In this chapter, we will explain the construction process behind the datasets that have been used to train the model and observe the feature's performance. The process is composed of three main steps: running simulations, translating simulation log files into time series, computing robot features time series. The simulations are executed through the simulators introduced in Section **??** and will be explained in more detail in Section 0.1.1. The translation of log files into time series data objects is explained in Section 0.2. Finally, the computation of the features' time series is described in Section 0.3.

0.1. Data Retrieval

Simulations data is collected through comma separated value files (`.csv`) that are directly written by the simulators. These files are then processed with `python` and all the robots' positions are translated into time-series objects.

0.1.1. Running Examples

In our experiments, we run several simulations with different settings. In this section, we will list the settings used for the simulations of each task and show an example of task execution.

ARGoS Flocking

For the flocking task, we have used a simulation environment of a 30×30 m^2 arena surrounded by walls and with 15 bots. Even though the walls have never worked as a constraint measure, they are necessary to put physical bounds to the arena. The coordinate system of the simulation is considered to have the $(0, 0, 0)$ position in the middle of the arena at ground height. The simulations have a duration of 800 seconds, this value has been chosen arbitrarily after some tests to allow the swarm to complete the task. The goal of the task is to reach a light positioned in the center of the arena at a height of 4 m from the ground, the light coordinates are $(0, 0, 4)$ and it is oriented toward $(0, 0, 0)$. The flocking task presumes that the robots manage to reach the light while keeping a formation, more precisely they have to keep a distance of 75 cm from all their neighbors. In our `.argos` settings we specified a `gain` parameter of 1000 for the flocking controller. The robots begin each flocking simulation from one of the 4 different starting areas, namely North, South, East, and West. We can see the starting positions depicted in Figure 1 with their corresponding names and the light in the center of the arena.
Faults have been injected in the 10%, 20%, or 33% of the swarm population and at 0 s, 50 s, 150 s, or 400 s from the beginning of the simulation. Completely nominal simulations are run together with faulty simulations in order to have a complete dataset of a real situation. An example of the robot trajectories can be seen in Figure 2 where nominal and faulty agents are separated in two different graphs.

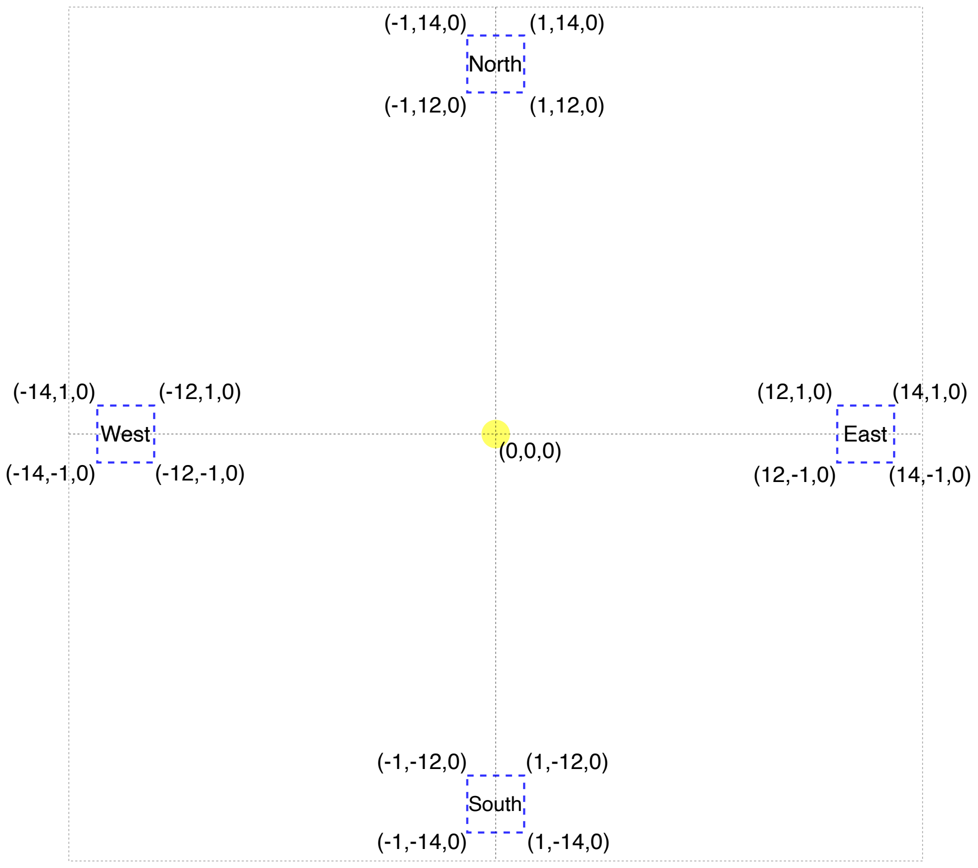
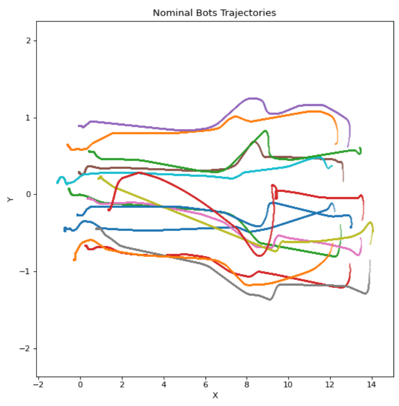
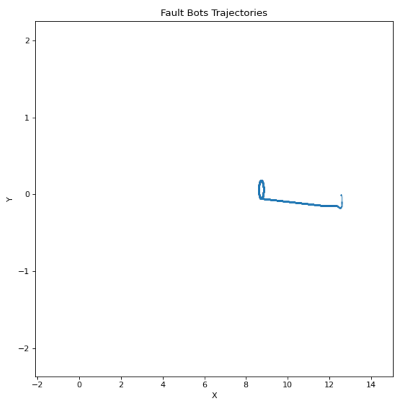


Figure 1: Flocking task robots starting positions.



(a) ARGoS flocking nominal bots trajectories.



(b) ARGoS flocking faulty bots trajectories.

Figure 2: ARGoS flocking plotted trajectories.

ARGoS Foraging

For the foraging task, we have used incremental settings that adapts with the increase of the arena size. The arena size starts from $5 \times 5\ m^2$ and ends in $8 \times 8\ m^2$ going trough each possible combination with sides of size $\{5, 6, 7, 8\}\ m$ thus using 16 different arena sizes. In this task, the arena is surrounded by walls and they're fundamental to keep the bot inside the arena while they're performing the random walk. The goal of the task is to retrieve the items in the resource area and bring them to the nest area. There are 4 lights to signal the position of the nest on the nest side of the arena, they're positioned at a height of 1 m from the ground. The lights' coordinates change with the size of the arena. At each simulation, they are positioned at the 10%, 30%, 70%, and 90% of the wall length in order to be adequately distanced. The robot number starts from 20 agents and scales in a directly proportional way with the arena size. The simulations last 800 seconds, this value has been chosen arbitrarily but the task execution does not depend on it as it is for flocking. The foraging task requires some parameters for the agents' foraging behavior, these values are scaled with the arena size in order to have coherent simulations among different arenas. The foraging task parameters are:

- `minimum_resting_time` parameter: initialized at 5 s.
- `minimum_unsuccessful_explore_time` parameter: initialized at 60 s.
- `minimum_search_for_place_in_nest_time` parameter: is initialized at 15 s.

Each foraging simulation requires a number of items to be scattered around in the resources area, this value is initialized at 15 items and scales with the arena size. An example of the foraging arena can be seen in Figure 3.
For each arena size, a number of nominal simulations are executed together with a number of simulations with injected faults. For the faulty simulations, we have considered 10%, 20%, or 33% of the swarm population to be infected at 0, 100, or 400 s from the task start. An example of the robot trajectories can be seen in Figure 4 where nominal and faulty agents are separated in two different graphs.

RAWSim-O Fulfillment System

The RMFS task is set in a warehouse sized by the number of horizontal and vertical aisles that the robots use to move. In our experiments we used different arena sizes starting from 5×5 horizontal aisles $\times$ vertical aisles size through 8×8 horizontal aisles $\times$ vertical aisles, using each combination of the aisles number in $\{5, 6, 7, 8\}$, thus 16 different warehouse sizes. The environment objects are placed by the simulators, between each aisle there is a group of 10 pods locations that do not have to be all occupied. The simulations have a duration of 80000 seconds which approximately simulates the duration of a workday. The goal of the task is to follow the orders of the RMFS and carry pods from the storage area to the pick stations and replenishment stations. Based on how many horizontal aisles are included in the warehouse there is a corresponding number of pick stations and replenishment stations, the rule the compute the stations' number is:

$$\text{stations number} = \lfloor \frac{\text{horizontal aisles number}}{2} \rfloor.$$

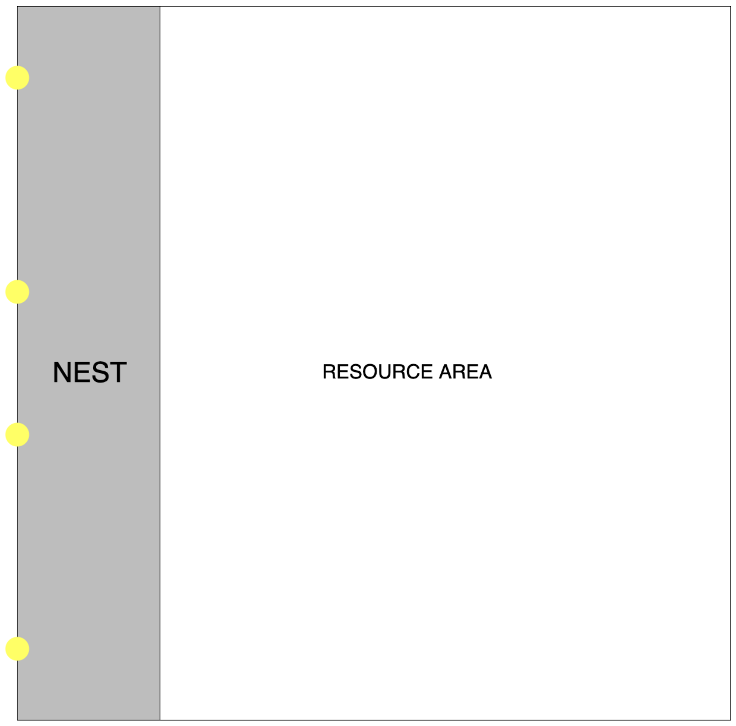
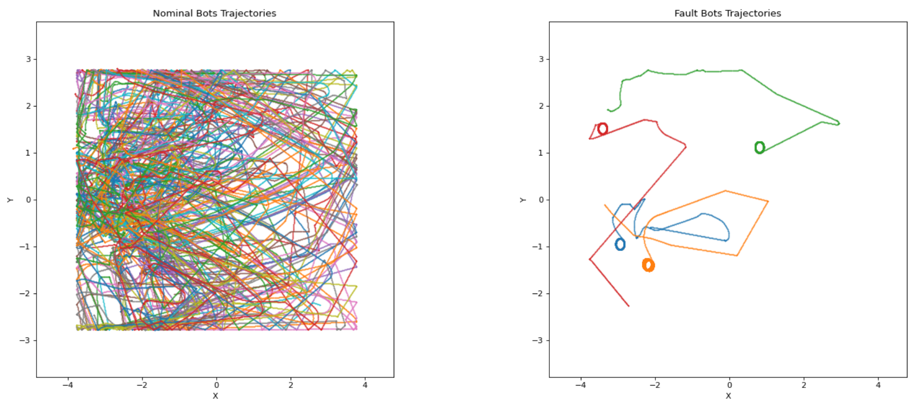


Figure 3: Foraging arena example.



(a) ARGoS foraging nominal bots trajectories.

(b) ARGoS foraging faulty bots trajectories.

Figure 4: ARGoS foraging plotted trajectories.

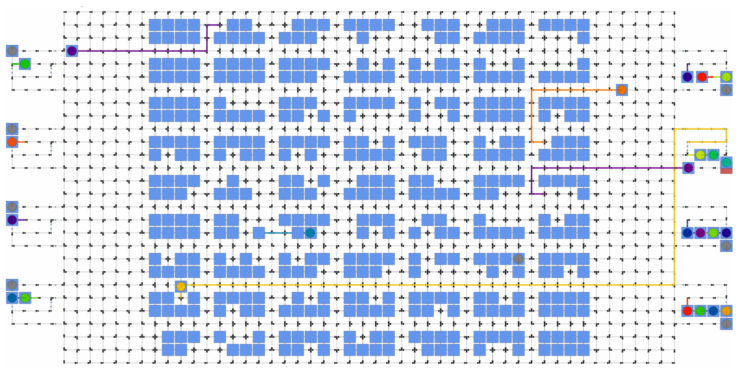
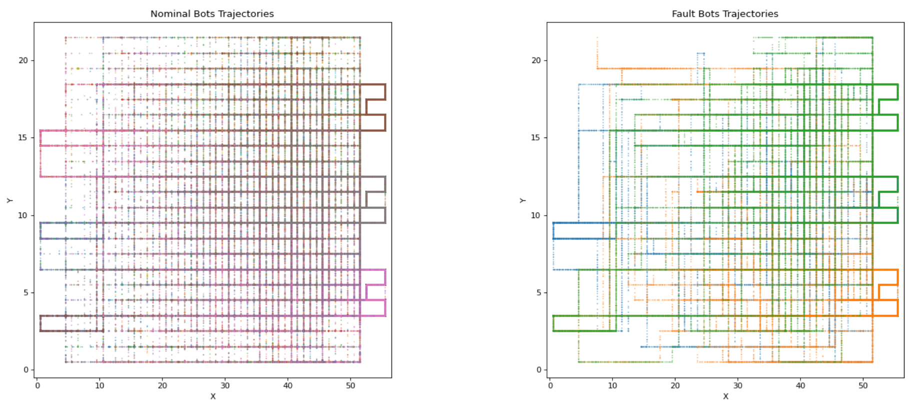


Figure 5: RMFS warehouse example.



(a) RAWSim-O nominal bots trajectories. (b) RAWSim-O faulty bots trajectories.

Figure 6: RAWSim-O plotted trajectories.

The robot number starts from 15 agents and scales in a directly proportional way with the arena size. We can see an example of the warehouse environment in Figure 5.
For each warehouse size, a number of nominal simulations have been executed together with a number of simulations with injected faults. For the faulty simulations, we have considered 5%,10%, 20%, or 33% of the swarm population to be infected from the start of the task with a reduced speed of 10% or 5% of the maximum speed. An example of the robot trajectories can be seen in Figure 6 where nominal and faulty agents are separated in two different graphs.

0.2. Simulation Data Translation

During each simulation, the simulator is responsible for writing a `.csv` file where the information of every bot at every timestep is written. At the end of the simulation, we are provided with a `.csv` file with a structure identical to that of Table 1 but filled with values for each timestep and robot identification code. This step is supposed to simulate the external controller that observes the execution of the task and identifies each agent's position. The Fault value is necessary to perform supervised learning, in real-world scenarios it will not be provided.

timestep	ID	PosX	PosY	Fault
1	0	12.5606	-0.0025857	0
1	1	12.2117	-0.222327	0
...	...	...	...	...
1501	0	8.72475	-0.0530649	1
...	...	...	...	...
8000	0	8.66732	-0.0364418	1
...	...	...	...	...
8000	14	-0.0665298	0.290923	0

Table 1: Comma separated values file structure.

0.3. Feature Computation

The feature list has is shown in Table 2. In this section we will explain how each feature has been computed.

0.3.1. Position, Direction, Speed, Cumulative Speed

The position feature is the transposition of the `PosX` and `PosY` values into a `numpy` vector of size [timestep number, 2]. For future references, we will refer to this feature as $pos_{x,i}$ where x represents the x-axis and $i \in [0, \ldots, T]$ is the timestep in a simulation of length T.
The direction feature is computed by subtracting the current position value from the previous position value for the x-axis and y-axis:

$$dir_{x,i} = pos_{x,i-1} - pos_{x,i}.$$

The speed feature is computed as the rate of change of position with respect to time:

$$v_i = \frac{\sqrt{(pos_{x,i-1} - pos_{x,i})^2 + (pos_{y,i-1} - pos_{y,i})^2}}{t_i - t_{i-1}} \tag{1}$$

with $v_0 = 0\frac{m}{s}$.
The cumulative speed feature for agent i at timestep t is computed as:

$$\text{cum-}v_{i,t} = \sum_{j=t-(\texttt{time-window})}^{T} v_{i,j}$$

Feature	**Description**
Position	Coordinates of the agent in a single timestep
Direction	Directional vectors on x and y axis
Speed	Velocity of the agent between two consecutive timesteps
Cumulative speed	Sum of the last n timesteps velocity values
Neighbors number	Number of agent within a specified radius
Neighbors average distance	Average distance from all the bots in the swarm
Centroid distance	Distance from the coordinates of the swarm centroid
Cumulative centroid distance	Sum of the last n centroid distance values
Position entropy	Entropy measure of the last n positions
Area coverage	Percentage of area covered by the agent
Speed area coverage	Area coverage between two consecutive timesteps
Swarm position	Coordinates of the swarm centroid
Swarm speed	Velocity of the centroid between two consecutive timestep
Swarm area coverage	Percentage of area covered by the whole swarm

Table 2: Single bot list of features.

0.3.2. Neighbors Number, Neighbors Average Distance

The neighbors number feature holds the information about how many other agent are within a predefined radius r. It is computed with the Algorithm 1 with N as the swarm cardinality, T as the total number of timestep. The neighbors average distance feature is

Algorithm 1 Neighbors Number Computation.

```
1: for $t = 1$ to $T$ do
2:     temp-number=0
3:     for $i = 1$ to $N$ do
4:         Compute $d$=distance between current agent and agent$_i$.
5:         if $d < r$ then
6:             temp-number+=1
7:         end if
8:     end for
9:     neighbors-number$_t$=temp-number
10: end for
```

computed as the mean value among the distance from the current agent and all the other agents. It is computed with the Algorithm 2 with N as the swarm cardinality, T as the total number of timestep.

Algorithm 2 Neighbors Average Distance Computation.

```
1: for $t = 1$ to $T$ do
2:     distances={}.
3:     for $i = 1$ to $N$ do
4:         Compute $d$=distance between current agent and agent$_i$.
5:         Add $d$ to distances.
6:     end for
7:     avg-$d_t$ = $\frac{\sum_{d \in \text{distances}} d}{N}$
8: end for
```

0.3.3. Position Entropy

The position entropy feature computes the entropy of the positions in the last `time-window`×10 timesteps. The entropy is computed with `scipy.stats.entropy` [?] module that uses the following formula:

$$S = -\sum_{v \in Values} (P(v) * log(P(v)))$$

The entropy computation procedure is stated in Algorithm 3.

0.3.4. Centroid Distance,Cumulative Centroid Distance

The centroid distance feature is the distance of the current agent from the centroid of the swarm. The centroid of the swarm is computed as the mean of the position of all the

Algorithm 3 Position Entropy Computation.

```
1: for $t = 1$ to $T$ do
2:     if $t <$ time-window then
3:         Compute vector of probabilities $P$ for each in position in the last $t$ positions.
4:         $entropy_t = -\sum_{p \in P}(p * log(p))$
5:     else
6:         Compute vector of probabilities $P$ for each in position in the last time-window
           positions.
7:         $entropy_t = -\sum_{p \in P}(p * log(p))$
8:     end if
9: end for
```

agents. Once the swarm centroid time series has been computed, each agent computes the distance between its current position and the position of the swarm centroid. The procedure is shown in Algorithm 4. The cumulative centroid distance feature for agent i

Algorithm 4 Centroid Distance Computation.

```
1: for $t = 1$ to $T$ do
2:     $centroid\text{-}pos_t = \frac{\sum_{i \in N} pos_i}{|N|}$
3: end for
4: for $i = 1$ to $N$ do
5:     for $t = 1$ to $T$ do
6:         $centroid\text{-}distance_{i,t} = centroid\text{-}pos_t - pos_{i,t}$
7:     end for
8: end for
```

at timestep t is computed as:

$$\text{cum-}centroid\text{-}distance_{i,t} = \sum_{j=t-(\texttt{time-window})}^{t} centroid\text{-}distance_{i,j}$$

0.3.5. Area Coverage, Speed Area Coverage

The area coverage feature represents the area percentage covered by the agent until that exact moment. It is computed with 4 different levels of detail: 4, 16, 64, and 256 subdivision of the total area available. Area subdivisions are computed with the maximum and minimum positions covered by any robot in each axis. An approximate algorithm of the procedure is shown in Algorithm 5, the real implementation is optimized using `numpy` array and python functional programming with list comprehension.
Once the area borders have been computed, these values are used to compute the area coverage percentage of each agent. The area coverage computation procedure is shown in Algorithm 6.
The area coverage speed feature holds the information about the value increment in the

area coverage timeseries for each timestep. The procedure to compute the area coverage speed feature is shown in Algorithm 7.

Algorithm 5 Area bounds computation.

```
1: Initialize variables to store positions. In this algorithm we will consider the order (agent-id, timestep-number,coordinate-axis) for the subscripts of $pos$
2: min-pos-x = $pos_{1,1,x}$
3: min-pos-x = $pos_{1,1,y}$
4: max-pos-x = $pos_{1,1,x}$
5: max-pos-x = $pos_{1,1,y}$
6: for $i = 1$ to $N$ do
7:     for $t = 1$ to $T$ do
8:         if min-pos-x > $pos_{i,t,x}$ then
9:             min-pos-x = $pos_{i,t,x}$
10:        end if
11:        if min-pos-y > $pos_{i,t,y}$ then
12:            min-pos-y = $pos_{i,t,y}$
13:        end if
14:        if max-pos-x < $pos_{i,t,x}$ then
15:            max-pos-x = $pos_{i,t,x}$
16:        end if
17:        if max-pos-y < $pos_{i,t,y}$ then
18:            max-pos-y = $pos_{i,t,y}$
19:        end if
20:    end for
21: end for
22: for $split \in \{2, 4, 8, 16\}$ do
23:     for $i = 1$ to $split$ do
24:         Compute segment$_{x,i}$ coordinates from the subdivision of segment [min-pos-x, max-pos-x] in equal $split$ parts
25:         Compute segment$_{y,i}$ coordinates from the subdivision of segment [min-pos-y, max-pos-y] in equal $split$ parts
26:         Save segments coordinates in an AreaPartition object.
27:         Save AreaPartition object in the area-subdivision list.
28:     end for
29:     Save area-subdivision list object in the area-partitions list.
30: end for
```

0.3.6. Swarm Position, Swarm Speed, Swarm Area Coverage

The swarm position feature represents the coordinates of the entire swarm, for this feature we've used the centroid of the positions of all the robots. The swarm position computation can be seen from line 1 to line 3 of Algorithm 4.
The swarm speed is computed as the speed feature for the single agent, in fact, we can refer to Equation 1 to compute the swarm speed considering the swarm centroid instead

Algorithm 6 Area coverage computation.

```
1: Consider current agent $i$.
2: for area-subdivisions list in area-partitions list do
3:     timestep=0.
4:     while any AreaPartition not visited & timestep $< T$ do
5:         for area-partition in not visited area-partitions do
6:             if $pos_{i,t}$ is in area-partition then
7:                 area-partition.visited=True.
8:             end if
9:         end for
10:        area-coverage$_{i,t} = \frac{\sum_{\text{area}\in\text{area-partitions-list}} I(\text{area.visited()==True})}{|\text{area-partitions-list}|}$
11:        timestep+=1.
12:    end while
13:    if iteration ends before visiting all timesteps then
14:        fill area-coverage$_i$ timeseries with the last computed value.
15:    end if
16: end for
```

Algorithm 7 Area coverage speed computation.

```
1: Consider current agent $i$.
2: for $t = 1$ to $T$ do
3:     coverage-speed$_i$ = area-coverage$_{i,t}$ - area-coverage$_{i,t-1}$
4: end for
5: Repeat for each area-subdivision level.
```

of the single agent position.
The swarm area coverage is considered as an overall area coverage performed by the whole swarm. The computation of the swarm area coverage feature is similar to the one shown in Algorithm 5 and Algorithm 6 with the only difference that, when we check if an area partition is visited, we check if any robot in the whole swarm is covering that area.