ALMA MATER STUDIORUM – UNIVERSITY OF BOLOGNA

MSc in Engineering and Computer Science

Herd Monitoring with Autonomous Drones: a Decentralized k-Coverage-inspired Approach

Thesis in:
Laboratory of Software Systems

Supervisor
Prof. Danilo Pianini

Co-supervisor
Dott. Jenna Kline

Candidate
Denys Grushchak

Fourth Graduation Session
Academic Year 2022-2023

Abstract

This thesis explores the innovative intersection of autonomous drones with wildlife monitoring, specifically focusing on herd monitoring through a decentralized k-coverage approach, aiming for a significant improvement in the quality and scope of ecological data collection. The work includes an in-depth analysis of drone-captured footage telemetry, enabling the reconstruction of animal dynamics and drone engagement. Reproducing this complex situation involved enhancing the existing herd behavior model and implementing it using the Alchemist simulator environment to model multiple sophisticated scenarios and conditions.

The significant contribution of this research lies in adapting aggregate algorithms for the Online Multi-Object k-Coverage (OMOkC) problem to the drone-based herd tracking scenario. The focus is on adopting a hierarchical clustering technique that optimizes target definition and assignment in a distributed network of drones while observing a highly dynamic environment. Through comprehensive simulations, the thesis assesses the performance of these algorithms, exploring the impact of clustering improvements and the effect of an adaptive hierarchical clustering algorithm.

The implications of this work extend beyond the immediate application to herd monitoring, suggesting a paradigm shift in how we approach biodiversity conservation and ecosystem management in the era of autonomous systems.

Dedicated to all who stand with Ukraine in these difficult times.

Acknowledgements

I am deeply grateful to my mother, whose unwavering support made my entire university journey and this thesis possible. I also thank my father and family for their encouragement and support throughout my studies.

A special thanks to my "Mangoh" friends - Riccardo, Eleonora, Amelia, Pippo, Rubba, Malloc, Matteo, Dillo, Brigo, and Gloria - for sharing this challenging journey with me.

Finally, my sincere thanks go to my supervisors, Danilo Pianini and Jenna Kline, for their guidance, patience, and invaluable support on this complex project.

Contents

Chapter 1

Introduction

1.1 In Situ Imageomics

The *in situ imageomics* [KSBW+23] represents a novel method for investigating ecological, biological, and evolutionary systems. This approach involves the collection of extensive image and video datasets in natural environments, with subsequent utilization of machine learning techniques to deduce biological characteristics of individual organisms, animal social groups, species, and entire ecosystems.

The utilization of *in situ imageomics* facilitates the monitoring of biological traits across vast areas and extended periods, opening opportunities for data-driven strategies in wildlife conservation, biodiversity, and sustainable ecosystem management.

The primary focus of this approach is the examination of organisms within the context of their natural environment, emphasizing an "in situ" perspective. To comprehend the collective dynamics of individual and group behaviors, as well as their responses to habitat and environmental conditions, observation of fine-grained details at both individual and population scales is essential. This involves tracking movements over potentially large distances and changing habitats.

1.1.1 Autonomous UAVs

For an *imagomics* application designed to automatically extract behavioral traits from videos, modern machine learning and computer vision techniques are es-

sential. However, these techniques require exceptionally large and high-quality datasets, surpassing the amount of data typically collected for biological studies.

While camera traps and unmanned aerial vehicle (UAV) generate substantial data, they face challenges in tracking individuals over large areas, capturing social dynamics, and dealing with issues like poor lighting and unfavorable view angles in images.

In contrast, small unmanned aerial systems (sUAS) consisting of one or more UAVs and control systems like those from DJI and Parrot, offer dynamic animal tracking and efficient traversal of remote terrain. This surpasses the limitations of heavy-duty sport utility vehicles (SUVs) often used in fieldwork, enabling the capture of fine-grained details, such as animal behaviors within their social and environmental context.

As commercial UAVs become cheaper and more readily available, more often they are employed for the capture of video and photo data in animal ecology, conservation, and agriculture applications.

However, these missions necessitate trained pilots capable of manual operation tailored to specific geographic regions and species. Utilizing autonomously controlled sUAS in ecological research presents a significant advantage over manual approaches that are associated with challenges such as high costs for hiring expert pilots, limited availability in certain areas, potential human errors, and difficulties in ensuring consistent data collection. Automating missions not only addresses these issues but also proves to be more cost-effective, safer, and provides better spatial accuracy compared to manually piloted flights, as evidenced by previous studies [BCSK19]. Furthermore, manual piloting introduces significant coordination difficulties, particularly as the scale of sUAS operations increases. This complexity amplifies with each additional sUAS introduced into the operation, making efficient large-scale manual organization challenging.

1.1.2 Nadir and Non-Nadir Views

The tracking of animals using UAVs commonly relies on two types of perspectives: nadir (or bird's eye view) [KSBW+23] and non-nadir [KDK+23] (illustrated in Figure 1.1). While nadir-angle photography allows for the individual identification

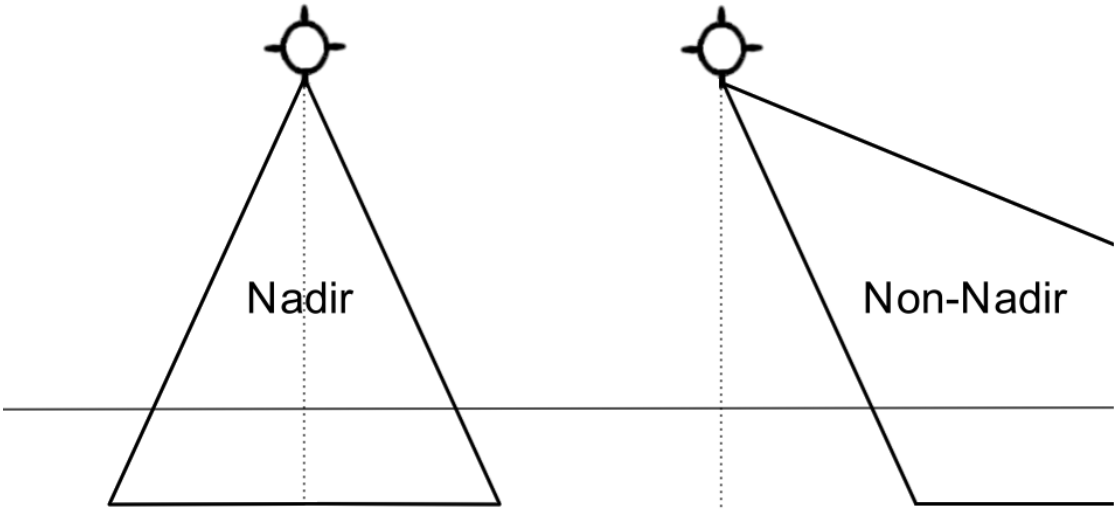

Figure 1.1: Visualizzation of Nadir and Non-Nadir views

of animals of interest, it may not be suitable for all species, such as zebras and giraffes, because animals' distinguishing markings on the hip and shoulder are not visible, and behavior is difficult to identify. Notably, the most valuable ecological data for animal ecology from sUASs is gathered through off-nadir views, where the UAV is positioned high enough to avoid surface-level obstructions, such as vegetation, as shown in Figure 1.2. However, nadir missions (see Figure 1.3) are generally much easier to plan and execute using existing methods compared to non-nadir missions.

1.2 Field of View Projection

The **Field of View (FoV)** of an UAV or an animal, when projected in two dimensions, can be characterized by the triple $\mathcal{V} = \langle \Theta, R, \frac{\beta}{2} \rangle$, where:

- Θ represents the orientation of the view relative to a fixed reference system;

Figure 1.2: Example of Non-Nadir view, from data collected at the Mpala Research Centre in Laikipia County, Kenya. Behavior and distinguishing markings are easily seen [KSBW+23].

Figure 1.3: Nadir-view, animals' distinguishing markings on hip & shoulder not visible and behavior difficult to identify [KSBW+23].

- R denotes the range of view, indicating the maximum distance at which the targets can be detected;
- $\frac{\beta}{2}$ signifies half of the view angle, defining the width of the FoV beyond which blind spots exist. The FoV is assumed to be symmetric.

A scheme of the FoV projection is illustrated in Figure 3.1.

1.3 Computer Simulation

The computer simulations can accurately replicate complex environments and scenarios, they become particularly advantageous for designing and evaluating diverse algorithms for autonomous UAV missions aimed at collecting in situ *imageomics* data. A *computer simulation* can be formally defined as "the use of a mathematical/logical model as an experimental vehicle to answer questions about a referent system" [PN94]. Computer simulation models find extensive applications in various domains such as cancer treatment, crowd dynamic movements, city traffic, financial markets, and numerous other fields. Over time, computer simulation has demonstrated benefits for [LTF+18]:

1. Visualizing complex interactions in dynamic systems;
2. Providing results much faster than would be possible in real-time; and
3. Allowing "what if" analysis when changes to an actual system are difficult to implement, costly, or impractical.

These capabilities enable researchers to design, develop, and assess complex systems more efficiently, cost-effectively, and safely than experimenting with actual systems.

1.4 Alchemist Simulator

The Alchemist[1] is an open-source general-purpose simulation framework designed for modeling complex multi-agent systems [PMV13]. Alchemist, as a stochastic

[1] `https://alchemistsimulator.github.io/index.html`

simulator, employs probability theory to model the unpredictability and variability inherent in chemical and biological systems. Unlike deterministic models that rely on precise initial conditions and interactions, it incorporates randomness and fluctuations due to molecular interactions, environmental changes, and biological variability, enhancing the modeling of complex systems.

The Alchemist simulator has been enhanced with smart camera capabilities, enabling it to simulate and assess algorithms aimed at solving the **Online Multi-object k-coverage (OMOkC)** problem [PPCE22], detailed in Section 3.1.1. Combining simplicity and extendibility, Alchemist emerges as the optimal platform for adapting and testing sUAS coordination algorithms within the context of herd tracking. It provides a robust environment for the development and evaluation of complex coverage algorithms.

1.4.1 The Alchemist Meta-Model

The namings of Alchemist's entities derive from its beginning: "The core of Alchemist is an event-based engine derived from chemistry-oriented simulators, and its computational meta-model in part reflects these origins" [PPCE22]. The world of Alchemist, that is illustrated on Figure 1.4, is composed of the following entities[2]:

- *Molecule*: Represents the name of *variable* (name of data item);
- *Concentration*: Quantifies the *value* associated to a particular *molecule*;
- *Reaction*: Defines how changes occur within the simulation. A reaction is modeled as a set of *conditions* on the state of the system, which triggers the execution of a set of *actions*;
- *Condition*: Dictate the prerequisites for *reactions* to occur;
- *Action*: Models a change in the environment;
- *Node*: Acts as the container for *molecules* and *reactions*;

[2] `https://alchemistsimulator.github.io/explanation/metamodel/index.html`

- *Environment*: The space in which *nodes* are placed and move;
- *Linking rule*: A function of the current status of the *environment* that associates to each *node* a *neighborhood*;
- *Neighborhood*: An entity composed of a node (center) and a set of *nodes* (neighbors).

Incarnation is a crucial concept, providing a customizable layer that specifies the types of molecules, reactions, and conditions applicable in a simulation, adapting the framework to specific domains or research needs. Currently, the Alchemist distribution includes four Incarnations:

- *Protelis*[3] [PVB15] - Aimed at simulating networks of devices running an aggregate program that is written in protelis programming language, as described in Section 1.5.1;
- *SAPERE* [ZOA+15] - Models a distributed computing system inspired by natural ecosystems;
- *Biochemistry* - The first incarnation of Alchemist, designed to model biochemical reactions in multicellular environments;
- *ScaFi* [CVAP22] - Designed for the simulation of device networks running programs that utilize ScaFi, a Scala-based framework for Aggregate Programming.

1.5 Aggregate Programming

One of the main features of Alchemist is its native and first-class support for *aggregate programming* (limited to the Protelis and ScaFi implementations). This paradigm is founded on the observation that in the development of distributed systems, conventional device-centric programming languages compel programmers to focus on individual devices and their communication protocols. As a result,

[3] `https://protelis.github.io`

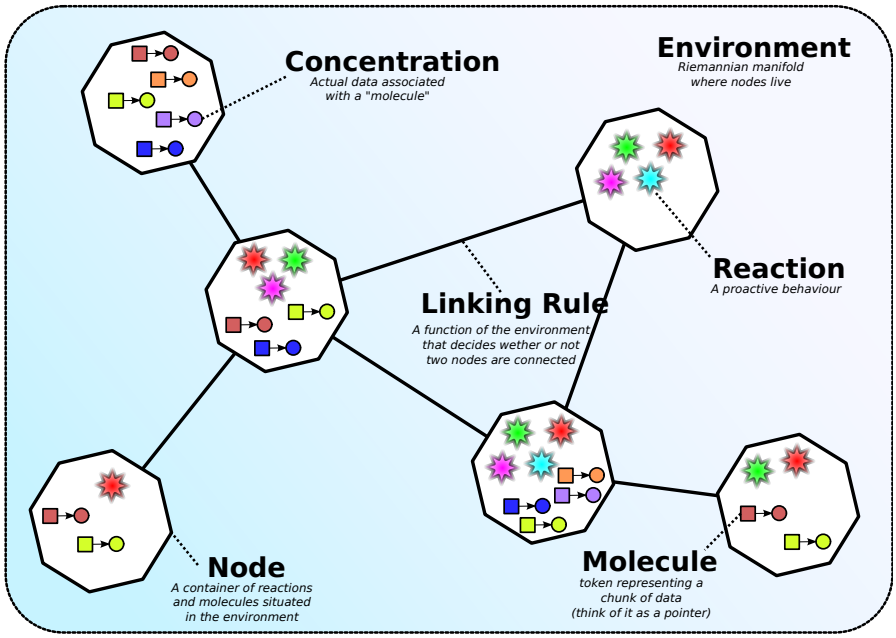

Figure 1.4: The alchemist metamodel

the design of device interaction and coordination becomes entangled with the application's implementation, leading to a fusion of various aspects of a distributed system. These aspects include the effectiveness and reliability of communications, coordination in the face of changes and failures, and the composition of behaviors across different devices and regions. This composition makes it difficult to effectively design, debug, maintain, and compose complex distributed applications [PVB15].

Aggregate programming aims to resolve this issue by offering composable abstractions that distinctively separate the global behavior from the implementation specifics [PVB15]:

- Communication between devices is generally made implicit, with higher-level abstractions for controlling efficiency/robustness trade-offs;
- Methods for distributed coordination are encapsulated within aggregate-level operations (e.g., measuring distance from an area, spreading a value by gossip, sampling a collection of sensors at a certain resolution in space and time); and

- The system as a whole is defined through the composition of aggregate-level operations. This high-level specification is subsequently translated into a comprehensive distributed implementation through an appropriate mapping process.

The computation in this paradigm is achieved through the manipulation of distributed data structures known as *computational fields* (or simply *fields*), which represent a global, distributed map from computational devices to computational objects (data values of any sort). For example: a collection of temperature sensors produces a field of ambient temperatures, a smartphone application produces a field of notification and a group of drones produces a field of visible objects. In this approach, the designer's focus shifts from individual devices and communication protocols to the evolution and composition of fields. The responsibility of translating these high-level abstractions into appropriate local interactions to achieve the desired global behavior falls to the language's interpreter (or compiler) [PPCE22].

1.5.1 Protelis

Protelis is a purely functional aggregate programming language with C-like syntax, that has full interoperability with Java type-system and API [PVB15]. Protelis is based on the *field calculus* a tiny functional language providing basic constructs to work with fields [AVD+19]. Fields are built and manipulated using four program constructs [BPV15]:

- *Functions*: $b(e_i, ..., e_n)$ applies a stateless function b to arguments $e_i ... e_n$;
- *Dynamics*: $rep(x \leftarrow v)\{s_1; ...; s_n\}$ define a local state variable x initialized with value v and at each *computational round*, it is updated with the result of executing its body statement $\{s_1; ...; s_n\}$. This constructs a field that evolved;
- *Interaction*: $nbr(s)$ gathers a field from all neighbors (including the device itself) to the latest value from computing s. A build-in "under the hood" functions can then summarize such maps: for example, $minHood(m)$ finds the minimum value in map m;

Listing 1.1: Example of network partition by if construct in Protelis.

```
// Assumption: we have 10 interconnected devices with ids 0..9
// the call self.getDeviceUID().getId() returns id of a device (integer >= 0)
rep (v <- self.getDeviceUID().getId()) { // inizialize v with device id
  if (self.getDeviceUID().getId() < 5) {
    maxHood(nbr(v)) // the first five devices will get 4 as a result
  } else {
    maxHood(nbr(v)) // the second five devices will get 9 as a result
  }
}
```

- *Restriction*: $if(e)\{s_1;...;s_n\}$ $else$ $\{s'_1;...;s'_m\}$ partitions the network into two regions: where the condition e is true $s_1;...;s_n$ is computed; elsewhere, $s'_1;...;s'_m$ is computed instead.

It's important to note that in aggregate programming, the use of if has different consequences than in traditional languages. Rather than serving as a "flow control" instruction, it acts as a domain restriction instruction, effectively dividing devices into two non-communicating domains. This makes devices get separated into two, non-communicating domains. An example of this division based on device ID is illustrated in Listing 1.1.

While if serves as an exclusive branching construct, the $mux(e)\{s_1;...;s_n\}$ $else$ $\{s'_1;...;s'_m\}$ construct functions as an inclusive multiplexing branch. It evaluates both branches and selects one of them for return, avoiding network partition.

1.6 Clustering

Clustering is a technique in unsupervised learning that aims to categorize a collection of objects into clusters, ensuring that objects within the same cluster are more similar to each other than objects from different clusters. It is a crucial method in data mining, aiming to achieve optimal partitioning of data without prior knowledge.

Clustering is widely applied across various domains; it facilitates statistical data analysis in fields such as machine learning, data mining, pattern recognition, image analysis, and bioinformatics [PE14]. Particularly, in the context of animal tracking by UAVs, clustering methods can be employed to organize sets of individuals into

groups. This organization can be leveraged to facilitate the assignment of each group to a specific UAV, aiming to diminish coverage overlap and enhance tracking efficiency.

To address the clustering problem from multiple angles, several clustering techniques have been developed [PE14]:

- *Partitional Clustering*: Divides the dataset into distinct non-overlapping subsets or clusters without any hierarchical structure. Partitioning clustering methods are useful for applications where a fixed number of clusters are required;
- *Density-Based Clustering*: Identifies clusters based on the density of data points in a region, allowing for the discovery of clusters with arbitrary shapes and the ability to handle noise and outliers;
- *Hierarchical Clustering*: Divide or merge a dataset into a sequence of nested partitions, often visualized as a dendrogram (see Figure 1.5), which allows for intuitive analysis of data at different levels of granularity.

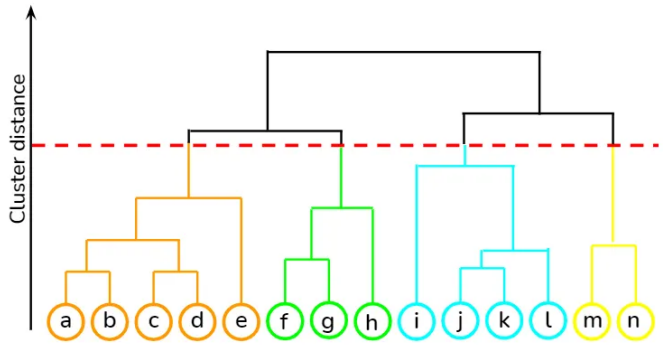

Figure 1.5: A hierarchical clustering dendrogram visualization[5]. Selecting a specific clustering distance allows cutting the hierarchy at the desired level, producing clusters of the intended granularity.

[5]Source: https://towardsdatascience.com/hierarchical-clustering-explained-e59b13846da8

1.7 Structure of the Thesis

The primary goal of this study is to establish a robust framework that supports the testing and evaluation of UAVs navigation algorithms, specifically tailored for herd tracking. Additionally, this project attempts to adjust existing OMOKC algorithms for their application within this novel context. The ultimate objective is to contribute to the improvement of the collection of high-quality biological data in large-scale scenarios.

The structure of this thesis is organized as follows:

- Chapter 2 investigate the telemetry data from manually piloted UAV missions and explore various methodologies for modeling herd behavior within simulations;
- Chapter 3 is dedicated to introducing and adapting existing aggregate algorithms to address the specific requirements of the herd tracking problem;
- Finally, Chapter 4 combines the adapted algorithms with diverse herd composition scenarios to measure their performance across a range of conditions. This evaluation aims to identify the strengths and limitations of each algorithmic approach under various circumstances.

Chapter 2

Herd Movement Model

2.1 Analysis and related work

Deploying and maintaining networks of UAVs in the real world is generally cumbersome, time-intensive, and requires manual labor. Experimenting with new methodologies in real networks can be costly and problematic—as experiments often cannot be reproduced precisely [PPCE22]. Given these constraints, simulations emerge as a viable solution. However, simulating herd tracking algorithms necessitates an accurate representation of herds, achievable through two primary methods:

- Replicating real-world herd movements within the simulator, as discussed in Section 2.1.1 and Section 2.1.2;
- Implementing a mathematical model of herd movement within the simulator, as examined in Section 2.1.3 and Section 2.1.4.

2.1.1 KABR dataset

The dataset, known as the Kenyan Animal Behavior Recognition (KABR), has been gathered directly from drone-captured footage to enrich the available resources for studying animal behavior [KKR+24]. This dataset is focused on the wildlife of Kenya, including the behaviors of giraffes, plains zebras, and Grevy's

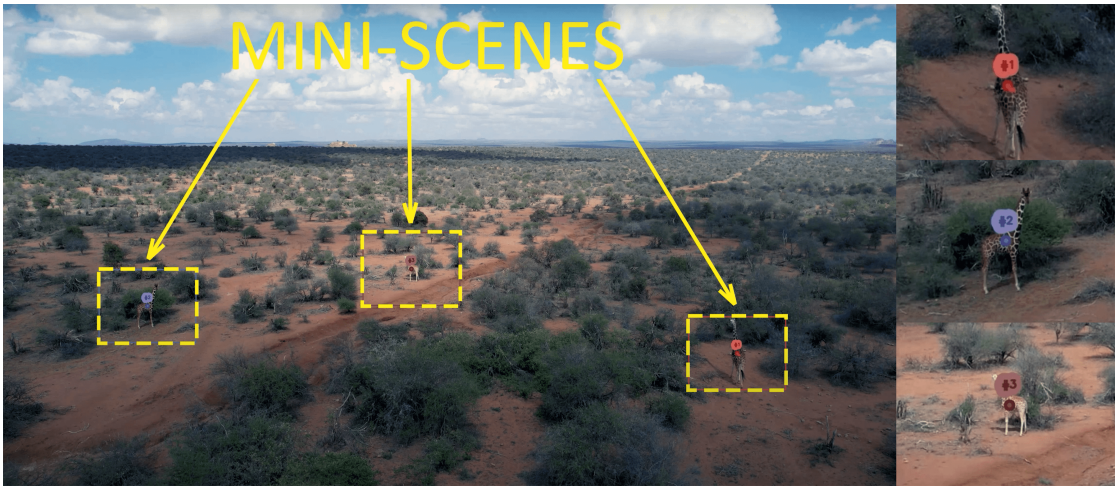

Figure 2.1: A mini-scene is a sub-image cropped from the drone video footage, centered on and surrounding a single animal. The KABR dataset consists of mini-scenes and their frame-by-frame behavior annotation [KKR+24].

zebras. These species were chosen due to their ecological significance and conservation status within the region. A novel technique introduced in this work involves detecting and tracking each animal across the high-resolution videos, subsequently linking the results into tracklets. For every tracklet generated, a separate video segment, referred to as a *mini-scene* (see Figure 2.1), is created by extracting a sub-image centered on each detection in a video frame.

KABR-telemetry analysis

The KABR-telemetry dataset [HDR24] complements the KABR dataset by providing telemetry data from the drone (DJI Mavic 2S) during its observational missions. This telemetry data includes critical details about the drone's operational status, such as its location and altitude, as well as the bounding box dimensions of observed wildlife within the video frame and annotations of their behaviors. However, the dataset lacks two crucial pieces of information necessary for the accurate reconstruction of animal positions: the compass direction of the drone and the gimbal pitch, which indicates the camera's vertical angle.

Despite this omission, for this work, an additional sample of data contain-

ing the missing drone telemetry was requested and subsequently obtained. This supplementary data enables the reconstruction of animal movements from the KABR-telemetry dataset.

2.1.2 Herd movement reconstruction from UAV flight data

To estimate the relative distance between a target and an UAV, a geometrical approach based on the UAV's camera perspective can be utilized. Firstly, δ is considered as the vertical angle from the UAV's altitude line to the optical center line of the UAV's camera. To determine the angle between the camera center and the target, the offset of the target's bounding box center within the camera's FoV can be calculated, as illustrated in Figure 2.2. Here, Δw and Δh represent the horizontal and vertical offsets, respectively, with w and h denoting the width and height of the image.

The angles of interest can be determined using the following formulas:

$$\theta = \frac{\Delta w}{w}\psi_w, \quad \alpha = \frac{\Delta h}{h}\psi_h$$

where ψ_w and ψ_h are the camera's horizontal and vertical FoV angles. The angles θ and α represent the relative horizontal and vertical angles between the camera center and the target. Furthermore, the angle γ is calculated as:

$$\gamma = \delta + \alpha$$

This angle γ represents the combined vertical angle from the UAV's altitude line to the target's position, as illustrated in Figure 2.3.

In certain research, the distance from the camera to the target is approximated using the proportional relationship [LWHH19]:

$$f/l = m/M, \quad l = Mf/m$$

where l is the distance between the target and the camera, f is the camera's focal length, M represents the actual height of the target, and m is the height of the target as captured on the camera's sensor. However, this approach faces two main

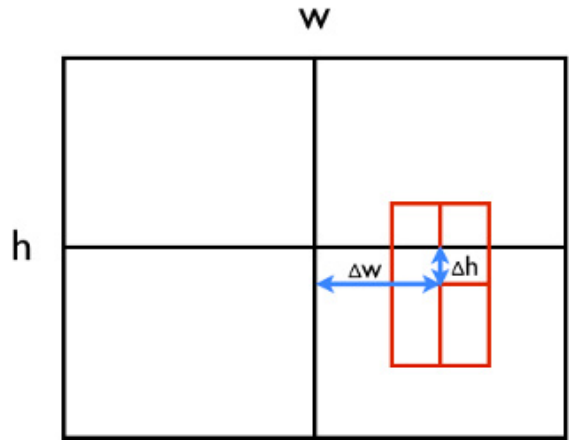

Figure 2.2: Position of a target's bounding box in the image [LWHH19].

challenges. Firstly, the precise height of animals, crucial for this calculation, cannot always be accurately determined from the available data. Secondly, the sizes of bounding boxes in the KABR-telemetry dataset can vary significantly from one frame to another, leading to abrupt changes in the estimated position of the target.

To address these issues, an alternative method is employed, leveraging the known value of the camera's altitude z from the dataset, along with the calculated angles θ and γ. These elements enable the estimation of the distance d from the camera to its optical center projection on the ground through the following formula (see Figure 2.3):

$$d = \frac{z}{\cos(\theta)}, \quad x = \tan(\theta) \cdot z, \quad y = \tan(\gamma) \cdot d$$

Here, x and y represent the relative distances on the ground from the drone to the target. By integrating these distances with the drone's location and compass direction, it is possible to create estimated trajectories of animal movements.

2.1.3 Models of self-organized collective behavior

Group dynamics models are categorized into two main types: **Eulerian** and **Lagrangian** [GLR96]. Eulerian models overlook individual identities, focusing instead on the number of individuals within a given unit of area or volume, often treating this space as a continuum. This method models population dynamics

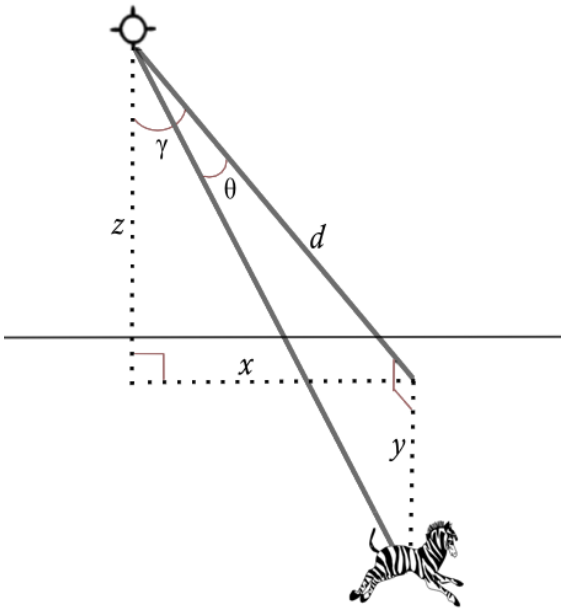

Figure 2.3: Graphical representation of estimating the relative distance from a drone to its target.

through a density function, employing equations to track changes in this density over time. Although effective for modeling the collective movement of densely packed, smaller organisms like bacteria, insects, and krill, the Eulerian perspective is less suitable for larger species such as fish, birds, and mammals. These larger species typically form groups (schools, flocks, herds) comprising only dozens or hundreds of individuals and typical spacing is relatively large in terms of body lengths.

In contrast, Lagrangian (or agent-based) models offer a more appropriate representation of these scenarios. Unlike their Eulerian counterparts, agent-based models of animal grouping assume behavioral rules at the level of the individual. Each agent follows a set of rules that dictate its behavior, including movement and interaction with other individuals, thereby capturing the complex dynamics of group cohesion and dispersion. In these agent-based models, collective behavior results from three simple and general behavioral rules followed by individuals [Gia08]:

- *Alignment*, which makes neighboring animals move in the same direction;

- *Attraction*, which ensures no animal remains isolated;
- *Short-range repulsion*, which prevents dangerous proximity.

At the group level, these three components should grant the directional polarity (alignment) and cohesion of the aggregation (attraction), preserving individual integrity (short-range repulsion).

The agent-based model can be interpreted as a variation of the n-body dynamics problem, where the movement of each agent is governed by Newton's second law. This movement is determined through the integration of Newton's equation [GLR96]:

$$m_i \ddot{x}_i = \sum_k F_{ik} = F_i \quad \text{for} \quad i = 1, 2, ..., n$$

Where x_i is the position of individual i, $\ddot{x}_i$ is the second derivative of x_i to time (represent acceleration) and m_i denotes the mass of the individual. Here, F_i signifies the sum of forces acting upon individual i (as attraction, repulsion and alignment), and n represents the total number of individuals.

2.1.4 The Dynamics of Herds Model

The article "*The Dynamics of Herds: From Individuals to Aggregations*" of Gueron *et al.* (1996) [GLR96] introduces a two-dimensional discrete stochastic agent-based model that describes the self-organized collective behavior of herds. Despite its age, this model establishes rules that can provide a foundation for the simulation of interactions between UAVs and herds.

This study presents the following assumptions and definitions: For $i = 1...n$, $r_i(t) = (x_i(t), y_i(t))$ indicates the location of the i-th individual at time t. The model conforms to the general class of n-body problems mentioned above where each individual is subject to only two forces:

- $\mathbf{w}_{i1}$: The intrinsic velocity vector of individual i;
- $\mathbf{w}_{i2}$: The velocity adjustment of individual i due to interactions with other individuals.

Thus, the total force summation can be expressed as:

$$\dot{r}_i = \mathbf{w}_{i1} + \mathbf{w}_{i2} \quad \text{for} \quad i = 1, 2..., n,$$

where $\dot{r}_i$ denotes the velocity of individual i, that represents the first derivative of r_i with respect to time, and n is the number of individuals in the herd.

The dimensions of each individual are defined by a *body length* l and a *body width* w, for which the ratio $w = l/2$ is applied, reflecting proportions typical of grazing mammals.

Intrinsic behavior

In the absence of neighbors, it is assumed that each individual can move in one of three directions at any given time step:

- *left*: negative x axis direction;
- *forward*: positive y axis direction; or to the
- *right*: positive x axis direction.

With the respective probabilities p_0, p_1, and p_2 ($p_0 + p_1 + p_2 = 1$). In an idealized scenario where the animal is precisely aware of the target location and can respond perfectly, p_1 would be set to 1, indicating a direct forward movement, while p_0 and p_2 would be reduced to 0, eliminating lateral movements.

The intrinsic movement vector $\mathbf{w}_{i1}$ is constituted by two components: intrinsic *forward velocity* v_i, and intrinsic *lateral velocity* u_i.

Influence zones and interactions with neighbors

The behavior of individuals is significantly influenced by the presence of neighbors within spatial zones (Figure 2.4) that are defined as rectangular regions $[a, b]\times[c, d]$, where the notation $[a, b]$ represents an interval. Each zone is characterized by specific reactions:

1. **Stress Zone (SZ)** is defined as $[x_i - a_1, x_i + a_1] \times [y_i - b_1, y_i + b_1]$. The presence in this zone triggers avoidance behaviors, varying according to the neighbor's position relative to individual i:

- *In front* leads to a speed reduction to a fraction q_1;
- A *lateral* presence prompts i to move in the opposite lateral direction;
- Neighbors *behind and to one side* also cause i to move laterally in the opposite direction but do not affect speed;
- Neighbors *behind on both sides* result in i being pushed forward, increasing its speed by a fraction q_1.

2. **Neutral Zone (NZ)** is delineated as $[x_i - a_2, x_i + a_2] \times [y_i, y_i + b_2] - SZ$, where $a_2 > a_1$ and $b_2 > b_1$. This zone's interactions are predicated on reducing predation risk: when all neighbors are on the same side, i displays *selfishness* by moving towards them without altering speed.

3. **Attraction Zone (AZ)** is outlined as $[x_i - a_3, x_i + a_3] \times [y_i, y_i + b_3] - SZ - NZ$, with $a_3 > a_2$ and $b_3 > b_2$. Interactions within this zone aim at maintaining group cohesion:

   - Neighbors only on one side cause i to accelerate to a multiple q_2 of its intrinsic speed and move towards the neighbors;
   - Neighbors on both sides drive i to increase its speed to a fraction q_2 without changing direction.

4. **Rear Zone (RZ)** is characterized as $[x_i - a_4, x_i + a_4] \times [y_i - b_4, y_i] - SZ$, where $a_4 > a_1$ and $b_4 > b_1$. The presence of neighbors in this zone distinguishes between leaders and trailers:

   - If there are no neighbors in the first three zones but there are in the rear zone, i is considered a *leader*, potentially reducing its speed to a fraction q_3 with a probability s_1;
   - Conversely, *trailers* accelerate to a multiple q_4 with a probability s_2.

The first three zones are mutually exclusive, and their influence on behavior determination follows a specific hierarchical procedure (Figure 2.5). The presence of neighbors in any one zone negates the influence of the others. For example, if neighbors are identified within the stress zone, only the effects of this zone are acknowledged, completely excluding the impact of the other two zones.

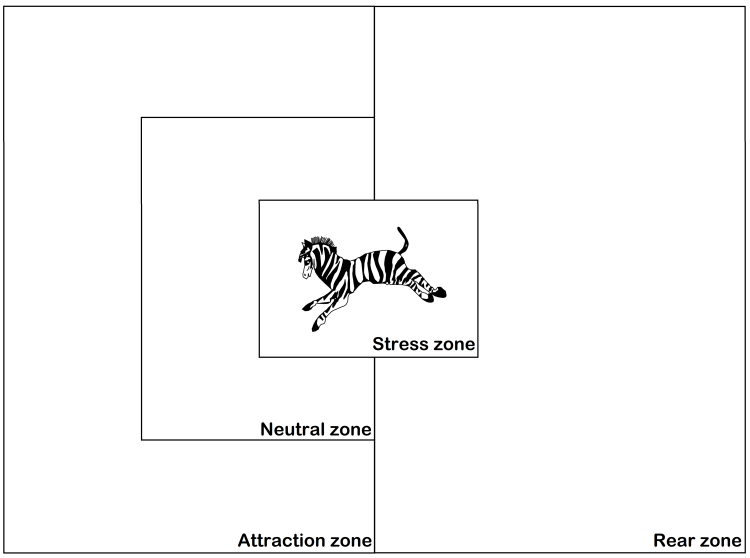

Figure 2.4: Idealized stress, neutral, attraction and rear zones for an individual [GLR96].

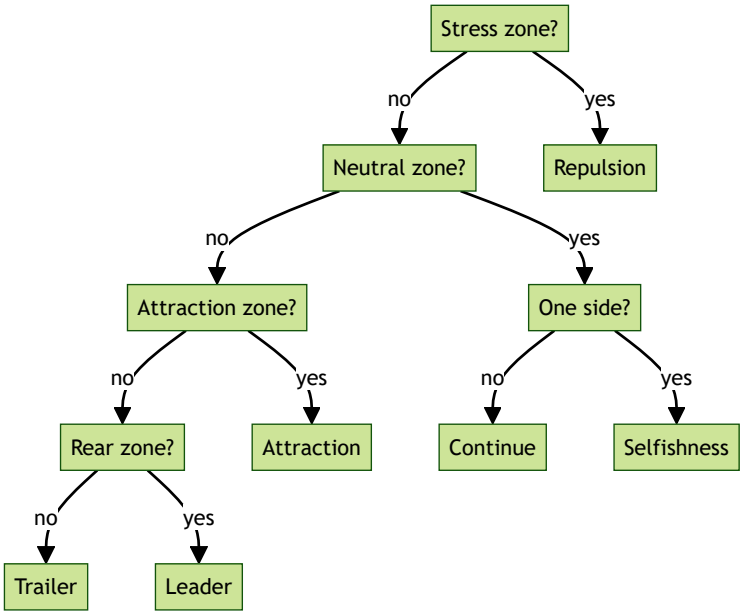

Figure 2.5: Flowchart of the hierarchical decision algorithm for an individual [GLR96].

The model flaw

The model developed by Gueron *et al.* (1996) incorporates essential behavioral rules for *attraction* and *short-range repulsion*. However, it lacks the individual alignment rules necessary for ensuring that neighboring animals move in the same direction. The absence of such rules means that individuals never change their direction, which may yield unrepresentative outcomes during the evaluation of UAV algorithms. An extension to this model, introducing alignment rules, is proposed in section 2.2.1, aiming to enhance the applicability of the model.

2.2 Design

2.2.1 Enhancements to the Model

Zone Geometry Transformation

A significant modification from the Gueron *et al.* (1996) model involves transitioning the geometry of behavioral zones from rectangular to circular sectors. This adjustment is motivated by two principal considerations:

1. The circular representation of behavior zones can be observed in more recent research of self-organized collective behavior [Gia08];

2. The implementation of directionality for zones' geometry is more straightforward with circular sectors compared to rectangles.

A circular sector zone can be conceptualized as a Field of View (FoV) $\mathcal{V}$, detailed in Section 1.2. It is characterized by a triple $\mathcal{V}_z = \langle \Theta, R, \frac{\alpha}{2} \rangle$, with z serving as the zone type identifier. A zone characterized by a full circular range with $\alpha = 360°$ can assume an elliptical shape. To accurately describe this geometry, we introduce an additional parameter, k, representing the ratio of the ellipse's length to its width.

Employing this framework, we can improve the zone delineations posited by Gueron *et al.* (1996) as follows:

- **Stress Zone** is delineated as $\mathcal{V}_{SZ} = \langle \theta_1, r_1, \frac{\alpha_1}{2} \rangle$. Typical setup for this zone is $\alpha = 360°$ with ellipse ratio $k = 2$;

- **Neutral Zone** is delineated as $\mathcal{V}_{NZ} = \langle \theta_1, r_2, \frac{\alpha_2}{2} \rangle - \mathcal{V}_{SZ}$, with $r_2 > r_1$ and $\alpha = 180°$;
- **Attraction Zone** is delineated as $\mathcal{V}_{AZ} = \langle \theta_1, r_3, \frac{\alpha_2}{2} \rangle - \mathcal{V}_{SZ} - \mathcal{V}_{NZ}$, where $r_3 > r_2$ and $\alpha = 180°$;
- **Rear Zone** is characterized by $\mathcal{V}_{RZ} = \langle \theta_2, r_4, \frac{\alpha_2}{2} \rangle - \mathcal{V}_{SZ}$, where $\alpha = 180°$, $\theta_2 = \theta_1 + 180°$ and $r_4 > r_1$.

This geometric transformation enables a more intuitive and practical implementation of the model.

Trailer redefinition

In the original model, an individual is considered as a *trailer* exclusively under conditions of complete isolation, indicated by the absence of neighbors across all zones. This scenario also implies that a *leader* may become a *trailer* if it will move faster than the rest of the herd. To address this issue, we apply the trailer acceleration parameter q_4, which probability s_2 to any individual lacking neighbors in the rear zone, regardless of the presence of neighbors in other zones. The only exceptions are single individuals who do not receive any acceleration boost and move only with their intrinsic velocity.

Herd direction alignment

The *heading* of an individual is defined as its orientation in degrees relative to a fixed reference system, represented by a vector $\vec{h}_{curr}$. The *heading* angle of individual i, denoted as Θ_i, aligns with the orientation of the stress, neutral, and attraction zones. Any adjustment to the individual's *heading* angle results in a corresponding shift in the orientation of all zones.

The canonical velocity vectors for movement (*left*, *forward*, *right*) are retained but are adjusted to align with the individual's *heading*, ensuring that movements are accurately directed relative to the individual's orientation in the environment.

Individuals have a static preference for turning direction, choosing either to increase or decrease their current direction angle Θ_i. The right to change direction is reserved for *leaders* or solitary individuals. Each one aligns its *heading* vector

with the average of the heading vectors $\vec{h}_{ngb}$ of neighbors located within the *neutral zone*. The formula for calculating the new aligned heading vector $\vec{h}_{new}$ is as follows:

$$\vec{h}_{new} = \epsilon \cdot \vec{h}_{curr} + (1 - \epsilon) \cdot \vec{h}_{ngb}$$

where $\epsilon \in [0; 1]$ is a constant that represents the importance of the individual's current direction.

To limit the simulation space, a circle with radius l defines the desired world boundary. Individuals within this radius may adjust their direction by γ_1 degrees with probability s_3, in their preferred turning direction. This mechanism ensures a unique movement trajectory for the herd.

If a herd exits beyond the designated world borders the following rules are applied: (I) the turning angle γ_1 is increased by an additional fraction γ_2; (II) individual turning preferences are overridden, compelling all to turn towards the direction that minimizes the angle between their *heading* and the vector pointing from their position to the world's origin. These rules in a natural way make the herd return to the desired simulation borders.

The Table 2.1 summarizes most of the relevant notation of this section.

Multiple herd support

To facilitate simulations involving multiple herds that remain distinct without intermixing, each individual is assigned exclusively to a single herd. Attraction and alignment forces are applied solely among individuals belonging to the same herd, ensuring cohesive movement and orientation within each group. Conversely, repulsion forces are universally applied across all individuals, regardless of their herd affiliation. This approach allows for the maintenance of separation and avoidance behaviors between different herds.

2.2.2 Desing of herd behavior action in Alchemist

Alchemist provides various environments that have different properties and dimensions. To model herd behavior the `ContinuousPhysics2DEnvironment` is utilized. This environment provides an unbounded 2D Euclidean space, enriched with

Table 2.1: Summary of notations for herd movement model [GLR96].

Symbol	Description
$\mathcal{V}_z = \langle \theta, r, \frac{\alpha}{2} \rangle$	field of view zone definition
r_1	stress zone radius
r_2	neutral zone radius
r_3	attraction zone radius
r_4	rear zone radius
θ_1	orientation of stress, neutral and attraction zones
θ_2	orientation of rear zone
α_1	stress zone circular sector angle (360°)
α_2	neutral, attraction and rear sector angle (180°)
$p_1 : p_2 : p_3$	intrinsic directionality
u, v	intrinsic velocity coordinates
q_1	slowing-down factor for repulsion
q_2	speeding-up factor for attraction
s_1	leaders' probability to "wait"
s_2	trailers' probability to "accelerate"
q_3	leaders' relative velocity
q_4	trailers' relative velocity
n	group size
$\vec{h}$	heading vector of an individual
l	prefered radius of the simulation world
γ_1	base individual direction turning angle
γ_2	additional individual direction turning angle
s_3	probability to change direction

physics capabilities and node shapes.

To integrate our model logic within Alchemist's framework, we introduce a new *action*, `HerdBehavior`. This action is designed to encapsulate all `Zones`, with key responsibilities including:

- Computing all forces affecting the node;
- Updating the node's position based on these forces;
- Aligning the node's direction with that of its neighbors in the neutral zone;
- Handling changes in the node's direction.

The integration of `HerdBehavior` with other components of Alchemist's architecture, crucial for the simulation, is depicted in Figure 2.6.

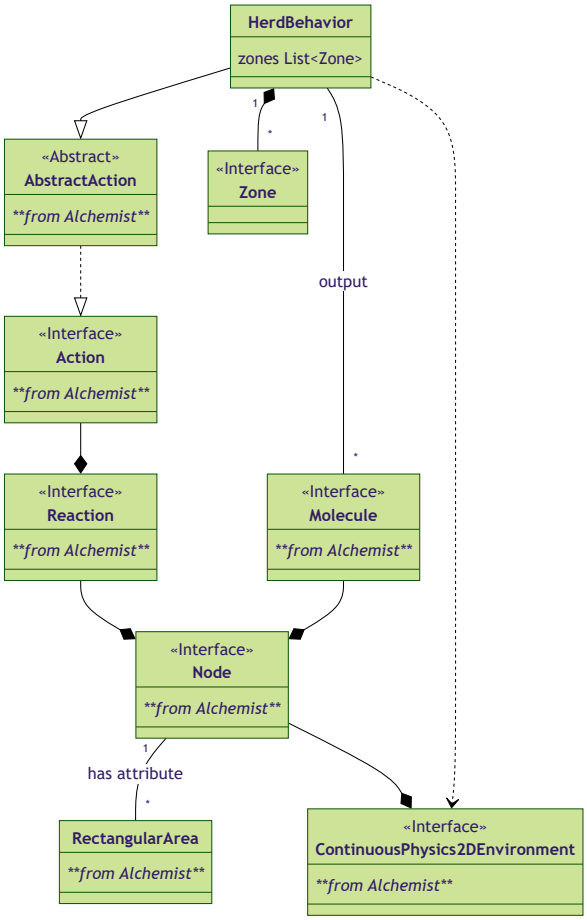

Figure 2.6: Class diagram illustrating the integration of `HerdBehavior` within Alchemist's architecture.

Figure 2.7 offers a detailed view of the zone hierarchy structure. The `AbstractZone` class implements common methods shared across all zones, while individual zone classes are tasked with executing their unique logic through `getNextMovement()`. Additionally, `StressZone` overrides the nodes filter to be able to repulse all nearby individuals, and `RearZone` overrides the default heading definition.

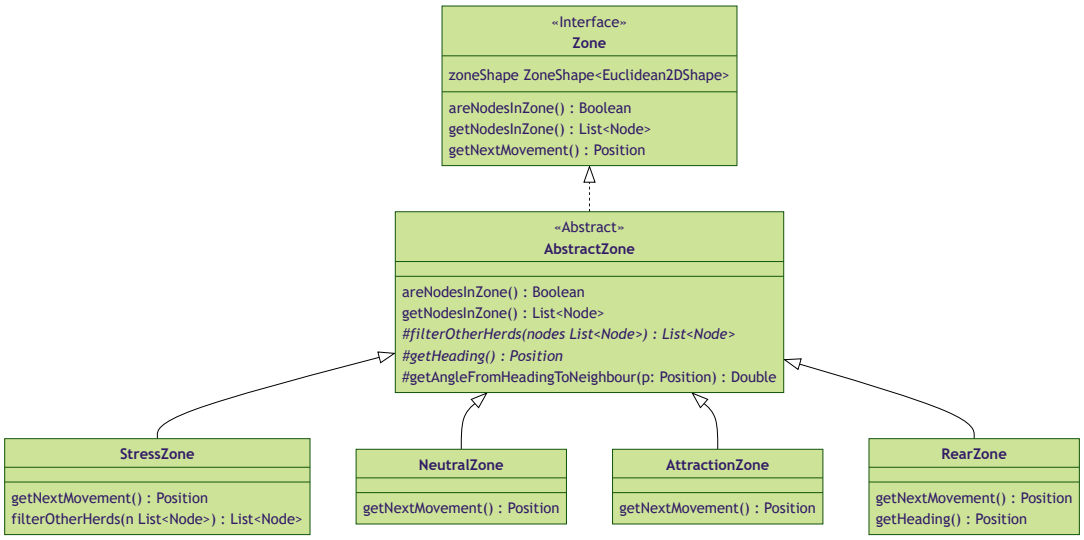

Figure 2.7: Class diagram detailing the zone hierarchy structure. `Position` is defined as an alias for `Euclidean2DPosition`.

2.3 Implementation Details

2.3.1 Implementing Herd Movement Reconstruction

Datasets merging

In the process of approximating animal movement, merging two datasets presents a significant challenge: the KABR-telemetry dataset (D_v) and an additional dataset (D_a) that includes the UAV's compass direction and gimbal pitch data. Dataset D_a contains records registered at a frequency of 10 records per second, whereas D_v shows that the video was captured at 30 frames per second. Each frame in D_v corresponds to one record for every detected animal within that frame. If no animals were detected in certain frames, this leads to missing records for those frames in the D_v dataset. Therefore, for each record in D_a, there can be from zero to three corresponding frames in D_v, with the number of records for each frame equal to the number of animals detected.

Due to the lack of unique identifiers that could facilitate a straightforward merging of these datasets, they are combined using an approximate join based

on three attributes: `longitude`, `latitude`, and `altitude`. Although this approximate merging method does not guarantee perfect accuracy, it can produce a completely merged dataset (D_m) from the available data sample.

Movement reconstruction results

Given the DJI Mavic 2S specifications[1] and the merged dataset D_m, the methodology outlined in Section 2.1.2 is applied to each record. This approach enables the estimation of the location of individuals at each frame, and by combining these estimates, the complete movement trace of the animals can be constructed.

This process encounters a second significant challenge: when the UAV makes rapid directional changes (turning left/right) or adjusts the gimbal pitch angle, a desynchronization occurs between video frame data and telemetry data. Such desynchronization leads to positional "jumps" of the individuals' position projections, which, after a few steps, return to their original location. This issue may partly arise from the imperfections in the approximated dataset joining method. However, the relatively prolonged duration of these "jumps" suggests a potential initial telemetry desynchronization. Nonetheless, the brief nature of these occurrences allows for their effects to be mitigated with data smoothing, represented by the formula:

$$\mu_i = \epsilon \cdot \mu_{i-1} + (1 - \epsilon) \cdot x_i$$

Here, ϵ is a smoothing hyperparameter, typically set to 0.98, μ_i represents the smoothed drone telemetry value at time i, and x_i is the actual UAV telemetry value at time i. Although this smoothing approach may slightly reduce the final precision of the localization, it significantly improves the smoothness of the resulting movement traces, making the overall analysis more coherent and visually consistent.

Conclusion

The current approach to reconstructing herd movements has several limitations, the primary one being the inability to evaluate the precision of computed local-

[1] `https://web.archive.org/web/20240229133723/https://www.dji.com/air-2s/specs`

izations based on the available data. Given these constraints, the reconstructed movement data are not suitable for evaluating UAV navigation algorithms in their present form. Despite this limitation, the processed results can still offer valuable insights, particularly regarding the behavior and responses of animals to UAV presence. This information is crucial for designing drone operations algorithms that minimize disturbance to wildlife.

Moreover, analyzing scenarios where animals' bounding boxes disappear from view, attributed to limitations of neural network-based detection systems under poor visibility or technical issues, is valuable for developing more sophisticated environmental models for simulating drone navigation algorithms. Additionally, this approach can be useful for gaining a better understanding of the dynamics of multiple drone missions, enabling a comparison of real-world operations with simulations.

2.3.2 Defining Simulations with YAML in Alchemist

Alchemist enables the definition and configuration of simulations via YAML files[2], adhering to the Alchemist meta-model outlined in Section 1.4.1. Listing 2.1 showcases the YAML configuration for simulating herd behavior: Initially, the configuration specifies the incarnation and the simulation environment type. To enhance readability and minimize repetition we introduce a long list of variables that establishes nearly all parameters for herd configuration. These parameters, detailed in Table 2.1, are aggregated under the variable `_herd_parameters` for input into the `HerdBehavior` *action*.

The `deployments` construct facilitates the stochastic placement of nodes within the environment. While Alchemist supports various deployment strategies, this project necessitated the development of a new deployment type, `GroupsDeployment`. This strategy randomly positions each node within a circle, ensuring that nodes belonging to the same group are positioned proximally. The assignment of nodes to groups is determined by the modulo operation:

$$\texttt{group\_id} = \texttt{node\_id} \ \% \ \texttt{number\_of\_groups}$$

[2] `https://alchemistsimulator.github.io/reference/yaml/index.html`

generating a balanced distribution of group members.

The `contents` parameter requires only a single molecule, `zebra`, which aids in differentiating animal nodes from UAV nodes. The `programs`' key establishes the `HerdBehavior` action, and the `properties` key assigns a rectangular shape to the node.

2.3.3 Global Herd Behavior Logic

The construction of the simulation model for herd behavior incorporates the `HerdBehavior` class, a part of it is illustrated in Listing 2.2. The primary functionality of any action within this model is encapsulated by the `execute()` method, which is inherited from the `AbstractAction` class. This method implementation combines individual directional alignment via `alignDirection()` and calculating the node's movement through `getNextPosition()`.

Within `getNextPosition()`, the process involves iterating through an ordered sequence of `Zone`s, ranging from the stress zone to the rear zone. The iteration seeks to determine if any zone contains neighbors. Upon encountering neighbors within a zone, the method calculates the subsequent movement and interrupts further iteration. Conversely, in scenarios where no zones possess neighbors, the node is considered as a single trailer. Consequently, it proceeds to move in a random direction, adhering to predefined intrinsic movement probabilities.

2.3.4 Extension of the Alchemist GUI

The Alchemist Swing-based Graphical User Interface (GUI) features the capability for users to select and apply a visual effect from a list of available effects to the ongoing simulation. The framework's extensibility facilitates the straightforward incorporation of new effects by developers. The `DrawZones` effect was developed, enabling the visualization of nodes' zones within the simulation environment. The application of this effect significantly aids in understanding the spatial dynamics and interactions of individuals within simulated environments. An illustration of the `DrawZones` effect in action is presented in Figure 2.8, showcasing the delineation of individual zones.

Listing 2.1: Definition of Alchemist simulation for herd behavior

```
incarnation: protelis
environment:
  type: ContinuousPhysics2DEnvironment
variables:
  ...

_herdParameters: &herdParameters
  - *zonesRadii
  - *velocities
  - *movementProbabilities
  - *stressRepulsionFactor
  - *attractionSpeedUpFactor
  - *leaderSeedChange
  - *trailerSeedChange
  - *NumberOfHerds

deployments:
  - type: GroupsDeployment
    parameters: [*ndividualsNumber, 0, 0, *worldRadius, *
        NumberOfHerds]
    contents:
      - molecule: zebra
        concentration: true
    programs:
      - time-distribution: 1
        type: Event
        actions:
          type: HerdBehavior
          parameters: *herdParameters
    properties:
      - type: RectangularArea
        parameters: [ *bodyLen, *bodyWidth ]
```

Listing 2.2: Implementation of movement logic of a herd individual

```
class HerdBehavior(node: Node<Any>, ...) : AbstractAction<Any>(node) {
  ...
  override fun execute() {
    alignDirection()
    environment.moveNode(node, getNextPosition())
  }

  private fun getNextPosition(): Euclidean2DPosition {
    for (zone in zones) {
      if (zone.areNodesInZone()) {
        if (zone == rearZone) turning() // Leader
        var movement = zone.getNextMovement()
        if (!rearZone.areNodesInZone()
          && nodeRandomizer.nextDouble() <= trailersSpeedUpProbability) {
          movement *= trailersSpeedUpFactor
        }
        return rotateVector(movement, getAngle(environment.getHeading(node)))
      }
    }
    // Single trailer
    turning()
    val movement = movementProvider.getRandomMovement()
    return rotateVector(movement, getAngle(environment.getHeading(node)))
  }
  ...
}
```

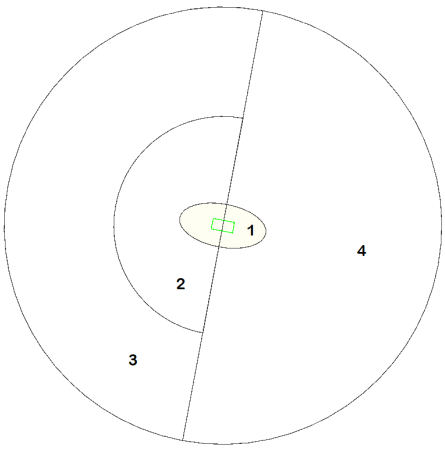

Figure 2.8: An extension to the Alchemist GUI showcasing the drawing of individual zones. Legend: (1) stress zone; (2) neutral zone; (3) attraction zone; and (4) rear zone. The rectangle within the stress zone indicates the individual's body shape.

Chapter 3

Aggregate algorithms for UAV-based herd tracking

3.1 Analysis and related work

3.1.1 Online Multi-Object k-Coverage with Mobile Smart Cameras

Approaches that track groups of animals from various perspectives using multiple autonomous UAVs, such as *in situ imageomics*, require sophisticated communication and coordination strategies. In the literature, this challenge is recognized as the ***Cooperative Multi-Robot Observation of Multiple Moving Targets (CMOMMT)*** problem [PE97], where multiple mobile robots, such as drones equipped with vision sensors, collaboratively observe and *cover* objects of interest, also known as *targets*. The specific instantiation of this problem is referred to as the ***OMOkC*** problem [EL17], where the number of cooperative robots and targets is unknown and potentially dynamic. The main objective is to efficiently operate the system to maximize the number of mobile targets covered by at least k robots (*k-covered*) over time while minimizing the associated costs. In the literature, various algorithms have been proposed for addressing the OMOkC problem [EL17, PPCE22] and have been evaluated through simulation.

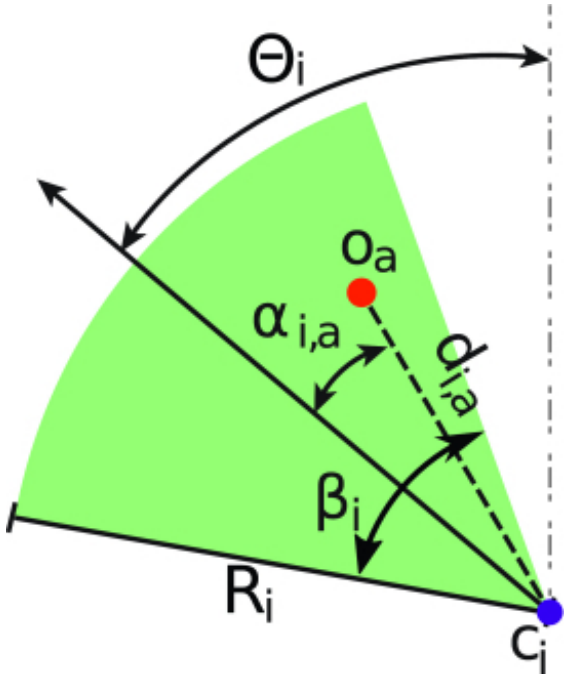

Figure 3.1: Illustration of an object o_a inside the FoV $\mathcal{V}_i$ of camera c_i [PPCE22].

OMOkC problem definition

The problem considers a set of n autonomous mobile robots as $C = \{c_1, c_2, ..., c_n\}$, equipped with vision sensors and capable of analyzing their *FoV*. These robots are assumed to have communication capabilities with other robots in the environment. The communication is based on a logical neighboring relationship, a few communication strategies outlined in Section 3.1.2.

The set of mobile objects is represented as $O = \{o_1, o_2, ..., o_m\}$. The original problem defines a subset $P \subseteq O$ containing important objects, but for herd monitoring, we will assume that all "objects" are important. Thus, all identified objects are considered targets.

The state of each robot equipped with vision sensors, operating under the assumption of a constant altitude, is modeled by a 4-tuple representation $c_i = \langle \vec{x}_i, \vec{v}_i, \omega_i, \mathcal{V}_i \rangle$. Where: *Location*, represented as $\vec{x}_i = (x_i, y_i)$, indicating the robot's position; *Velocity*, expressed as $\vec{v}_i = (v_i^X, v_i^Y) = \left(\frac{dx_i}{dt}, \frac{dy_i}{dt}\right)$; *Angular Velocity*, denoted by ω_i, describing the rate of rotation; and the *Field of View (FoV)* of robot's camera c_i, symbolized by $\mathcal{V}_i$ and is characterized by the triple $\langle \Theta_i, R_i, \frac{\beta_i}{2} \rangle$, as de-

tailed in Section 1.2.

An object o_a is considered covered at a specific time t if the object is geometrically within the field of view $\mathcal{V}_i$ of a camera c_i, as illustrated in Figure 3.1. A target is k-covered if k or more cameras cover this object at the same moment. Perfect localization is assumed.

The global objective is to maximize the number of important targets detected by the set of cameras. Consequently, achieving coverage of each target with precisely k cameras emerges as the predominant strategy for the collective, given that cameras not tracking known targets are available to explore and identify new targets. In a scenario where the ratio between the number of targets and sensors is less than k, it is automatically preferable to sacrifice some targets and achieve k-coverage in others rather than attempting to monitor all of them.

In the context of herd monitoring, where multiple animals tend to be and move near each other, to mitigate redundancy, the definition of a target should be reformulated from a single "object" to a group of objects. This introduces a new challenge: how to partition the partially covered collection of objects into distinct groups in a manner that allows each camera to cover a separate group, thus increasing the global coverage of individual objects.

3.1.2 Coordination Algorithms for OMOkC

In this study, we delve into the coordination algorithms presented in the related work "A Collective Adaptive Approach to Decentralised k-Coverage in Multi-robot Systems" [PPCE22]. The evaluation of these algorithms is based on the following parameters:

- *Exploration strategy*: outlines the base behavior of robots when the response model does not identify a target to follow;
- *Communication strategy*: specifies the criteria for selecting a subset of neighbors that each robot will communicate with;
- *Response model*: defines the approach a robot adopts in reaction to the data it receives and subsequent actions.

Table 3.1: Summary of notations for OMOkC problem [PPCE22].

Symbol	Description
C	set of cameras
O	set of objects
n	number of robots with cameras
m	number of objects
$c_i = \langle \vec{x}_i, \vec{v}_i, \omega_i, \mathcal{V}_i \rangle$	i-th camera/robot
o_i	i-th object
$\vec{x}_i = (x_i, y_i)$	i-th robot's location vector
$\vec{v}_i$	i-th robot's velocity vector
ω_i	i-th camera's angular velocity
$\mathcal{V}_i = \langle \Theta_i, R_i, \frac{\beta_i}{2} \rangle$	i-th camera's field of view
R_i	range of the i-th camera's field of view
Θ_i	orientation of the i-th camera's field of view
β_i	angle of the i-th camera's field of view
α_{ij}	angle of the j-th object w.r.t. the i-th camera's field of view
d_{ij}	distance of the j-th object w.r.t. the i-th robot

Exploration Strategy

For the *exploration strategy*, the ***force field* (FF)** exploration approach is adopted based on findings from prior research, which indicate that "Data shows that force field-based exploration outperforms the baseline ZigZag algorithm [EL17] during the bootstrap phase, however, this edge gets lower and lower with time. Data shows that *force field* exploration is a valid companion for any response model compared to the baseline" [PPCE22]. The FF algorithm is inspired by the concept of attraction and repulsion fields, where each robot emanates a repulsive force field ϕ. To prevent the system from becoming stuck in a static situation, an additional concept, termed willpower (denoted as W), is used. This allows robots to adhere to their prior decisions despite the prevailing force fields. The force fields are formulated as functions of the distance (d) between entities, delineated as follows:

$$\phi(d) = \frac{W}{2} \cdot \frac{(2\mathcal{V}_R)^2}{\max(1, d)^2}$$

where $\mathcal{V}_R$ is the distance of the field of view. This algorithm represents a coordinated method of exploration that can be efficiently implemented within the framework of aggregate computing.

Additionally, robots are programmed to rotate at maximum angular velocity ω to increase the probability of intercepting a target.

Communication Strategy

For the communication strategies of our algorithms, we will explore the following approaches:

- **no communication (NoComm)**, where robots function autonomously without any form of interaction with other robots;
- **neighborhood broadcast (BC)**, which enables a robot to communicate with all other robots within its communication range;
- **smooth (SM)**, a strategy that restricts communication based on a "spatiotemporal closeness" metric. This metric measures how long robots within communication range have been close to each other for long periods. Robots are considered close if they observe identical objects simultaneously. Conversely, when robots no longer observe the same objects, they *forget* their connections, resulting in a decrease in the measured closeness [EL17].

Response model

The comparison of response models focuses on the robots' behavior in reaction to requests received from other robots. We focus on four algorithms:

- ***Linear Programming-based Algorithm* (LinPro)**: This method derived from the concept of "... continuously solving multiple local linear programming problems defining the target selection strategy to minimize the robots' movements while attaining coverage" [PPCE22]. It involves robots sharing their fields of view with neighbors, and then each robot locally solves a classic optimization problem. Although it does not guarantee a globally optimal solution, it facilitates effective local behavior;

- ***LinPro Fair***: This variant of the LinPro algorithm aims to distribute the cameras fairly among the detected targets. The "fairness" should prevent the situation where k robots follow the same target at the cost of other targets having inadequate coverage [PPCE22];

- ***Available* (AV)**: In this method, a robot, if and only if it is not already busy following a target, attempts to cover the most recently requested target from another robot; If faced with multiple requests, the nearest target is selected based on the newest-nearest approach [EL17];

- ***Received calls* (RE)** A robot currently not following a target will provision the target with the least number of requests, as this corresponds to a small number of robots currently observing it [EL17].

Given the intrinsic requirement for communication within each response model, the *NoComm* communication strategy does not adopt a particular response model, illustrating a scenario where robots function completely autonomously.

3.1.3 Groups clustering

The primary limitation of algorithms tailored to the original OMOkC problem formulation lies in their conceptualization of the target as a singular object. This approach becomes problematic in scenarios involving animal herds, where individuals are frequently nearby. Such granularity in target definition may lead to considerable redundancy, as numerous robots may end up tracking different individuals within the same herd. Consequently, while one individual may receive excessive coverage, other herds might remain unmonitored.

To address this issue, it is proposed to shift the target concept from individual objects to groups of objects. *Clustering*, a method for identifying natural groups based on data similarity (introduced in Section 1.6), emerges as a viable solution. The core strategy involves employing a clustering algorithm to aggregate nearby individuals into the same cluster. However, the effectiveness of this approach depends on the appropriate sizing of clusters. Excessively small clusters could continue the initial problem of redundant coverage, whereas too many large clusters might exceed a single robot's coverage capacity, thus diminishing the overall

coverage effectiveness. Ideally, a *perfect clustering* approach would generate clusters that a single robot can fully cover, minimizing the overlap of clusters within the robots' FoVs.

Considering this perspective, the k-coverage process does not gain any modification. The only distinction lies in the objective that k robots aim to achieve: instead of covering a singular object, they should collectively cover the same group of objects.

Requirements for Clustering

To select an appropriate clustering algorithm from the wide variety of available approaches, it is essential to establish a set of specific requirements dictated by the current problem:

- Dynamic determination of cluster numbers is required: A fixed number of clusters may lead to excessive partitioning of the initially detected herd and neglecting the potential discovery of other animal groups;
- The number of clusters may not reflect original herd membership: Animals in proximity, despite belonging to different species, may be considered as a single target;
- Clustering should be complete: Each individual must be assigned to a cluster;
- Clusters can be heterogeneous: Clusters may vary significantly in size, shape, and density;
- The clustering algorithm must be lightweight: Given the high dynamics of the environment and the need for real-time navigation decisions, the algorithm's computational complexity must be manageable.

Hierarchical clustering

Hierarchical clustering can be a potentially more suitable solution for the problem at hand, primarily due to its natural flexibility that facilitates splitting clusters at any granularity level. Unlike *partitional clustering* algorithms, which typically necessitate pre-determining the number of clusters and struggle with heterogeneous

clusters, and *density-based clustering* algorithms, which are primarily focused on outlier detection — a feature that, in the context of animal tracking, might completely ignore the outliers that potentially have significant observational value — hierarchical clustering offers a versatile approach.

By nature, *hierarchical clustering* doesn't support varying density cluster divisions. However, this limitation can be effectively bypassed. Different groups of robots, tasked with monitoring herds of varying densities, can adjust differently their hierarchical clustering slitting distance, in this way fitting their analysis to the specific density of each animal group.

Despite the numerous advantages of *hierarchical clustering*, it may not be the best fit for datasets with a large number of elements due to its excessive computational complexity, which reaches $O(n^3)$ for Agglomerative Hierarchical Clustering [PE14]. In scenarios involving the observation of numerous individuals, a workaround may be required. For example, initially can be convenient to partition all individuals into larger clusters using faster clustering algorithms such as K-means. Subsequently, each robot could perform more detailed, fine-grained hierarchical clustering on one of these pre-calculated clusters, allowing efficient processing while maintaining the benefits of hierarchical analysis.

3.2 Design

3.2.1 Adaptive clustering

The primary distinction among various agglomerative hierarchical clustering implementations lies in the methods used to calculate the distance between clusters. This metric determines the ordering in which elements and clusters are merged throughout the hierarchy construction process. Among the methods are:

- *Single-link*: Considers the shortest distance between any two points in the two clusters;
- *Complete-link*: Looks at the longest distance between any two points in the two clusters;

- *Average-link*: Calculates the average distance between all pairs of points in the two clusters;
- *Centroid distance*: Measures the distance between the centroids of two clusters;
- *Ward's Method*: The similarity between two clusters is given by the increase in quadratic error when the two clusters are merged.

Each similarity measuring method significantly influences the resulting cluster shapes. For instance, *single-link* clustering permits the formation of non-spherical clusters but is highly sensitive to outliers, whereas *complete-link* clustering is less affected by outliers but tends to produce spherical-shaped clusters. Choosing the optimal hierarchical clustering measure necessitates a thorough evaluation. In our analysis, we will focus on the *average-link* method, which is expected to offer a balanced compromise between the characteristics of *single-link* and *complete-link* clustering methods.

Clustering distance

When selecting a clustering similarity measurement method, it's necessary to specify a distance value that will determine the number and granularity of the clusters. A typical approach to address this issue includes testing various distance values across different scenarios and then selecting a single static clustering distance value that shows the best performance. However, this strategy has drawbacks such as a lack of adaptability to environmental changes and the necessity to select a specific distance value for each hierarchical clustering method, given that each employs different units of reference.

In addition, an alternative approach is proposed: a distributed, Adaptive Hierarchical Clustering Distance Selection (AHCDS) algorithm. This method is designed to help a group of robots automatically adjust their clustering distance in real time, responding dynamically to changes in the environment.

In the *LinPro* response model, predicting neighbors' behavior necessitates that each device maintains a view of the world as similar as possible to that of its neighbors. This similarity ensures that each device can perform nearly identical

computations, resulting in the same device-to-target assignments on each device. To meet this requirement we should maintain a uniform clustering distance value across neighbors.

The algorithm description

The AHCDS is primarily designed for algorithms implementing the *LinPro*-based response model. This model facilitates the explicit *assignment* of targets to specific robots, enabling each robot to distinguish between nodes within assigned clusters and those outside. Conversely, other response models, such as AV (Available) and RE (Received Calls), require model extensions to accommodate the notion of *assigned* targets. Additionally, the term *extraneous* nodes refers to nodes that are visible to a camera but do not belong to the camera's target cluster.

The core principle of the AHCDS algorithm is to dynamically adjust the clustering distance in each computational round, following one of three options:

- **Increment distance**: Aims to expand clusters and is applicable in two scenarios:
  - When a robot successfully covers its target as well as numerous unassigned nodes, it indicates an excess of clusters relative to the number of available robots and suggests that the clusters are sufficiently small to be covered by a single camera;
  - When a camera captures a large number of extraneous nodes, signifying substantial overlap between clusters in the FoV.
- **Decrement distance**: Reduce the size of clusters. This strategy is intended to be employed when all nodes covered by camera c belong to the target cluster a, and a significant number of nodes of a are detected by other cameras but not by c.
- **Adapt distance**: Balance the clustering distance among neighboring devices, ensuring a uniform adjustment across the network.

A detailed explanation of the clustering distance adaptation process is depicted in Figure 3.2.

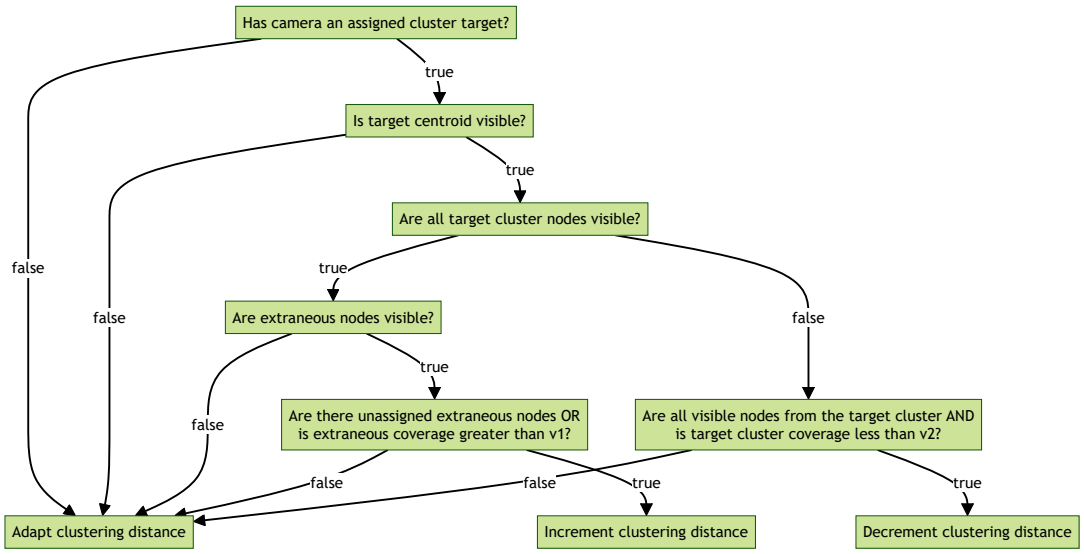

Figure 3.2: Flowchart of the hierarchical decision algorithm for the update of clustering distance value.

The principal downside of this algorithm is its difficulty in reducing cluster sizes. Such a reduction in clustering distance mainly occurs when a camera (c_1) is exploring the area around another camera (c_2) and detects new target nodes assigned to c_2 which c_2 fails to cover. Conversely, without additional exploration behavior, the clustering distance tends to not decrease.

3.2.2 Non-Nadir View Blind Spot Extension

The current implementations of smart cameras within Alchemist do not consider the UAV blind spot caused by non-nadir observational angles, which results in the terrain directly below the UAV being obscured from view. In a 2D model, this blind spot is represented as a circular sector with radius $\mathcal{V}_b$ within a FoV that has a radius $\mathcal{V}_R$, where $\mathcal{V}_b < \mathcal{V}_R$. For a more comprehensive 3D model, the blind spot would need to be dynamically calculated based on several factors, including the UAV's altitude, the vertical FoV angle, and the gimbal pitch angle, to accurately reflect the obscured area below the UAV.

3.3 Implementation Details

3.3.1 Algorithms Adaptation

The coordination algorithms for OMOkC were previously developed and integrated with the Alchemist simulator in the work of Pianini *et al.* (2022) [PPCE22], with the entire codebase made accessible through a public repository[1]. However, to ensure compatibility with the latest Alchemist API, some modifications to the code are necessary. These algorithms utilize a combination of Kotlin and Protelis programming languages. Specifically, the Kotlin segment is tasked with solving the simplex problem, whereas the Protelis portion is responsible for device coordination.

To adapt the existing code for the problem of herd tracking, a revised version of each algorithm is necessary, incorporating the clustering of visible nodes into the codebase. Since a robot cannot track an entire cluster simultaneously, the implemented solution involves following the cluster's centroid, which is the mean position of all points within a cluster. Future works could explore the use of a medoid (the most centrally located data point in a cluster), which may offer different results.

The original version of the `avoidCameraCollision()` function, designed to facilitate the coordination of robots in observing the same target from various angles, cannot work correctly with centroids, as a centroid represents a point in space rather than an actual object. A workaround for this issue involves rounding each centroid's coordinates to the nearest rounding point, ensuring that centroids nearby are rounded to the same coordinate.

For the clustering functionality, the *Smile*[2] library is employed. *Smile* is a comprehensive machine-learning library that provides a straightforward API for JVM (Java Virtual Machine) systems.

The original implementation of the *LinPro* algorithm is delineated in Listing 3.1, illustrating its initial configuration and operational logic. In contrast, the enhanced version, which incorporates clustering capabilities, is presented in Listing 3.2. This upgraded version includes the option of updating device clustering

[1] `https://github.com/DanySK/Experiment-2019-Smartcam`
[2] `https://haifengl.github.io`

Listing 3.1: Protelis code for *LinPro* response model algorithm. This code polls the neighboring robots for information about their position and the nodes they have in sight. The information is collected and sent to a local process in charge of solving the linear programming problem [PPCE22].

```
rep(solver <- getLinproSolver()) {
    let targets = foldUnion(nbr(localTargets()))
    let cameras = nbr(getCenterOfFov())
    let myTarget = solver.solve(cameras, targets, getMaxCamerasPerTarget(), false)
        .getOrDefault(getUID(), noTarget())
    followOrExplore(myTarget, fieldExploration)
    avoidCameraCollision(myTarget, localTargets)
    solver
}
```

distance with `getNewClusteringDistance()` function. It is important to note that the `foldUnion()` operator serves as an equivalent to the deprecated built-in `unionHood` operator introduced in Section 1.5.1.

3.3.2 Extension of the Alchemist GUI

The Alchemist GUI has been enhanced with two additional visualization effects:

- **FoV blind spot visualization**: This effect delimitates the sector within the field of view that remains invisible to the UAV due to its viewing angle.
- **Coloring individuals based on their cluster**: This visualization shows how drones categorize visible individuals into clusters. It is important to understand that this effect displays clustering from the perspective of a single robot. Given that each robot conducts clustering at different times and under varying initial conditions, the precise clustering outcome may differ from one robot to another.

The robots in action, along with the implemented effects, are showcased in Figure 3.3.

Listing 3.2: Protelis code for the clustering-enhanced version of the *LinPro* response model algorithm. The `linpro()` function parameters facilitate algorithm configuration by enabling the selection of an exploration strategy, the option to employ a fair version of the algorithm and the activation of adaptive clustering.

```
1
2  public def linpro(isFair, isClusteringAdaptive, explorationStrategy) =
3      rep(solver <- getLinproClusterSolver()) {
4          let localVisibleNodes = getLocalTargets()
5          let allVisible = foldUnion(nbr(localVisibleNodes))
6          let cameras = nbr(getCenterOfFov())
7
8          let clusters = getClusters(allVisible, getClusteringDistance())
9          let assignedClusters = solver.solve(cameras, clusters,
               getMaxCamerasPerTarget(), isFair)
10         let myCluster = assignedClusters.getOrDefault(getUID(), emptyCluster())
11
12         followOrExploreCluster(myCluster, explorationStrategy)
13         avoidCameraCollisionForClusters(myCluster)
14
15         if(isClusteringAdaptive){
16             let assignedNodes = getAssignedNodes(assignedClusters, clusters)
17             let newDistance = getNewClusteringDistance(myCluster,
                   localVisibleNodes, assignedNodes)
18             setClusteringDistance(newDistance)
19         }else{ 0 }
20         solver
21     }
```

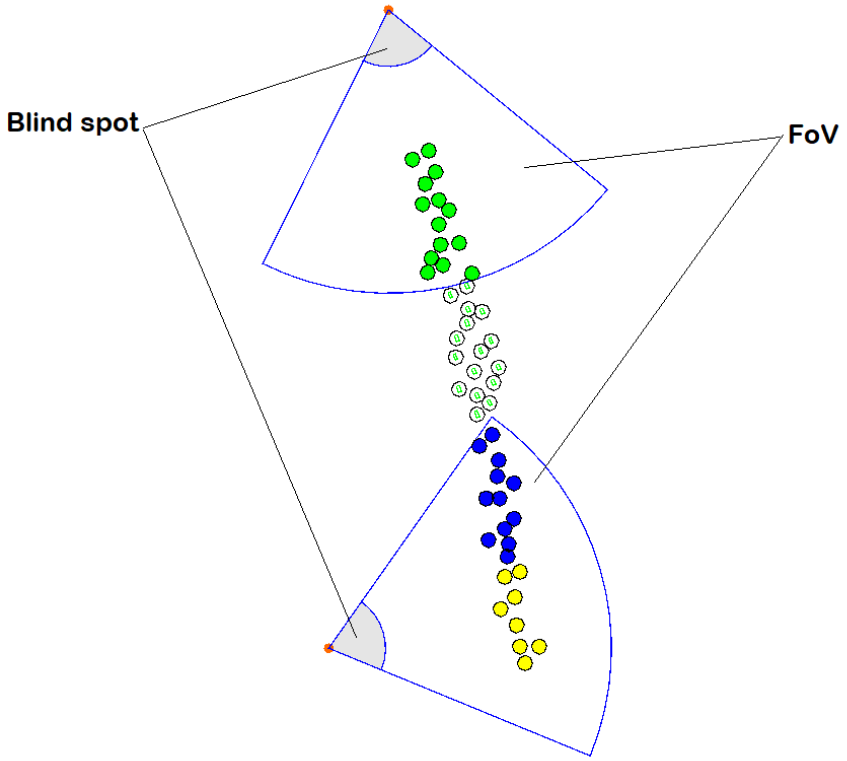

Figure 3.3: Visualization of an Alchemist GUI extension showcasing (I) UAVs blind spots; (II) the coloring of individuals based on their cluster membership, with white indicating individuals that do not belong to any cluster due to lack of coverage.

Chapter 4

Evaluation

4.1 Experimental Setup

Referring to Table 4.1, within a Euclidean two-dimensional environment, a set of zebras z is evenly distributed across m herds. These herds are randomly positioned within an area defined by a radius L, with n robots positioned at the center of this area. This setup simulates the k-coverage problem within a dynamic context, where each zebra, as a member of the herds, moves by following the rules of the herd behavior model introduced upon in Chapter 2, with specific configuration parameters detailed in Table 4.2. It should be noted that the quantity of m herds, especially in simulations involving larger-sized herds, may increase over time. This increase is resulting from the nature of the herd movement model used, which does not guarantee the integrity of the herds, potentially resulting in the splitting of a herd into smaller groups.

The configuration of each robot model is aligned with the specifications of the DJI Mavic 2S[1], detailed as follows:

- Moves with speed $\vec{v}_c$: Given the lack of specific data on the optimal speed for detecting animals, the drone's speed is derived as the mean of its maximum speed in normal mode (15 m/s) and cinematic mode (5 m/s);
- Rotates with angular velocity ω: The maximum cinematic mode angular

[1] `https://web.archive.org/web/20240229133723/https://www.dji.com/air-2s/specs`

velocity of the drone, 60°/s, is adopted;

- The depth of the field of view, $\mathcal{V}_R$: A selected depth of 100 m is used, approximating the 70-80 m distances estimated by using the KABR-telemetry dataset in the absence of precise measurements;

- The horizontal field of view angle, $\mathcal{V}_\beta$: The nominal FoV of the DJI Mavic 2S is expressed as a visual angle FOV_d across the diagonal of the screen, with an aspect *ratio*. Thus, the horizontal and vertical fields of view, $FOV_h = \mathcal{V}_\beta$ and FOV_v, are calculated as follows [SBK+11]:

$$FOV_h = 2 \cdot atan\left(\frac{tan(FOV_d/2)}{\sqrt{1 + \frac{1}{ratio^2}}}\right)$$

$$FOV_v = 2 \cdot atan\left(\frac{tan(FOV_d/2)}{\sqrt{1 + ratio^2}}\right)$$

- To calculate the depth of the blind spot, $\mathcal{V}_b$: Assuming the drone operates at a fixed altitude of 16 m, close to the median altitude found in the KABR-telemetry dataset, and a typical gimbal pitch angle of -16° combining with drone's FOV_v angle, the depth of the blind spot is approximated to be 18 m using the method outlined in Section 2.1.2.

Robots are tasked with achieving k-coverage and operate by executing an aggregate algorithm at a predefined frequency f. Each robot can communicate with others within a certain distance, denoted as the communication range R. The assumption of perfect localization and communication is applied, with idealized settings assuming that real-world inaccuracies, such as localization errors or communication disruptions, do not affect the robot's performance. All variables are documented in a summary table Table 4.1.

The simulation is conducted over $T = 1800$ seconds. For every possible combination of variable values, representing the Cartesian product of the possible values for each variable, a total of 20 simulation runs with different seeds are performed to ensure statistical significance.

The algorithms utilized in the simulations are detailed in Table 4.3, including their communication and response strategies. Notably, algorithms enhanced with clustering functionalities employ hierarchical clustering with the *average-link* method.

The data generated during these simulations is analyzed using the xarray library [HH17]. For the visualization of results, matplotlib [Hun07] is utilized, enabling the creation of detailed visual reports that illustrate the performance and outcomes of the simulation.

Table 4.1: List of the variables and their values for the simulations

Symbol	**Description**	**Values**
z	animal count	100
n	robot count	-
m	minimum herd count	2, 4, 6, 8
n/m	robots/herd ratio	0.5, 1.0, 1.5, 2.0
$\bar{v}_c$	robot linear velocity	10 m/s
ω	robot's camera angular velocity	$\frac{\pi}{3}$ rad/s
$\mathcal{V}_R$	FOV depth	100 m
$\mathcal{V}_b$	FOV blind spot depth	18 m
$\mathcal{V}_\beta$	FOV angle	80°
k	desired maximum coverage	2
L	environment arena radius	1000 m
R	robots' communication range	2000 m
f	round frequency	1 Hz
T	simulation end time	1800 s
W	Willpower for force field exploration strategy	40
–	hierarchical clustering method	average-link

4.2 Results

In Section 4.2.1, various static clustering distances are evaluated, with a more significant value selected for use in Section 4.2.2, where the original OMOkC algorithms are compared against their enhanced versions. Finally, the performance and effectiveness of the adaptive clustering algorithm are explored in Section 4.2.3.

Table 4.2: List of the herd configuration variables and their values for the simulations

Symbol	Description	Values
-	individual body length	2 m
-	individual body width	1 m
r_1	stress zone radius	3 m
r_2	neutral zone radius	20 m
r_3	attraction zone radius	40 m
r_4	rear zone radius	40 m
$p_1 : p_2 : p_3$	intrinsic directionality	1:2:1
v	intrinsic forward velocity	2 m/s
u	intrinsic lateral velocity	1 m/s
q_1	slowing-down factor for repulsion	0.5
q_2	speeding-up factor for attraction	1.5
s_1	leaders' probability to "wait"	80%
s_2	trailers' probability to "accelerate"	40%
q_3	leaders' slowing-down factor	0.7
q_4	trailers' speeding-up factor	2.0
l	prefered radius of the simulation world	1000 m
γ_1	base individual direction turning angle	1-3°
γ_2	additional individual direction turning angle	5°
s_3	probability to change direction	10%

4.2.1 Static Clustering Distances Evaluation

Initially, a comparative analysis of various clustering distances for average-link hierarchical clustering was conducted to identify a distance that provides optimal coverage results. The evaluated distances were $[10, 30, 50, 70, 90]$, with Figure 4.1 illustrating the mean coverage evolution over the simulation period. This comparison reveals the distinct impacts of clustering distances on different algorithms, with LinPro algorithms being more significantly affected, whereas BC-RE algorithms show the least impact. Clustering distances of 10 and 30 are considered too low, leading to notably lower coverage, whereas distances in the range of 50 to 70 exhibit optimal performance. An increase in distance from 70 to 90 does not show any significant impact on performance.

Notably, LinPro and Fair LinPro algorithms demonstrate nearly identical re-

Table 4.3: Algorithms considered in evaluations, described by component. All algorithms use *force field* exploration strategy.

Name	**Communication**	**Response**	**Clustering**
FF-LinPro	Neighborhood Broadcast	LinPro	None
FF-LinPro-C	Neighborhood Broadcast	LinPro	Static
FF-LinPro-AC	Neighborhood Broadcast	LinPro	Adaptive
FF-LinProF	Neighborhood Broadcast	Fair LinPro	None
FF-LinProF-C	Neighborhood Broadcast	Fair LinPro	Static
FF-NoComm	None	None	None
FF-NoComm-C	None	None	Static
SM-AV	Smooth	Available	None
SM-AV-C	Smooth	Available	Static
BC-RE	Neighborhood Broadcast	Received Calls	None
BC-RE-C	Neighborhood Broadcast	Received Calls	Static

sults, suggesting that their differences are negligible for the current configuration of the problem. Consequently, the Fair LinPro algorithm can be excluded from further evaluations as redundant.

4.2.2 Algorithms' Clustering Improvement

Following the analysis in Section 4.2.1, a clustering distance of 60 was established for the algorithms that incorporated clustering capabilities. The data visualized in Figure 4.2 depict the average k-coverage levels achieved for $k = 1$ and $k = 2$ across the simulations. This analysis reveals that, except for the LinPro algorithm, general coverage trends among most algorithms remain consistent across different simulation setups. However, LinPro's strategy of balanced target distribution across the network, when each animal is considered a target, leads to inefficiency because all robots tend to track only a single herd or even a part of a larger herd, resulting in overall unproductive coordination.

Figure 4.3 provides a comparative analysis of the average coverage results of the algorithms, highlighting that the versions enhanced with clustering capabilities outperform their original counterparts under various conditions. Among these, the FF-LinPro-C algorithm emerges as the most effective, particularly for its scalability. It demonstrates exceptional efficiency in managing multiple small herds and

expertly distributing a large number of robots. Conversely, in situations characterized by large herds and a limited number of drones, the BC-RE-C algorithm has notable results. However, it struggles to adapt to other conditions, limiting its versatility compared to LinPro.

4.2.3 Adaptive Clustering Evaluation

The final analysis, illustrated in Figure 4.4, compares the performance over time between LinPro algorithms utilizing an adaptive clustering approach (detailed in Section 3.2.1) and those that do not incorporate this feature. The simulation parameters from the initial experiment are reused here, testing various starting clustering distances $[10, 30, 50, 70, 400]$. The resulting coverage progression, as depicted in Figure 4.4, highlights the adaptive algorithm's ability to compensate for the initialization of too-low clustering distances. However, as expected, the algorithm encounters difficulties when dealing with excessively high initial clustering distances.

When properly initialized, static hierarchical clustering significantly outperforms the proposed implementation of distributed adaptive clustering. Despite this, the overall approach still exhibits its utility and potential efficacy in adapting to and effectively responding to changes within the observed environment. This indicates that, with further refinement and optimization, adaptive clustering techniques could offer valuable solutions for dynamic and complex scenarios.

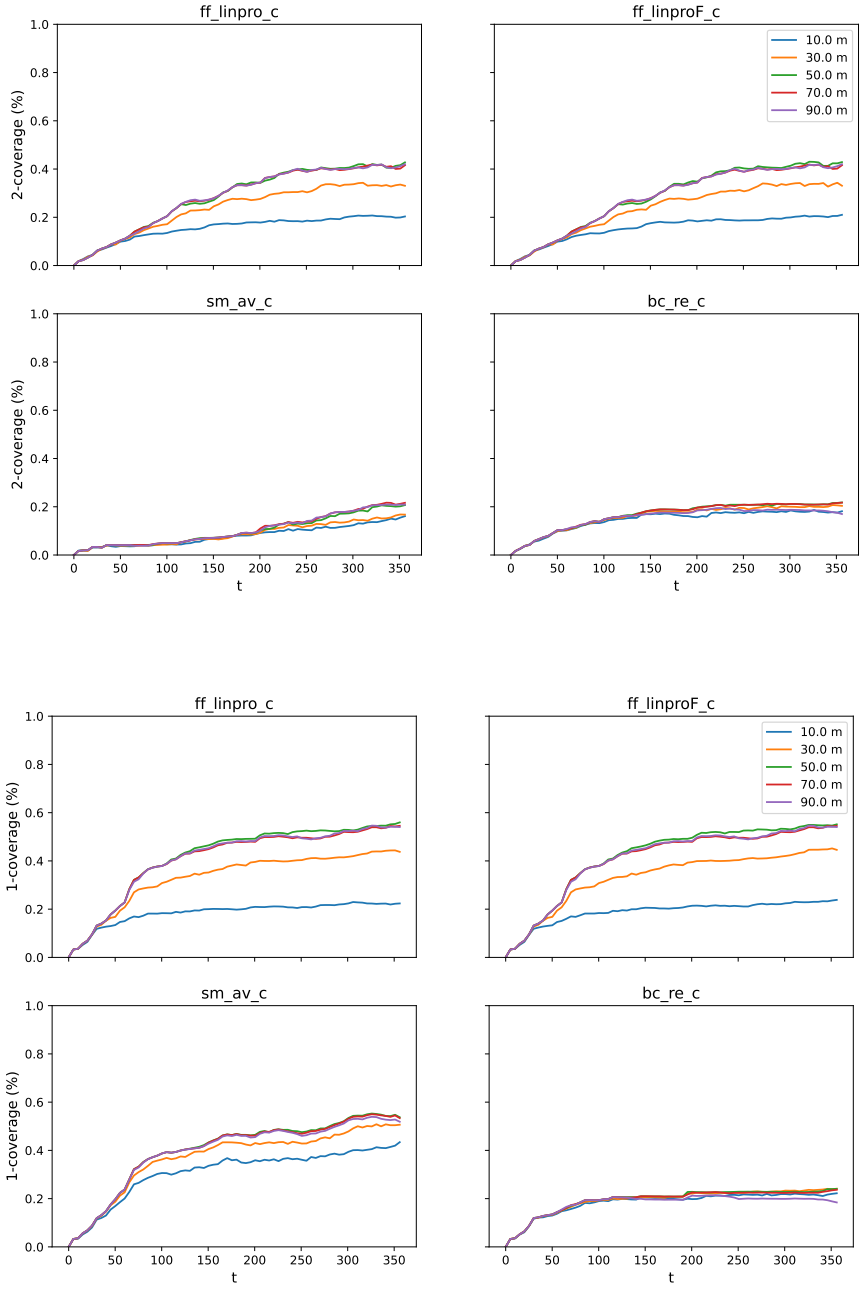

Figure 4.1: Mean coverage observed for different average-link clustering distances (in meters) over a simulation duration of 1800 seconds, sampled at 5-second intervals, with a fixed number of 6 herds. The results indicate that excessively short clustering distances lead to reduced coverage.

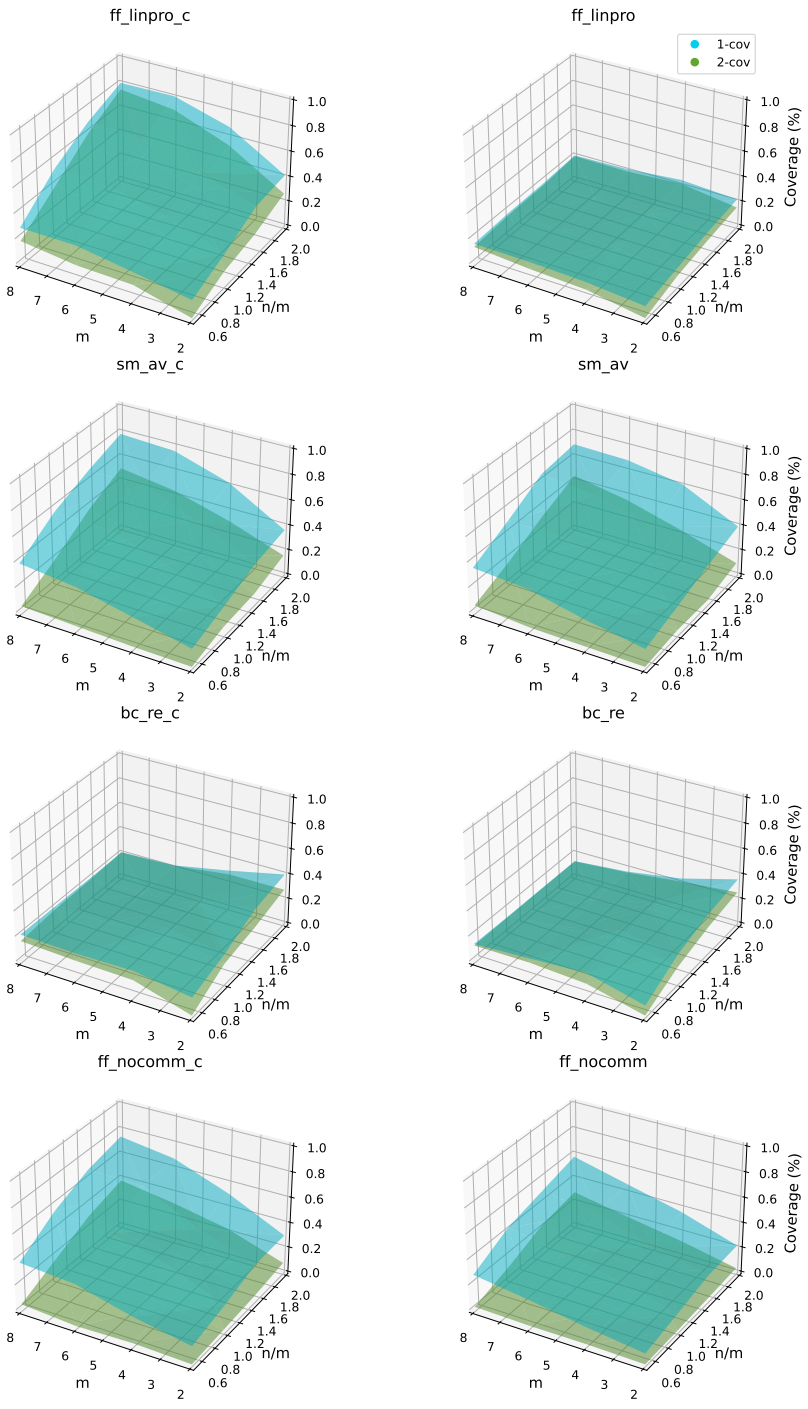

Figure 4.2: Compact representation of the performance of the algorithms under test varying the robots/herds ratio $\frac{n}{m}$ and the number of herds m. Blue surfaces are the 1-coverage levels, green surfaces are 2-coverage.

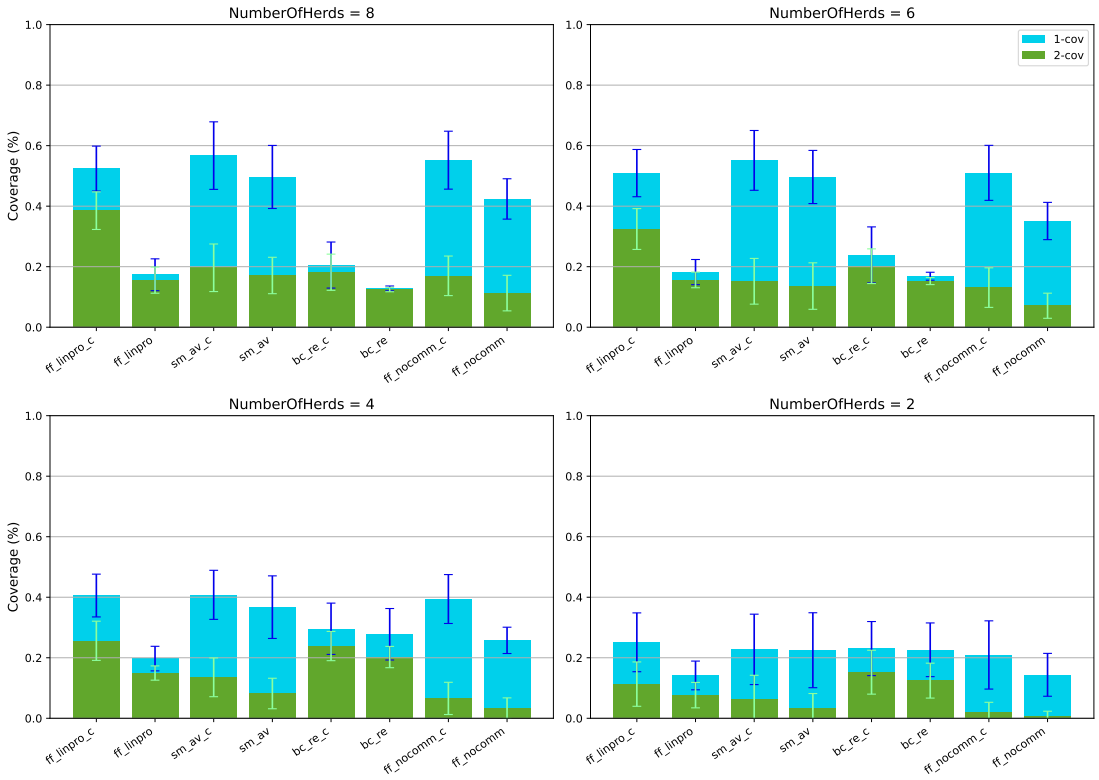

Figure 4.3: Comparison of algorithm coverage with a fixed ratio $\frac{n}{m} = 1.0$. The error bars indicate the standard deviation. The results demonstrate that algorithms enhanced with clustering capabilities surpass the outcomes of their original versions.

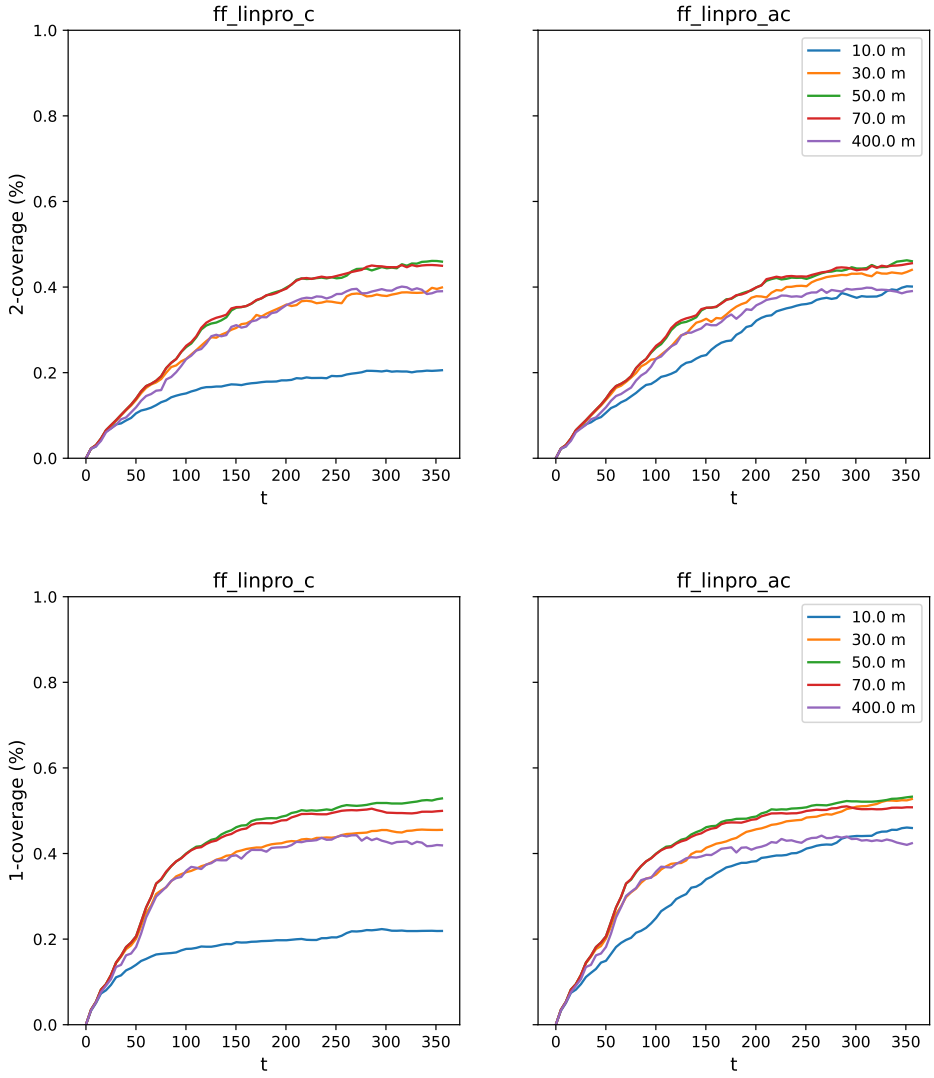

Figure 4.4: Mean coverage observed for different average-link clustering distances (in meters) over a simulation duration of 1800 seconds, sampled at 5-second intervals, with a fixed number of 8 herds. The results demonstrate the adaptive clustering algorithm's ability to mitigate coverage gaps for clustering distances initialized too low.

Chapter 5

Conclusion

In this thesis, we explore how UAVs can be utilized to autonomously track and monitor herds by applying advanced aggregate algorithms and simulation techniques. The significant contributions of this thesis include:

- **Development of a geometric approach for reconstructing animal movement paths**: This included an extensive analysis of KABR-telemetry data, implementing an approximate join technique to include missed data, and mitigating the challenge of reconstructed location jumps with a smoothing technique;
- **Development of a Herd Movement Model**: The thesis introduced a detailed model for simulating herd movements, incorporating enhancements to the existing model by adding alignment rules and transforming zone geometries for more realistic simulations;
- **Adaptation of Aggregate Algorithms for Herd Tracking**: It adapted existing OMOkC aggregate algorithms to the specific requirements of UAV-based herd tracking, focusing on optimizing the coverage of moving herds;
- **Evaluation of Algorithms through Simulation**: An extensive evaluation was conducted to compare the performance of the algorithms under various conditions, using the Alchemist simulation environment. This included testing different exploration, communication, and response strategies to determine the most effective approaches for UAV-based herd tracking;

- **Implementation of Adaptive Clustering**: To address the challenge of tracking groups of animals in a dynamic environment;
- **Extension of the Alchemist GUI**: Enhancements were made to the Alchemist graphical user interface to more comprehensively visualize herd behavior and UAV coordination.

5.1 Future works

This project makes a significant step forward in the development of distributed UAV-based herd tracking algorithms, yet there is substantial room for enhancement and refinement. Key areas for future development could include:

- **Simulation of animal responses to UAV presence**: The disturbance caused by the proximity of noisy devices to animals is a critical factor. Accurately simulating the impact of UAVs necessitates a comprehensive analysis of real-world interactions between animals and drones to model animal responses accurately. It will then be essential to develop algorithms that minimize disturbance by avoiding close contact with animals;
- **Incorporating errors in individual localization**: Given that localization algorithms can only approximate target locations, their potential inaccuracies must be accounted for, especially in distributed systems where consensus on the observed environment is crucial for effective coordination;
- **Accounting for errors in individual detection**: Object detection algorithms may face challenges in identifying animals under suboptimal visibility conditions. The environmental model needs to incorporate obstructions (like trees or other animals) that impede visibility, as well as to consider the technical limitations of cameras;
- **Introducing altitude dimensionality**: Properly modeling the animals' response to UAVs, as well as accommodating different scenarios involving various altitudes and camera angles, necessitates incorporating the dimensionality of drone altitude into the model;

- **Exploring new clustering approaches**: While the project currently focuses on implementing and evaluating the hierarchical clustering average-link method, this is not the only viable solution. Investigating alternative clustering methods may produce improved outcomes for the task at hand.

Bibliography

[AVD+19] Giorgio Audrito, Mirko Viroli, Ferruccio Damiani, Danilo Pianini, and Jacob Beal. A higher-order calculus of computational fields. *ACM Transactions on Computational Logic*, 20(1):1–55, January 2019.

[BCSK19] Jayson Boubin, John Chumley, Christopher Stewart, and Sami Khanal. Autonomic computing challenges in fully autonomous precision agriculture. In *2019 IEEE International Conference on Autonomic Computing (ICAC)*, pages 11–17, 2019.

[BPV15] Jacob Beal, Danilo Pianini, and Mirko Viroli. Aggregate programming for the internet of things. *Computer*, 48(9):22–30, 2015.

[CVAP22] Roberto Casadei, Mirko Viroli, Gianluca Aguzzi, and Danilo Pianini. Scafi: A scala DSL and toolkit for aggregate programming. *SoftwareX*, 20:101248, 2022.

[EL17] Lukas Esterle and Peter R. Lewis. Online multi-object k-coverage with mobile smart cameras. In *Proceedings of the 11th International Conference on Distributed Smart Cameras*, ICDSC 2017. ACM, September 2017.

[Gia08] Irene Giardina. Collective behavior in animal groups: Theoretical models and empirical studies. *HFSP Journal*, 2(4):205–219, August 2008.

[GLR96] Shay Gueron, Simon A. Levin, and Daniel I. Rubenstein. The dynamics of herds: From individuals to aggregations. *Journal of Theoretical Biology*, 182(1):85–98, September 1996.

[HDR24] HDR Imageomics Institute. Kabr-telemetry, 2024.

[HH17] Stephan Hoyer and Joe Hamman. xarray: N-d labeled arrays and datasets in python. *Journal of Open Research Software*, 5(1):10, April 2017.

[Hun07] John D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science and Engineering*, 9(3):90–95, 2007.

[KDK+23] Benjamin Koger, Adwait Deshpande, Jeffrey T. Kerby, Jacob M. Graving, Blair R. Costelloe, and Iain D. Couzin. Quantifying the movement, behaviour and environmental context of group-living animals using drones and computer vision. *Journal of Animal Ecology*, 92(7):1357–1371, 2023.

[KKR+24] Maksim Kholiavchenko, Jenna Kline, Michelle Ramirez, Sam Stevens, Alec Sheets, Reshma Babu, Namrata Banerji, Elizabeth Campolongo, Matthew Thompson, Nina Van Tiel, Jackson Miliko, Eduardo Bessa, Isla Duporge, Tanya Berger-Wolf, Daniel Rubenstein, and Charles Stewart. Kabr: In-situ dataset for kenyan animal behavior recognition from drone videos. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) Workshops*, pages 31–40, January 2024.

[KSBW+23] J. Kline, C. Stewart, T. Berger-Wolf, M. Ramirez, S. Stevens, R. Babu, N. Banerji, A. Sheets, S. Balasubramaniam, E. Campolongo, M. Thompson, C. V. Stewart, M. Kholiavchenko, D. I. Rubenstein, N. Van Tiel, and J. Miliko. A framework for autonomic computing for in situ imageomics. In *2023 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, pages 11–16, Los Alamitos, CA, USA, sep 2023. IEEE Computer Society.

[LTF+18] Lauren F. Laker, Elham Torabi, Daniel J. France, Craig M. Froehle, Eric J. Goldlust, Nathan R. Hoot, Parastu Kasaie, Michael S. Lyons, Laura H. Barg-Walkow, Michael J. Ward, and Robert L. Wears. Un-

derstanding emergency care delivery through computer simulation modeling. *Academic Emergency Medicine*, 25(2):116–127, 2018.

[LWHH19] Yisha Liu, Qunxiang Wang, Huosheng Hu, and Yuqing He. A novel real-time moving target tracking and path planning system for a quadrotor uav in unknown unstructured outdoor scenes. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(11):2362–2372, 2019.

[PE97] L.E. Parker and B.A. Emmons. Cooperative multi-robot observation of multiple moving targets. In *Proceedings of International Conference on Robotics and Automation*, volume 3, pages 2082–2089 vol.3, 1997.

[PE14] Shraddha K Popat and M Emmanuel. Review and comparative study of clustering techniques. *International journal of computer science and information technologies*, 5(1):805–812, 2014.

[PMV13] Danilo Pianini, Sara Montagna, and Mirko Viroli. Chemical-oriented simulation of computational systems with alchemist. *Journal of Simulation*, 7(3):202–215, 2013.

[PN94] Ernest H. Page and Richard E. Nance. Parallel discrete event simulation: a modeling methodological perspective. *SIGSIM Simul. Dig.*, 24(1):88–93, jul 1994.

[PPCE22] Danilo Pianini, Federico Pettinari, Roberto Casadei, and Lukas Esterle. A collective adaptive approach to decentralised k-coverage in multi-robot systems. *ACM Trans. Auton. Adapt. Syst.*, 17(1–2), sep 2022.

[PVB15] Danilo Pianini, Mirko Viroli, and Jacob Beal. Protelis: practical aggregate programming. In Roger L. Wainwright, Juan Manuel Corchado, Alessio Bechini, and Jiman Hong, editors, *Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13-17, 2015*, pages 1846–1853. ACM, 2015.

[SBK+11] Frank Steinicke, Gerd Bruder, Scott Kuhl, Pete Willemsen, Markus Lappe, and Klaus Hinrichs. Natural perspective projections for head-mounted displays. *IEEE Transactions on Visualization and Computer Graphics*, 17(7):888–899, 2011.

[ZOA+15] Franco Zambonelli, Andrea Omicini, Bernhard Anzengruber, Gabriella Castelli, Francesco L. De Angelis, Giovanna Di Marzo Serugendo, Simon A. Dobson, Jose Luis Fernandez-Marquez, Alois Ferscha, Marco Mamei, Stefano Mariani, Ambra Molesini, Sara Montagna, Jussi Nieminen, Danilo Pianini, Matteo Risoldi, Alberto Rosi, Graeme Stevenson, Mirko Viroli, and Juan Ye. Developing pervasive multi-agent systems with nature-inspired coordination. *Pervasive Mob. Comput.*, 17:236–252, 2015.