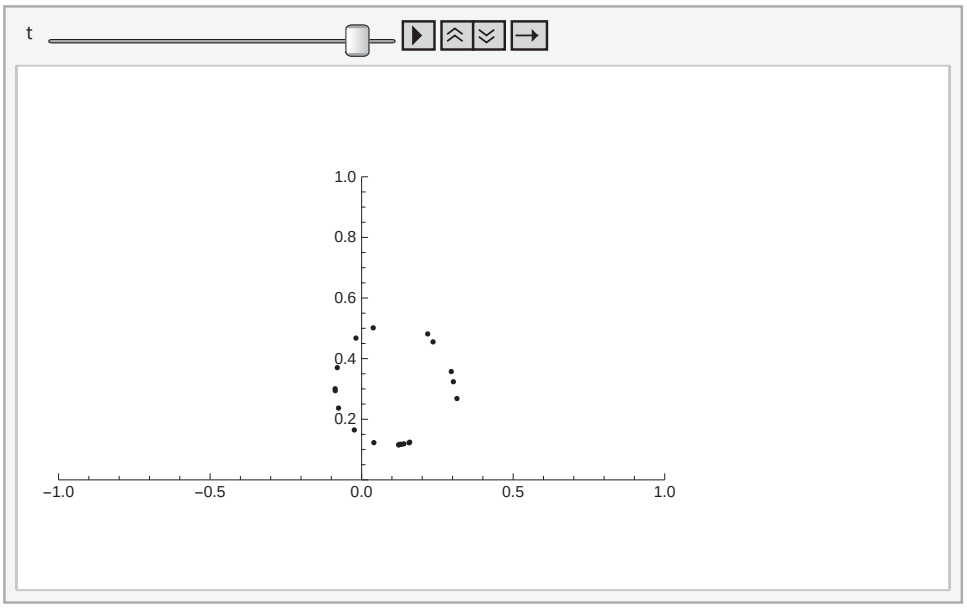
We solve the resulting second-order differential equation.

In[52]:=
```
sol = First@NDSolve[eqns, unknownFun, {t, 0, totaltime},
     "Method" → {"EquationSimplification" → {Automatic, "TimeConstraint" → 10}}];
(* Solve the diff eq *)
swarmpos = unknownFun /. sol; (* Calculate the unknowns given the solution *)
```

We animate the results on the hyperbolic half-plane.

In[54]:=
```
(*Animation on fundamental domain*)
Animate[
 Show[{Graphics[Table[Point[Table[swarmpos[[i]][[j]][t], {j, dim}]], {i, numofbots}],
    PlotRange → {{-1, 1}, {0, 1}}, Axes → True]}], {t, totaltime}, AnimationRate → .5]
```

Out[54]=



Oval Swarming on Flat Metric

This code generates a parabolic potential swarm with oval-shaped limit cycles, using the methods described in the paper.

First, we initialize all necessary parameters.

In[56]:=
```
ClearAll["Global`*"] (* Clear all variables and definitions *)
numofbots = 20; (* Number of agents *)
dim = 2; (* Dimension *)
totaltime = 30; (* Total time to simulate for *)
initpos = RandomReal[{1, 2}, {numofbots, dim}]; (* Initial positions *)
initvel = RandomReal[{-2, 2}, {numofbots, dim}]; (* Initial velocities *)
(* Distance function - depends on the metric tensor,
and difficult to calculate in general*)
g = {{1, 2}, {0, 2}}; (* Metric tensor for hyperbolic space *)
dist[pos1_, pos2_] := [pos1 - pos2(.g.[pos1 - pos2( (* Distance function *)
u[pos1_, pos2_] := [dist[pos1, pos2](;
(* Potential function, based on distance function*)
ginv = Inverse[g]; (* Inverse of the metric tensor *)
christoffel[a_, i_, j_] := 1) 2 * Sum]Inverse[g][[a]][[k]] *
      [D[g[[i, k]], x[j]] + D[g[[j, k]], x[i]] - D[g[[i, j]], x[k]](, {k, 2}/;
(* Christoffel symbols for the space *)
```

Next, we use these parameters to generate the equations of motion.

In[67]:=
```
eqns = Table]With]{i = i}, {Table[r[i][j]'[t] } v[i][j][t], {j, dim}],
      Table]v[i][j]'[t] } [1 - Sum]v[i][k][t] * v[i][l][t] * [g[[k, l]] /. Table[
                  x[ii] == r[i][ii][t], {ii, dim}](, {k, dim}, {l, dim}/( * v[i][j][t] -
         1) numofbots * Sum]Sum][ginv[[j, jj]] /. Table[x[ii] == r[i][ii][t], {ii, dim}]( *
             D[u[Table[r[i][k][t], {k, dim}], Table[r[ii][k][t], {k, dim}]],
              r[i][jj][t]], {jj, dim}/, {ii, numofbots}/ -
         Sum][christoffel[j, k, l] /. Table[x[jj] -> r[i][jj][t], {jj, dim}]( *
           v[i][k][t] * v[i][l][t], {k, dim}, {l, dim}/,
        {j, dim}/, Table[r[i][j][0] } initpos[[i]][[j]], {j, dim}],
       Table[v[i][j][0] } initvel[[i]][[j]], {j, dim}]
      →/, {i, numofbots}/; (* Equations for the model *)
unknownFun = Table[r[i][j], {i, numofbots}, {j, dim}];
(* unknowns for the model: positions of each agent *)
```

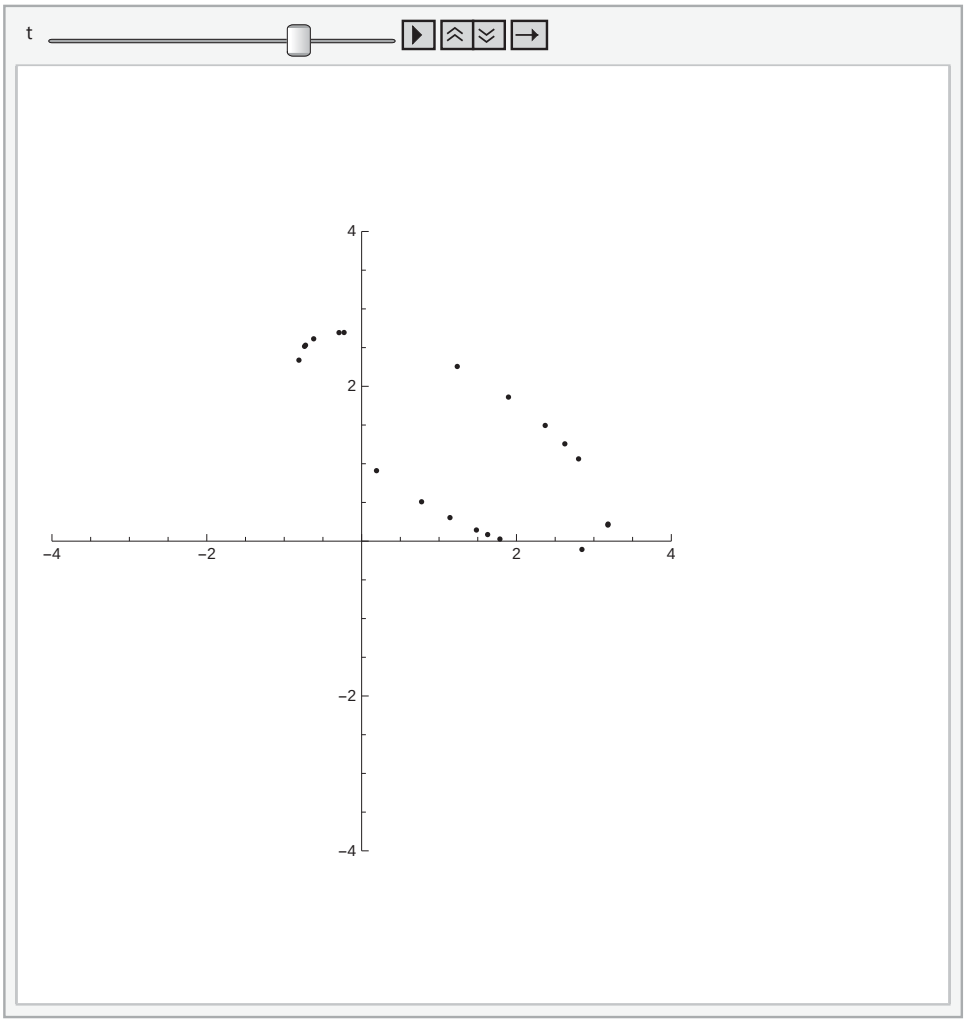
We solve the resulting second-order differential equation.

In[71]:=
```
sol = First@NDSolve[eqns, unknownFun, {t, 0, totaltime},
     "Method" == {"EquationSimplification" == {Automatic, "TimeConstraint" == 10}}];
(* Solve the diff eq *)
swarmpos = unknownFun /. sol; (* calculate the unknowns given the solution *)
```

Finally, we display our animation.

In[73]:=
```
(* Animation on "stretched" space with flat metric *)
Animate[
 Show[{Graphics[Table[Point[Table[Re[swarmpos[[i]][[j]][t]], {j, dim}]], {i, numofbots}],
    PlotRange == {{-4, 4}, {-4, 4}}, Axes == True]}], {t, totaltime}, AnimationRate == .5]
```

Out[73]=



Multi-Hector Control

Run this document in your ROS workspace to control agents spawned with the multihector package.

```
ClearAll["Global`*"]
homepath = "~/my_ros_pkgs"; (* Directory to save/read files to and from *)
saveToFile[M_] := Export[FileNameJoin[{homepath, "dataExchangeMath2ROS.csv"}], M, "CSV"];
(*Format: [velX,velY,velZ,angVelX,angVelY,angVelZ]*)
loadFromFile := Import[FileNameJoin[{homepath, "dataExchangeROS2Math.csv"}]];
(*Format: [posX,posY,posZ,linVelX,linVelY,linVelZ,angVelX,angVelY,angVelZ,yaw]*)

numofbots := Length[dataoutput]; (* Number of agents *)
dim = 2; (* Dimension of the simulation *)
(* Parameters for the model *)
α = 1;
β = 5;
λ = 10;
c = 10/9;
l = 3/4;
h = 4; (* Step size *)
rtf = .18; (* Approximate real time factor of the system. Displayed in Gazebo *)
u[pos_] := c * e^(-Sqrt[Sum[pos[[i]]^2 + .01, {i, 2}]]/l) -
   e^(-Sqrt[Sum[pos[[i]]^2 + .01, {i, 2}]]); (* Swarm potential function *)
(* Gradient of potential function: *)
gradu[pos_] :=
  Grad[u[Table[x[i], {i, 3}]], Table[x[i], {i, 3}]] /. Table[x[i] → pos[[i]], {i, 3}];
(* Measure the velocity from Gazebo. Note that Gazebo gives the
 velocity in terms of global, not local coordinates. *)
velocityMeasured[data_] :=
 Table[{data[[i]][[4]], data[[i]][[5]], data[[i]][[6]]}, {i, numofbots}]
(* Measure position *)
position[data_] :=
  Table[{data[[i]][[1]], data[[i]][[2]], data[[i]][[3]]}, {i, Length[data]}];
(* Equation for the model *)
rhs[data_] :=
 Table[(α - β * Sum[data[[i]][[j]]^2, {j, 4, 6}]) * Table[data[[i]][[j]], {j, 4, 6}] -
   (λ/numofbots) * Sum[gradu[Table[data[[i]][[k]] - data[[j]][[k]], {k, 3}]],
     {j, numofbots}] + {0, 0, 5 - data[[i]][[3]] - data[[i]][[6]]}, {i, numofbots}]
```

```
(* Find desired velocity, rotate it, and then format it to export *)
vDesired[data_] := velocityMeasured[data] + h * rtf * rhs[data];
vRotated[data_] := ParallelTable[
    RotationMatrix[-data[[i]][[10]], {0, 0, 1}].vDesired[data][[i]], {i, Length[data]}];
commandMatrix[vec_] := Table[{vec[[i]][[1]], vec[[i]][[2]], vec[[i]][[3]], 0, 0, 0},
  {i, Length[vec]}]
```

```
(* Run this to create the scheduled task *)
(* REMEMBER TO ENABLE THE MOTORS *)
RunScheduledTask[dataoutput = loadFromFile;
  vmatrix = vRotated[dataoutput];
  saveToFile[commandMatrix[vmatrix]];, h];
```

```
(* Run this to remove all scheduled tasks and stop the robots *)
RemoveScheduledTask[ScheduledTasks[]]
saveToFile[commandMatrix[Table[{0, 0, 0}, {i, numofbots}]]];
```

```
{}
```

```
(* Run this to time how long the process takes. Use a step size a little bigger than
 the output. Make sure all the Gazebo processes are running when you do this,
or else this will underestimate the computation time.*)
AbsoluteTiming[dataoutput = loadFromFile;
  vmatrix = vRotated[dataoutput];
  saveToFile[commandMatrix[vmatrix]];][[1]]
```

```
3.38634
```

Appendix B Python Code

installhectorquadrotor.bash

```
#!/bin/bash

#before running, move this file into your catkin workspace/src folder.
#after installation, cd into your catkin workspace main folder, then run
       source devel/setup.bash
sudo apt-get install ros-kinetic-ros-control
sudo apt-get install ros-kinetic-gazebo-ros-control
sudo apt-get install ros-kinetic-unique-identifier
sudo apt-get install ros-kinetic-geographic-info
sudo apt-get install ros-kinetic-laser-geometry
sudo apt-get install ros-kinetic-tf-conversions
sudo apt-get install ros-kinetic-tf2-geometry-msgs
sudo apt-get install ros-kinetic-joy

git clone -b kinetic-devel https://github.com/tu-darmstadt-ros-pkg/
    hector_quadrotor
git clone -b catkin https://github.com/tu-darmstadt-ros-pkg/
    hector_localization
git clone -b kinetic-devel https://github.com/tu-darmstadt-ros-pkg/
    hector_gazebo
git clone -b kinetic-devel https://github.com/tu-darmstadt-ros-pkg/
    hector_models
git clone -b catkin https://github.com/tu-darmstadt-ros-pkg/hector_slam

sed -i -e 's/option(USE_PROPULSION_PLUGIN "Use a model of the quadrotor
    propulsion system"ON)/option(USE_PROPULSION_PLUGIN "Use a model of
    the quadrotor propulsion system" OFF)/g' hector_quadrotor/
    hector_quadrotor/hector_quadrotor_gazebo/urdf/CMakeLists.txt

sed -i -e 's/option(USE_AERODYNAMICS_PLUGIN "Use a model of the quadrotor
     aerodynamics" ON)/option(USE_AERODYNAMICS_PLUGIN "Use a model of
    the quadrotor aerodynamics" OFF)/g' hector_quadrotor/
    hector_quadrotor/hector_quadrotor_gazebo/urdf/CMakeLists.txt

# this is to deactivate warnings
sed -i -e 's/add_dependencies(landing_action
    hector_uav_msgs_generate_message_cpp)//g' hector_quadrotor/
    hector_quadrotor/hector_quadrotor_actions/CMakeLists.txt

sed -i -e 's/add_dependencies(pose_action
    hector_uav_msgs_generate_message_cpp)//g' hector_quadrotor/
    hector_quadrotor/hector_quadrotor_actions/CMakeLists.txt

sed -i -e 's/add_dependencies(takeoff_action
    hector_uav_msgs_generate_message_cpp)//g' hector_quadrotor/
    hector_quadrotor/hector_quadrotor_actions/CMakeLists.txt

sed -i -e 's/add_dependencies(hector_quadrotor_controllers
    hector_uav_msgs_generate_message_cpp)//g' hector_quadrotor/
    hector_quadrotor/hector_quadrotor_controllers/CMakeLists.txt
```

```
cd ..
catkin_make
source devel/setup.bash
```

createhectorlaunchfile.py

```
#!/usr/bin/env python
"""
Args: nBots
Returns: a launch file with nBots hector quadrotors
"multi_quadrotor_world_nBots.launch"
"""
import sys
import math

def createLaunchFile():
    sideLength = int(math.sqrt(nBots)/2)+1
    botPoses = [(i,j) for i in range(-sideLength,sideLength) for j in
        range(-sideLength,sideLength)]
    fileContent = '''<?xml version="1.0"?>

<launch>
  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="gui" default="true"/>
  <arg name="headless" default="false"/>
  <arg name="debug" default="false"/>
  <arg name="model" default="$(find hector_quadrotor_description)/urdf/
      quadrotor.gazebo.xacro" />

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="paused" value="$(arg paused)"/>
    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
    <arg name="gui" value="$(arg gui)"/>
    <arg name="headless" value="$(arg headless)"/>
    <arg name="debug" value="$(arg debug)"/>
  </include>
    \n
    '''
    for i in range(1,nBots+1):
        robot = '''<!-- BEGIN ROBOT {0}-->
        <group ns="robot{0}">
          <include file="$(find hector_quadrotor_gazebo)/launch/
              spawn_quadrotor.launch">
          <arg name="name" value="Robot{0}" />
          <arg name="tf_prefix" value="robot{0}" />
          <arg name="model" value="$(arg model)" />
          <arg name="x" value="{1}"/>
          <arg name="y" value="{2}" />
          </include>
        </group>
        \n'''.format(str(i),str(botPoses[i][0]),str(botPoses[i][1]))

        fileContent = fileContent + robot

    fileContent = fileContent+'''</launch>'''
    launchFile = open('multi_quadrotor_world{}.launch'.format(str(nBots))
        ,'w')
```

```
    launchFile.write(fileContent)
    launchFile.close()


if __name__ == '__main__':
    try:
        nBots = int(sys.argv[1])
    except ValueError:
        print "The 'nBots' parameteter is required. \n Try '***.py nBots
            '"
        sys.exit()

    createLaunchFile()
```

multihectorListener.py

```
#!/usr/bin/env python
"""
ABOUT:
This python script listens to the gazeboModelStates topic
to get actual positions and velocities of the bots in Gazebosim.
The velocities and positions are saved to a csv file.

Args:
nBots read from the command line

Returns:
We save the following info about each bot to the csv file
[posX,posY,posZ,velLin.x,velLin.y,velLin.z,velAngular.x,velAngular.y,
    velAngular.z,yaw]
Each row contains the info about one bot.
Thus, if we are monitoring 5 bots, the csv will contain 5 rows.

How to execute this script:

1. Place this file in the folder with "filename.csv"
2. In the command line run
python fultibotListener.py nBots
"""

import os
import sys
import tempfile
import csv
import tf

import rospy
# Messages
from std_msgs.msg import String
from gazebo_msgs.msg import ModelStates
from nav_msgs.msg import Odometry
from geometry_msgs.msg import Point, Quaternion

filename = 'dataExchangeROS2Math.csv'

def callback(data):
    #rospy.loginfo(rospy.get_caller_id() + '%s', data)
    names = data.name
    poseDictBots = {data.name[i] : data.pose[i] for i in range(nBots+1)}
    twistDictBots = {data.name[i] : data.twist[i] for i in range(nBots+1)
        }


    #rospy.loginfo('%s', data.pose.pose.position.x)
    with tempfile.NamedTemporaryFile('w', dir=os.path.dirname(filename),
        delete=False) as fake_file:
        with fake_file as fake_csv:
            writer = csv.writer(fake_csv, delimiter=',')
```

```
            for i in range(1,nBots+1):
                orienX = poseDictBots["Robot"+str(i)].orientation.x
                orienY = poseDictBots["Robot"+str(i)].orientation.y
                orienZ = poseDictBots["Robot"+str(i)].orientation.z
                orienW = poseDictBots["Robot"+str(i)].orientation.w


                (roll, pitch, yaw) = tf.transformations.
                    euler_from_quaternion([orienX, orienY, orienZ,orienW])
                velLin = twistDictBots["Robot"+str(i)].linear
                velAngular = twistDictBots["Robot"+str(i)].angular
                posX = poseDictBots["Robot"+str(i)].position.x
                posY = poseDictBots["Robot"+str(i)].position.y
                posZ = poseDictBots["Robot"+str(i)].position.z
                odometryTuple = [posX,posY,posZ,velLin.x,velLin.y,velLin.z
                    ,velAngular.x,velAngular.y,velAngular.z,yaw]
                writer.writerow(odometryTuple)

        tempFileName = fake_file.name
    os.rename(tempFileName, filename)


def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # name are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('listener', anonymous=True)

    #Reading the position directly from Gazebo
    rospy.Subscriber('/gazebo/model_states', ModelStates, callback)

    # Use the following piece to get the position from Odometry. Note
        that it is not very accurate.
    #rospy.Subscriber('/odom', Odometry, callbackOdometry)

    rospy.spin()

if __name__ == '__main__':
    try:
        nBots = int(sys.argv[1])
    except ValueError:
        print "The 'nBots' parameteter is required. \n Try '***.py nBots
            '"
        sys.exit()

    listener()
```

multihectorPublisher.py

```
#!/usr/bin/env python
"""
Args: in the command line run
python multibotPublisher.py nBots

Returns:

This python scripts loops indefinitely and reads from the csv file "
    pubRate" times
per second and publishes the velocity commands to ROS /mobile_base/
    commands/velocity for each bot.

The input file must use the csv format with each row representing the
    velocities
for one bot. Each row should follow the format
[velX,velY,velZ,angVelX,angVelY,angVelZ]

"""
import time
import sys
import csv

# ROS Interface
import rospy
from geometry_msgs.msg import Twist
from geometry_msgs.msg import Vector3

fwdVel = 0.0
angVel = 0.0
pubRate = 30

def talker():
       pub = [None]*nBots
       for i in range(nBots):
           topicName = 'robot'+str(i+1)+'/cmd_vel'
           pub[i] = rospy.Publisher(topicName,Twist,queue_size=10)

       rospy.init_node('act', anonymous=True)
       rate = rospy.Rate(pubRate) #hz
       while not rospy.is_shutdown():
           with open('dataExchangeMath2ROS.csv', 'r') as f:
               try:
                   reader = csv.reader(f)
                   velList = list(reader) # list of the form [velX,velY,
                       velZ,angVelX,angVelY,angVelZ]
                   print "New cmnd", velList
                   print "nBots:", nBots

                   for i in range(0,nBots):

                       linX = float(velList[i][0])
                       linY = float(velList[i][1])
```

```
                    linZ = float(velList[i][2])
                    angX = float(velList[i][3])
                    angY = float(velList[i][4])
                    angZ = float(velList[i][5])
                    linear = [linX,linY,linZ]
                    angular = [angX,angY,angZ]
                    pub[i].publish(Twist(Vector3(linear[0],linear[1],
                        linear[2]),Vector3(angular[0],angular[1],
                        angular[2])))

                except:
                    pass

            rate.sleep()
            #time.sleep(0.1)

if __name__ == '__main__':
    try:
        nBots = int(sys.argv[1])
    except ValueError:
        print "The 'nBots' parameteter is required. \n Try '***.py nBots
            '"
        sys.exit()
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

enablemotors.bash

```
#!/bin/bash
for i in ‘seq 1 $1‘; do
rosservice call /robot$i/enable_motors true
done
```