COMP 461 Final Report - The Drone Rangers

Riley Kuhlman
Rice University
Houston, USA
rak15@rice.edu

Joel Villarino
Rice University
Houston, USA
jav15@rice.edu

Roxana Gutierrez
Rice University
Houston, USA
reg12@rice.edu

Supanat Khaodhiar
Rice University
Houston, USA
sk186@rice.edu

Abstract—**Herding, particularly herding large flocks using multiple shepherds, remains a largely unsolved problem in the computer science literature as well as in industry. Existing hand-coded heuristic based algorithms are often simplistic, and machine learning strategies often result in unwieldy development and results that don't scale well. We detail an attempt to develop a framework for evaluating herding algorithm performance, several iterations of hand-coded herding algorithm development, and a domain-specific tool for ranchers to perform autonomous drone herding using the algorithm. The tool, the Drone Rangers app, would give a rancher the ability to select a location on a digital map of their farm, then attend to other tasks while drones intelligently herd livestock to that goal position. While the primary focus of this specific tool is on sheep, the flexible core for iteratively testing and developing herding algorithms, coupled with a modular drone interface, makes the technology adaptable not only for other animals such as cattle, goats, pigs, and horses, but also for entirely different domains like herding trash on the ocean surface, guiding wildlife away from hazardous areas, or even orchestrating autonomous vehicles in crowded environments. We present results for the algorithm's performance, as well as a description of the architecture, performance, and features of the frontend and backend tools produced.**

I. Introduction

Humans have been herding livestock for over 10,000 years. Today, in a world with self-driving cars and ChatGPT, we still use the same herding methods we did all those years ago. Some ranchers herd on horseback - putting their physical safety at risk. Using herding dogs can minimize that, but a human still needs to be out commanding the dog step by step, so herding remains incredibly time-intensive. By making use of emerging robotic technology and artificial intelligence, we hope to innovate this practice that is as old as agriculture itself.

Herding is the practice of managing and moving livestock across large areas of land, referring to both the general ongoing care of free-ranging animals as well as the process of their active relocation as a group. On modern ranches and farms, three primary herding methods remain common: the use of trained herding dogs, horseback riding, and mechanized alternatives such as motorbikes or all-terrain vehicles (ATVs). While effective, these approaches entail substantial costs (such as the upkeep of additional working animals) and pose significant safety risks to riders navigating uneven terrain at speed.

Herding has relied on animals such as dogs and horses since its earliest practice, and animal use has persisted through the large-scale cattle drives of the 19th and 20th centuries where large groups of cattle were transported between pastures and markets hundreds of miles apart. Although herding today typically occurs on smaller scales—within a single ranch or between adjacent lands—the same methods remain central.

In recent decades, precision agriculture has increasingly adopted robotic systems to improve efficiency, reduce labor burdens, and enhance safety. Unmanned aerial vehicles, i.e. drones, are seeing applications in crop monitoring, mapping, and livestock surveillance, with demonstrated success in extending farmers' reach and providing new spatial perspectives. However, their application has been primarily observational rather than interventionist.

Our project aims to address this gap by leveraging drone technology to innovate herding practices. By extending the existing technology beyond monitoring, we seek to reduce operational costs, improve safety, and introduce new levels of precision and automation in livestock management.

II. Related Work

A. Competition in Industry

The current commercial landscape for drone-based livestock management is dominated by "Precision Livestock Farming" (PLF) solutions focused on observation rather than intervention. Major industry players, such as Cargill with its *CattleView* technology, utilize drones primarily for inventory tracking, health assessment, and feed optimization [1]. Similarly, hardware manufacturers like DJI offer enterprise packages (e.g., the Mavic 3 Multispectral) tailored for aerial surveying and vegetation indexing rather than active animal interaction. While these systems effectively extend a rancher's visual range, they remain passive tools that require human labor to act on the data they collect.

A smaller segment of the market has begun to address active herding. BeeFree Agro, for example, offers the "Joe" system, which markets itself as a "flying cowboy" capable of moving cattle autonomously [2]. While the system has been demonstrated herding cattle, reliability remains a concern, and ranchers are required to remain in close proximity to the drone should something go wrong. Since the market share for ranching activities is enormous and BeeFree Agro's system requires some level of manual oversight, there is ample room for competition in this space. In particular, we believe we can find several aspects in which our product has a competitive edge:

1) Features for industrial-scale farms, e.g. support for multi-drone path-planning,

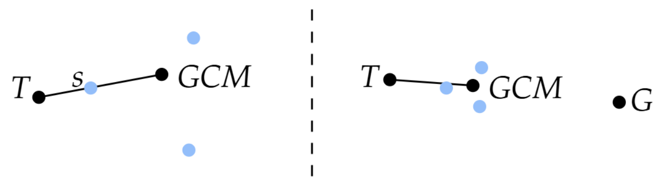


Fig. 1. Diagram depicting the collecting (left) and driving (right) phases of Strombom's algorithm. Sheep are represented as blue dots, and the position that the algorithm directs the drone towards is T. On the left, we show how the algorithm picks a sheep s farthest from the GCM and directs the drone to a target where it would push the sheep towards the GCM. On the right, we show how the drone is directed towards a target that will propel the center of mass of the herd towards the goal G.

2) automated task scheduling to minimize the need for continuous operator supervision, and
3) better coordination and management tools for working with multiple drones on multiple jobs.

B. Herding Algorithms Literature

Our heuristic herding strategy, described in Section III-A, implements the sheepdog interaction models proposed by Strombom [3]. The algorithm dynamically switches between two behaviors based on the dispersion of the flock relative to its Global Center of Mass (GCM).

1) **Collecting Phase:** Triggered when any sheep is further than distance f_N from the GCM. The drone targets the sheep s furthest from the GCM, moving to a position T behind s relative to the GCM to urge it back toward the flock (Fig. 1, left).
2) **Driving Phase:** Active only when all sheep are within the threshold f_N. The drone positions itself behind the GCM relative to the goal to propel the consolidated flock towards the desired destination (Fig. 1, right).

In experimenting with the algorithm, we encountered several problems:

1) deadlock switching between collecting and driving,
2) deadlock switching between multiple clusters, and
3) herding past and circling the target position.

The problems and the resolution we arrived at will be discussed in Section III-A, and showcase the main technical innovation of our project.

III. Methodology

We focused first on developing a robust herding algorithm (Section III-A), improving the realism of the simulation it interacts with (Section III-B), and optimizing the performance of that simulation (Section III-C). We built a backend server (Section III-D to expose this behavior and then a frontend Drone Rangers app to interact with it (Section III-E).

A. Heuristic-Based Herding

Development on the algorithm proceeded by incrementally modifying Strombom's algorithm to address the problems identified above.

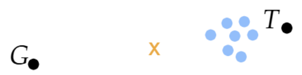


Fig. 2. The drone (an orange cross) attempts to reach position T in order to push the flock of sheep (blue dots) towards G. However, flying towards position T pushes the sheep away from the goal, often causing long delays before herding task completion.

1) Problem 1: Deadlock Switching Between Collecting and Driving: In the strategy presented by Strombom, the distinction between the strategy is discrete—the algorithm is only ever attempting to collect the herd or to drive the herd, never both. We found that for large flocks, this would result in frequent toggling between phases, with the drone spending most of its time moving back and forth from the sheep farthest from the GCM to its desired driving position. For our strategy, we removed the driving phase and modified the way that the collecting phase determines which sheep a drone should target. Rather than picking the sheep with the maximum distance from the GCM, the drone will target the sheep maximizing the below score function f:

$$f(s) = a_1 d(s, GCM) + a_2 d(s, G)$$

where d is the distance function, s is a given sheep, a_1 and a_2 are tunable parameters, and G is the goal position for the herd. By removing the discrete state switches from the algorithm, we eliminated one source of maladaptive behavior from the algorithm. Notice that while the algorithm only ever explicitly pushes sheep directly towards the GCM, pressuring sheep farther from the goal towards the GCM will also result in overall motion towards the goal.

2) Problem 2: Deadlock Switching Between Multiple Clusters: We also observed that when there were two clusters of approximately equal size, Strombom's algorithm would alternately target sheep in either cluster, often flying back and forth between clusters since the sheep in either cluster had approximately the same distance from the GCM. To mitigate this problem, we introduced the distance from a drone r to a given sheep as another term into our score function f:

$$f(s, r) = a_1 d(s, GCM) + a_2 d(s, G) + a_3 d(s, r).$$

3) Problem 3: Herding Past and Circling the Goal Position: We found that Strombom's algorithm would frequently arrive in a state where the drone was targeting a position behind the flock, but it was stuck on the other side of the flock - since the drone flies directly towards the target and constantly repels the sheep, the flock's natural cohesion would prevent the drone from pushing through—a typical situation is shown in Figure 2.

Rather than attempting to path plan the drone around the herd, we relaxed one of the requirements of the original paper. Instead of the flock being repulsed by the drone at all times, we allow the algorithm to determine whether the drone should apply repulsion or not. This constrains the Drone Rangers app and internal algorithm to only have compatibility with drones that support conditional repulsion. In general, we suspect that this will often be the case—drones that apply repulsion via a sound can pause the sound or simply fly higher to have less effect on the flock. A limitation of our model is that the switch between applying repulsion and not applying repulsion is instantaneous, whereas there would likely be a transitionary period when working with actual animals.

Our modified herding algorithm instructs drones to only apply repulsion when they're within some threshold distance of their target position. In this way, the drone can simply fly over the herd to reach its target location, rather than pushing along behind it.

While this modification resulted in much higher success rates, there was still often suboptimal behavior very near to the goal—e.g. pushing the herd in circles around the goal and pushing the farther half of the herd out of the target area while trying to get the closer half to be inside the target area. To address these issues, we changed the tunable weights in the score function to be calculated dynamically rather than fixed constants. We used the following computation for each weight:

$$C = \frac{f_N}{\text{mean}(d(s, \text{GCM}))}, \quad R = \frac{\max(d(s, \text{Goal}))}{f_N}$$
$$a_1 = \text{interp}(0.8, 0.6, 0.3, 1.5, C) \cdot \text{interp}(0.5, 1.0, 1.0, 3.0, R)$$
$$a_2 = \text{interp}(0.2, 0.4, 0.3, 1.5, C)$$
$$a_3 = \text{interp}(1.0, 0.2, 0.3, 1.5, C) \cdot \text{interp}(0.2, 1.0, 2.0, 4.0, R)$$

where interp is defined by

$$\text{interp}(a, b, t_1, t_2, t) = a + (b - a) \cdot \max\left(0, \min\left(1, \frac{t - t_1}{t_2 - t_1}\right)\right).$$

Effectively, this calculation changes which aspects the herding algorithm prioritizes as the state of the herding shifts over time. The algorithm weights more heavily which sheep are farthest from the goal whenever all of the sheep are very near to goal and puts less emphasis on targeting sheep close to the drone. This dynamic weighting helped ameliorate some of the difficulties the algorithm tended to encounter near the finish conditions.

4) Extension to Multiple Drones: As a last extension to the algorithm, we modified the score function to support assigning different target sheep to multiple drones. We define

$$f(s, r) = a_1 d(s, GCM) + a_2 d(s, G) + a_3 d(s, r) + a_4 b(s, r)$$
$$b(s, r) = \begin{cases} 1 & \text{if drone } r \text{ is the closest drone to sheep } s \\ 0 & \text{otherwise} \end{cases}$$

Effectively, a drone is much more likely to choose to target a sheep if it's the closest drone to that sheep. This naturally results in divide-and-conquer strategies in the cases where there are multiple clusters, and typically results in good results when multiple drones attempt to drive one large cluster.

B. Bespoke Sheep Behavior Model

In order to iterate our development of the algorithm, it's critical to have a model of agent behavior in flocking that's accurate to real-world entities. Again, we started with Strombom's paper [3], which described a slightly modified version of the standard Reynolds' model of a flock [4]. In this model, sheep exhibit grazing behavior by default, only flocking when a drone is near. Grazing behavior is characterized by each entity moving in a random walk that steps p_{graze} proportion of the model's iterations—we compute grazing velocity by

$$v_{\text{graze}} = \begin{cases} 0.2 \cdot \boldsymbol{\eta}, & \text{when } \boldsymbol{\mu} < p_{\text{graze}} \\ 0, & \text{otherwise} \end{cases}$$

where $\boldsymbol{\eta}$ is a vector in a random direction with magnitude from the standard normal distribution and μ comes from a uniform distribution over $[0, 1]$.

We compute flocking velocity according to the following rules, which come from Strombom's paper:

1) **Cohesion:** To maintain group integrity, each agent calculates the centroid (center of mass) of its k_{NN} nearest neighbors. An attractive force vector F_a pointing toward this local centroid is applied. This ensures the agents remain aggregated as a flock rather than dispersing.
2) **Separation:** To prevent entity overlap, each agent is repulsed by every other agent within a radius r_a. Vectors away from every other agent, inversely proportional to the distance to the other agent, are summed and applied to the movement of a given agent. Call the net vector F_a.
3) **Shepherd Avoidance:** This rule extends the standard model to include an external threat. If a shepherd agent is detected within the radius r_s, a strong repulsive force is applied away from the shepherd's position. This force is weighted significantly higher than peer attraction to prioritize survival and escape behavior. Call this force F_s.
4) **Inertia:** To prevent erratic, jittery movement and simulate physical mass, a momentum term is applied based on the agent's velocity from the previous time step $t-dt$. Call this force F_m.

We compute the next position for each sheep as a weighted sum of these direction vectors, allowing us to tune the influence of each rule:

$$H = w_a F_a + w_r F_r + w_s F_s + w_m F_m$$

where H is a vector that we use to set velocity v for a given agent:

$$v_{\text{flock}} = v_{max} \cdot \frac{H}{\|H\|}.$$

While Strombom's initial paper defined a sheep to be flocking if and only if it was within r_s of a drone, we instead allowed a sheep to be partially in between the grazing and flocking state. Every sheep has some value $\alpha \in [0, 1]$, which is closer to 1 when the sheep should be influenced more by flocking behavior, and closer to 0 when a sheep should be more influenced by grazing behavior. This allows us to have more complex state transitions between flocking and grazing - particularly, it means that our simulation enforces that sheep remain in a herd for a period of time after a drone finishes herding them, which qualitatively matches the behavior of real-world sheep. Mathematically, we compute v_{graze} and v_{flock} for every sheep, and set the velocity at every iteration to

$$v = \alpha v_{\text{flock}} + (1 - \alpha) v_{\text{graze}}.$$

For each sheep, we update the α value at every iteration based on what it was in the previous step, α_{prev}, and that sheep's proximity to drones. We compute the affect of a given drone r on a sheep s as being proportional to $\max\left(0, 1 - \frac{d(r,s)}{r_s}\right)$, and we combine the effect of a set of multiples drones R as being proportional to

$$\alpha_{\text{target}} = 1 - \prod_{r \in R} \min\left(1, \frac{d(r,s)}{r_s}\right).$$

For every iteration, we interpolate α progressively closer to α_{target} from α_{prev}:

$$\alpha = \text{interp}(\alpha_{\text{prev}}, \alpha_{\text{target}}, 0, T, \Delta t).$$

where Δt is the timestep for that iteration, and

$$T = \begin{cases} 2 & \text{when } \alpha_{\text{target}} > \alpha_{\text{prev}} \\ 60 & \text{otherwise} \end{cases}$$

We define T in cases so that it takes approximately 2 seconds for a sheep to go from α_{prev} to α_{target} when it would be exhibiting more flocking behavior, and 60 seconds when it would be exhibiting less. This mirrors the initial reaction that sheep have to encountering a threat, and the time that it takes for their response to that threat to wear off.

C. Simulator Performance

A key requirement for this project was a world simulator that was both performant and easy to iterate on. Because we chose Python for rapid development, careful performance engineering was essential. Each simulation step updates all sheep according to local flocking rules and interactions with nearby agents, making the cost of neighbor search the critical bottleneck. Across several optimization stages, vectorization, JIT compilation, and an adaptive neighbor-caching strategy, we improved simulator performance by roughly $50\times$–$130\times$ compared to the original implementation. *Note: All performance profiling was completed on a Macbook Pro M1 running Python 3.14.0*

1) Baseline: Original Object-Oriented Implementation: The simulator originally followed a classic object-oriented design. Each sheep was a `Sheep` instance storing its own state, and each simulation step iterated over a Python list of these objects. Although simple and extensible, this structure incurred heavy overhead from Python-level looping, dynamic attribute access, repeated small NumPy calls, and poor memory locality. We benchmarked this legacy implementation using a dedicated profiling harness to determine the per-step runtime.

TABLE I
LEGACY OBJECT-ORIENTED SIMULATOR PERFORMANCE.

Flock Size	Legacy Runtime (ms)
64	17.7
128	68.6
256	274.4
512	1089.1
1024	4402.7

These measurements illustrate that the simulator became prohibitively slow even before flocking dynamics themselves became interesting, as interpreter overhead dominated the workload.

2) Vectorization: Structure-of-Arrays (SoA) Rewrite: To eliminate Python dispatch overhead and enable batch computation, we refactored the simulator into a data-oriented, vectorized Structure-of-Arrays (SoA) layout. All positions, velocities, and intermediate state were stored in large contiguous NumPy arrays:

```
P: (N, 2) positions
V: (N, 2) velocities
```

This improved cache locality, enabled batched operations, and removed tens of thousands of Python object accesses per step.

TABLE II
PERFORMANCE OF THE VECTORIZED (SOA) SIMULATOR.

Flock Size	Vectorized (No JIT) Runtime (ms)
64	4.20
128	11.3
256	19.6
512	43.6

This step alone yielded a 4–25$\times$ improvement, but pairwise neighbor search still dominated the runtime.

3) Compilation: Numba-JIT Accelerated Kernels: Once the simulator used contiguous arrays with no Python objects, the entire update pipeline became compatible with Numba's `@njit` compilation. This eliminated NumPy dispatch overhead, fused loops, and allowed tight machine-code execution.

This provided an additional 5–30$\times$ improvement beyond vectorization alone. At this stage, nearly all Python overhead had been eliminated, leaving the inherent $O(N^2)$ neighbor-search cost as the primary bottleneck.

TABLE III
PERFORMANCE AFTER NUMBA JIT COMPILATION (PER-STEP RUNTIME).

Flock Size	Vectorized + JIT Runtime (ms)
64	0.350
128	0.380
256	2.17
512	8.50
1024	32.0

4) Unsuccessful Approaches: With the JIT-optimized baseline established, we next explored algorithmic alternatives to reduce the neighbor-search cost. Two promising ideas, uniform spatial grids and dense distance caching, proved slower in practice due to Python overhead or unfavorable constant factors.

a) Uniform Grid Neighbor Cache.: Spatial hashing recreated Python lists and dictionaries each step, preventing JIT optimization. Grid construction alone took 1.35 s in a scenario where the naive brute-force approach required only 0.62 s, making this method $1.9\times$ slower overall.

b) Dense $O(N^2)$ Distance Cache.: Constructing a full distance matrix removed redundant scalar operations but introduced new bottlenecks from repeatedly building dense arrays and invoking `np.argpartition`. This caused a 29% regression, with the cache builder consuming 58% of runtime.

Although unsuccessful, these experiments were necessary to rule out algorithmic alternatives before committing to an optimized brute-force design.

5) Successful Optimizations: With Python overhead eliminated and alternative structures proving slower, the most effective strategy was to retain brute-force neighbor search but amortize its cost for large flocks.

a) Hybrid Neighbor Cache for Large Flocks.: We introduced an ε-movement neighbor cache, allowing each agent to reuse its previous neighbor list unless it moved beyond a displacement threshold. A Numba-compatible Linear-K selector maintained the k nearest neighbors without Python objects or heap structures.

We benchmarked each configuration over 300 steps and computed mean per-step runtimes:

TABLE IV
MEAN PER-STEP RUNTIME WITH THE ε-MOVEMENT NEIGHBOR CACHE (300-STEP BENCHMARK).

Flock Size	Cache OFF (ms)	Cache ON (ms)	Speedup
64	1.07	1.14	-6.1%
128	2.19	2.22	-1.5%
256	4.76	4.58	$+3.8\%$
512	11.0	9.08	$+17.3\%$
1024	27.9	19.4	$+30.5\%$

These results validate the intended behavior: the cache introduces overhead at small N, provides negligible benefit around the crossover point at $N = 256$, and yields substantial improvements for dense flocks.

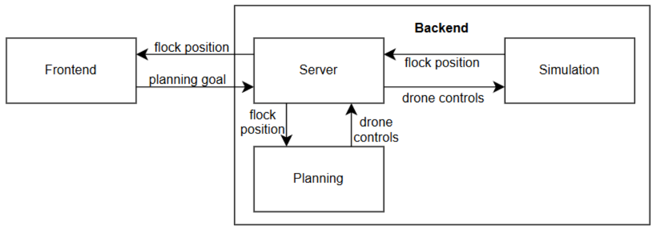


Fig. 3. Diagram showing the main components of the backend and how a typical frontend would interact with them.

6) Final Strategy and Impact: The simulator now employs an adaptive strategy:

- For $N < 256$: use pure Numba-compiled brute-force search (minimal overhead).
- For $N \geq 256$: use the ε-movement neighbor cache with Linear-K selection.

Across all flock sizes, the final simulator delivers sustained $50\times$–$130\times$ speedups over the original design. At very large scales, the adaptive neighbor-cache provides an additional 17–30% improvement, expanding the practical limits of the system. These optimizations transform the simulator from a prohibitive bottleneck into a scalable foundation for ongoing research, with clear pathways toward GPU acceleration, signed-distance-field obstacle models, and significantly larger flock sizes.

D. Backend Design

The backend is organized into modular, highly-reusable components, roughly laid out in Figure 3. As the business scales from focusing solely on the original Drone Rangers app to providing additional swarm-control apps in different fields, we'll reuse the same core parts of the backend: the jobs database and scheduler, the herding algorithm, and the swarm simulation. Note that the swarm simulation will not be used directly by production apps—rather, these apps will communicate directly with physical hardware. However, it will be useful to begin with a simulation when initially developing each new domain-specific application.

Our internal job database and scheduler maintains the status of each job, and keeps the data from each job synchronized with persistent storage. A job's status must be one of SCHEDULED, PENDING, RUNNING, COMPLETED, or CANCELED (as shown in Figure 4). Jobs will initially be SCHEDULED, and will transition to RUNNING once their scheduled time is reached. The user can choose to pause or unpause jobs, which will cause a transition from RUNNING to PENDING or vice versa. Upon completion, jobs transition to COMPLETED and are no longer shown; likewise for cancellation. The jobs are fed into the herding algorithm so it can determine which drones to allocate to which jobs, prioritizing jobs that started earlier.

The herding algorithm and swarm simulation are decoupled, which facilitates the ability to swap sim components

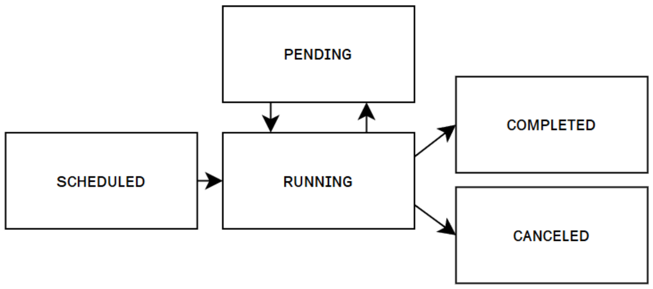

$$\begin{tikzcd} & \text{PENDING} \arrow[d, shift right] & \text{COMPLETED} \\ \text{SCHEDULED} \arrow[r] & \text{RUNNING} \arrow[u, shift right] \arrow[ur] \arrow[dr] & \\ & & \text{CANCELED} \end{tikzcd}$$

Fig. 4. Diagram displaying possible statuses that a job could have and transitions between those statuses.

out for real hardware. The general flow of communication between the components is as follows: 1) The herding algorithm receives the state of the world and returns the action that the drone should take. 2) The update for the drones is applied to the state of the world. 3) The swarm simulation steps through how the sheep should be updated according to the new drone positions. Implementation specifics are discussed in III-A and III-B.

E. Drone Rangers App Design

The customer facing application gives ranchers the ability to manage and view herding jobs on their farm. To prioritize usability and intuitive flow, the app is organized into three main views navigable via a tabbed header:

1) Live View: A farmer opens the app to this page that displays a live aerial view of their livestock, drones, and herding jobs. Each herding job is represented on the image of the land as a pin point icon at the positioning of the job's target, colored red if currently active (i.e. drones are actively herding livestock to this location), and gray otherwise. In addition, herding jobs are displayed as an ordered list to one side of the map, allowing the farmer to scroll through and select any one job to view individually. In the event that a large number of total herding jobs are scheduled, the farmer can reduce clutter on the map by choosing to only display jobs that will occur in a specified time interval.

2) Schedule View: A farmer can navigate to this page to view and manage their herding job schedule. To create a new job, the farmer must click on the single "Create Job" button at the top of the page and input their desired parameters of the job via a pop up modal. Jobs can be created with a scheduled status of "immediately" to be started as soon as there is no other job running or "scheduled" to be started at a specified time. Upon submission, a request is sent to the backend to add the job to the database, and the job is displayed on the calendar at its scheduled time. To edit a job, the farmer can click on the desired job in the calendar and input the desired modifications via a pop-up modal.

3) Drone Management: A farmer can view their fleet of usable drones from this page. When a farmer acquires a new drone, it can be added by name to their fleet to be utilized in future herding jobs. When the drone is retired, it can be deleted from the deleted from the database. This page also provides a live focused view of the currently selected drone.

IV. Evaluation

In our initial project proposal, we identified several goals for this project; we evaluate our progress for each of them below.

A. Herding Algorithm

The proposal identified a separate collection task and a driving task on which to evaluate our model. For the collection task, the herding algorithm is required to be able to autonomously gather 100 flock agents randomly distributed around a 2-square-mile area. We successfully demonstrated excellent progress in this area, achieving a 100% completion rate in this task using our evaluation framework. In fact we were able to demonstrate potential much greater scalability for our algorithm, achieving an 80% completion rate for completing the same task in thirty simulated minutes with 500 sheep.

For the driving task, we aimed for our algorithm to be functional with multiple drones in herding a gathered flock of size 100 from one point to another. We successfully demonstrated this behavior in class, driving a herd of sheep to a user-defined endpoint as shown in Figure 7. However, in development we typically combined the collection task and driving task into a composite task, which has the characteristics given below (unless otherwise specified):

1) $N = 100$ sheep randomly distributed in a 250 meter by 250 meter area.
2) 3 drones starting in the bottom left corner.
3) A goal position for the sheep at $x = 240, y = 240$, in the top right corner of the area, and the goal radius is given by $4 \cdot \sqrt{N}$. This tends to be a radius that is just big enough to fit all of the sheep with comfortable spacing.
4) Sheep are driven by the same herding simulation as given in III-B, although we frequently vary the k_{NN} parameter.
5) A task is evaluated as a success if it is able to herd the entire flock into less than 2000 seconds, or approximately 30 minutes.

The results of running this composite test on a variety of different input conditions are shown in Figure 5, and a full recording of a typical run is available in Supplemental Material 1.

To further analyze our effectiveness in the driving task, we've evaluated how the amount of time required to herd varies as the distance from the target increases. As expected, the algorithm seems to drive sheep towards the goal at a roughly constant speed regardless of how far away the goal is, as displayed in Figure 6. Roughly estimated, the algorithm drives sheep at 1km/hr, which is comparable to existing drone-based herding algorithms.

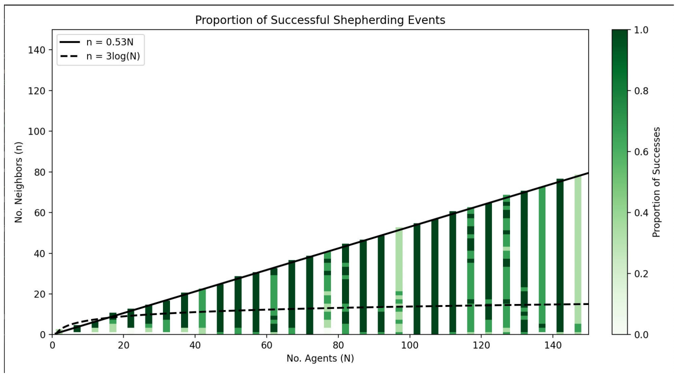


Fig. 5. Graph showing the proportion of successful herding tasks in a variety of situations. For every colored square on the graph, we ran the described simulated scenario with the number of sheep indicated on the x-axis and the sheep's behavior model set up to have the k_{NN} value indicated on the y-axis.

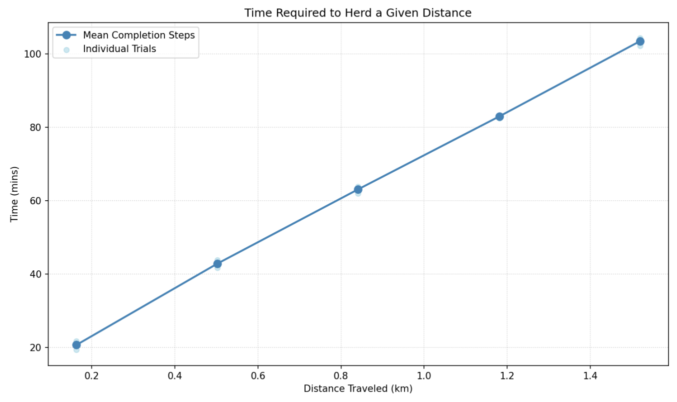


Fig. 6. Graph displaying the amount of time required to shepherd a flock of 100 sheep a given distance under the default assumptions of the simulation.

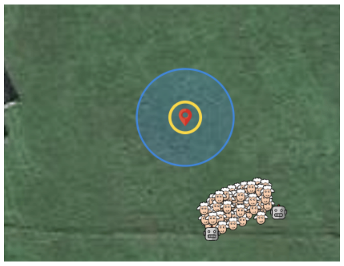
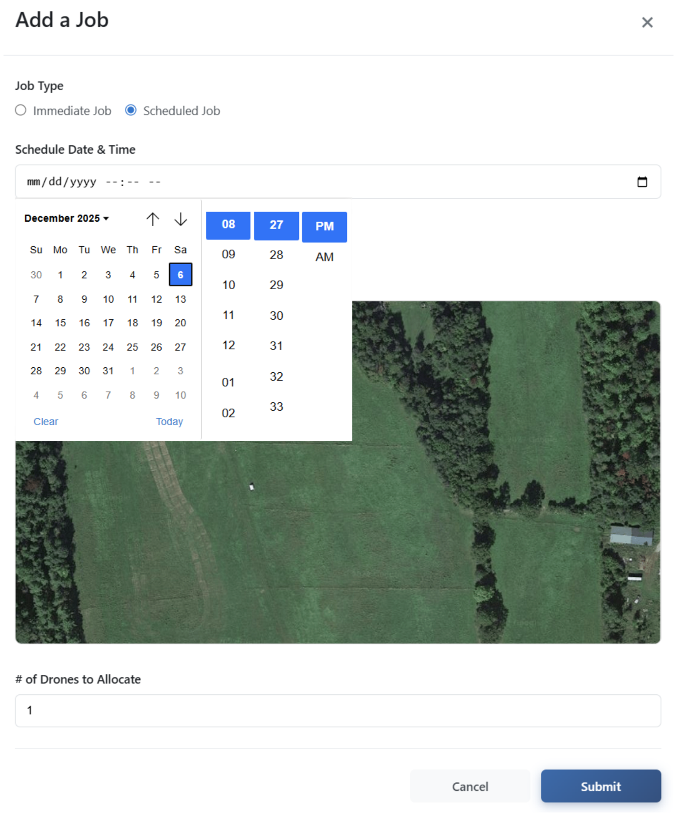


Fig. 7. Multiple drones work to drive a cluster of sheep towards a goal position. Taken as a screenshot of the Drone Rangers app.



Add a Job

Job Type

Immediate Job [✓ Scheduled Job]

Schedule Date & Time

mm/dd/yyyy --:-- --

# of Drones to Allocate

1

Fig. 8. Display used to create a new job, showing the calendar view that can be used to set the date/time when it should execute.

B. *End-to-End Workflow Validation*

Our user-facing goal for the project was for users to be able to use a frontend GUI to choose a destination for the herd on a map, then the app to schedule a corresponding herding task, and for this herding task to successfully run to completion. We've demonstrated this, and in fact have introduced several additional options. For any herding task, users are able to specify a date and time for the task to execute, the number of drones to use to execute this task, and the size and position of the herding goal region.

V. Conclusion

This project presents a software platform capable of directing autonomous drones to manage sizable flocks within a comprehensive simulation-backed control framework. The system supports interactive task scheduling, real-time monitoring, and dynamic adjustment, and it establishes a modular foundation for domain-specific applications such as the Drone Rangers App. To ensure extensibility beyond livestock management, we developed a general-purpose simulation environment that shares the same backend components and exposes configurable parameters for flocking behavior, herding strategy, and world dynamics. This complementary pairing of an operator-facing application and an internal simulation tool demonstrates the adaptability of the system and creates a foundation for research, development, and eventual real-world deployment across a broad range of multi-agent settings.

A central contribution of this work is the set of improvements introduced to Strombom's original herding model. By replacing the simple nearest-agent heuristic with a more expressive scoring function, the system can identify target sheep more effectively and avoid many of the deadlock scenarios that limit the traditional algorithm. We also implemented a modified flocking model that more accurately reflects observed sheep behavior, which yields smoother group motion and more realistic responses to drone influence. Together, these algorithmic enhancements increase reliability, reduce intervention complexity, and create a stable substrate upon which scalable and predictable herding behavior can be built. They also position the platform within a largely unexplored space in agricultural robotics as existing systems focus almost exclusively on environmental monitoring, while our platform demonstrates that software-driven, autonomous intervention is tractable and repeatable in simulation at scale.

Usability played a deliberate and significant role throughout the system design. The interface is structured around established heuristics that emphasize clarity, consistency, and user control. A shallow navigation hierarchy helps operators complete tasks efficiently without encountering irrelevant complexity, while continuous visual feedback allows them to verify whether an action has been executed successfully. Labeling conventions and interface grouping reinforce expected behaviors, and error prevention features reduce the likelihood of misconfiguration or invalid commands. These considerations ensure that a user does not need expertise in robotics, simulation systems, or multi-agent control to manage tasks effectively. By reducing cognitive overhead, the platform becomes a viable tool for domain experts such as ranchers and technicians who require reliability and transparency more than algorithmic detail.

Scalability in both computation and deployment is essential to support the long-term viability of autonomous herding systems. The core herding algorithm operates in linear time with respect to the number of agents, which ensures predictable performance for both small testing scenarios and large commercial-scale flocks. The simulation environment is constructed with a similar philosophy. While optimized for rapid iteration and small to medium experimental sets, it can be expanded to support distributed computing if needed. This flexibility allows the system to grow without requiring a full architectural redesign.

Although the present work is simulation-based, the underlying software architecture is designed for real-world deployment that will exist in a dedicated edge compute unit. This unit would execute all latency-sensitive tasks locally, including telemetry handling, flock-state updates, and real-time control decisions, while the centralized infrastructure would provide diagnostics, configuration management, and scheduled updates. This division of responsibilities creates a scalable business model in which each installation carries its own computational load. As a result, the cost of supporting new deployments does not increase proportionally with the number of clients, and reliability is sustained even in rural environments with intermittent network connectivity.

The broader social impact of this work emerges from its potential to reshape both agricultural labor and general-purpose multi-agent coordination. In the agricultural context, autonomous herding reduces the physical strain and danger associated with working directly with large animals. It also mitigates the labor shortages that challenge smaller farms, which often lack access to advanced tools that industrial operations can afford. By combining affordable automation with accessible user interaction patterns, the platform supports more equitable participation in modern agricultural practices.

At a wider scale, the control logic developed here generalizes to many multi-agent environments where decentralized entities must be guided safely and predictably. The same principles that enable cohesion, separation, and directed movement within a flock are directly applicable to other domains in which agents respond to local interactions rather than global instructions. With appropriate domain-specific adaptations, the system can be extended to coordinate drone swarms for environmental containment, influence human movement during emergency evacuations, and guide autonomous aerial vehicles in search and rescue scenarios. These applications illustrate that the software is not limited to agricultural use but instead serves as a foundation for managing complex, agent-driven dynamics in a variety of critical environments.

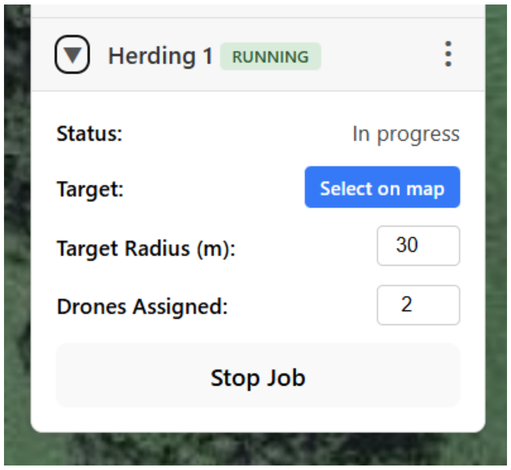
By integrating algorithmic robustness, clear usability principles, predictable performance scaling, a credible deployment pathway through an edge compute unit, and meaningful societal impact, this project establishes a framework that can support both immediate agricultural applications and future advances in multi-agent autonomy. The work outlined here demonstrates not only that autonomous herding is feasible in simulation at scale, but also that the underlying ideas provide a versatile and extensible toolset for addressing broader challenges in distributed control and coordinated behavior.

VI. Contributors and Contributions

A. *Contributions (Riley Kuhlman)*

Before the midterm presentation, I focused primarily on herding policy implementation. I modified the existing herding algorithm to flexibly target sheep according to a variety of different features, and I implemented code for extending our algorithm to function with multiple drones. I developed a framework for evaluating the performance of the algorithm, which was used to generate the data for Figures 5 and 6, as well as to aid in various development decisions throughout the semester. I also assisted in cleaning up a few components of the simulation, fixing bugs and tuning parameters to better match sheep behavior.

I also developed the first iteration of our backend, which maintained a single simulation and handled to API requests from the frontend to interact with it. On the frontend side, I built the first iteration of the live jobs panel, the final version of which is shown in Figure 9.



Status: In progress

Target:

Target Radius (m): 30

Drones Assigned: 2

Fig. 9. Panel displaying the status of a herding job, along with options for editing how the job should be carried out.

After the midterm presentation, I continued development on the herding policy, implementing the dynamically tuned weights described in section III-A3. I also implemented algorithmic support for targets of arbitrary shape, as well as adding the ability to request that a drone maintain the flock in a given area for a set period of time.

I continued fullstack work on the app as well, adding a drone management page alongside persistent storage for which drones a user had on their farm. I fixed bugs along the way, including issues we had with panning and visual bugs with scrolling.

B. Contributions (Joel Villarino)

My primary contribution to the project was the design, optimization, and stabilization of the simulation engine, which had become the central bottleneck limiting both experimentation and system reliability. I was the sole developer on the simulator and introduced Numba-accelerated computational kernels, and developed the hybrid neighbor-caching strategy that enabled large-flock experiments. These changes collectively transformed the simulator into one of the fastest and most stable components of the system and are explored in Section III-C.

In addition to the performance work, I led the separation of the Simulator from the Live Farm application. The two systems had grown tightly coupled, making it difficult to evolve features independently and increasing the risk of cross-system breakages. By introducing a clear boundary between them, we reorganized the codebase around explicit responsibilities, enabling safer iteration, more predictable testing, and cleaner long-term extensibility.

I also upgraded our real-time visualization pipeline by implementing Server-Sent Events (SSE), which allowed the frontend to receive continuous state updates with significantly lower latency. This improved the responsiveness of both the simulation and live farm views. Alongside this, I built the scenario-creation pipeline and the metrics dashboard, giving the team a structured way to configure environments and analyze the behavior of different herding strategies.

Throughout the semester, I served as the primary full-stack maintainer, resolving integration issues across the backend, simulator, and frontend, ensuring stability during rapid iteration, and supporting teammates as new features were introduced. This is supported by being the contributor with most lines written and commit velocity on Github insights. After the final presentation, I created our website deliverable and overhauled our testing suite by adding more than seventy new tests. This enabled a safe large-scale refactor and documentation pass across the entire backend codebase, improving maintainability for future teams.

C. Contributions (Supanat Khaodhiar)

My main contribution was building and refining the sheep–herding simulator and using it to test our herding policies. I focused on making the environment as realistic as possible, while still fast and stable enough to run large numbers of trials.

I started by creating a lightweight simulator for testing the herding algorithm and tuning its parameters to match the setup in our reference research paper. From there, I added more realistic sheep dynamics, like momentum and inertia. I also added the blended movement model that lets each sheep smoothly shift between grazing and flocking instead of snapping between modes. Finally, I limited each sheep's k-NN perception radius so the flock no longer reacts to unrealistically distant neighbors.

On the control side, I implemented the conditional repulsion option. This allows the drone to move over the herd and target the furthest sheep without disrupting the entire cluster. It also ensures the drone only applies pressure when doing so is actually beneficial.

I also spent some time improving realism around obstacles and boundaries. I refined sheep movement near obstacle edges so they no longer clip, slide, or bounce inside obstacles. Throughout this process, I repeatedly re-tuned the parameters so everything stayed stable as the environment changed.

Finally, I worked on performance and testing. I vectorized computations such as the k-NN and dog distance calculations to speed up large simulations. I also ran large batches of experiments to compare our herding algorithm against the baseline from the paper and to test different drone counts, herd sizes, and parameter settings.

D. Contributions (Roxana Gutierrez)

I led development of the frontend of our user-facing application. I built the main map component, designing a basic UI to display labeled entities (drones, livestock, targets) over a satellite image of our farmland. I implemented the coordinate transformation system to map backend world positions to frontend pixel coordinates. To integrate with the backend, I initially established HTTP polling to retrieve the positional state of all of the entities. To reduce latency, I later modified the frontend infrastructure to accept SSE from the backend, while retaining the HTTP polling option as fallback.

I developed the initial user flow to create and manage a single simulated job, as seen in our midterm demo. This included a page to customize the simulation being run, where the user could select the farm to view as well as initialize scenarios via presets or a dynamic customization modal. I also implemented interactive map controls allowing a user to manage the current simulation including playing/pausing, restarting, and selecting/changing the target.

Following the initial prototype, I reorganized the application architecture by adding a tabbed navigation system. To this, I added a comprehensive scheduling component. I built 3 navigable calendar views (daily, weekly, and monthly) that each display herding jobs at their scheduled times. I integrated these views with the backend API to support management (creation, modification, and deletion) of herding jobs.

Throughout development, I added and modified minor features to improve UX, including the scheduled job filter, contextual tooltips for job targets, among others.

References

[1] H. R. E. H. Bouchekara *et al.*, “Livestock management with unmanned aerial vehicles: A review,” *IEEE Access*, vol. 10, pp. 45001–45028, Apr. 2022.

[2] Z. Dukowitz, “Drones in precision ranching—How BeeFree Agro uses autonomous drones and AI-powered software to herd cattle,” *UAV Coach*, Dec. 2020. [Online]. Available: https://uavcoach.com/beefree-agro/

[3] D. Strombom, R. P. Mann, A. M. Wilson, S. Hailes, A. J. Morton, D. J. T. Sumpter, and A. J. King, “Solving the shepherding problem: heuristics for herding autonomous, interacting agents,” J. R. Soc. Interface, vol. 11, no. 100, p. 20140719, Nov. 2014.

[4] C. W. Reynolds, “Flocks, herds and schools: A distributed behavioral model,” *Comput. Graph.*, vol. 21, no. 4, pp. 25–34, July 1987.

VII. Supplementary Data

1) **Example Composite Test of Herding Algorithm:** https://drive.google.com/file/d/1XlHKfIRohRZFu0MO19Yry2VIcuQvWBaV/view?usp=sharing