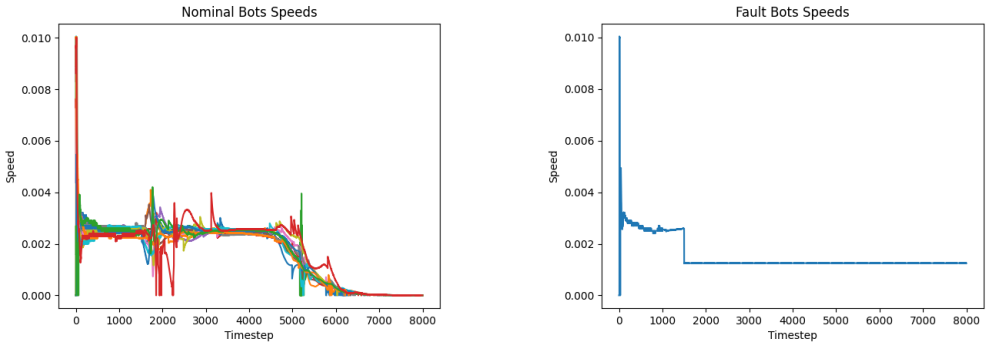


(a) Flocking nominal agents speed feature values.

(b) Flocking faulty agents speed feature values.

Figure 1: Flocking speed feature values.

In this chapter, we will analyze the experiment we have made and discuss the measurements we have collected. The experiments will be analyzed from three different perspectives: feature analysis, performance metrics, and model performances. At the beginning we will analyze the feature time series values of nominal robots against faulty robots, we will then proceed on explaining which metrics we have used to judge the model performances, and, in the end, we will evaluate the model performances themselves.

0.1. Data Sets Analysis

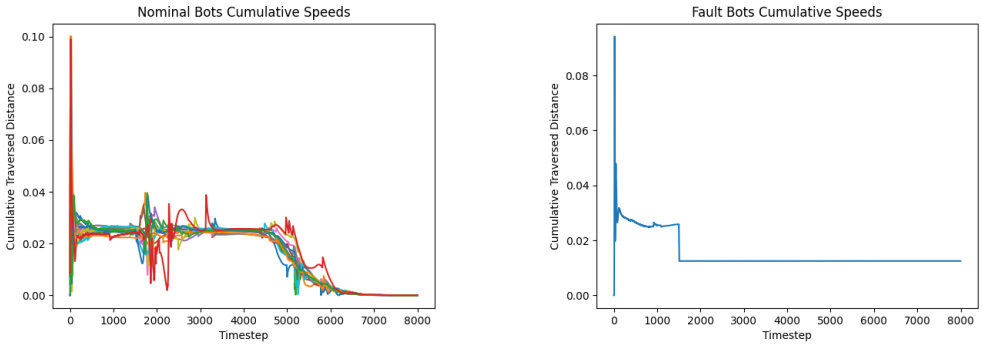
In this section we will graphically show the values assumed by each feature and if they present some distinguishing characteristics between nominal and faulty agents.

0.1.1. Flocking Features Analysis

The flocking task is explained in Section **??**. The robot trajectories can be seen in Figure **??**. From now on, we will analyze the features explained in Section **??**. The direction feature will not be analyzed since it is unfeasible to show it graphically and useless to numerically analyze it. In this example, we will analyze a simulation of 15 agents with 1 faulty agent. In this section, we will analyze a flocking task starting from the East position. The fault is injected at 150 s and the robot rotates around a fixed point until the end of the simulation. For each second we have 10 timesteps.

Speed

The speed feature is one of the most characteristics since all of our faults influence this measure. It is possible to immediately distinguish the faulty robot behavior from the nominal robots. In the graphs in Figure 1 we can see that the faulty robot does not change speed after the fault is injected.



(a) Flocking nominal agents cumulative speed feature values.

(b) Flocking faulty agents cumulative speed feature values.

Figure 2: Flocking cumulative speed feature values.

Cumulative Speed

The cumulative speed feature (Figure 2) is a rounded version of the speed feature that holds some information about the timestep in the `time-window`. This feature has identical characteristics to the speed feature and it is possible to distinguish the faulty robot at first sight. The `time-window` parameter is set to 10 and the values of the feature show the correlation between the two features.

Neighbors Number

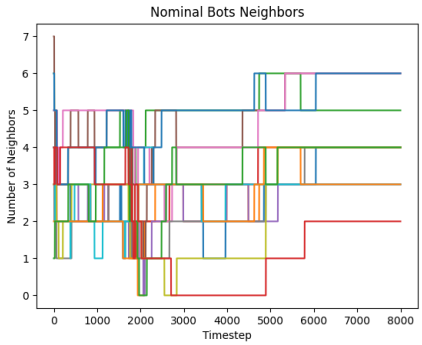
The neighbors number feature is very interesting for the flocking task since the agents have to keep a formation at a certain distance from one another. In Figure 3 we can see that the faulty agent has a high number of neighbors at the start of the simulation, however, soon after the fault is injected, this value drastically drops denoting a separation of the agent from the swarm.
In this situation the neighborhood radius value has been set to 2 m. This value has to be fine tuned in order to have significant results.

Neighbors Average Distance

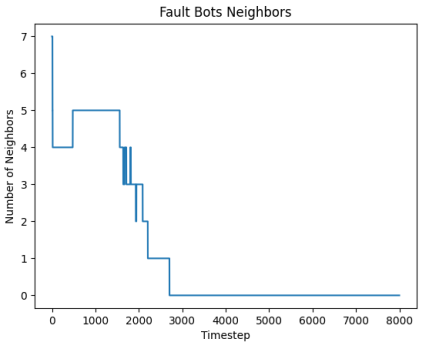
The neighbors average distance feature highlights information similar to the neighbors number feature with opposite values, in this situation, the high value of the feature denotes a separation of the agent from the swarm. This feature is interesting for the flocking task, as the neighbors number feature, because it describes the behavior of the agent with respect to the other agents. In Figure 4 we can see that the faulty agent detaches from the swarm after the fault is injected.

Centroid Distance

The centroid distance feature is a simplified version of the neighbors average distance feature since it is easier to compute. From the graphs in Figure 5 we can see that the

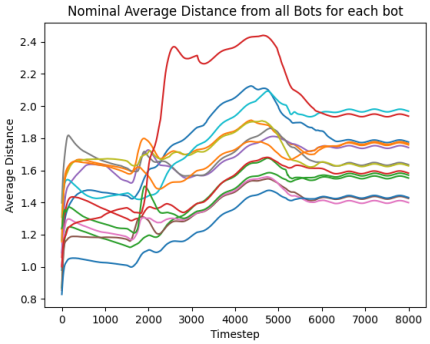


(a) Flocking nominal agents neighbors number feature values.

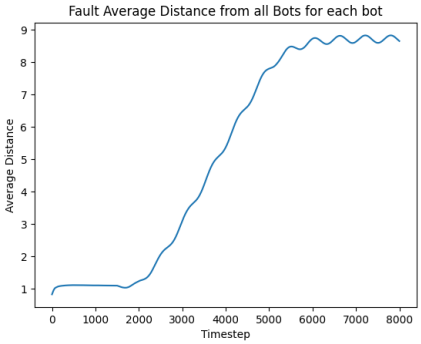


(b) Flocking faulty agents neighbors number feature values.

Figure 3: Flocking neighbors number feature values.

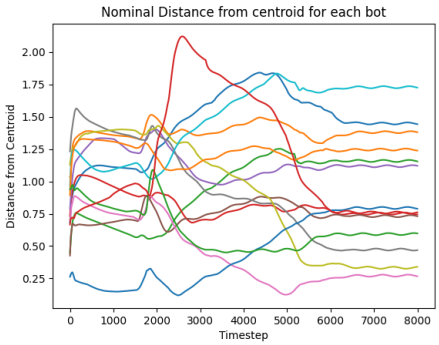


(a) Flocking nominal agents neighbors average distance feature values.

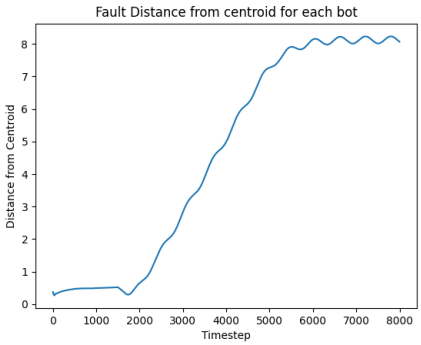


(b) Flocking faulty agents neighbors average distance feature values.

Figure 4: Flocking neighbors average distance feature values.

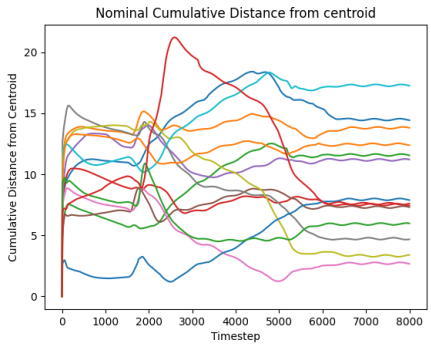


(a) Flocking nominal agents centroid distance feature values.

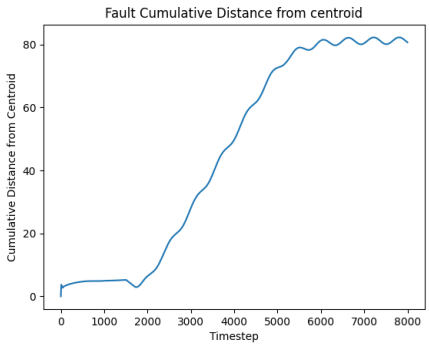


(b) Flocking faulty agents centroid distance feature values.

Figure 5: Flocking centroid distance feature values.



(a) Flocking nominal agents centroid distance feature values.



(b) Flocking faulty agents centroid distance feature values.

Figure 6: Flocking centroid distance feature values.

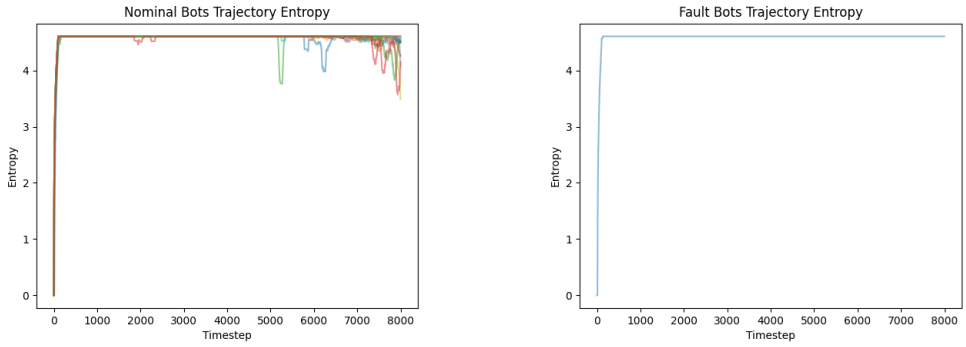
values have a similar shape to the values shown in Figure 4 and show similar values, this is because the distance from the centroid can be seen as a mean value of the distance from the other agents.

Cumulative Centroid Distance

The cumulative centroid distance feature is a feature that holds some information about the past values. The concept behind this feature is similar to the one of the cumulative speed feature. Since this feature is a sum of the past values of the centroid distance feature, these two feature show high correlation and the values in the Figure 6 depicts the values of Figure 5 multiplied by the `time-window` parameter.

Position Entropy

The position entropy feature aims at identifying how many different positions are covered by the agent in the last `time-window` timesteps. The graphs shown in Figure 7 do not



(a) Flocking nominal agents position entropy feature values.

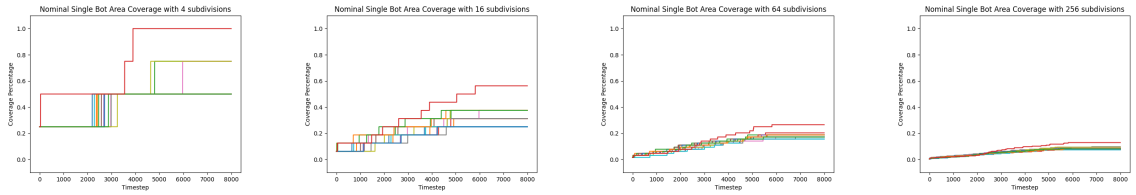
(b) Flocking faulty agents position entropy feature values.

Figure 7: Flocking position entropy feature values.

show any interesting information about the task execution since the faulty agent doesn't have different values than nominal agents. Like we said for the `neighborhood-radius` parameter, the `time-window` parameter has to be fine tuned for each different task such that the position entropy feature shows interesting values.

Area Coverage

The area coverage feature is divided into 4 different time series. The time series shown in Figure 8 depict the values assumed by the area coverage feature of the nominal agents with different levels of area partitioning. From the left, we can see the values in the case of 4 subdivisions, 16 subdivisions, 64 subdivisions, and then 256 subdivisions. We can see that the maximum value reached by the feature decreases with the increase of the area subdivisions, this is because, with the increase of the definition in area partitioning, some areas are too small to be covered by any agent.
The time series shown in Figure 9 depict the values assumed by the area coverage feature of the faulty agents. The graphs are organized like in Figure 8. The most important difference between nominal and faulty agents is that the faulty agent feature usually reaches lower values than the nominal agents.



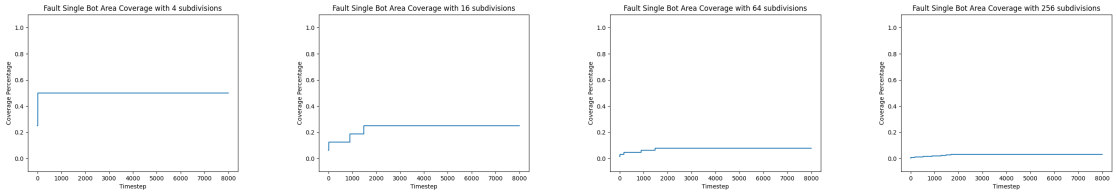
(a) 4 subdivisions.

(b) 16 subdivisions.

(c) 64 subdivisions.

(d) 256 subdivisions.

Figure 8: Flocking nominal agents area coverage feature values.

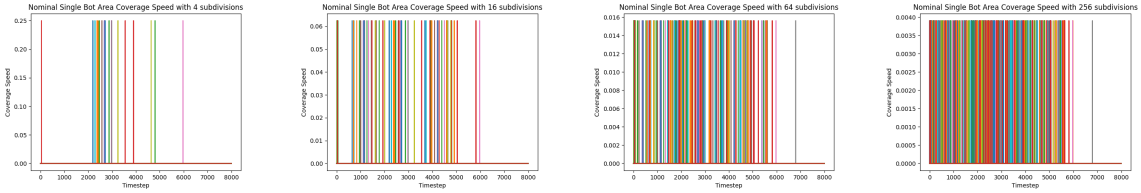


(a) 4 subdivisions. (b) 16 subdivisions. (c) 64 subdivisions. (d) 256 subdivisions.

Figure 9: Flocking faulty agents area coverage feature values.

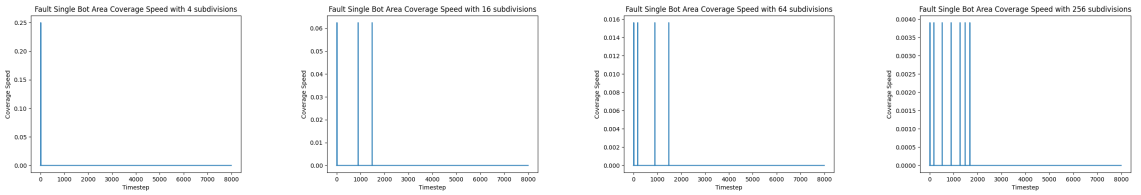
Area Coverage Speed

The area coverage speed feature holds information about the value increments of the area coverage feature. The graphs are organized like the ones of the area coverage feature. The most distinguishing aspect of this feature is that we can observe the faulty robot that stops exploring earlier than the nominal agents due to the injection of the fault.



(a) 4 subdivisions. (b) 16 subdivisions. (c) 64 subdivisions. (d) 256 subdivisions.

Figure 10: Flocking nominal agents area coverage speed feature values.

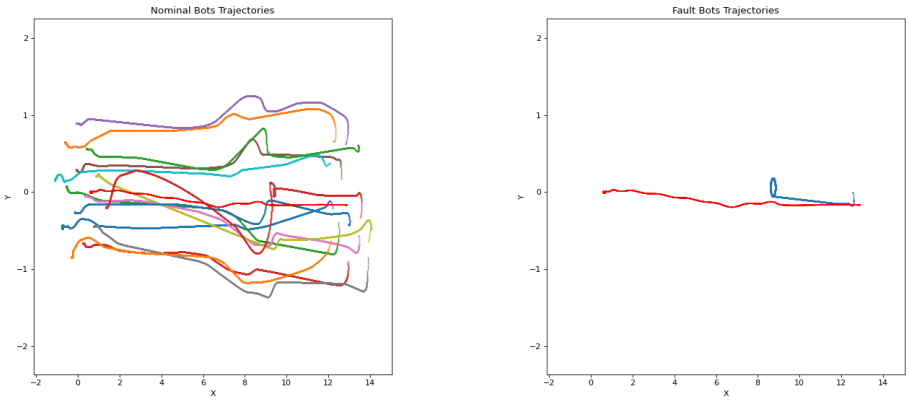


(a) 4 subdivisions. (b) 16 subdivisions. (c) 64 subdivisions. (d) 256 subdivisions.

Figure 11: Flocking faulty agents area coverage speed feature values.

Global Features

The global features are those features that contain information about the overall behavior of the swarm in its entirety. These features should depict information about the execution of the task but do not contain information about the single agent, this characteristic is important because this kind of feature can not be considered fundamental for the objective of fault detection. The presence of these features is important whenever we use a model that can infer complex information from different features of the sample, i.e. convolutional



(a) Swarm trajectory among nominal agents.

(b) Swarm trajectory among faulty agents.

Figure 12: Swarm trajectory.

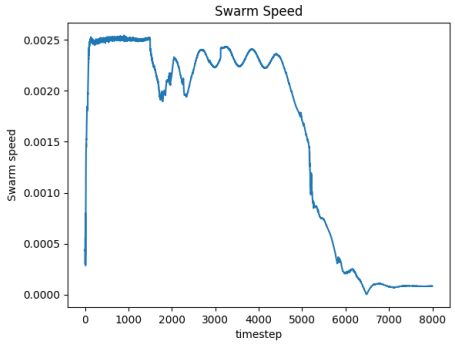


Figure 13: Swarm speed.

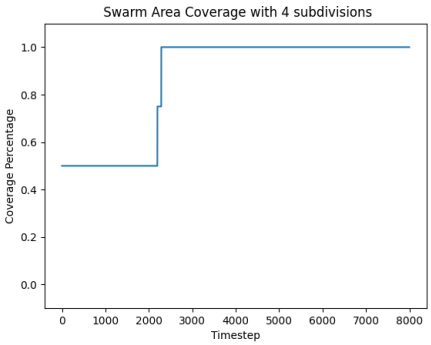
neural networks.

Swarm Position

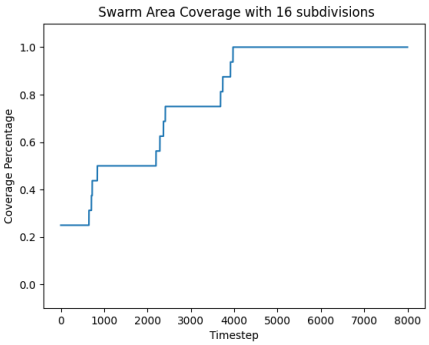
The swarm position feature is the coordinates of the centroid of the swarm. We can see the trajectory followed by the swarm centroid in Figure 12 as the bright red line in the middle of the plot among nominal agents (Figure 12a) and faulty agents (Figure 12b).

Swarm Speed

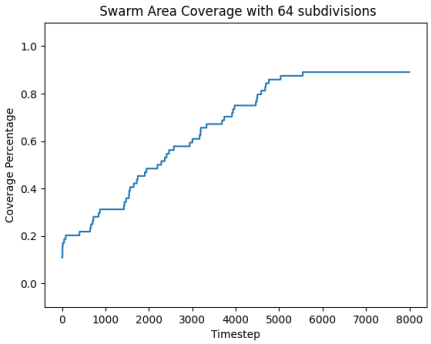
The swarm speed feature represents the velocity of the swarm centroid. In Figure 13, we can see that the values have a high spike at the beginning when the robots move to get in flocking formation. The feature value stays constant until 1500 timesteps, namely 150 s, and then slowly drops to lower values after the fault is injected. This feature behavior perfectly mirrors the swarm centroid movement that is slowed down by the faulty agent.



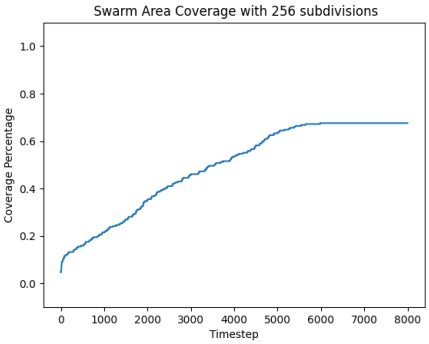
(a) Flocking swarm 4 subdivisions area coverage feature values.



(b) Flocking swarm 16 subdivisions area coverage feature values.



(c) Flocking swarm 64 subdivisions area coverage feature values.



(d) Flocking swarm 256 subdivisions area coverage feature values.

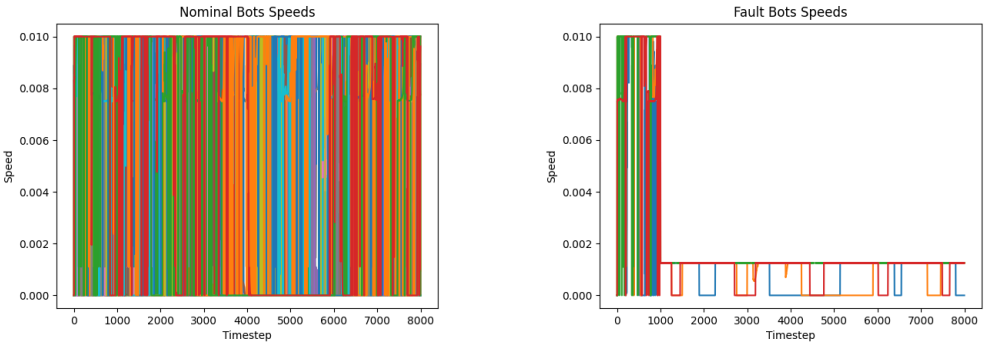
Figure 14: Flocking swarm area coverage feature values.

Swarm Area Coverage

The swarm area coverage features illustrate the percentage of area covered by the entirety of the swarm considered each agent. This feature is similar to the area coverage feature seen before in section Area Coverage. As stated above, this is a global feature and does not have the objective to directly identify the fault injection. Nonetheless, the graphs shown in Figure 14 do not shown any sign of the presence of the fault injected after 150 s.

0.1.2. Foraging Features Analysis

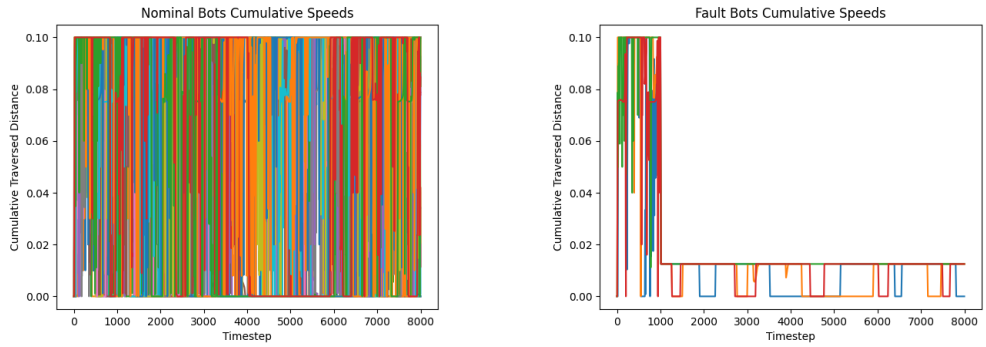
The foraging task is explained in Section **??**. The robot trajectories can be seen in Figure **??**. In this example, we will analyze a simulation of 38 agents with 4 faulty agents. The fault is injected at 150 s and the robots rotate around a fixed point until the end of the simulation. For each second we have 10 timesteps.



(a) Foraging nominal agents speed feature values.

(b) Foraging faulty agents speed feature values.

Figure 15: Foraging speed feature values.



(a) Foraging nominal agents cumulative speed feature values.

(b) Foraging faulty agents cumulative speed feature values.

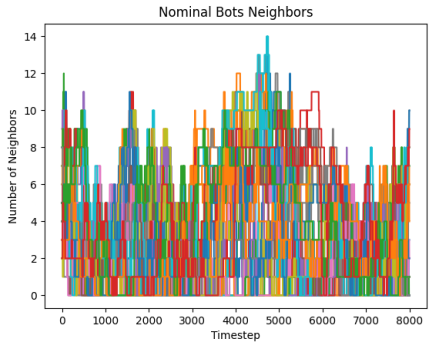
Figure 16: Foraging cumulative speed feature values.

Speed

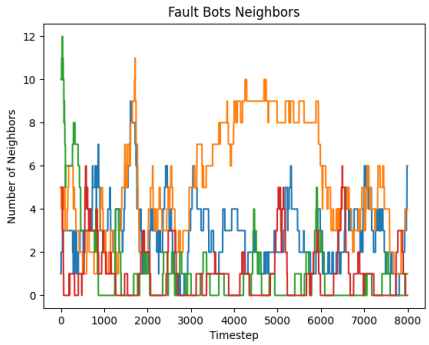
As stated for the flocking task, this feature is highly significant due to the nature of the faults injected. In Figure 15 we can see that the faulty robots rotate do not reach high speeds after the fault is injected but their speed oscillates at values lower than the nominal agents. The oscillations of faulty agents' speed can be traced back to obstacle avoidance maneuvers or blockages in the nest area.

Cumulative Speed

This feature has identical characteristics to the speed feature and it is possible to distinguish the faulty robot at first sight (Figure 16). The `time-window` parameter is set to 10 and the features values show the correlation between the two features.

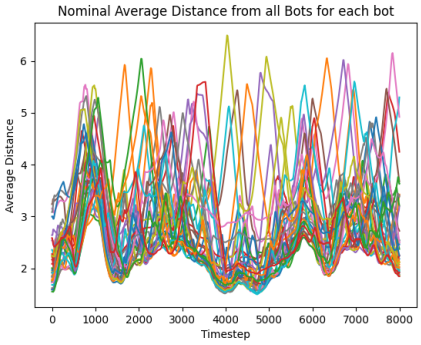


(a) Foraging nominal agents neighbors number feature values.

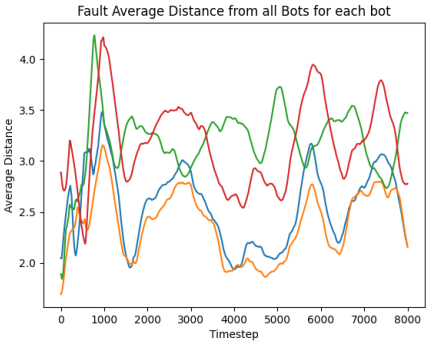


(b) Foraging faulty agents neighbors number feature values.

Figure 17: Foraging neighbors number feature values.



(a) Foraging nominal agents neighbors average distance feature values.



(b) Foraging faulty agents neighbors average distance feature values.

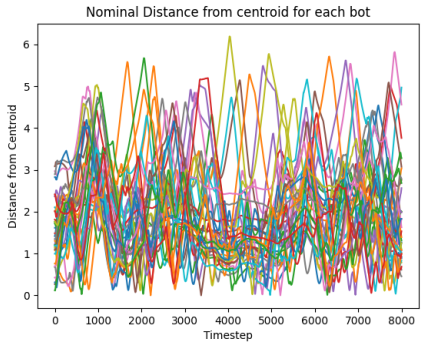
Figure 18: Foraging neighbors average distance feature values.

Neighbors Number

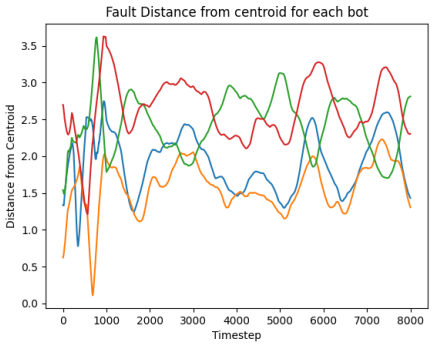
The neighbors number feature is not very interesting for the foraging task since the agents spend most of the time wandering around. In Figure 17 we can see that the faulty agents have similar values with respect to the nominal agents.
In this situation the neighborhood radius value has been set to 2 m. This value has to be fine tuned in order to have significant results.

Neighbors Average Distance

The neighbors average distance feature highlights information similar to the neighbors number feature with opposite values, in this situation, the high value of the feature denotes a separation of the agent from the swarm. As stated before, this feature does not contain interesting information due to the nature of the task. In Figure 18 we see that faulty agents and nominal agents have mostly similar values.

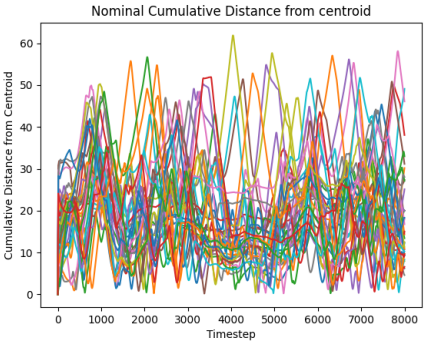


(a) Foraging nominal agents centroid distance feature values.

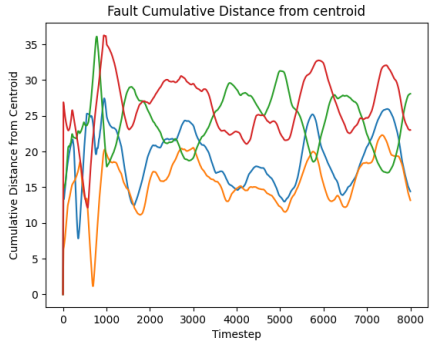


(b) Foraging faulty agents centroid distance feature values.

Figure 19: Foraging centroid distance feature values.



(a) Foraging nominal agents centroid distance feature values.



(b) Foraging faulty agents centroid distance feature values.

Figure 20: Foraging centroid distance feature values.

Centroid Distance

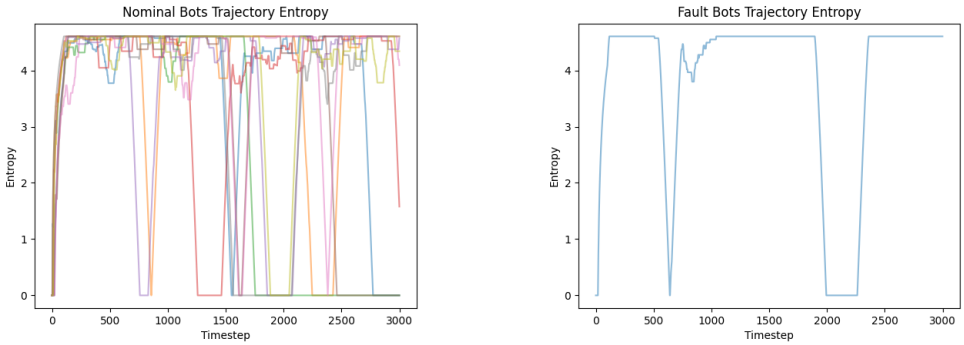
From the graphs in Figure 19 we can see that the values have a similar shape to the values shown in Figure 18, this is because the distance from the centroid can be seen as a mean value of the distance from the other agents.

Cumulative Centroid Distance

Similarly to the flocking task, these two feature show high correlation and the values in Figure 20 depicts the values of Figure 19 multiplied by the `time-window` parameter.

Position Entropy

To analyze this feature we consider a reduced sample of timesteps and a lower number of agents. The graphs shown in Figure 21 shows more different values than the graph in Figure 7, however there are not any clearly discriminant characteristic on the graphs that



(a) Foraging nominal agents position entropy feature values.

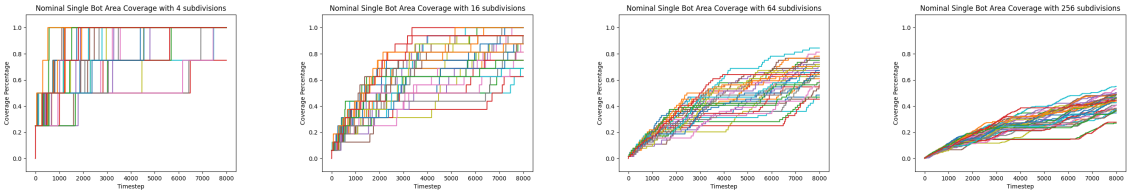
(b) Foraging faulty agents position entropy feature values.

Figure 21: Foraging position entropy feature values.

allows us to distinguish nominal agents from faulty agents. As stated before, the values of this feature depend on the tuning of the `time-window` parameter.

Area Coverage

The area coverage feature is identically organized as the flocking task. In Figure 22 we can see that the maximum value reached by the feature decreases with the increase of the area subdivisions, this is because, with the increase of the definition in area partitioning, some areas are too small to be covered by any agent.
The most important difference between nominal (Figure 22) and faulty agents (Figure 23) is that the faulty agents feature reaches lower values than the nominal agents and it does not increase after the fault is injected.



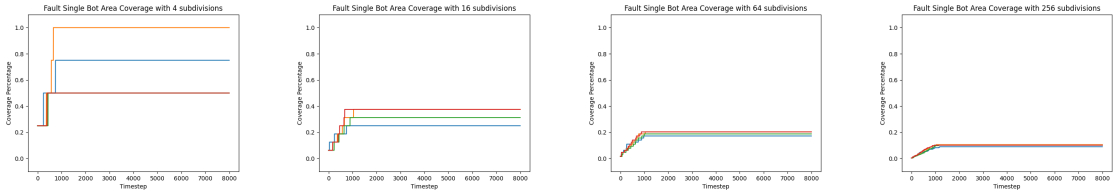
(a) 4 subdivisions.

(b) 16 subdivisions.

(c) 64 subdivisions.

(d) 256 subdivisions.

Figure 22: Foraging nominal agents area coverage feature values.

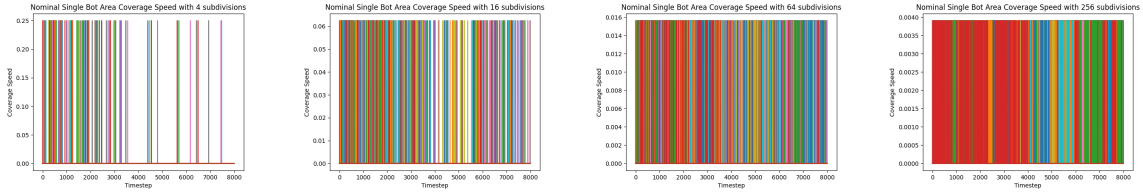


(a) 4 subdivisions. (b) 16 subdivisions. (c) 64 subdivisions. (d) 256 subdivisions.

Figure 23: Foraging faulty agents area coverage feature values.

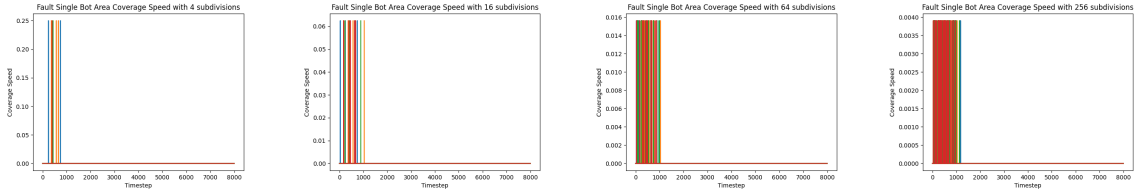
Area Coverage Speed

The graphs are organized like the ones of the area coverage feature. The most distinguishing aspect of this feature is that we can observe the faulty robot that stops exploring earlier than the nominal agents due to the injection of the fault.



(a) 4 subdivisions. (b) 16 subdivisions. (c) 64 subdivisions. (d) 256 subdivisions.

Figure 24: Flocking nominal agents area coverage speed feature values.



(a) 4 subdivisions. (b) 16 subdivisions. (c) 64 subdivisions. (d) 256 subdivisions.

Figure 25: Foraging faulty agents area coverage speed feature values.

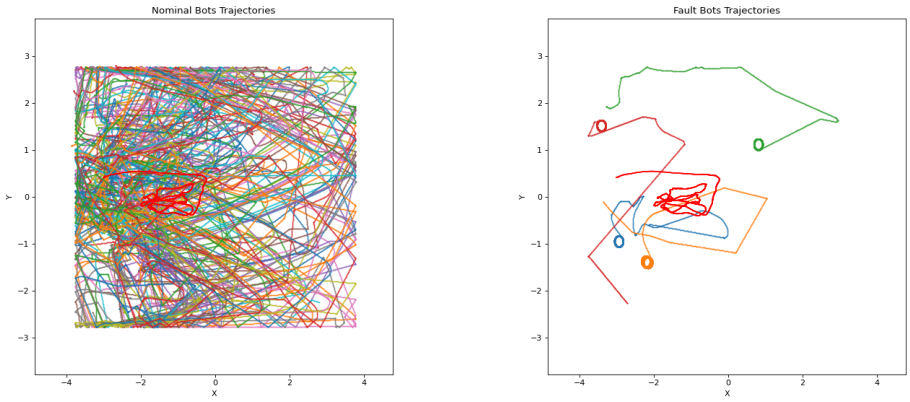
Global Features

Swarm Position

The swarm position feature is the coordinates of the centroid of the swarm. We can see the trajectory followed by the swarm centroid in Figure 12 as the bright red line in the middle of the plot among nominal agents (Figure 12a) and faulty agents (Figure 12b).

Swarm Speed

The swarm speed feature represents the velocity of the swarm centroid. In Figure 27 we



(a) Swarm trajectory among nominal agents.

(b) Swarm trajectory among faulty agents.

Figure 26: Swarm trajectory.

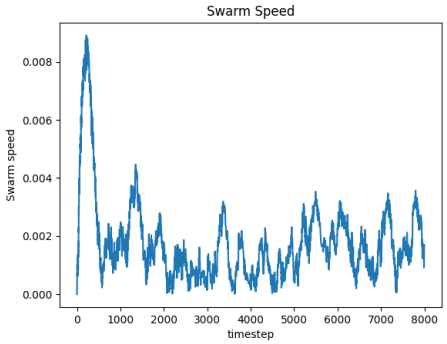
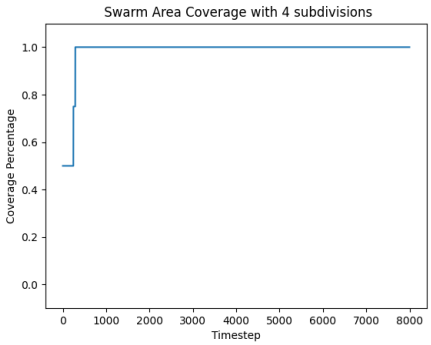


Figure 27: Foraging swarm speed.

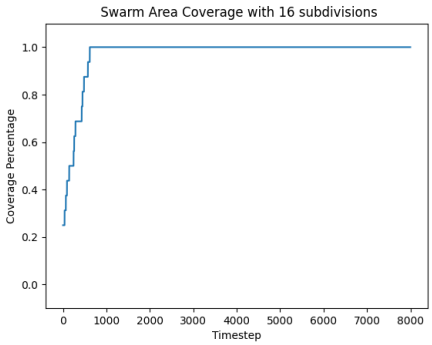
can see that the values have a high spike at the beginning, when most of the robots start exploring. Despite this feature showing exactly the fault injection in Figure 13, in this situation we are unable to observe the fault injection moment due to the nature of the task that does not have a homogeneous behavior.

Swarm Area Coverage

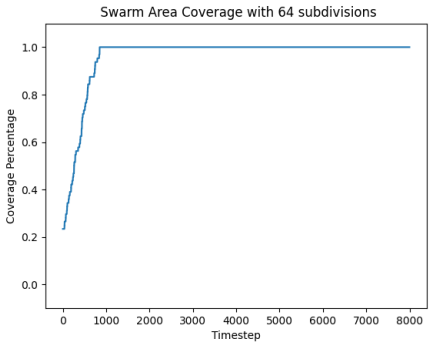
The swarm area coverage features illustrate the percentage of area covered by the entirety of the swarm considered each agent. This feature is similar to the area coverage feature seen before in section Area Coverage. As stated above, this is a global feature and does not have the objective to directly identify the fault injection. Nonetheless, the graphs shown in Figure 14 do not shown any sign of the presence of the fault injected after 150 s.



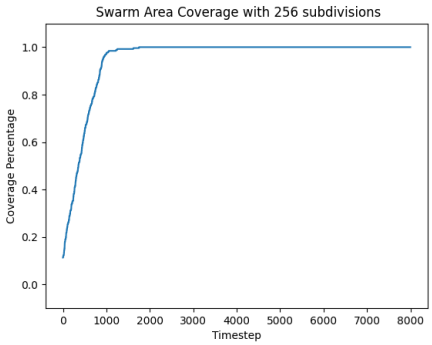
(a) Foraging swarm 4 subdivisions area coverage feature values.



(b) Foraging swarm 16 subdivisions area coverage feature values.

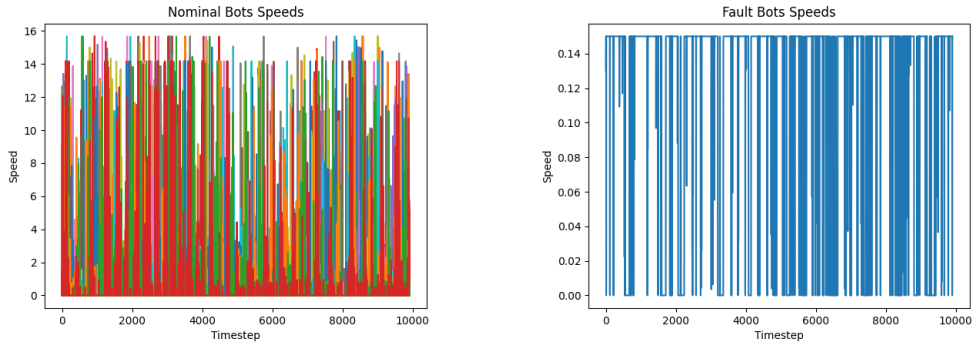


(c) Foraging swarm 64 subdivisions area coverage feature values.



(d) Foraging swarm 256 subdivisions area coverage feature values.

Figure 28: Foraging swarm area coverage feature values.



(a) RMFS nominal agents speed feature values.

(b) RMFS faulty agents speed feature values.

Figure 29: RMFS speed feature values.

0.1.3. RMFS Features Analysis

The RMFS task of the RAWSim-O simulator is explained in Section **??**. The robot trajectories can be seen in Figure **??**. In this example, we will analyze a simulation of 25 agents with 1 faulty agent. The fault is injected from the beginning of the simulation and the faulty robot travels at 5% of the maximum speed until the end of the simulation. The positions of all the robots are recorded only when a robot executes its `Act()` function, since this function is not called at every timestep, the time series data points are not evenly distributed in time.

Speed

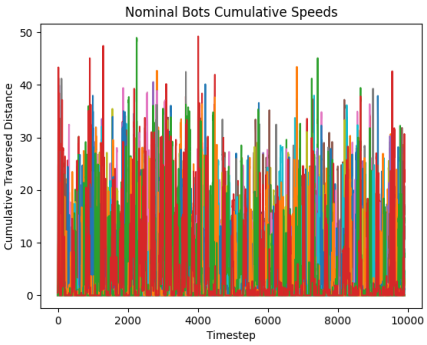
As stated before, the speed feature is highly discriminating and we can observe this characteristic in the graph in in Figure 29. We can see that the faulty robot does not reach high speeds like nominal robots.

Cumulative Speed

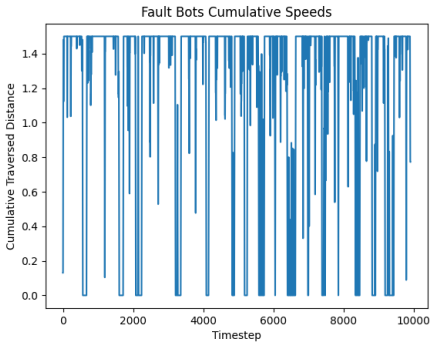
From the graphs in Figure 30 we can appreciate the same characteristic of the graphs in Figure 29 with fewer spikes and higher values. The speed graphs do not show any discriminating information other than the maximum speed value a robot can reach.

Neighbors Number

The neighbors number feature (Figure 31) does not show any particular difference between nominal robots and faulty robots. Like we have seen in Figure 17 for the foraging task, the faulty robots have similar values to nominal ones. The RMFS task does not have the same behavior as the foraging task where robots wander around performing a random walk, in this situation the agents have to follow specific routes and reach specific locations, however, the continuous movements in the warehouse force the robots to visit crowded locations with the same chance of visiting locations with any other robot.

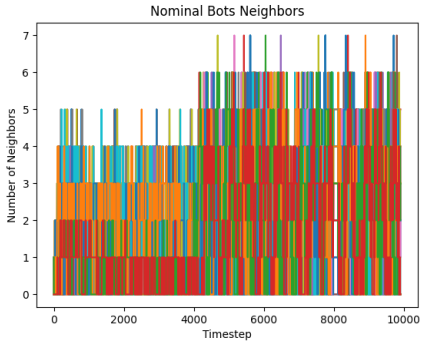


(a) RMFS nominal agents cumulative speed feature values.

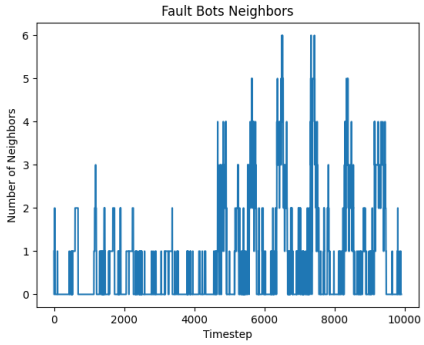


(b) RMFS faulty agents cumulative speed feature values.

Figure 30: RMFS cumulative speed feature values.



(a) RMFS nominal agents neighbors number feature values.



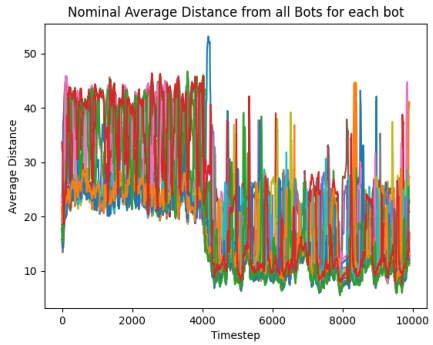
(b) RMFS faulty agents neighbors number feature values.

Figure 31: RMFS neighbors number feature values.

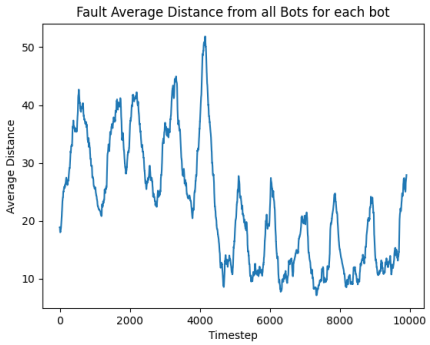
In this situation the neighborhood radius value has been set to 2 m. This value has to be fine tuned in order to have significant results.

Neighbors Average Distance

The graphs shown in Figure 32 have some similarities with the neighbors number feature. Neighbors average distance and the number of neighbors feature are inversely correlated but depict the same behaviors. In Figure 32b we can assume that the lowest values are reached when the robot approaches the pick stations. Instead, the higher values depict situations in which the robot is searching for a pod in less visited regions. From the two graphs, we can observe two phases, a first phase in which the feature assumes higher values and a second phase in which the feature assumes on average lower values. This behavior should be analyzed with a complete recording of the simulation, however, we can assume that these values depict a first phase in which the robots visited both pick and replenishment stations, ending in covering a broader area of the warehouse, and a

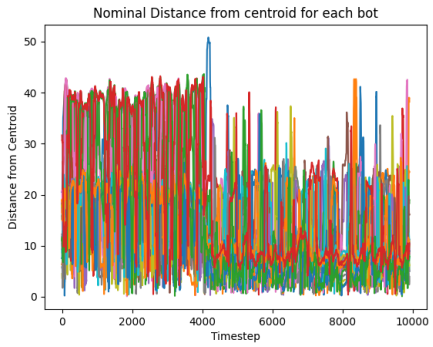


(a) RMFS nominal agents neighbors average distance feature values.

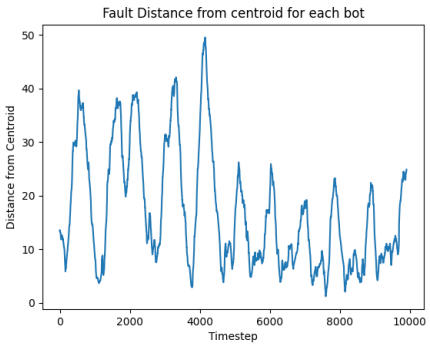


(b) RMFS faulty agents neighbors average distance feature values.

Figure 32: RMFS neighbors average distance feature values.



(a) RMFS nominal agents centroid distance feature values.



(b) RMFS faulty agents centroid distance feature values.

Figure 33: RMFS centroid distance feature values.

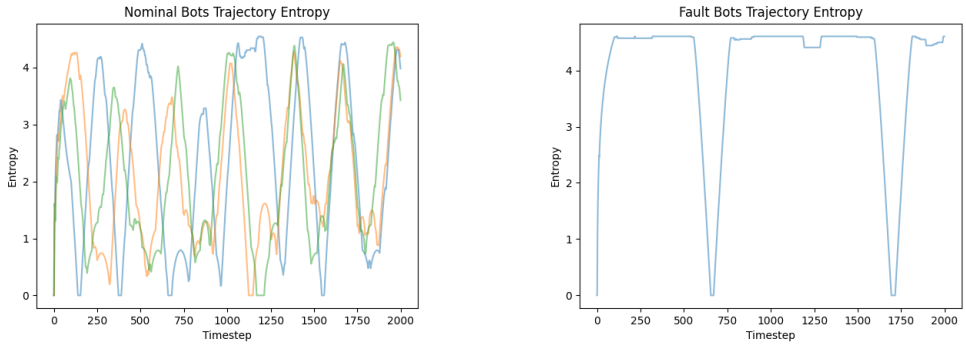
second phase where the robots were closer together may be due to a congestion on the replenishment station or due to an excess of replenishment orders.

Centroid Distance

In Figure 33 we can observe the same behavior of the neighbors average distance feature shown in Figure 32. The only difference we can see between the two features is that the centroid distance shows higher oscillations and less distinguished phases. We could note that the faulty agents have fewer spikes with respect to the nominal agents, we can assume this is a consequence of the lower speeds of faulty robots but it is necessary to implement a model able to interpret time series to exploit these characteristics.

Position Entropy

In Figure 34 we analyze a reduced time series with fewer nominal bots to underline the differences between nominal and faulty agents. In particular, we can see that the



(a) RMFS nominal agents position entropy feature values.

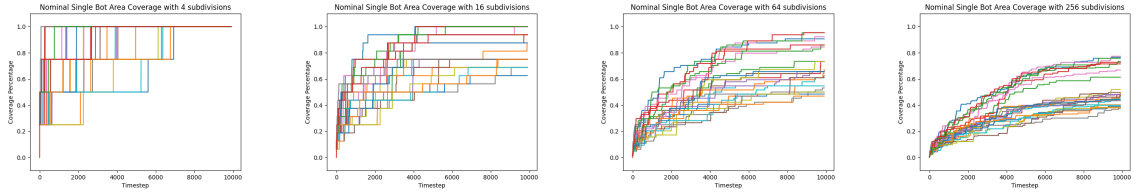
(b) RMFS faulty agents position entropy feature values.

Figure 34: RMFS position entropy feature values.

nominal robots have a higher number of spikes, as stated before this is a consequence of the difference in speed. In Figure 34 we can see that the faulty agent reaches higher values on average than its nominal counterparts. This is an undesired behavior since the slowest robot should visit fewer locations and it should not have high entropy values. However, this feature characteristic is a direct consequence of the `time-window` parameter, confirming once again that each parameter has to be fine tuned for each specific task.

Area Coverage

The area coverage feature does not depict any new behavior with respect to the graphs analyzed in Figure 8 or Figure 22. The only clear difference we can appreciate is between Figure 35c and Figure 36c or between Figure 35d and Figure 36d where the faulty agent reaches lower values on average than the nominal agents.



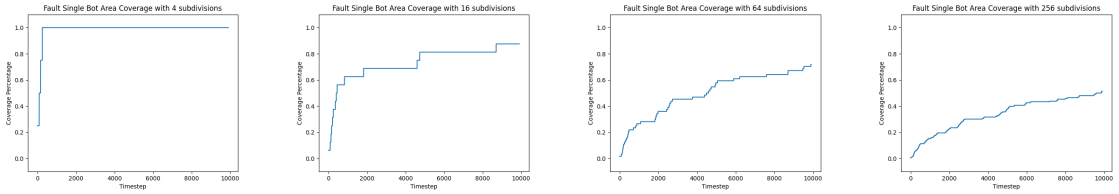
(a) 4 subdivisions.

(b) 16 subdivisions.

(c) 64 subdivisions.

(d) 256 subdivisions.

Figure 35: RMFS nominal agents area coverage feature values.



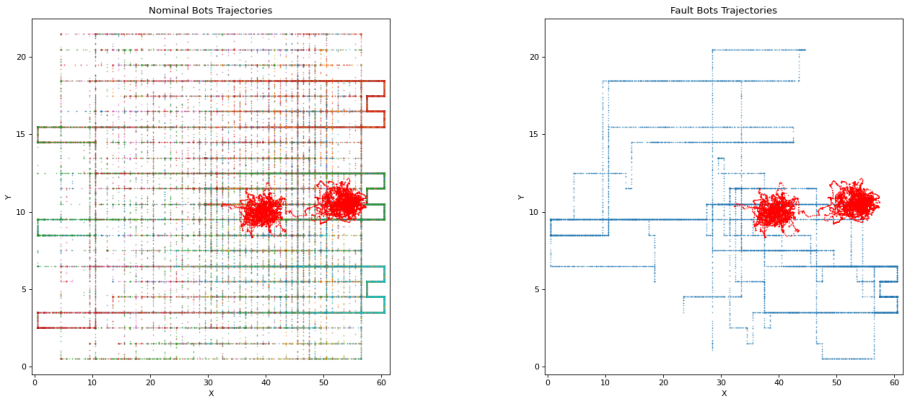
(a) 4 subdivisions. (b) 16 subdivisions. (c) 64 subdivisions. (d) 256 subdivisions.

Figure 36: RMFS faulty agents area coverage feature values.

Global Features

Swarm Position

The swarm position feature is the coordinates of the centroid of the swarm. We can see the trajectory followed by the swarm centroid in Figure 37 as the bright red line in the middle of the plot among nominal agents (Figure 37a) and faulty agents (Figure 37b). Since the data is not evenly distributed in time and we do not have a continuous representation of the robot movements, the swarm position is presented as a set of points mixed with robots trajectories. In particular, we can see that are two main points aggregations, one closer to the middle of the warehouse and one closer to the pick stations. These positions reflect the situation observed for the neighbors average distance graphs in Figure 32.



(a) Swarm trajectory among nominal agents. (b) Swarm trajectory among faulty agents.

Figure 37: RMFS Swarm trajectory.

0.2. Metrics

In this section, we will introduce the metrics used to evaluate the model performance and explain how to interpret them.

0.2.1. Confusion Matrix

The distinction between nominal and faulty agents is a classification task that falls into the anomaly detection category. Most of the time, anomaly detection implies the presence of an unbalanced set since it is easy to imagine that the anomalies should be less frequent than the nominal data samples. In this situation, analyzing the model performances solely from the accuracy (Equation 1) values can be misleading.

$$\text{classification accuracy} = \frac{\text{correct predictions}}{\text{total predictions}} \tag{1}$$

If we imagine a dataset with very few samples belonging to the faulty class, a trivial model that predicts only the nominal class should obtain high accuracy performance even though it is not rightfully classifying the faulty data samples. To overcome this difficulty we use the confusion matrix which highlights the number of samples classified correctly for each class. The goal of the confusion matrix is to show how confused is our model while making predictions on the test dataset. The confusion matrix used in our experiments is organized with the actual class instances in the rows and the predicted class instances in the columns, we consider faulty agents for the positive class and negative agents for the negative class. We can see an example of the confusion matrix structure in Table 1. More precisely, we can see each cell in the matrix as:

- **True positive:** correctly predicted event values.
- **False positive:** incorrectly predicted event values.
- **True negative:** for correctly predicted no-event values.
- **False negative:** for incorrectly predicted no-event values.

In particular, a good performance indicator would be to have the majority of the classified samples on the diagonal of the matrix, namely in the true negative and true positive regions.
Furthermore, to extract more information from the confusion matrix, we compute the recall, precision, and f1-score indexes. The precision is defined as:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positive} + \text{False Positives}}.$$

This measure is used to "evaluate the fraction of correct classified instances among the ones classified as positive" [**?** , pg.52]. The precision values range from 0.0 to 1.0 with 0.0 denoting precision absence and 1.0 denoting perfect precision.
The recall is defined as:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positive} + \text{False Negatives}}$$

and it quantifies the number of correct positive predictions retrieved from all the positive predictions that are possible. The recall values range from 0.0 to 1.0 and have the same behavior of precision values.
The use of these two metrics depends on what is the objective of the classifier, if the

True Label	**n′**	True Negative	False Positive
	p′	False Negative	True Positive
		n	**p**
		Predicted Label	

Table 1: Confusion matrix structure.

classifier has to minimize the number of false positives, then it must have high precision, if the classifier has to minimize the number of false negatives, then it must have a high recall.
In order to combine the precision and recall measure, we compute the F1-score. This metric is the harmonic mean of precision and recall and it is computed with the following formula:

$$\text{F1-score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

The F1-score can assume values from 0.0 to 1.0 and, like all the previous measures, 0.0 identifies poor performances while 1.0 identifies perfect performances.

0.2.2. Precision Recall Curve

In classification tasks, it is useful to measure how the performances change when using different thresholds to distinguish a sample between positive and negative classes. In binary classification, we can imagine the threshold as the numeric boundary that draws a line between the negative and positive classes. The receiver operating characteristic (ROC) curve and the precision recall curve are two useful metrics to evaluate the classification performance of a model when using different thresholds. In this section, we will explain how the precision-recall curve works and how it can be interpreted.
We have to underline that precision and recall are not directly correlated. If we try to increase the precision of a model by lowering the threshold to identify more positive events, this does not imply that the number of false negatives will decrease so that the recall will increase.
The precision recall curve graphs will also indicate the average precision (AP) measure. The average precision is defined as follows:

$$AP = \sum_n (R_n - R_{n-1}) P_n$$

where P_n and R_n are the precision and recall at the n-th threshold. This measure identifies the area subtended by the precision recall curve. The AP will range from 0.0 to 1.0, values

lower than 0.5 usually mean that the model is performing worse than a random guesser or a trivial model predicting only one class, values higher than 0.5 mean that the model is able to classify and distinguish a fair amount of samples. A model with perfect performances will have an AP value around 1.0 while a model with values around 0.0 usually presents some errors in the parameters or is trying to classify corrupted data.

0.3. Gradient Boosting Performance Evaluation

In this section, we will analyze the performance of our model on the fault detection for the 3 different tasks introduced in Section **??**. The performances are divided into 3 different sets of features, these sets aim at collecting different points of view. Even though the goal of our work is to perform fault detection from an external point of view, it is useful to analyze different sets in order to understand which effects and consequences they could have in other approaches.
The first set represents the point of view of the single agent, in this set we have included the following features: position, speed, direction, cumulative speed, position entropy, area coverage, and area coverage speed.
The second set represents the point of view of the single with the acknowledgment of being surrounded by other robots. This feature set includes the following features: position, speed, direction, cumulative speed, neighbors number, neighbors average distance, centroid distance, cumulative centroid distance, position entropy, area coverage, and area coverage speed.
The third and last set represents the point of view of the single agent with complete knowledge of its surroundings and the agents of the swarm. This feature set includes the following features: position, speed, direction, cumulative speed, neighbors number, neighbors average distance, centroid distance, cumulative centroid distance, position entropy, area coverage, area coverage speed, swarm centroid positions, swarm speed, and swarm area coverage.
The gradient boosting model is trained with the following settings:

- `loss = ‘‘deviance’’`: deviance referes to the logistic regression loss function.
- `learning_rate = 0.1`: it is the default value and it is needed to shrink the contribution of each tree.
- `max_depth = 3`: this parameter defines the maximum depth of the classification trees. Since the gradient boosting method uses a series of simple classifiers we set this parameter to a low value. However, the documentation states that it should be cross-validated and the performances may be influenced by this parameter.
- `n_estimators=500`: it is the maximum number of tree estimators that are introduced in the model. The documentation suggests setting this number to high number since the gradient boosting model is very robust against overfitting.
- `max_features=‘‘sqrt’’`: it is the maximum number of feature to consider when looking for a split, in this situation, our model consider a number of feature that is the square root of the maximum number of features. The documentation says that for classification tasks it is better to consider a smaller subset of features.

- `n_iter_no_change = 5`: this parameter is used to define how many steps without improving validation score are needed to stop the training before its termination. This value is needed to perform early stopping, a technique that prevents the model to overfit on the training data.
- `tol = 1e-3`: 1e-3 represent the scientific notation in python and it is equal to $1 \cdot 10^{-3} = 0.001$. This is the minimum improvement required to detect a change in the validation set performance while performing early stopping.

0.3.1. Flocking Fault Detection

The flocking task simulations we will analyze in this section are described in Section **??**. In section **??**, we have described a summary of the parameters we have used to collect all the simulations used in the dataset. In general, the overall procedure to execute all the simulations is explained in Algorithm 1. It is implied that every time a simulation is executed its log files are automatically collected. Once the simulation labeled positions have been collected, they are processed to build each feature time series. The elaborated data is processed to obtain an unique time series for the train data set and one for the test dataset.

Confusion Matrix

The confusion matrix describes how many samples of each class the model is able to correctly predict. In Figure 38 we can see that the gradient boosting model has quite perfect perfomances on the flocking task, the majority of the classified samples is on the diagonal of the confusion matrix and the false positive and false negative cells are almost empty. More precisely, in Table 2 we can see that the accuracy, precision, and recall metrics have perfect scores and the F1-score never goes below the 99 percent. Based on which behavioral aspect is more important we can focus either on the precision or on the recall metric. For example, if it is more important to detect immediately a faulty agent because it is dangerous to leave it operating, it is better to have a high recall. On the other hand, if the recovery procedure is expensive in term of times or resources, and the task can be completed without the faulty agent, then its better to focus on the precision and not have a lot of false positives. In this situation all the measures are very promising and it is useless to make any considerations about these values.
In Table 2 we can see the exact values of the performance metrics for each feature subset, given the high values and the very small differences among them it does not make sense to assess any influence of the different feature sets.

Precision Recall Curve

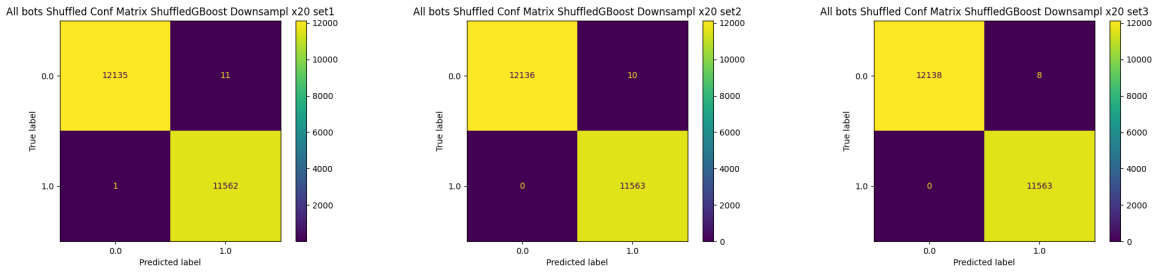
In Figure 39 we can see the precision recall curves of the different feature subsets. It is important to underline that the y-axis of all the graphs starts at values above 0.99 and the AP metric does not show values below 1.00. The values shown in this section reflect a perfect classification performed by the model. The precision recall curve stays at 1.00 values for the precision for the majority of recall values and lowers to values higher than 0.99 for recall values around 0.9. The plots shown in Figure 39 describe a model

Algorithm 1 Flocking Simulation Execution.

```
nominal-repetitions=15.
positions= { North, South, East, West }.
for i = 1 to nominal-repetitions do
    for position in positions do
        execute nominal simulation from position.
    end for
end for
faults= {10%, 20%, 33%}.
fault-timesteps = { 0, 500, 1500, 4000 }
single-bot-fault-repetitions = 4.
for i = 1 to single-bot-fault-repetitions do
    for position in positions do
        for fault-time in fault-timesteps do
            execute single bot fault simulation from position with fault injection after fault-time.
        end for
    end for
end for
for fault-percentage in faults do
    for position in positions do
        for fault-time in fault-timesteps do
            execute fault-percentage simulation from position with fault injection after fault-time.
        end for
    end for
end for
```

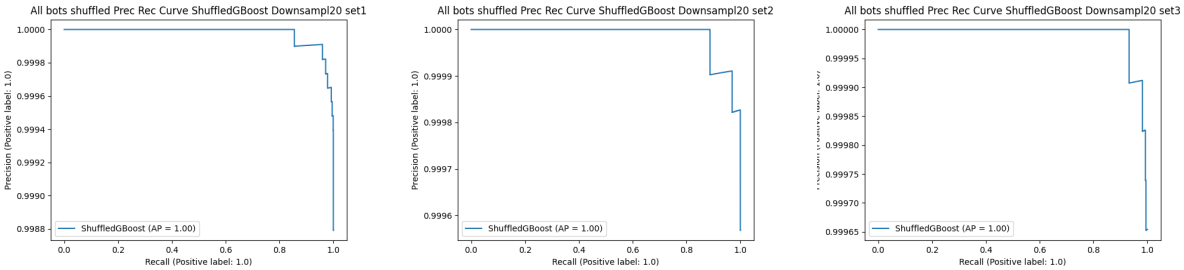
Feature Set	Accuracy	Precision	Recall	F1
1	0.9995	0.999	0.9999	0.9995
2	0.9996	0.9991	1.0	0.9996
3	0.9997	0.9993	1.0	0.9997

Table 2: Performances of the gradient boosting model on the flocking task.



(a) Feature set 1. (b) Feature set 2. (c) Feature set 3.

Figure 38: Confusion matrices of the gradient boosting model performances on the flocking task.



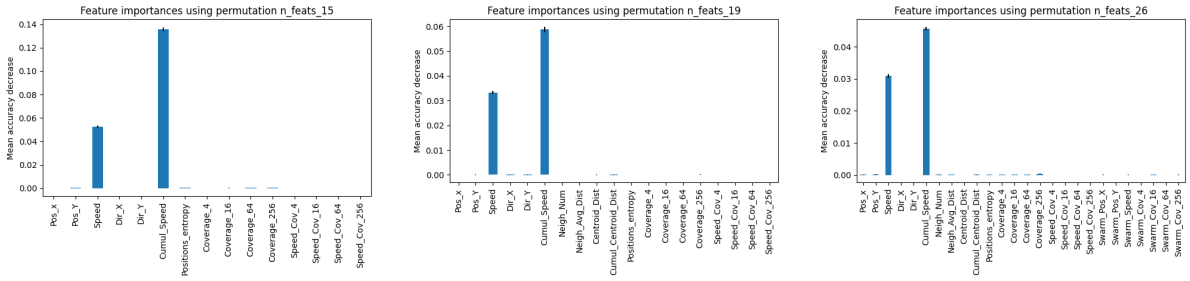
(a) Feature set 1. (b) Feature set 2. (c) Feature set 3.

Figure 39: Precision recall curves of the gradient boosting model performances on the flocking task.

that is able to perfectly distinguish the positive and negative class for every level of the classification threshold.

Feature Permutation Results

The graphs shown in Figure 40 gather the result of the feature permutation results for the flocking task for the different features sets. As anticipated in Section 0.1.1, the speed feature is highly discriminating and the cumulative speed is completely correlated with speed. These characteristics are confirmed by the feature permutation results, in Figure 40a we see clearly that the cumulative speed feature has a significant influence on the prediction results. Immediately after the cumulative speed feature importance we see the speed feature, this feature has a relevant importance in the classification but not as high as the cumulative speed. The importance values reached by these two features are relevant but not fundamental, a feature importance value of 0.14 (Figure 40a) for the cumulative speed feature means that the accuracy score of the model on the test set would decrease of 0.14 if the cumulative speed feature is randomly shuffled.This values are significant but do not represent the majority of the accuracy score reached by the model, this means that the feature has discriminant characteristics but is not fundamental for the classification task. In order to appreciate more significant results it would be useful to inspect more



(a) Feature set 1. (b) Feature set 2. (c) Feature set 3.

Figure 40: Feature permutation results on the flocking task.

subsets combination of feature and see how the feature are correlated.

0.3.2. Foraging Fault Detection

The foraging task simulations we will analyze in this section are described in Section **??**. In section **??**, we have described a summary of the parameters we have used to collect all the simulations used in the dataset. In general, the overall procedure to execute all the simulations is explained in Algorithm 2. It is implied that every time a simulation is executed its log files are automatically collected. Once the simulation labeled positions have been collected, they are processed to build each feature time series. The elaborated data is processed to obtain an unique time series for the train data set and one for the test dataset.

Confusion Matrix

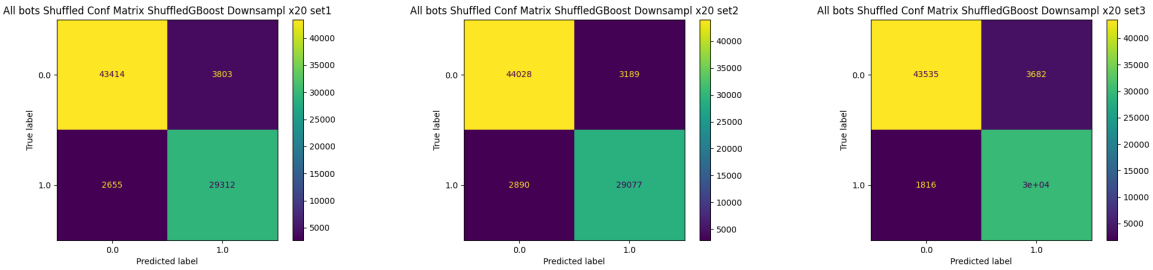
The confusion matrix graphs shown in Figure 41 depict very promising results like in the previous section for the flocking task. The confusion matrices have most of the samples on the diagonal and the cells of the false positive and false negative class have very low values. We can appreciate a small difference between the confusion matrix of the feature set 3 (Figure 41c) and the other two confusion matrices of feature set 1 and 2 (Figure 41a and Figure 41b), in particular we can see that the feature set 3 has a lower number of false negative samples and an higher number of false positive samples. A possible exaplnation of this behavior may be that the hyperspace defined by the feature set 3 arranges the samples such that the model ends up in selecting a threshold more keen to detect positive events instead of negative one, however this interpretation requires further analysis and has to be taken with a grain of salt.
In Table 3 we can see the exact values of the accuracy, precision, recall, and F1-score metrics. In this situation the differences between each features set are more evident than the ones of the flocking task (Table 2). In particular we can see some small improvements in the F1-score performances with the increase of the feature set size. The performance improvements of the feature set 3 are a consequence of the high values in the recall metric, as observed before in the confusion matrix, the feature set 3 detects an higher number of positive samples thus increasing the recall while lowering the precision. The interpretation

Algorithm 2 Foraging Simulation Execution.

nominal-repetitions=15.
arena-dimensions= { 5, 6, 7, 8 }.
for $i = 1$ to nominal-repetitions **do**
  for (x, y) in arena-dimensions $\times$ arena-dimensions **do**
    execute nominal simulation with arena of size $x \times y$.
  end for
end for
faults= {10%, 20%, 33%}.
fault-timesteps = { 0, 1000, 4000 }
single-bot-fault-repetitions = 4.
for $i = 1$ to single-bot-fault-repetitions **do**
  for (x, y) in arena-dimensions $\times$ arena-dimensions **do**
    for *fault-time* in fault-timesteps **do**
      execute single bot fault simulation with arena of size $x \times y$ with fault injection after *fault-time*.
    end for
  end for
end for
for *fault-percentage* in faults **do**
  for (x, y) in arena-dimensions $\times$ arena-dimensions **do**
    for *fault-time* in fault-timesteps **do**
      execute *fault-percentage* simulation with arena of size $x \times y$ with fault injection after *fault-time*.
    end for
  end for
end for

Feature Set	Accuracy	Precision	Recall	F1
1	0.9184	0.8852	0.9169	0.9008
2	0.9232	0.9012	0.9096	0.9054
3	0.9306	0.8912	0.9432	0.9164

Table 3: Performances of the gradient boosting model on the foraging task.



(a) Feature set 1. (b) Feature set 2. (c) Feature set 3.

Figure 41: Confusion matrices of the gradient boosting model performances on the foraging task.

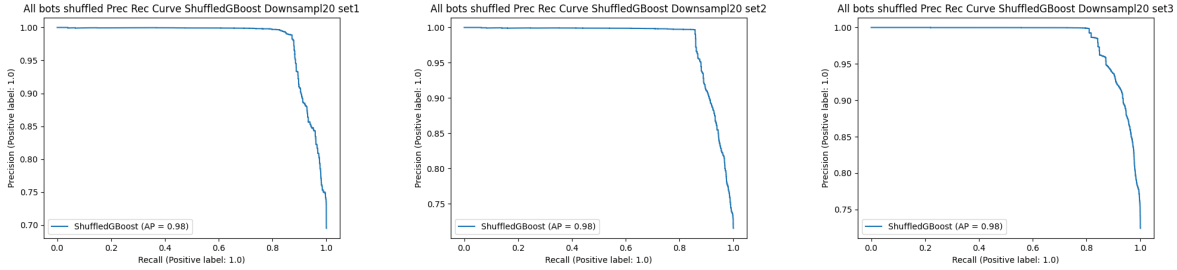
of these results depends on where the model has to put more attention. If the detection of positive samples is critical then it should be better to have an high recall. If the presence of false positives is troublesome and not worth the time, then it is better to have an high precision.

Precision Recall Curve

The precision recall curve graphs shown in Figure 42 depict interesting results and reflect the satisfactory performance shown by the confusion matrices in Figure 41. At first sight, the precision recall curves do not shown any particular difference between each feature set. The AP metric is equivalent among all the feature set and the curve have almost the same slopes with similar values. Since there is very small or none difference at all between the graphs, it is not possible to make any assumption on the influence of the different feature sets.

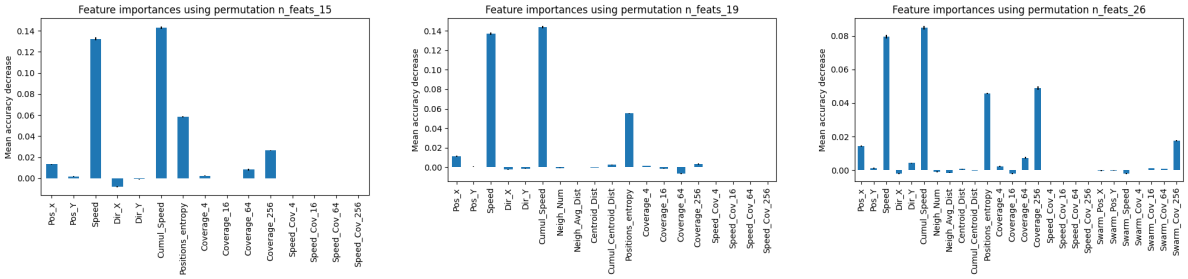
Feature Permutation Results

The feature permutation results shown in Figure 43 present some interesting details. In general , we can see that the speed and cumulative speed feature show moderate levels of importance like in the previous graphs for the flocking task. This is a consequence of the nature of the fault injected that forces the robot to have a specific speed. We can observe that these graphs show higher importance for the cumulative speed feature rather than the speed feature, this could suggest that it is better to embed past information into



(a) Feature set 1. (b) Feature set 2. (c) Feature set 3.

Figure 42: Precision recall curves of the gradient boosting model performances on the foraging task.



(a) Feature set 1. (b) Feature set 2. (c) Feature set 3.

Figure 43: Feature permutation results on the foraging task.

an unique feature.The importance values of the speed and cumulative speed feature are relevant but not very significant. As explained in Section 0.3.1 in the feature importance analysis for the flocking task, the feature importance values shown in Figure 43 describe the increase or decrease for the accuracy score. The feature importance graphs for the flocking task show importance values of at most 0.14 that are not a major percentage of the the accuracy scores obtained by the model. Although the importance feature values are modest, we can not make any precise assumption on the real importance of these features since there different subset of features to analyze.

Differently from the flocking task, in this situation we can observe that there are some other features that have small but considerable importance values. More in particular: in Figure 43a the position entropy feature has relevant values for the first time; in Figure 43b we can see that the position entropy feature has similar values to the ones in Figure 43a; lastly, in Figure 43c there is the position entropy feature that has relevant values but it is overcome by the area coverage with 256 subdivisions. The values reached by these features do not have significant influence on the model performances but they definitely have some influence in the classification task and can be further analyzed in more detailed experiments.

0.3.3. RMFS Fault Detection

The RMFS task simulations we will analyze in this section are described in Section **??**. In section **??**, we have described a summary of the parameters we have used to collect all the simulations used in the dataset. In general, the overall procedure to execute all the simulations is explained in Algorithm 3. It is implied that every time a simulation is executed its log files are automatically collected. Once the simulation labeled positions have been collected, they are processed to build each feature time series. The elaborated data is processed to obtain an unique time series for the train data set and one for the test dataset.

Algorithm 3 RMFS Simulation Execution.

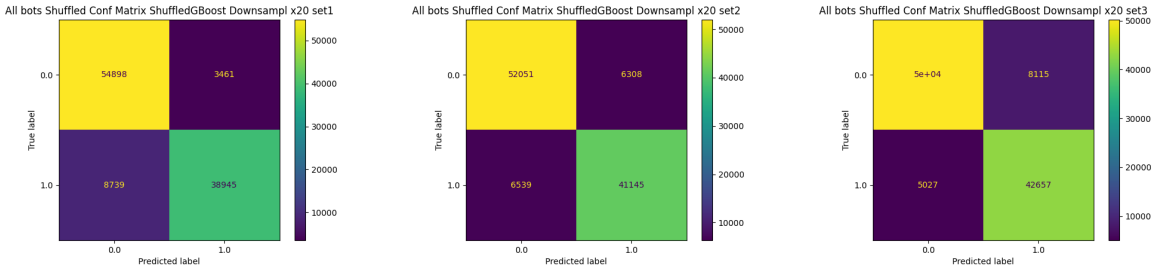
```
nominal-repetitions=4.
aisles-numbers = { 5, 6, 7, 8 }.
for i = 1 to nominal-repetitions do
    for (x, y) in aisles-numbers × aisles-numbers do
        execute nominal simulation with warehouse of size x × y.
    end for
end for
faults = {10%, 20%, 33%}.
fault-timesteps = { 0, 1000, 4000 }
single-bot-fault-repetitions = 4.
for i = 1 to single-bot-fault-repetitions do
    for (x, y) in aisles-numbers × aisles-numbers do
        for fault-time in fault-timesteps do
            execute single bot fault simulation with warehouse of size x × y with fault
            injection after fault-time.
        end for
    end for
end for
for fault-percentage in faults do
    for (x, y) in arena-dimensions × arena-dimensions do
        for fault-time in fault-timesteps do
            execute fault-percentage simulation with warehouse of size x × y with fault
            injection after fault-time.
        end for
    end for
end for
```

Confusion Matrix

In Figure 44 we can see the confusion matrices of the model performances for the RMFS task. The confusion matrices graphs depict promising performances, in particular, the samples are mostly in the diagonal of the matrix and the false negative and false positive cells have fairly low numbers. From the confusion matrices graphs we can observe there

Feature Set	Accuracy	Precision	Recall	F1
1	0.885	0.9184	0.8167	0.8646
2	0.8789	0.8671	0.8629	0.865
3	0.8761	0.8402	0.8946	0.8665

Table 4: Performances of the gradient boosting model on the RMFS task.



(a) Feature set 1. (b) Feature set 2. (c) Feature set 3.

Figure 44: Confusion matrices of the gradient boosting model performances on the RMFS task.

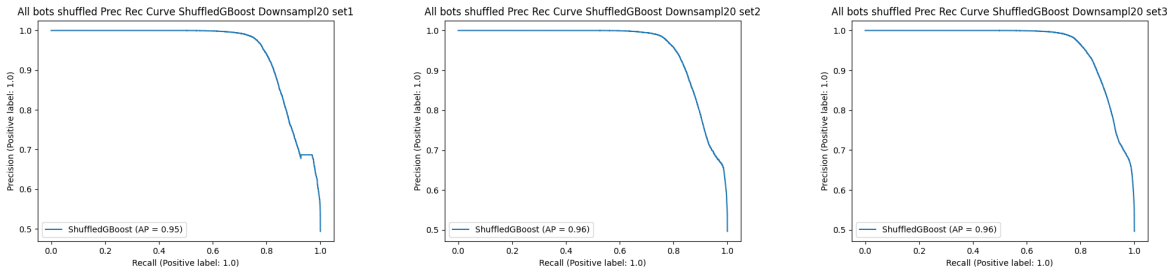
is a mild trend from feature set 1 to feature set 3 on selecting more positive samples. This trend can be seen on the colors of the predicted positive class column that gets more intense from the left to the right. The bias towards the positive class can also be confirmed in Table 4, we can see that from feature set 1 to feature set 3 the precision gets lower while the recall improves significantly.

Precision Recall Curve

The precision recall curve graphs shown in Figure 45 reflect the behavior seen in Table 4. From the left, we can see that the precision recall curve of the feature set 1 has an average precision of 0.95 while the other two graphs have an average precision of 0.96. More in particular, the precision recall curve of feature set 3 has an area under the curve bigger than feature set 2.
From the behaviors of the precision recall curve graphs we can say that the samples mix together in some areas of the hyperspace because the model drops in performance with the modification of the classification threshold. This characteristic can be seen in the decrease of the curve before the small step at the end.

Feature Permutation Results

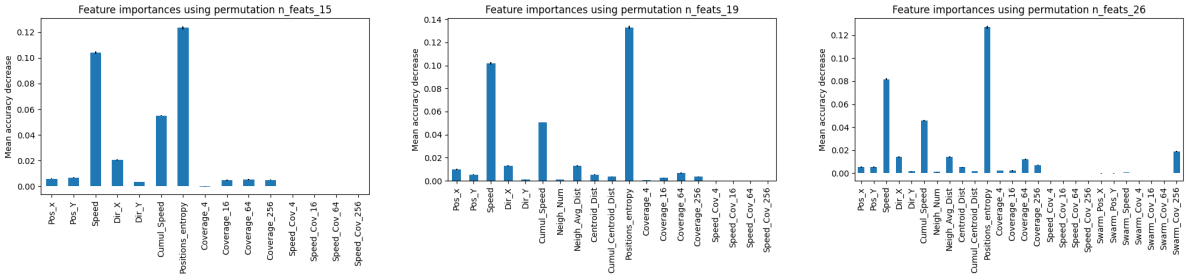
The feature importance graphs shown in Figure 46 present a new characteristic that did not appear in the previous tasks. The graphs show that the position entropy feature has the highest feature importance value, immediately followed by the speed and cumula-



(a) Feature set 1. (b) Feature set 2. (c) Feature set 3.

Figure 45: Precision recall curves of the gradient boosting model performances on the RMFS task.

tive speed features. Since the speed feature and the cumulative speed feature still have modestly high values, we can not say that the position entropy is the most discriminant feature. Considering the small value difference between the feature importance of the speed feature and the position entropy feature on the feature set, this behavior could vanish with a different permutation of the dataset or a new training execution. In order to confirm the importance of the position entropy feature this results should be cross validated with multiple training executions. However, it is interesting to see that the position entropy features presents significant importance values across the three feature sets. This result can be a hint on which features to analyze in the future analysis.
All the other features in the feature sets, other than speed and cumulative speed, have insignificant values and do not show any interesting quality.
In the end, when we perform feature importance using highly correlated features, like speed and cumulative speed for example, there are some precautions to take. When we compute the feature permutation importance of two correlated features, the model still has access to the correlated feature and the final results will have a lower importance score for the feature under analysis. This underlines that using correlated features should be avoided and when done it has to be taken under consideration for the feature importance scores.
Another aspect that would be interesting to analyze is the difference of the feature importance between the training dataset and the test dataset. This analysis would highlight features that may lead the model to overfit. If a feature is important on the training set but not on the test set, it is likely to drive the model to overfit.



(a) Feature set 1. (b) Feature set 2. (c) Feature set 3.

Figure 46: Feature permutation results on the RMFS task.