Technical Document

Version 1.0

Millena Cavalcanti

December, 2023

Contents

1 Introduction

Worldwide, approximately 40% of land is utilized for agricultural and livestock production (Alston and Pardey , 2014). Brazil has a significant role in this sector, being the predominant beef exporter with approximately 25% of global exports (USDA , 2022) and the second largest beef producer. The country also relies on agribusiness as a primary source of economic growth, contributing 27% to the national Gross Domestic Product (GDP) (CNA-Brasil , 2021).

Because of this multi-billion dollar market and the fact that worldwide demand for meat consumption is increasing, there is a growing need to improve livestock management. This includes improving breeding, nutrition, and overall animal health to increase productivity (Liaghat and Balasundram , 2010). An emerging approach is Precision Livestock Farming (PLF), that combines digital information and communication technologies with low-cost sensors to monitor and track animals in real-time. This animal-centric approach enables the remote monitoring and efficient management of any changes in the health or state of the smallest production unit, i.e. groups of animals (Berckmans , 2006). In particular PLF enables prompt and precise identification and handling of issues, even on extensive farms with many cattle.

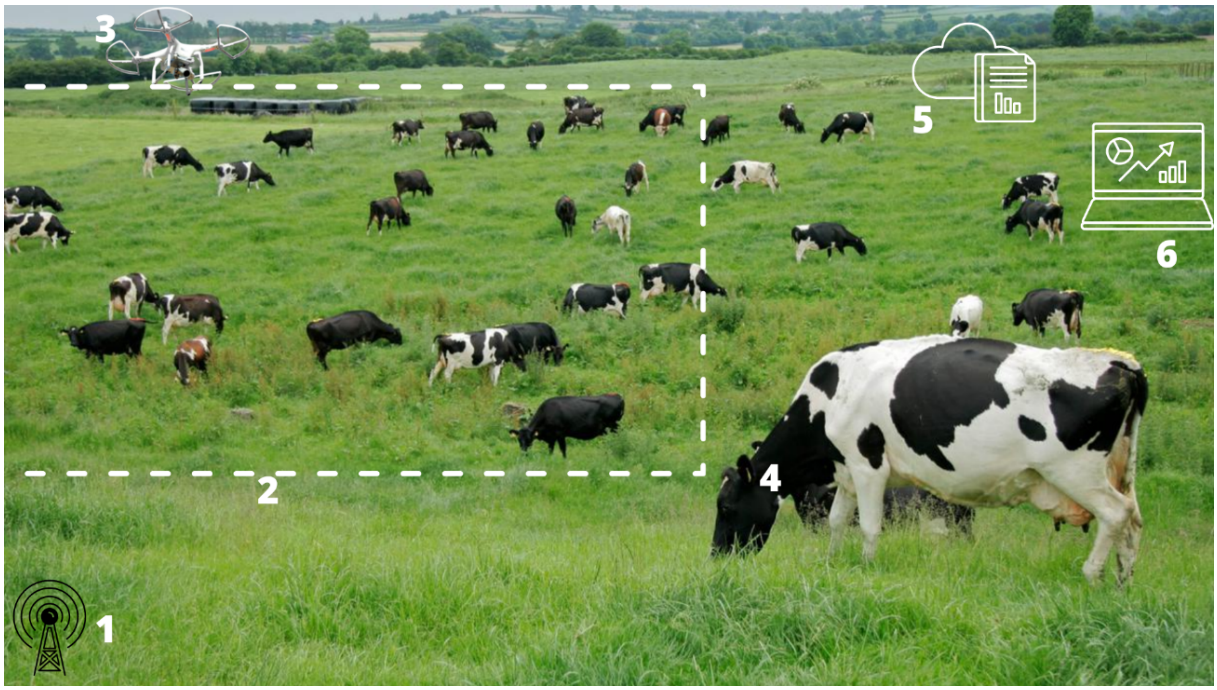

Figure 1: HMT system overview.

PLF is already widely used in intensive farming systems, where animals are kept confined in certain places, but so far rarely used in pasture-based ranching. These types of farms are large and the management of the livestock is made more difficult by the spatial variability of the feed base (i.e. quality of the pasture) and by the distances covered by the animals (Aquilani et al. , 2021). Thus, drone-assisted sensing and tracking of the livestock on these farms can be beneficial - or even necessary - for maintaining animal health and well-being, as well as for maintaining overall pasture conservation

and decision support by reducing costs.

Based on the context described above, the HMT system proposes a software for herd management and tracking using the PLF approach connected to a farm with a swarm of drones flying in coordination to cover the pasture areas, and exchanging information among them for optimized cattle monitoring and a better use of their energy resources and a greater area coverage.

2 System Overview

The HMT (Herd Management and Tracking) system is a comprehensive cattle management platform that takes an individualized, animal-centric approach, from identifying each animal to tracking veterinary and pasture data. In addition to supplying animal-specific information, it also allows for herd enumeration and the detection of absent animals in particular property areas, such as pastures.

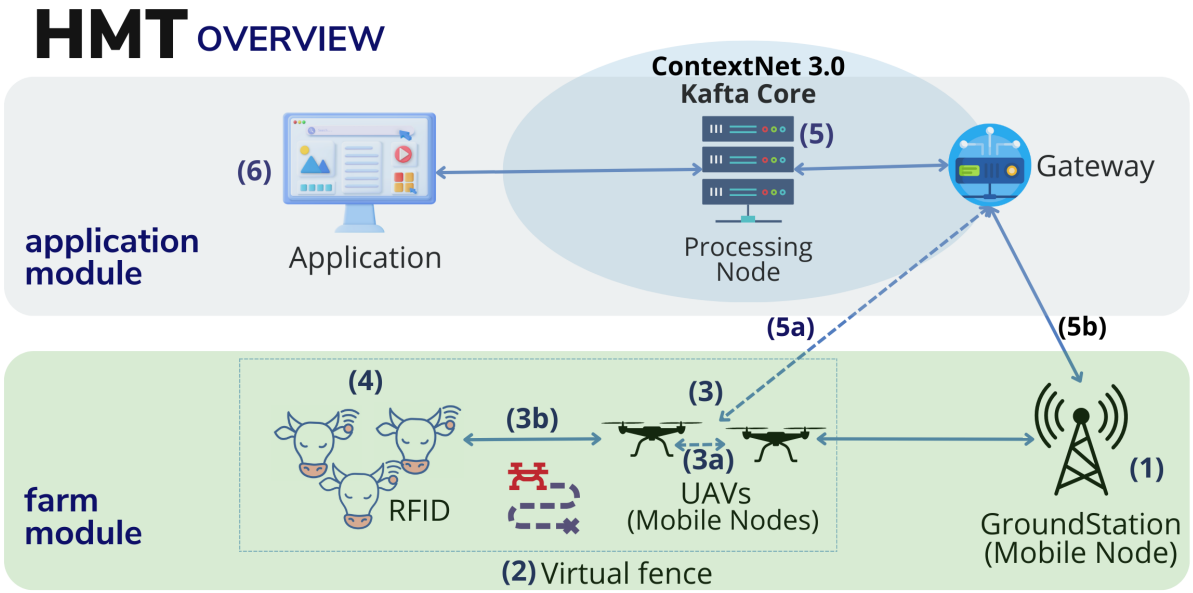

Figure 2: HMT Overview: system elements and communication.

Figure 2 presents an overview of the HMT system components. The ground station (1) deploys the UAVs (3) to detect and count the cattle (4) using identification algorithms modeled for the farm module (see Section 2.2). The acquired information flows to the cloud (5) via the ground station. The management system (6) gathers data from the ground station, UAVs, and associated devices. It also sends parameters for virtual fence settings (2) to the ground station in order to delineate the path of the tracking UAVs.

The management system is tasked with storing herd data and capturing key information about each animal, such as ID, breed, weight, and pedigree. The system enables users to create a virtual fence along the property boundary using pre-loaded satellite images and designate specific grazing zones. By monitoring animal activity within the restricted area, any lost or stolen animals can be promptly detected, allowing for quick and necessary counter measures to be taken.

The information regarding the virtual barrier is sent to the main station, which then transfers it to the drone(s) responsible for overseeing the animals inside the virtual barrier. These drone(s) fly (at low altitudes with a portable RFID reader[1]) over the region to obtain identification of each animal by communicating with RFID tags.

To ensure the transfer of sensor-collected data, UAVs on farms are equipped with external wireless communication. This collected information is periodically transmitted

[1]For example, SimplyRFiD ID-001 Mobile Reader System

to the user's application, allowing for real-time tracking without waiting for the area scan to be completed. Initially, the external communication interface can be exclusively possessed by the ground station. In this scenario, the pasture information tracking will only be transmitted after completion. One or more UAVs may possess the external communication interface to transmit tracking information prior to completion, enabling two distinct scenarios. All communication between the ground station and mobile devices with the central application (management system) and vice versa is facilitated by the ContextNet middleware [2] (Endler and Silva , 2018). The mobile devices (drones) thus act as Mobile-Hubs(Talavera et al. , 2015) and function as terminals for accessing the application.

In order to establish a clear vision and structure, the system consists of two primary, autonomous modules - Application (see 2.1) and Farm (see 2.2) - linked by a well-defined communication structure based on the ContextNet middleware.

2.1 Application Module

The Application Module supervises the HMT system and communicates with the Farm subsystem over the Internet using the ContextNet middleware, which pre-processes information. The infrastructure is separate from the user interface, which is easily accessible from any location via a web browser. The web interface displays system information, including UAV and herd listings, tracking information, and permits requests to read data. Users can modify the system by adding new animals, tracks, and fences, examining the herd's status, including individual animal information, and initiating tracking requests through this web interface.

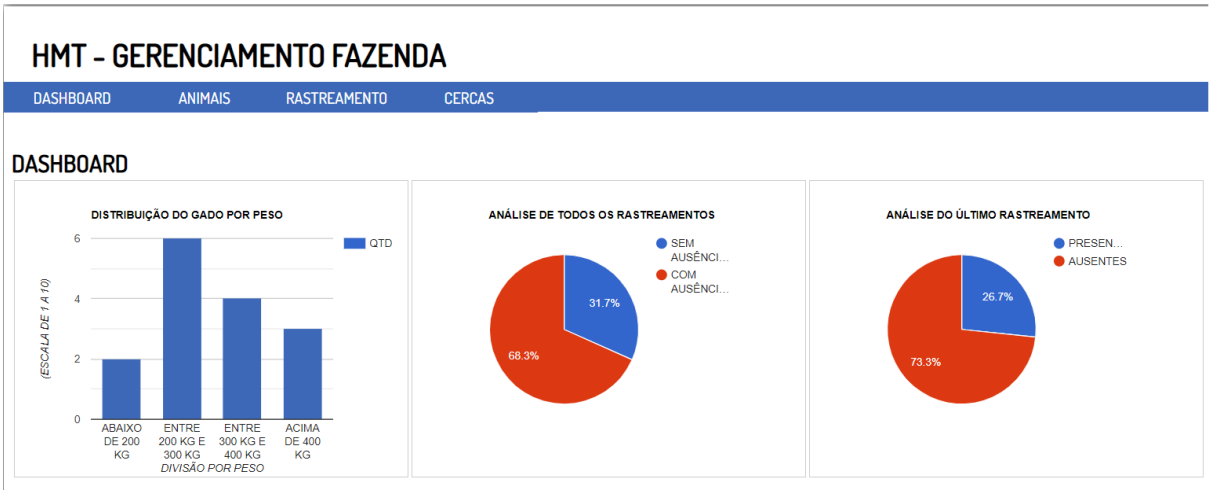

Figure 3: Screenshot of the *Dashboard* menu with general farm information

2.1.1 Functional Requirements

APP-FR 001 Website front end
The software must have a user interface in the format of a website.

[2] ContextNet webpage. https://wiki.lac.inf.puc-rio.br/

APP-FR 002 Ox management
The software's webpage front end should enable users to input and manage information pertaining to individual oxen. This information includes the oxen's breed, age, lineage, weight, birth and death dates.

APP-FR 003 Ox identification by UUID
Every ox is identified by a unique UUID generated by the system upon insertion. The UUID consists of 32 lowercase hexadecimal digits, displayed in five groups separated by hyphens, in the format 8-4-4-4-12. Its text representation is: aaaaaaaa-bbbb-1ccc-8ddd-eeeeeeeeee.

APP-FR 004 Graze tracking
The software should have a remote feature that enables users to initiate a graze tracking process to identify all the cattle within a particular grazing area. The software should display information on the ongoing tracking, including its starting time and the list of identified oxen.

APP-FR 005 Graze tracking history
All completed grazing tracking must be saved and accessible to the user, displaying the start and end time of the tracking process, and the list of identified and absent cattle.

APP-FR 006 Dashboard
The application must have a dashboard that displays real-time data about cattle health, cattle weight information, and actual and past graze tracking information.

APP-FR 007 Grazing alerts
The application must show notifications for important events related grazing tracking, as missing ox(en).

APP-FR 008 Virtual fence
The application must present the virtual fences available.

APP-FR 009 UAVs
The application must show the available UAVs.

2.1.2 Non Functional Requirements

APP-NFR 001 Scalability
The system should be able to scale up to accommodate an increasing number of oxen and grazing tracking data, as well as to handle additional modules and functionalities as the farm's needs evolve.

APP-NFR 002 Interoperability
The system should be able to seamlessly communicate with other modules and systems via the ContextNet middleware, ensuring smooth integration and data exchange between different components of the farm management system.

2.1.3 Architecture

The HMT software consists of two main layers: frontend and backend (Figure 4). Communication between the two takes place via the Farm interface of the backend, with message exchange occurring at this interface and database level. The front-end deploys the graphical user interface, retrieving data from the database and displaying it to the user through visual dashboards.

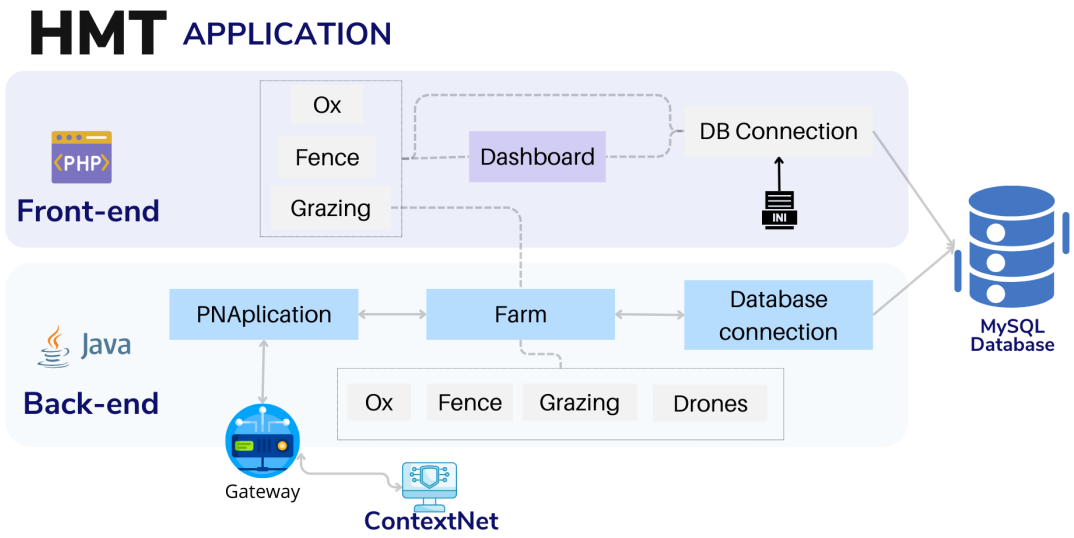

Figure 4: HMT Application simplifyied architecture.

The backend layer interfaces the business layer of the application, connecting with all external elements like the database, web user interface, and ContextNet for communication with the farm simulator.

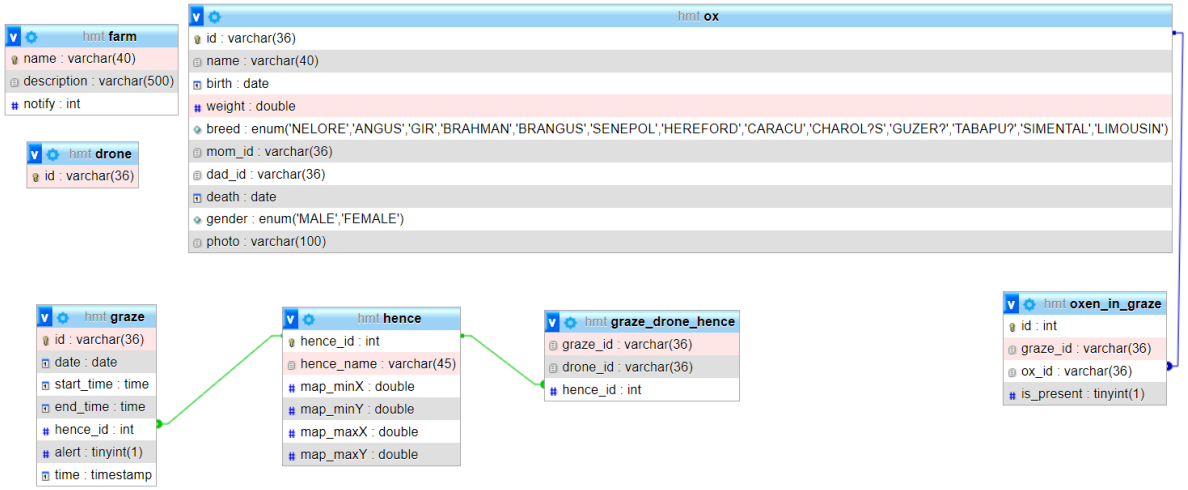

Figure 5: SQL database diagram.

The database, in turn, was modeled to reflect the attributes and relationships defined

by the business layer. In this way, it reflects the class structure of the backend layer. The relationships and entities can be seen in Figure 5.

2.1.4 Technologies Used

The back-end of the application module was implemented in Java, while the front-end was implemented using PHP 7.4.3, HTML5 and CSS3 for user visualization. All the information generated is stored in a MySQL 8.0.31 database

2.1.5 Use Cases

In this section, we provide use cases concerning the HMT application, covering both successful and problematic scenarios.

Successful use cases

APP-SUC-SCENARIO 01 Insert an ox into the farm animals list

User: A farmer, a farm manager or a researcher in PLF (Precision Livestock Farming).

A new ox was born or bought and the user wants to insert its information in the livestock managment system. He/She navigates to the *Animais* (Animals) section of the HMT farm webpage and clicks on the *Cadastrar* (Insert) option to add new information. A blank form is displayed for the user to fill in the new ox details such as name, breed, gender, date of birth, weight, sire and dam. He/She also have the option to upload a photo of the ox. Once all the information is entered, the user saves the new ox entry and it is added to their livestock inventory. The application then updates the inventory database with the new ox's information, including its unique identification number. The user can now easily track and manage the ox's through the application.

APP-SUC-SCENARIO 02 Visualize a past graze tracking

User: A farmer, a farm manager or a researcher in PLF (Precision Livestock Farming).

The user wants to view grazing track data from a specific day or period. To do so, they will go to the *Rastreamento* (Tracking) section of the HMT farm webpage and click on *Listar/Rastrear* (List/Track). A table will display showing all the tracks, with each row representing a track and columns showing the date, ID, number of identified and absent cattle. In each column, there is an empty field where the user can enter information to search for the desired track. The database is searched for relevant trackings, which are then displayed to the user. The

user then selects a specific grazing track record from the list, which they wish to view. The web application presents comprehensive information about the selected grazing track, which includes the ID of the grazing track, the initial date and time of the grazing period, the final date and time of the grazing period, a list of cattle present during the grazing period, and a list of cattle absent during the grazing period.

Problematic use cases

APP-PROB-SCENARIO 01 Request a tracking without farm simulation

User: A researcher in PLF (Precision Livestock Farming) or software tester.

The user needs to track their cattle in the pasture in order to locate any missing ones. They can access the tracking section of the HMT farm webpage by clicking on *Listar/Rastrear*(List/Track). The left side of the screen displays a list of all trackings, while the right side houses the *Rastrear* (Track) button. However, pressing the button does not initiate any tracking. The tracking will begin and return to the previous screen without updating the tracking list or indicating a tracking is in progress.

How to fix: Since the farming module is still being tested in the simulation phase, the user must initialize the GrADyS-SIM simulator with the Farm Module simulation project. Once initialized, the tracking request feature will function properly.

APP-PROB-SCENARIO 02 Request a tracking without gateway connection

User: A researcher in PLF (Precision Livestock Farming) or software tester.

The user needs to track their cattle in the pasture in order to locate any missing ones. They can access the tracking section of the HMT farm webpage by clicking on *Listar/Rastrear*(List/Track). The left side of the screen displays a list of all trackings, while the right side houses the *Rastrear* (Track) button.However, pressing the button does not initiate any tracking. An error message regarding communication will appear instead.

How to fix: Communication errors occur when a gateway to access the ContextNet middleware is not found. It occurs when a gateway is not initialized on the same machine as the backend application or if the IP address of the gateway is incorrect. To resolve this issue, users should follow the

instructions in the installation guide to initiate the gateway (section 4.1).

2.2 Farm Module

This module holds the utmost significance for the HMT system as it provides an outline and monitoring mechanism for all activities related to the control and maintenance of PLF on farms. The livestock farm elements, including the devices and animals, are structured within the Farm subsystem, which comprises elements (1), (3), (4), and (5) as depicted in Figure 2

- The **stationary ground station** serves as the central communication hub between the devices present on the property and the externally hosted management system;
- **Drones** are utilized for monitoring the perimeter of the pasture field and have the ability to read cattle data (by communicating with the RFID ear tags of each animal using an on-board portable RFID reader), the ground station, and the management system;
- The animals of the **herd** are individually identified by semi-passive RFID ear tags;
- Other **Sensors** are used to collect animal health information.

2.2.1 Functional Requirements

FARM-FR 001 Customized number of drones
The module should grant users the capability to deploy drones for conducting aerial surveys of the virtual grazing region.

FARM-FR 002 Customized number of cattle
The module should allow users to determine the quantity of cattle within the grazing area.

FARM-FR 003 Stationary ground station
The module should have a stationary ground station that communicates with the drones and the application module.

FARM-FR 004 Drone flight path definition
The module must enable the user to specify flight path parameters, via geografic waypoint files, for the drones to ensure comprehensive scanning of the grazing area.

FARM-FR 005 Cattle movement
The farm simulation module should accurately simulate the movement of a group of oxen within a virtual grazing area.

FARM-FR 006 UAV movement
The farm simulation module should accurately simulate the movement of a drone UAV.

FARM-FR 007 Real-time cattle identification
The farm module should simulate the identification of cattle by drones when they are within the communication area.

FARM-FR 008 Drone-drone communication
The farm module should simulate communication between drones when they are within the communication area. While communicating, they should exchange the identified UUIDs of the oxen.

FARM-FR 009 Communication with the application module
The farm module needs to communicate with the application module by receiving configuration information, including the number and IDs of drones and cattle, to begin the tracking process. The list of identified cattle should then be sent to the application module using either the stationary ground station or drones. Data exchange occurs through the internet using the ContextNet middleware.

FARM-FR 0010 Tracking algorithm and protocol
The user should be able to choose the cooperation algorithm used by the drones to scan the grazing area delimited by the virtual fence and the communication protocol used by them to communicate.

2.2.2 Non Functional Requirements

FARM-NFR 001 Drone-drone communication standard
The drones must communicate using a wifi radio frequency standard.

FARM-NFR 002 Drone-cattle communication standard
The drones need to communicate with the cattle using either RFID or Wifi as a radio frequency standard.

FARM-NFR 003 Interoperability
The farm module should be able to seamlessly communicate with the application module via the ContextNet middleware, ensuring smooth integration and data exchange between different components of the farm management system.

2.2.3 Architecture

The Farm Module underwent simulation through the GrADyS-SIM simulator (Lamenza et al. , 2022; Olivieri et al. , 2021).This simulator expands upon the INET++(INET , 2022) networking library, which is supported by the OMNET++(OMNet++ , 2022) framework. GrADyS-SIM is ideal for implementing and simulating UAV swarm coordination strategies for collecting sensor data in the field. It is used to observe UAV movement, exchange messages, and validate communication implementation between the UAVs, oxen, ground station, and cloud, as well as the quality of the area tracking algorithm. Versions 6.0.1 of OMNeT++ and 4.5 of INET++ are necessary.

The nodes were developed by extending the modules *MobileNode*, *MobileSensorNode*, and *GroundStation* of GrADyS-SIM. For farm device integration, we added the FarmNode module as an extension to the GroundStation. The FarmNode module simplifies external communication with the application for simulation configuration. This process activates and configures UAVs to scan a virtually delimited area, known as a "fence," for animal tracking.

Mobility behavior of the simulation's nodes is defined by the mobility models for UAVs, oxen, GroundStation, and FarmNode, respectively labeled as *UAVMobility*, *LinearMobility* and *AttachedMobility*, and *StationaryMobility*, all provided by GrADyS-SIM and/o INET++. All mobility models transfer telemetry data to the node's communication protocol, as can be seen in Figure 7, which then defines movement commands based on the gathered information and input from other nodes.

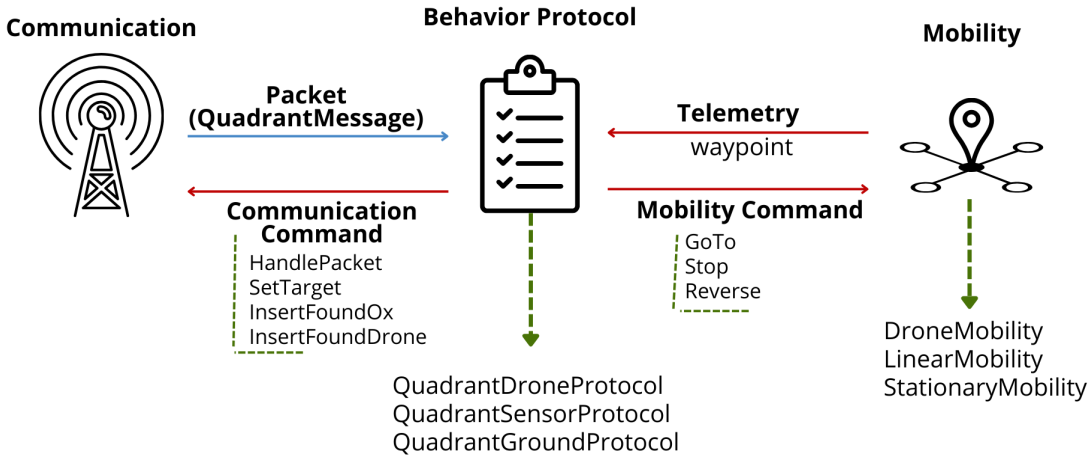

Figure 7: Diagram of the UAV, sensor, and ground station modules.

To ensure the transfer of data collected by sensors, drones are equipped with external wireless communication on the farm. The collected information is periodically transmitted to the user's application, allowing for real-time tracking without waiting for completion of the area scan. First, the ground station can exclusively possess the external communication interface. In this scenario, the tracking of pasture information will be transmitted only after completion. Second, one or more drones can possess the external communication interface and transmit the tracking information before completion.

Thus, the system enables two different scenarios. The communication between the drones and between the drones and the ground station utilizes the protocol chosen by the user, which can be QuadrantProtocol or HICAProtocol, which were specifically designed for this system. The user can also implement its own protocol by following the the instructions given by GrADyS[3]. The chosen colaborative swarm drones algorithm and protocol comprises several components: (i) mobility information, including nearby node and return point identification; (ii) telemetry data; (iii) a list of identifiers (IDs) used to store and communicate the drones that reader nodes have interacted with; and (iv) the data collected from mobile sensors. The chosen protocol, which include the initiating node's data, are used to transmit this information, including: (i) the ID and type of the

[3]GrADyS User Manual: https://brunoolivieri.github.io/gradys-simulations

sending module (DRONE, SENSOR, or GROUND_STATION); (ii) a list of sensor IDs that the sender module has identified; and (iii) a list of drone IDs that it has already discovered.

2.2.4 Use Cases

In this section, we provide use cases concerning the Farm module, covering both successful and problematic scenarios.

Successful use cases

FARM-SUC-SCENARIO 01 Start a pre-configured simulation
User: A distributed system or UAVs swarm coordination researcher.

The user wants to simulate and analyze a swarm of drones tracking a grazing area to identify present and absent cattle using a virtual fence. To achieve this, using GrADyS-SIM, the user should open the "*showcases/sim/quadrant_simulation*" directory, select the *omnetpp.ini* file, right-click and choose "*Run As* $\rightarrow$ *OMNeT++ Simulation*" from the menu options. This will open a new window displaying the simulation view. The user presses either the '*Run*' or '*Fast*' button (refer to Figure 8) to initiate the simulation. Throughout the simulation, the user can observe drone and cattle movements as well as exchanged messages.

FARM-SUC-SCENARIO 02 Start a simulation with pre-configured settings that enable communication with the HMT application.
User: A distributed system or UAVs swarm coordination researcher.

The user wants to simulate the grazing area tracking by drones and the communication with the external HMT application. To achieve this, using GrADyS-SIM, the user should open the "*showcases/sim/quadrant_simulation*" directory, select the *omnetpp.ini* file, right-click and choose "*Run As* $\rightarrow$ *OMNeT++ Simulation*" from the menu options. This will open a new window displaying the simulation view. The user presses either the '*Run*' or '*Fast*' button (refer to Figure 8) to initiate the simulation. Throughout the simulation, the user can monitor the movements of drones and cattle, as well as the messages exchanged between the HMT application and the ground station.

Figure 8: Run button in farm simulation.

Problematic use cases

FARM-PROB-SCENARIO 01 Execute a new simulation using different communication protocols in drones and sensors (cattle)
User: A distributed system or UAVs swarm coordination researcher.

The user creates a new simulation settings file (.ini), defining and initializing drones, sensors (that represents the cattle in the simulation), and ground station. When they are defining the protocol of each element, the user chooses different protocols for drones and sensors or drones and ground station. Then they select the new *.ini* file, right-click and choose "*Run As* $\rightarrow$ *OMNeT++ Simulation*" from the menu options. This will open a new window displaying the simulation view. The user presses either the '*Run*' or '*Fast*' button (refer to Figure 8) to initiate the simulation. Throughout the simulation, the drones fail to detect any cattle, even when they are within the drone's communication range, tracking is completed with full missing cattle information.

How to fix: The cattle identification fails because communication between drones and cattle is not possible. To facilitate a successful exchange of information, users must edit the simulation settings file and ensure compatibility between the protocols selected for drones and sensors. This can be verified by checking the protocol name at the beginning. For instance, if the drones are using the QuadrantDroneProtocol, the sensors should use the QuadrantSensorProtocol.

FARM-PROB-SCENARIO 02 Execute a new simulation using very low communication timeout
User: A distributed system or UAVs swarm coordination researcher.

The user creates a new simulation settings file (.ini), defining and initializing drones, sensors (that represents the cattle in the simulation), and ground station. When they are defining the timeout of each element, the user considers that the

communication can be done realy fast, not considering any packages loss or communications interferances, chooses values near zero. Then they select the new *.ini* file, right-click and choose "*Run As* $\rightarrow$ *OMNeT++ Simulation*" from the menu options. This will open a new window displaying the simulation view. The user presses either the '*Run*' or '*Fast*' button (refer to Figure 8) to initiate the simulation. Throughout the simulation, the drones were unable to complete communication with the majority of the cattle and failed to complete the animal identification process. The tracking was done with almost no information about the cattle at all.

How to fix: The cattle identification is unsuccessful due to incomplete communication between drones and cattle. To facilitate successful information exchange, users must modify the simulation settings file to ensure the timeout value is at least 5 seconds.

2.3 External Communication

The communication between the application and farm module (via base station) was implemented in Java using the ContextNet 3.0 (ContextNet webpage) Kafka Core middleware libraries. Figure 2 shows this communication through items 2 and 3, where the Farm and Application Modules communicate through the Mobile Node (MN) and Processing Node (PN), respectively. MN and PN are elements defined by ContextNet. Via the PN, the application sends and receives messages from an MN identified by a unique ID, which can be either a drone or base station. ContextNet is responsible for controlling and processing the communication directly linked to the PN, while also coordinating it. The MNs access the PN communication through network gateways.

The messages exchanged between these components are formatted as *NodeMessage*, with message fields such as SENDER, ACTION, TRACKID, STATUS, DRONES, and OXIDs, which are all delimited on the left. Only the first two fields are required for the action command. Control messages sent by the PN use unicast forwarding to the MN for requested instructions. Control messages are used for setting up tracking to inform drones and the area to fly over, as well as requesting a grazing scan to track animals.

The mobile nodes (MNs) send two types of messages based on the communicating element. MNs used by base stations transmit availability information, while MNs associated with drones transmit collected information for processing in the processing node (PN). The received messages include a list of animal identifiers detected by drones in the pasture.

3 HMT Application - User Manual

3.1 Installation Guide

The HMT application is composed of two layers: the backend application, implemented in Java and responsible for all business and communication modeling, and the frontend application, developed in PHP and responsible for the user interface. The backend application runs independently, but the web interface cannot execute all functionalities without the backend module, such as initializing a grazing tracking or getting information from the farm module.

3.1.1 Backend application

In order to run the backend application, you need to have JavaSE 17 or higher and Maven installed.

To run the application, first you need to configure the database, by installing MySQL version 8.0.31, or higher, and configuring it with the database structure avaliable in file htm.sql. For initializing it, run the following command:

```
$ mysql -u <username> -p hmt < htm.sql
```

After configuring the database, verify that the computer's PATH variable contains Java's *bin* directory. If auto-run is enabled, attempt to run the ***HMT-PN-APPLICATION-1.0-jar-with-dependencies.jar*** file by double-clicking it.

If double-clicking fails, execute the JAR file through the command line or terminal window, using the following command:

```
jar -jar HMT-PN-APPLICATION-1.0-jar-with-dependencies.jar
```

It will run the programm, using the following default settings for the mysql database connection

```
MYSQL_ADDRESS = "jdbc:mysql://45.56.126.161:3306/"
DATABASE_USERNAME = <username>
DATABASE_PASSWD = <password>
```

and ContextNet gatway connection.

```
GATEWAY_IP_ADDRESS = "127.0.0.1"
GATEWAY_IP = "scp.inf.puc-rio.br"
```

For other settings values, run the .jar file with the arguments in the ordem presented:

```
jar -jar HMT-PN-APPLICATION-1.0-jar-with-dependencies.jar
      <mysql_address> <username> <database_password>
      <gateway_ip_address> <gateway_ip>
```

To connect with the farm module simulation, you must also run the ContextNet Kafka Core (ContextNet webpage) middleware. The application attempts to connect with ContextNet at all times to exchange information with the farm module. However, it provides the ability to use other functionalities that do not rely on new tracking information. To initiate ContextNet gateways, download the ContextNet Kafka Core from github:

```
$ git clone https://gitlab.com/lac-puc/context-net-kafka-core.git
```

And raise the Docker image of the gateway:

```
$ sudo docker-compose -f /opt/ContextNet/CKC/docker-start-gw.yml
   up
```

If you want to check if the images are running:

```
$ sudo docker ps -a
```

3.1.2 Frontend web application

In order to run the web application, you need to have installed PHP 7.4.3, HTML5, CSS3, and a web server (we recomend Apache). This web server will host the front-end hmt web application in PHP.

If you want the data to persist, install a MySQL version 8.0.31 or higher, create the database and import the SQL file using the following command:

```
$ mysql -u <username> -p hmt < hmt.sql
```

Replace <username> with the name of your database user.

Copy the *hmt-web* directory to the PHP server directory. Then, enter the new location of hmt-web and open the config.ini file to modify the settings values (servername, username, password, and dbname). Make sure that the updated data reflects the actual database access information, and that <username> and <password> are the same used in the MySQL database (section 3.1.1), in the backend application settings (section 3.1.1), and in the frontend application config.ini file (section 3.1.2).

Open the terminal and navigate to the root directory of the *hmt-web*. Proceed to run the following command to initiate the embedded PHP server.

```
$ php -S localhost:8000
```

This will allow the hmt application to be accessed via *http://localhost:8000*.

3.2 Application Overview

The HMT web application, as can be seen in Figure 9 offers four menus: Dashboard, Animals, Tracking, and Fences. The ***Dashboard*** is the index page with graphics displaying cattle and grazing data. In the ***Animais*** menu, users can manage cattle by listing, adding, and editing animal information. In the ***Rastreamento*** menu, users can view and initiate grazing tracks. Lastly, the ***Cercas*** menu displays virtual fence information.

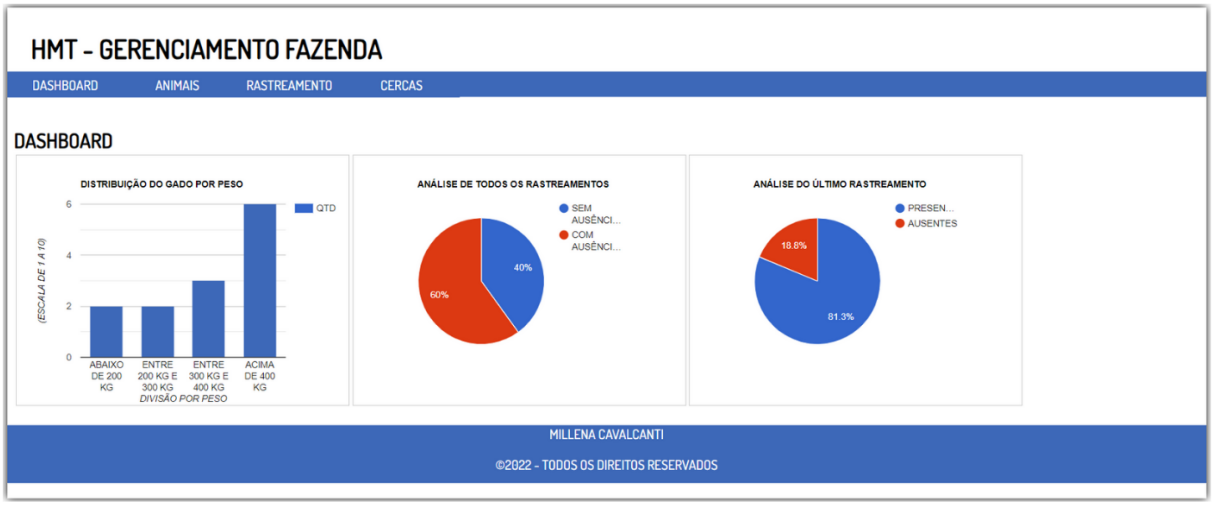

Figure 9: Dashboard menu screenshot.

3.3 Dashboard

In *Dashboard* menu three interactive graphs shows actualized information about cattle. The board *cattle distribution by weight*, presents the number of animals that are in each range of weight. This information is important to the farmer identify how many oxen are ready or near to abate, how many must have some attention. With the mouse over the bar, the user can see the exact number, as presented in Figure 10.

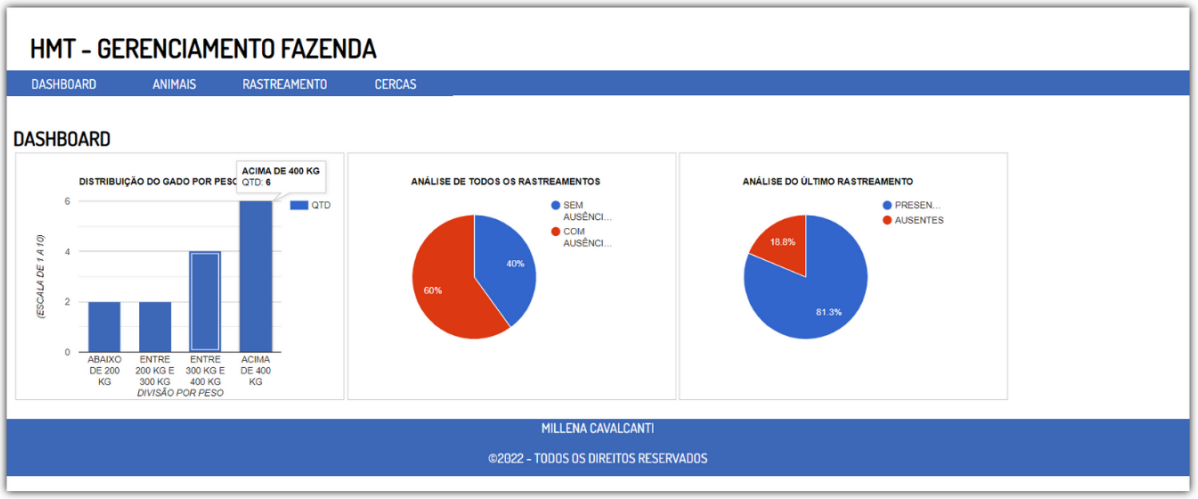

Figure 10: Cattle distribution by weight board information.

The dashboard displaying grazing tracking history (Figure 11) shows the count and percentage of tracking executions that had an alert for a missing animal.

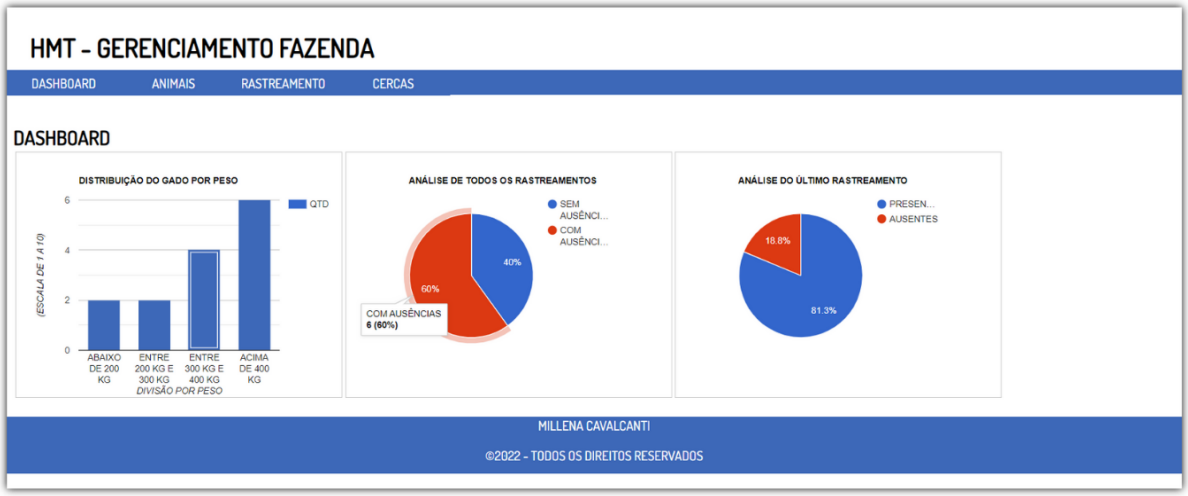

Figure 11: Cattle distribution by weight board information.

Information about the last grazing track execution is presented in last tracking analysis board. In this board the user can see if it was identified one or more ox absences and how many animals were detected or not.

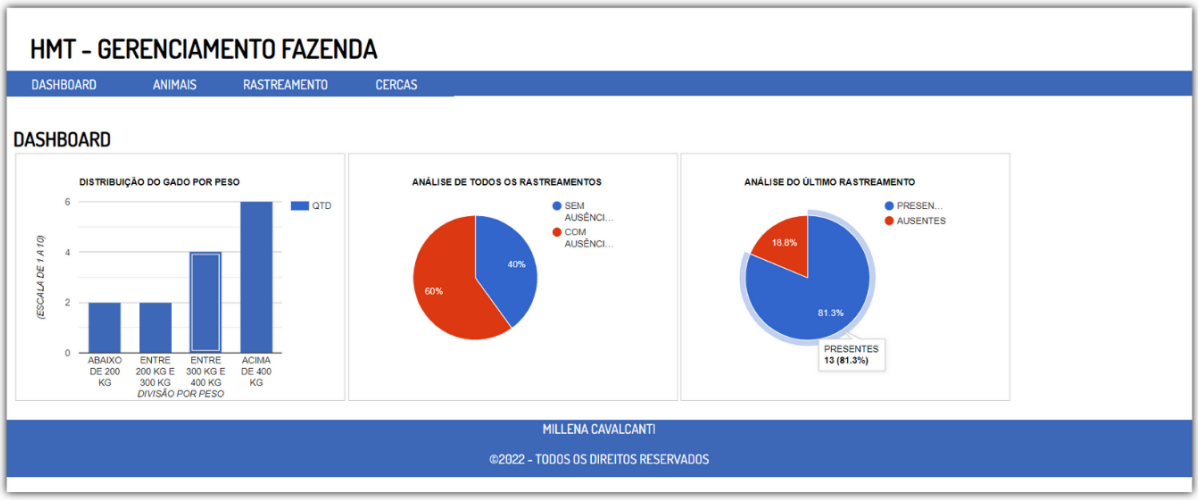

Figure 12: Last summarized grazing tracking information.

3.4 Herd Management

The herd management information can be found in the Animals menu, which offers two options (refer to Figure 13): list the cattle or add a new ox. The List animals option displays a table listing all animals that are, or have been part of, the farm's cattle. The table includes each animal's UUID, name, breed, weight, and gender. To make searching easier, users can enter the desired information in any table field, narrowing down the list to the animals that exhibit the searched characteristic. When a table row is selected, all the relevant information regarding that particular ox is displayed on the right-hand side of the table.

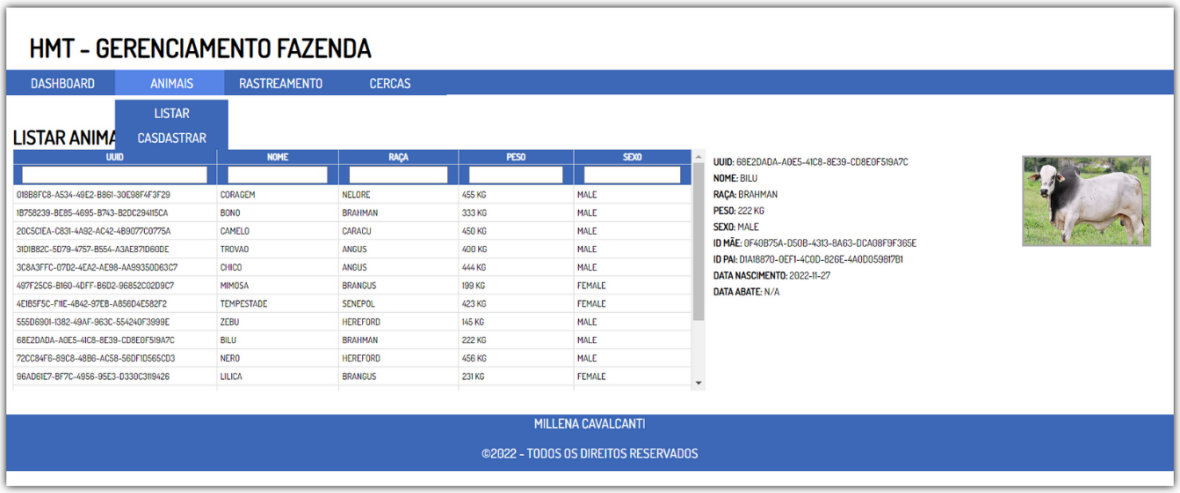

Figure 13: Animals menu screenshot.

To add a new ox to the database, click "Add Ox" and enter the necessary information requested in the screen presented in Figure 14. The animal ID can be obtained from an RFID tag or randomly generated. Additionally, provide the breed, weight, name, gender, mother, father, birth date, and death date. A picture of the animal can also be added to the database by clicking on the image field and selecting a file from the system. Once all the necessary information has been entered, the user should press the OK button to save the animal information to the database. The fields mother, father and death date are not mandatory.

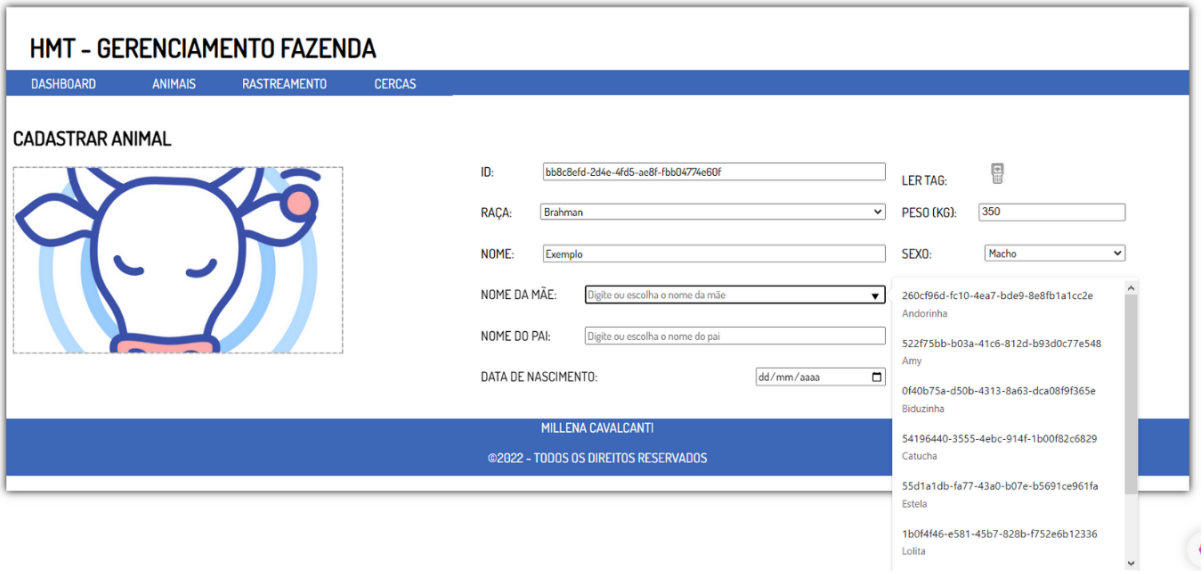

Figure 14: Insert an ox menu screenshot.

3.5 Grazing

The grazing tracking information is located in the Tracking (*Rastreamento*) menu. A table format displays the history of all tracking events by date, ID, and the number of found and absent oxen. Users can enter specific information in any field to narrow down the list of tracking events, making searching easier. When a row in the table is selected, the relevant information about that grazing track is displayed on the table's right-hand side.

The user can initiate a grazing track in the farm simulation module by pressing the Track (*Rastrear*) button at any time, but it only works if the simulator (refer to section 4) is already open.

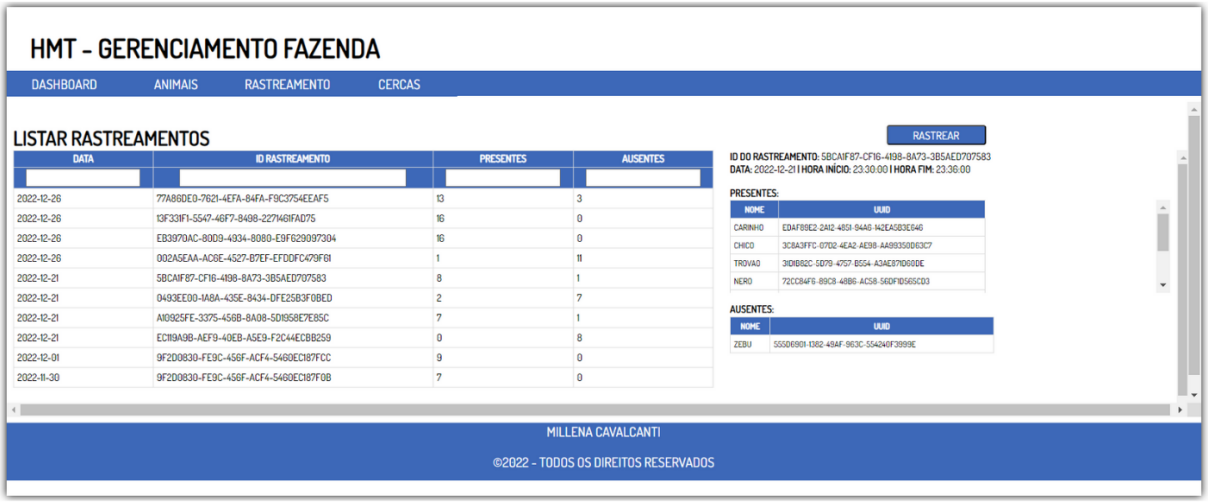

Figure 15: Grazing track menu screenshot.

3.6 Virtual Fence

The last menu displays an image of the farm area with a virtual fence boundary. The boundary is highlighted with red lines in Figure 16.

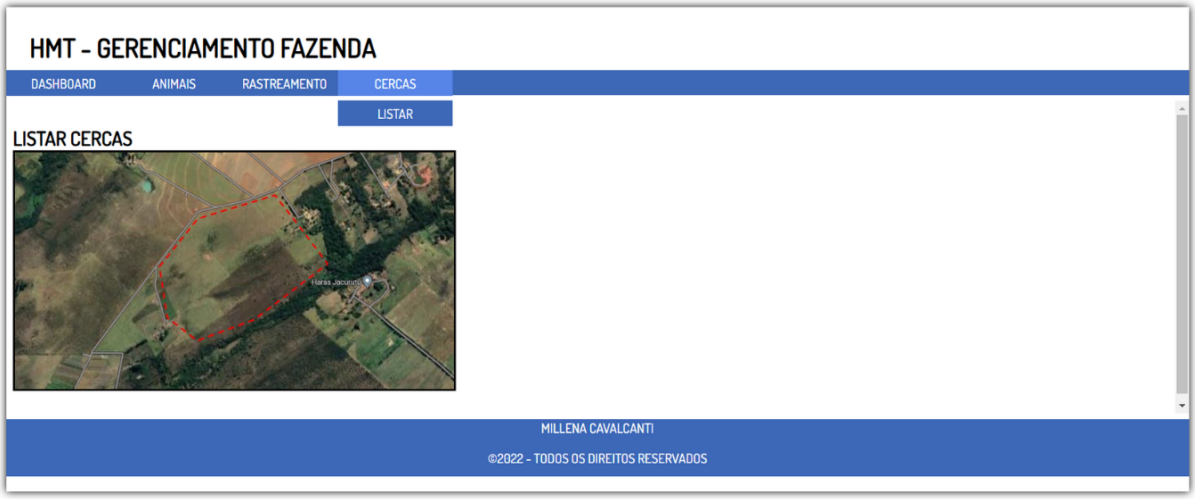

Figure 16: Virtual fence menu screenshot.

4 Farm Simulation - User Manual

4.1 GrADyS Installation Guide

There are three methods for installing and utilizing Gradys-Sim, the essential tool for simulating the farm module identification and tracking algorithm. The initial approach is to obtain the offered virtual appliance, which can be imported into any virtual machine software supporting the .ova file format. And finally, the third step is to execute a Docker image with the installation of GrADys-SIM.

4.1.1 Virtualization

Download the virtual appliance file (.ova) at ***this link*** and import it into any virtual machine software that supports the .ova format. The imported virtual machine should already contain a user named "gradys" with the password "gradys". We recommend using ***Oracle's VirtualBox***.

4.1.2 Local Instalation

In order to run the simulations and use the components in this repository you need to have both OMNeT++ and the INET framework installed.

Version 6.0.1 of OMNeT++ is required, to install it just follow ***these instructions***. INET version 4.5.0 is also required, when first opening the OMNeT++ IDE you should be prompted with the option to install INET and all you need to do is accept it but if you need help ***check out the installation instructions***.

After installing both OMNeT++ and INET, you should be able to clone the repository into your active OMNeT++ IDE workspace. To do this select File → Import... then open the "git" section and select "Projects from git" then "Clone Url". After that just enter the URL for this repository and finish the process following the instructions on the screen.

Make sure you have INET installed in your workspace and that it is selected as the project reference. If you see a directory named inet, then INET4.5 is installed in your workspace. You also need to check if it is selected for the *gradys-simulations* project. This can be done as follows: Right click on the *herd_identification project* → *Properties* → *Project References*. Make sure that INET is selected.

4.1.3 Docker image

If you are using a Linux machine...

Requirements: docker
Note: *We recommend the use of Ubuntu 20.04 LTS distribution.*

If there is no docker installed, update apt's package index and install packages to allow apt to use a repository over HTTPS:

```
$ sudo apt update
$ sudo apt install apt-transport-https ca-certificates curl
   software-properties-common
```

Add the official GPG key of the Docker repository:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg
$ sudo apt-key add -
```

Add the Docker repository to the system:

```
$ sudo add-apt-repository "deb [arch=amd64] https://download.
   docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

Update apt's package index again and install Docker:

```
$ sudo apt update
$ sudo apt install docker-ce
```

Check that Docker has been installed correctly:

```
$ sudo systemctl status docker
```

If you are using Windows 10 or higher...

Requirements: WSL2, DOcker Desktop, Ubuntu 20.04

First you will need to install the Docker Desktop, following ***these instructions***.

If you do not have the WSL2 installed in your machine, open the Microsoft Store app and search for WSL2 (Windows Subsystem for Linux) and click in the *Get* button. Reboot your machine after the initial installation to complete the setup.

Note: You can also install Ubuntu from the command line, following ***these instructions***.

With WSL2 installed, open the Microsoft Store app and search for Ubuntu 20.04.6 LTS and click in the *Get* button. After installation, you have two options to access the application: launch it directly from the store or search for Ubuntu using the Windows search bar.

Note: You can also install Ubuntu from the command line, following ***these instructions***.

On the first use, it will execute a setup, which can take a few time, and you will need to create a username and password (this does not need to match your Windows user credentials). When it is ready for use, we recomend to install the latest updates with the following commands, entering your password when prompted.

```
$ sudo apt update
$ sudo apt full-upgrade
```

Press Y when prompted.

Now that your linux instalation is set and updated, install a GUI package to be able to run and see the simulation in a 2D or 3D maps in GrADyS-SIM. Follow ***these instructions*** to install and configure the GUI package.

Execute docker...

Configure the enviroment to run the docker image, enabling the execution of GUI application throught docker, entering the following commands:

```
$ export DISPLAY=:0.0
$ xhost +local:docker
$ export LIBGL_ALWAYS_SOFTWARE=1
$ export MESA_GL_VERSION_OVERRIDE=3.3
```

Then, execute the docker using the command:

```
$ docker run -ti -e DISPLAY=$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-
   unix kamysek/gradys-simulations bash
```

If you want to store the information executed in docker in a local directory, you can map a directory changing the command above by:

```
$ docker run -v <mapping_dir>:<mapping_dir> -ti -e DISPLAY=
   $DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix kamysek/gradys-
   simulations bash
```

4.1.4 Starting a farm simulator

Once you have installed omnet++ locally or in a docker image, you can open the program by double-clicking on the icon or by using the command line.

```
$ omnetpp
```

With omnet++ open, clone the farm module project from GitHub by right-clicking on the Project Explorer view. Then select "*Import*" from the menu bar, followed by "*Git*" and "*Project from Git*". Enter the repository address for the project.

```
https://github.com/milliandrade/herd_identification.git
```

The cloned project consists of GrADyS-SIM and simulation files for the farm module.

Note: on the first use, INET++ will be installed in OMNeT++.

4.2 Initializing a simulation

GrADyS-SIM is an OMNeT++ framework project with the farm module serving as a subproject named *quadrant_simulation*. The Project Explorer view, on the left side of the Figure 17, this project's organization. The simulation settings are defined in the *quadrant_simulation* directory, utilizing communication and mobility models, protocols, and messages created within GrADyS (found in the src/gradys_simulations/ directory).

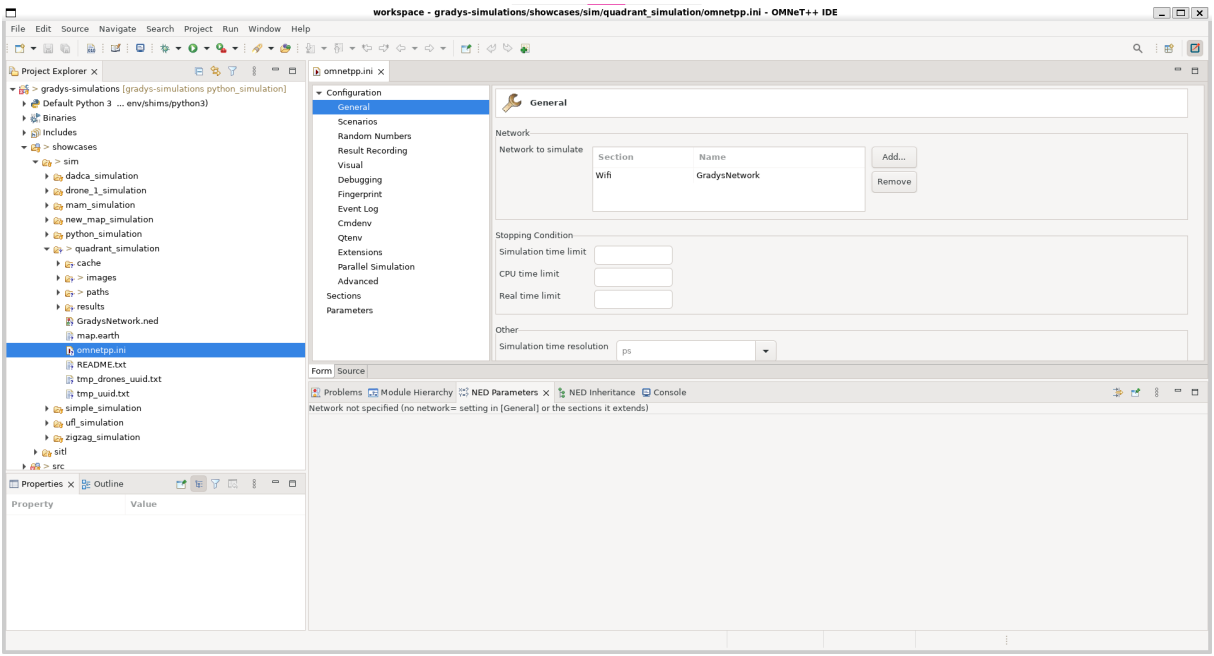

Figure 17: Simulator overview.

The *showcases/sim* directory contains a list of simulation projects that are configured and ready to run. To run any simulation using the omnetpp-GrADyS-SIM tool, select the simulation project folder in the *showcases/sim* directory labeled *<projetc>_simulation*. There is a ***.ini*** file in this folder. Right click on this file and choose *Run As* $\rightarrow$ *OMNeT++ Simulation* from the menu bar to run a simulation.

To run a farm module tracking simulation, select the *quadrant_simulation/omnetpp.ini*. The *quadrant_simulation* performs a cattle identification and tracking using the quadrant tracking strategy and protocol. In this strategy, each drone has a defined mission path configured by waypoints and they follow the path searching for cattle using radio frequency identification signals. During the mission, when they find other drones, they exchange the list of identified cattle.

The *Run As* $\rightarrow$ *OMNeT++ Simulation* option opens a new window displaying the simulation environment. The simulation window is presented in Figure 18. The main view, detached with the number 1, graphically shows the simulation environment and its elements. The scenario presents a 3D visualisation of the simulation, accompanied by a map view that aligns with the actual geographic coordinates designated for the farm. The 2D mode can display the drones, ground station, and cattle, identified by their corresponding icons and labels - quads, groundStation, and sensors - each with its own unique simulation identification number. Simply select the mode by clicking on the buttons marked in 2.

To initiate the simulation, the user is required to press the run button in 3. If the user desires to adjust the running mode, the user can select step, fast, or express. In *step mode*, each simulation event executes per time, whereas in *fast mode*, message animation is disabled, resulting in a potential 10-fold increase in speed compared to run mode. *Express mode* is the quickest, but it disables all tracing and limits interaction with the simulation. All module settings and information are located in view 4, while communication and event logs can be found in view 5.

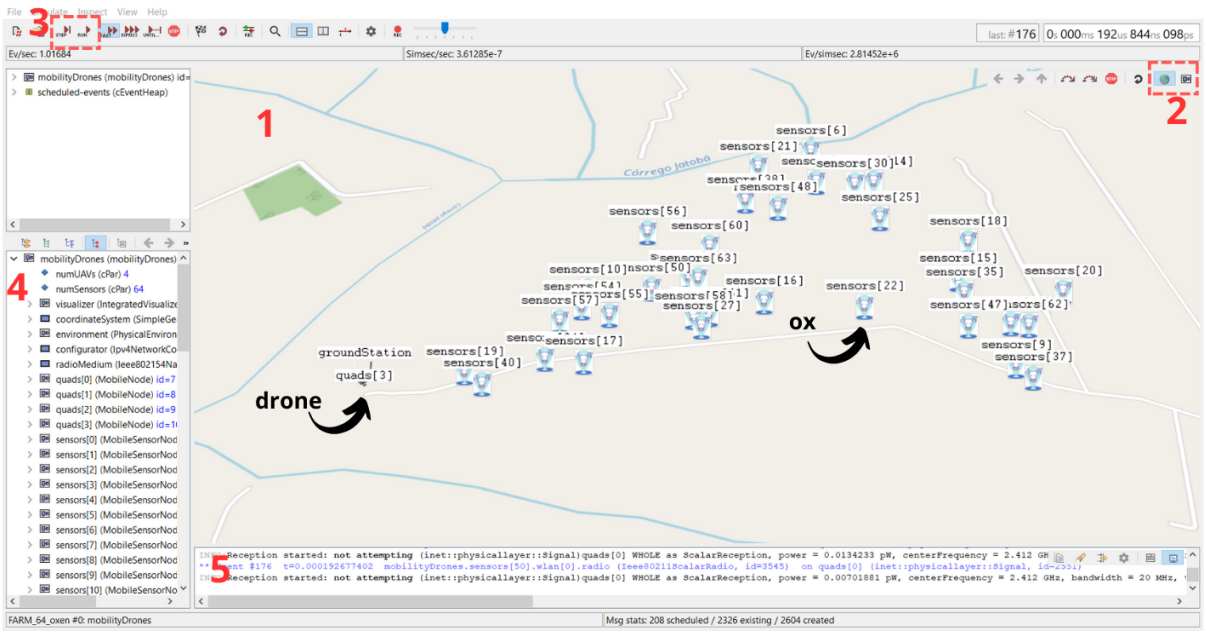

Figure 18: Simulator overview.

4.3 Configuring a simulation

In the .ini file, it is necessary to configure the geographic coordination of the farm and all its elements (farm, sensors, drones and ground stations).

4.3.1 Farm

The first thing to do is to define the geographic coordinates of the central point of the simulation map. It coincides with the location of the farm, which is defined by its longitude and latitude in degrees, as shown in the code below.

```
##FARM
*.coordinateSystem.sceneLongitude = -48.09779867095303deg
*.coordinateSystem.sceneLatitude = -15.630554093095277deg
*.coordinateSystem.sceneHeading = 0deg # scene orientation
```

To change the latitude and longitude values, we recommend that you use *Google Maps* to identify the correct geographic coordinates of the desired location. In the map, click on the desired location and it will display the tuple (latitude,longitude) values.

4.3.2 Drones Setup

In *[Config Wifi]* section, configure the drone protocol, mobility model, initial geolocation and mission path waypoints. If any other configuration is added, the values defined in the default configuration (*[Config Wifi]*) are overwritten.

Number and connections

```
*.numUAVs = 4
...
*.quads[0].app[0].destAddresses = "quads[1] quads[2] quads[3]
   sensors[0] sensors[1] sensors[2] sensors[3] sensors[4]
   sensors[5] sensors[6] sensors[7] groundStation"
*.quads[1].app[0].destAddresses = "quads[0] quads[2] quads[3]
   sensors[0] sensors[1] sensors[2] sensors[3] sensors[4]
   sensors[5] sensors[6] sensors[7] groundStation"
*.quads[2].app[0].destAddresses = "quads[0] quads[1] quads[3]
   sensors[0] sensors[1] sensors[2] sensors[3] sensors[4]
   sensors[5] sensors[6] sensors[7] groundStation"
*.quads[3].app[0].destAddresses = "quads[1] quads[2] quads[0]
   sensors[0] sensors[1] sensors[2] sensors[3] sensors[4]
   sensors[5] sensors[6] sensors[7] groundStation"
```

The destAddresses field defines the list of devices with which the drone can communicate. Devices that are not in this list will be ignored during identification, as well as any messages sent by them.

Protocol

```
# --- communication parameters -------
...
*.quads[*].protocol.typename = "QuadrantDroneProtocol"
```

Mobility and Geographic location

```
# --- mobility parameters -------
*.quads[*].mobility.typename = "DroneMobility"
...
*.quads[*].mobility.homeLongitude = -48.097798deg
*.quads[*].mobility.homeLatitude = -15.630554deg
...
*.quads[0].mobility.startTime = 2s
*.quads[1].mobility.startTime = 4s
*.quads[2].mobility.startTime = 6s
*.quads[3].mobility.startTime = 0s
...
*.quads[0].mobility.waypointFile = "paths/farm-100ha-drone-A.
   waypoints"
*.quads[1].mobility.waypointFile = "paths/farm-100ha-drone-B.
   waypoints"
*.quads[2].mobility.waypointFile = "paths/farm-100ha-drone-C.
   waypoints"
*.quads[3].mobility.waypointFile = "paths/farm-100ha-fence.
   waypoints"
```

4.3.3 Ground Station Setup

In *[Config Wifi]* section, configure the ground station protocol, mobility model, and geolocation. If any other configuration is added, the values defined in the default configuration (*[Config Wifi]*) are overwritten.

Protocol

```
# --- communication parameters -------
...
*.groundStation.protocol.typename = "QuadrantGroundProtocol"
```

Mobility and Geographic location

```
# --- mobility parameters -------
*.groundStation.mobility.typename = "StationaryMobility"
...
*.groundStation.mobility.initialAltitude = 0m
*.groundStation.mobility.initialLongitude = -48.097798deg
*.groundStation.mobility.initialLatitude = -15.630554deg
```

Connections

```
*.groundStation.app[0].destAddresses = "quads[0] quads[1] quads
   [2] quads[3]"
```

The destAddresses field defines the list of devices with which the ground station can communicate. Devices that are not in this list will be ignored during identification, as well as any messages sent by them. As only information sent to or received from drones are considered by the ground station.

4.3.4 Sensors/Cattle Setup

In *[Config Wifi]* section, configure the mobile sensors, used to represent the cattle, protocol, mobility model, and geolocation.

Number and connections

```
*.numSensors = 8
...
*.sensors[*].app[0].destAddresses = "quads[0] quads[1] quads[2] 
   quads[3]"
```

The destAddresses field defines the list of devices with which the sensor can communicate. Devices that are not in this list will be ignored during identification, as well as any messages sent by them.

Protocol

```
        # --- communication parameters -------
        ...
        *.sensors[*].protocol.typename = "QuadrantSensorProtocol"
```

Mobility and Geographic location

```
        # --- mobility parameters -------
        *.sensors[*].mobility.typename = "LinearMobility"
        *.sensors[*].mobility.constraintAreaMinX = 10m
        *.sensors[*].mobility.constraintAreaMinY = 10m
        *.sensors[*].mobility.constraintAreaMinZ = 0m
        *.sensors[*].mobility.constraintAreaMaxX = 150m
        *.sensors[*].mobility.constraintAreaMaxY = 150m
        *.sensors[*].mobility.constraintAreaMaxZ = 0m
        *.sensors[*].mobility.speed = 5mps
        *.sensors[*].mobility.initialMovementHeading = 10deg
```

References

Jilian M. Alston and Philip G. Pardey. Agriculture in the global economy. *Journal of Economic Perspectives*, 28:121–146, 2014. doi: http://dx.doi.org/10.1257/jep.28.1.121.

C. Aquilani, A. Confessore, R. Bozzi, F. Sirtori, and C. Pugliese. Review: Precision livestock farming technologies in pasture-based livestock systems. *Animal Behaviour Science*, 16(1), 2021. doi: https://doi.org/10.1016/j.animal.2021.100429.

D. Berckmans. Automatic on-line monitoring of animals by precision livestock farming. Wageningen Academic Publishers, 2006.

CNA-Brasil. Panorama do agro. Technical report, CNA Brasil, 2021. URL https://cnabrasil.org.br/cna/panorama-do-agro.

ContextNet webpage. URL https://wiki.lac.inf.puc-rio.br/.

M. Endler and F. S. Silva. Past, Present and Future of the ContextNet IoMT Middleware. *OJIOT*, 4(1):7–23, 2018.

INET. Inet framework, 2022. URL https://inet.omnetpp.org/.

Thiago Lamenza, Marcelo Paulon, Breno Perricone, Bruno Olivieri, and Markus Endler. Gradys-sim - a omnet++/inet simulation framework for internet of flying things. In *Anais Estendidos do XL Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 9–16, Porto Alegre, RS, Brasil, 2022. SBC. doi: 10.5753/sbrc_estendido.2022.222426. URL https://sol.sbc.org.br/index.php/sbrc_estendido/article/view/21413.

S. Liaghat and S. K. Balasundram. Past, present and future of the contextnet iomt middleware. *Open Journal of Internet of Things (OJIOT)*, 5(1):50–55, 2010. doi: https://doi.org/10.3844/ajabssp.2010.50.55.

Bruno Olivieri, Thiago Lamenza, and Marcelo Paulon. Gradys-sim simulator., 2021. URL https://github.com/brunoolivieri/gradys-simulations.

OMNet++. Omnet++ : Discrete event simulator, 2022. URL https://omnetpp.org/.

L. E. Talavera, M. Endler, I. Vasconcelos, R. Vasconcelos, M. Cunha, and F. J. da Silva e Silva. The Mobile Hub concept: Enabling applications for the Internet of Mobile Things. In *PerCom Workshops*, pages 123–128, Mar. 2015.

USDA. Livestock and poultry: World markets and trade. Technical report, United States Department of Agriculture - USDA, 2022. URL https://downloads.usda.library.cornell.edu/usda-esmis/files/73666448x/f1882v52q/765389829/livestock_poultry.pdf.