Run-time Detection of Faults in Autonomous Mobile Robots Based on the Comparison of Simulated and Real Robot Behaviour

Alan G. Millard
Department of Computer Science
York Robotics Laboratory
University of York
York, UK
Email: millard@cs.york.ac.uk

Jon Timmis
Department of Electronics
York Robotics Laboratory
University of York
York, UK
Email: jon.timmis@york.ac.uk

Alan F.T. Winfield
Bristol Robotics Laboratory
University of the West of England
Bristol, UK
Email: alan.winfield@uwe.ac.uk

***Abstract*— This paper presents a novel approach to the run-time detection of faults in autonomous mobile robots, based on simulated predictions of real robot behaviour. We show that although simulation can be used to predict real robot behaviour, drift between simulation and reality occurs over time due to the reality gap. This necessitates periodic reinitialisation of the simulation to reduce false positives. Using a simple obstacle avoidance controller afflicted with partial motor failure, we show that selecting the length of this reinitialisation time period is non-trivial, and that there exists a trade-off between minimising drift and the ability to detect the presence of faults.**

I. Introduction

It has long been assumed that swarm systems are robust, in the sense that the failure of individual robots will have little detrimental effect on a swarm's overall collective behaviour. However, Bjerknes and Winfield [1] have recently shown that this is not always the case, particularly in the event of partial failures (such as motor failure). The reliability modelling in [1] shows that overall system reliability rapidly decreases with increasing swarm size, therefore this is a problem that cannot simply be solved by adding more robots to the swarm. Instead, future large-scale swarm systems will need an active approach to dealing with failed individuals if they are to achieve a high level of fault tolerance.

Christensen et al. [2] proposed one such approach, inspired by synchronised flashing behaviour seen in fireflies, that allows failed robots to be detected and physically removed by other operational members of the swarm. This ability of robots to detect faults in each other is referred to as exogenous fault detection [2]. The work of Christensen et al. represents the state-of-the-art in exogenous fault detection for robot swarms, but it only addresses the case of completely failed robots, the effect of which on collective behaviour has been shown to be relatively benign by Winfield and Nembrini [3]. The occurrence of partial failures is of far greater concern, as highlighted by Bjerknes and Winfield [1].

In [4] we proposed a novel method of exogenous fault detection capable of detecting partial failures, based on the comparison of expected and observed robot behaviour. Rather than having robots learn the expected behaviour of others over time, they would instead possess a copy of each other's controller code, which could be instantiated within an internal simulator. The model of expected behaviour therefore comprises a copy of another robot's controller, and a simulator that is able to execute the controller code in a simulated environment. Each robot would initialise its internal simulation such that it reproduces the relative positions and orientations of the other robots in reality. The simulation can then be run for a short period of time, and the resulting position and orientation of each robot may be compared against the observed state of the robots in reality. If there is a significant discrepancy between the predicted and observed position of a particular robot, then this may indicate that it has developed a fault that can be detected.

As an intermediate step towards the realisation of our goal, we must first solve the problem of predicting the behaviour of an isolated robot. Although this is a simpler task than predicting the behaviour of a robot in a swarm, it is still non-trivial. This paper reports our initial work towards solving this problem.

A. Predicting Robot Behaviour

Model-based fault detection approaches typically attempt to model the expected behaviour of a system using an abstract mathematical model [5]. Our motivation for using a simulation to predict robot behaviour, is that the controller code provides an executable model of the robot's expected behaviour. If the real world scenario can be reproduced in simulation, then assuming that the simulation is able to mimic the real world controller's sensory inputs and actuator outputs, the simulated robot should behave like the real robot, thus allowing us to predict its behaviour.

This approach also has advantages over some data-driven fault detection methods. For example, if we simply monitor a robot's velocity over time, we may not be able to distinguish between a fault that causes the robot to stop, and a situation when the robot stops for a legitimate reason. Instead, if we instantiate a model of the robot in simulation, the robot controller encodes the additional context required to differentiate these two scenarios.

The main problem with this method of predicting robot behaviour is that there will always be some discrepancy between simulation and reality, referred to as the reality gap [6]. Existing robot simulators are typically only used to develop controllers offline, or to evolve robot controllers

online [7]. Neither of these applications is specifically geared towards accurately predicting the behaviour of a real robot, so a crude model of robot behaviour is often sufficient.

There is a trade-off between the fidelity of the simulation and the speed at which it can be executed. A high fidelity simulation may model the robot's behaviour very well, but will run slowly. On the other hand, a low fidelity simulation may run very quickly, but provide poor predictions of robot behaviour. The simulation must run faster than real-time for it to be useful for predicting robot behaviour. However, it must also provide reasonably accurate predictions.

B. Fault Detection

Assuming that a simulation can provide sufficiently accurate predictions of real robot behaviour, we propose that it can be used for fault detection. The robot controller can be instantiated within the simulation, embodied in a simulated model of the real robot, and used to generate predictions of non-faulty behaviour. A significant discrepancy between these simulated predictions and the real robot's observed behaviour may indicate the presence of a fault.

Bjerknes and Winfield [1] demonstrated that motor failure had the most detrimental affect on a swarm's ability to carry out its task. For this reason, we use motor failure as case study in this paper. However, instead of testing complete motor failure, which would be very easy to detect due to rapid divergence of non-faulty and faulty behaviour, we investigate partial motor failure. The particular fault we consider here is a permanently slow left wheel. Over time, this fault will cause the robot to veer gently to the left. This is a minor fault, and therefore quite difficult to detect.

The focus of this paper is not upon finding optimal solutions for the case study considered here. We are simply interested in investigating the fundamental issues inherent in this new fault detection approach, primarily as an indication of whether similar issues might exist in other scenarios.

II. Experimental Setup

This section describes the experimental setup that was used to investigate the proposed method of fault detection.

A. Task Description

We chose the task of obstacle avoidance as our case study, because it is well-understood, and relatively simple to model in simulation. The robot performs obstacle avoidance in an enclosed circular arena with a diameter of 800mm free of obstructions, as shown in Figure 1.

B. The e-puck Robot & Linux Extension Board

We use a single open-hardware e-puck robot [8] augmented with a Linux Extension Board (LEB) [9], which improves its processing and memory resources, and enables Wi-Fi communication. The e-puck uses two differential drive stepper motors to move around. These afford it precise movement with virtually no inertia, which makes its behaviour easier to predict in simulation. The robot has eight active infra-red (IR) proximity sensors distributed around its body, which we use to provide input to the obstacle avoidance controller.

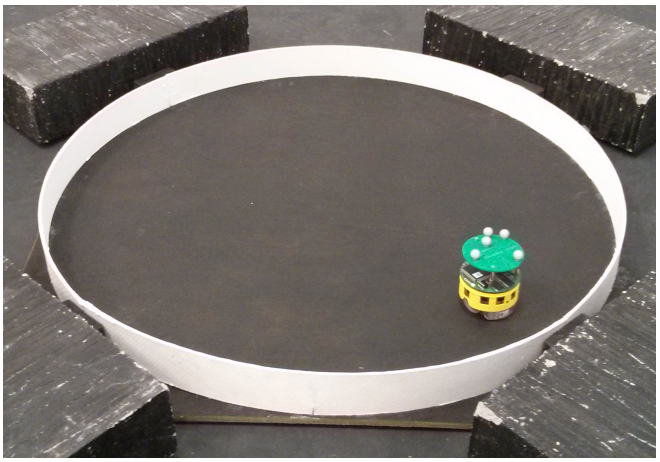

Fig. 1. The e-puck robot with Linux Extension Board and tracking hat, inside a circular 800mm diameter enclosed arena.

C. Robot Controller

The robot controller implements a simple obstacle avoidance behaviour that sets the wheel speeds based on IR sensor readings. The IR sensors are the only input to the robot controller, and the readings obtained from them are directly translated into left and right motor speeds using a vector of weights. This tight coupling of the sensors and actuators effectively results in a Braitenberg vehicle. IR sensor values below a certain threshold are ignored, so the robot's behaviour is insensitive to IR interference. Unless any of the IR sensor values are above this threshold, the robot will move in a straight line at 2.6cm/s.

The controller is deliberately stateless. This is because from the perspective of an outside observer, given only a snapshot the of the system at a particular instant in time, it would not possible to determine the internal state of the robot controller. It may be possible to infer the internal state given a history of the robot's behaviour, however this is beyond the scope of our initial work.

D. Simulator

We use a minimal 2D robot simulator developed by O'Dowd [7], as shown in Figure 2. It was initially designed for the embedded online evolution of robot controllers, and is specific to the e-puck robot platform. This simulator was chosen over more general purpose robot simulators due to its minimal nature, and the eventual goal of embedding behaviour prediction on the LEB. However, it was not originally developed with the aim of accurately predicting robot behaviour, so it was necessary to close the reality gap further by modifying the simulation to more accurately model the e-puck robot.

O'Dowd et al. [7] divided the reality gap into three categories of correspondence between simulation and reality:

- Robot-robot correspondence
- Robot-environment correspondence
- Environment-environment correspondence

In order to accurately predict robot behaviour, all three categories must be modelled with sufficient fidelity. We are

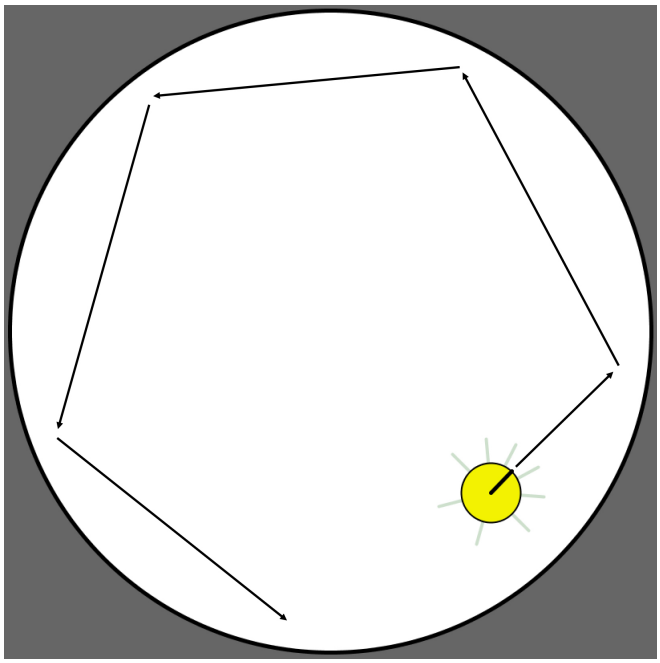

Fig. 2. Simulation of the e-puck in the circular arena shown in Figure 1. The lines protruding from the robot's body represent the range of each IR sensor. The arrows represent a typical trajectory resulting from the obstacle avoidance controller.

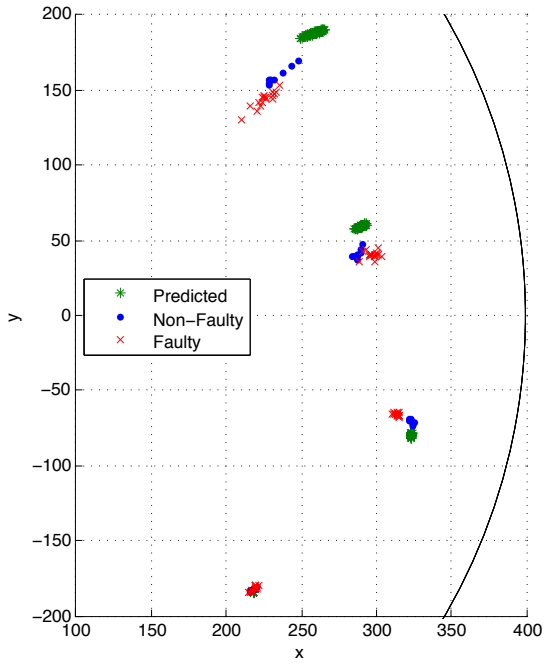

Fig. 3. Real robot non-faulty and faulty class distributions, and the predicted non-faulty class distribution over time. The robot begins at the coordinates (200, -200) facing the wall at 45°, as shown in Figure 2.

able to satisfy robot-robot correspondence quite easily. The e-puck can be modelled simply as a circle of the same circumference, and its wheel speeds are calculated from measurements of the real robot. Its position is updated using two-wheel differential drive kinematics. Each step of the simulation represents 10ms of real-time, so the granularity of movement is quite fine. Due to the absence of other obstacles, environment-environment correspondence is also easily satisifed by modelling the arena wall as a circle.

Robot-environment correspondence is harder to achieve, as it relies upon the use of an accurate IR sensor model. It has been shown that the response of active IR sensors depends not only on the distance from an obstacle, but also the angle, and the proportion of the beam that is reflected [10]. However, in this minimal simulator the IR sensor readings depend only on the distance from the wall, and are emulated using raw data obtained from the real robot's sensors, with the addition of uniform noise [7]. This deliberately simplistic sensor model allows us to explore the effect of imprecise simulation on the proposed fault detection method.

The simulated robot's controller code is a direct translation of the real robot's controller code. Assuming the real robot is non-faulty, this ensures that any deviation between expected and observed behaviour is due solely to the reality gap and observation inaccuracies.

The parameters of the simulation have been calibrated manually, to produce a sufficiently faithful reproduction of real robot behaviour. Our focus here is not upon obtaining perfect predictions of the real robot's future behaviour, we merely seek predictions accurate enough that we can achieve reliable fault detection.

E. Tracking System

In order to perform the comparison of expected and observed behaviour, we require a method of observing the real robot's behaviour. An OptiTrack^{TM} motion capture system is used to monitor the position and orientation (or *pose*) of the robot over time. The cameras detect the pattern of retro-reflective markers placed on the 'hat' that the robot wears (see Figure 1). The tracking data is used for post-experiment analysis, and transmitted to the robot over Wi-Fi so that it can be instructed to drive to a desired initial pose at the start of each experimental run.

III. Problem Analysis

This section presents the results of initial experiments that were carried out to investigate the issues inherent in using simulated predictions of robot behaviour for fault detection.

A. Real Robot Behaviour

For any particular initial pose, the robot's endpoint after a certain time period may be recorded. However, even with deterministic controller code, a real robot's behaviour is stochastic. This is because the sensory inputs to the controller are often prone to noise, particulary in the case of active IR sensors. There may also be a small discrepancy between the wheel speeds output by the controller and the actual speed of the robot, due to variations in the surface that the robot is driving on. Therefore, for any particular initial pose, there will in fact be a probability distribution of possible endpoints that the robot may reach after a given time period.

To demonstrate this, we used the tracking system to instruct the robot to drive to the coordinates (200, -200), and

TABLE I

ALL POSSIBLE CLASSIFICATIONS AND THEIR OUTCOMES

True class	Classification	Outcome
Faulty	Faulty	True Positive (TP)
Faulty	Non-faulty	False Negative (FN)
Non-faulty	Faulty	False Positive (FP)
Non-faulty	Non-faulty	True Negative (TN)

turn to an angle of 45°, such that it is facing the arena wall as shown in Figure 2. Once in position, the robot executes the obstacle avoidance controller for 20 seconds. Throughout the duration of the run, the robot's position is recorded using the tracking system. The robot then drives back to the same initial pose, and repeats the run. The experiment was first carried out using a non-faulty robot, and then repeated with a permanently 'faulty' robot that used a modified controller which reduced the left wheel speed output. For each robot, 15 repeat runs were carried out. This allowed us to sample from the underlying probability distribution of possible endpoints for each class of behaviour.

Figure 3 shows the results from the experiment. Initially, there is little variation in the robot's behaviour. However, over time the spread of the probability distribution increases. This is because the robot's trajectory after detecting the arena wall varies due to differences in the IR sensor readings between runs. It can be seen that the spread of the faulty robot's endpoint distribution similarly increases over time.

B. Simulated Prediction

In order to predict the behaviour of the non-faulty robot, the simulation is initialised with the same initial pose. The simulated robot's behaviour is similarly stochastic, due to noise added to the simulated IR sensor readings, and a small amount of noise added to the motor speeds. The simulation runs significantly faster than reality, so we repeat the same run 100 times to sample from the distribution of endpoints.

It can be seen from Figure 3 that the simulated predictions of the real robot's non-faulty behaviour are not perfect, and the difference between the classes increases over time. This drift occurs due to the reality gap — specifically due to imperfect robot-environment correspondence resulting from the simplified model of the IR sensors.

C. Behaviour Classification

Distinguishing between non-faulty and faulty real robot behaviour is essentially a classification problem. It would be possible to simulate multiple different classes of fault, and train a classifier to detect them. However, this would require a priori knowledge of all possible failure modes. Instead, we make the assumption that only non-faulty behaviour is known from the robot controller, and that any significant deviation from this should indicate the presence of a fault. If a fault is so subtle that it is indistinguishable from the non-faulty class, then it is assumed to be benign and not worth detecting. This approach is an example of one-class anomaly detection [11].

We classify the real robot's position based on a simple uniform distance threshold from the mean of the predicted distribution. This implicitly defines a circular 2D spatial decision boundary. Any test point within this region will be classified as non-faulty, and any point outside it as faulty. From Figure 3 we can see that the classes could be modelled better using an ellipse, especially after a longer time period, but a circular boundary is sufficient for this initial work.

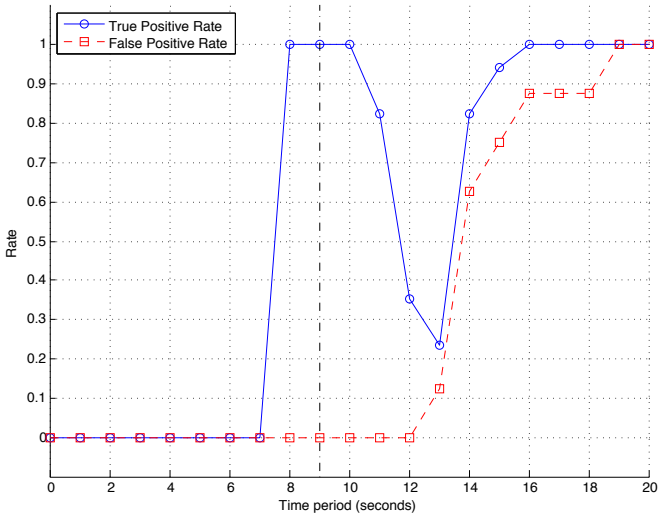

Fig. 4. TPR and FPR vs time from one particular initial pose. The robot detects the arena wall 9 seconds after initialisation.

Table I enumerates the four possible outcomes of the classification. For any time period after initialisation, we can classify each real robot endpoint shown in Figure 3 based on its distance from the mean of the predicted distribution. The classification outcomes are aggregated to produce the total number of TPs, FNs, FPs, and TNs at a specific time. We can then calculate the True Positive Rate and False Positive Rate using the following equations.

$$\text{True Positive Rate (TPR)} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{1}$$

$$\text{False Positive Rate (FPR)} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{2}$$

Figure 4 shows how the TPR and FPR vary over time when the robot starts in the same initial pose as in Figure 3. Initially the robot behaves as if it is traversing unbounded space, because the arena walls are beyond the range of its IR sensors. Immediately after initialisation, faulty behaviour is indistinguishable from non-faulty behaviour so the TPR is 0, but quickly increases to 1 after 7 seconds as the classes separate and the faulty class moves outside the decision boundary. If the robot were actually traversing unbounded space, the TPR would remain at 1 because the classes would simply separate further with time. The FPR would also remain at 0 for a long time. This is because the simulator is able to predict the straight line movement of the real robot accurately, so drift is minimal and the non-faulty class would remain contained within the decision boundary.

The non-faulty robot starts to detect the arena wall after approximately 9 seconds, and it turns away. This causes

its trajectory to intersect that of the faulty robot, and the once separable classes begin to overlap again. The result is a temporary drop in the TPR due to an increase in the number of FNs. This short drop in the TPR is relatively benign, as it only briefly reduces the likelihood of detecting a fault when the robot is near the arena wall. If the fault is persistent, then it will be detected once the robot starts moving away from the wall and the TPR recovers as the classes diverge again.

Of greater concern, is the effect of drift on the FPR over time. Figure 4 shows that shortly after the robot detects the wall, the FPR beings to increase. This is caused by increasing drift between the simulation and reality, which results in the non-faulty class moving outside the decision boundary that encircles the predicted class, and a rise in FPs. We would like to minimise FPs, as they may result in action being taken against a non-faulty robot mistakenly suspected of being faulty, which could be costly.

It is important to note that this increase in the FPR would not occur in unbounded space, and is due to poor robot-environment correspondence. The effect could be reduced by improving the simulated model of robot behaviour, but there will always be some amount of drift, because a simulation cannot hope to model the complexity of the real world in its entirety. Therefore, it is necessary to periodically reinitialise the simulation after a certain time period, to keep drift at a manageable level.

D. Reinitialisation Time Period

Selecting an appropriate reinitialisation time period is non-trivial. A long time period allows the non-faulty and faulty classes to separate, making them easier to differentiate, and improves our ability to detect minor faults. Unfortunately, a long time period increases the likelihood of the robot encountering an obstacle before reinitialisation, and therefore runs the risk of increased drift and potentially FPs.

A shorter time period is desirable because it minimises drift, and reduces the latency of fault detection, but may only allow us to detect major faults. Clearly, selection of the reinitialisation time period must trade-off multiple objectives. Furthermore, the optimal time period for the motor fault considered in this paper may not be optimal for another class of fault. Our focus here is not on finding the optimal time period, but rather to illustrate that a trade-off exists.

We suggest that the time period should be chosen under the assumption that the robot is always traversing unbounded space. From Figure 4 we can see that after a time period of 8 seconds it should be possible to reliably detect the fault. However, there is little benefit in using a time period much longer than this, as the classes are already be separable.

IV. Fault Detection at Runtime

In the previous analysis, the TPR and FPR were calculated by classifying endpoints generated from repeated non-faulty and faulty real robot runs from the same initial position. When performing fault detection at run-time, these distributions of data are unavailable. Instead, the simulation is initialised using tracking data and then predicts non-faulty behaviour

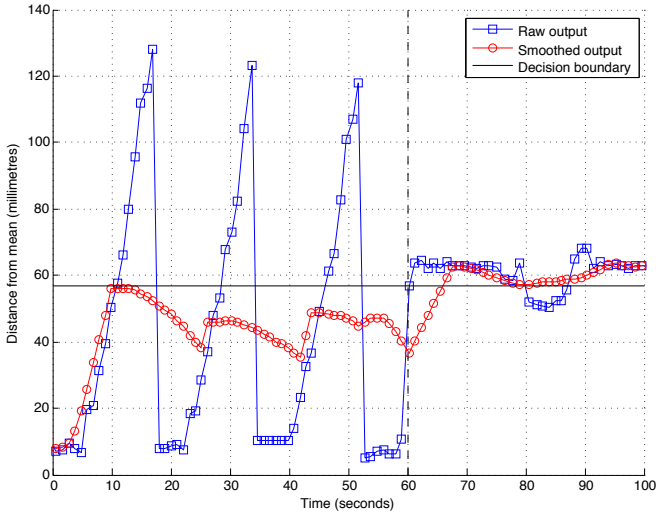

Fig. 5. Raw and smoothed classifier output (distance between the real robot endpoint and the predicted mean) over time. The reinitialisation time period is 10 seconds. A fault is injected after 60 seconds.

over the specified time period. The real robot's observed position after the same time period (now a single test point, rather than a distribution) is then classified according to its distance from the predicted mean. Once the classification is complete, the process repeats, and the simulation is initialised with new tracking data. The correctness of the classification is highly dependent on the robot's initial pose. Figure 5 shows how the distance between the real robot's endpoint and the predicted mean varies with time, with a reinitialisation time period of 10 seconds. Initially, the robot is non-faulty, but a fault is injected after 60 seconds.

When the simulation is first initialised, the robot does not encounter the arena wall within the 10 second time period, so drift is minimal. As it approaches the wall, drift begins to increase because the robot interacts with the wall before the simulation is reinitialised. The level of drift peaks when the simulation is initialised at the point where the robot first detects the wall, as this maximises the amount of post-wall drift that can occur within the time period. As soon as the robot's initial pose advances past the interaction with the wall, drift immediately drops back to minimal levels, as the robot is essentially moving through unbounded space again.

After the fault is injected, the robot's baseline distance from the predicted mean increases. This appears more stable because the circular arena causes the faulty robot's curved trajectory to remain at more consistent distance from the predicted mean. The brief drop after 80 seconds is caused by an overlap in the classes when the robot reaches the wall. If it were not for the spikes in the classifier output caused by drift when the robot is non-faulty, the non-faulty and faulty classes could be quite easily differentiated. For example, a decision boundary at 40mm would maximise the TPR, but the spikes in the output would result in many FPs.

Christensen et al. [12] demonstrated that by thresholding a moving average of the output of a fault detector, the number of FPs could be reduced by filtering out spikes

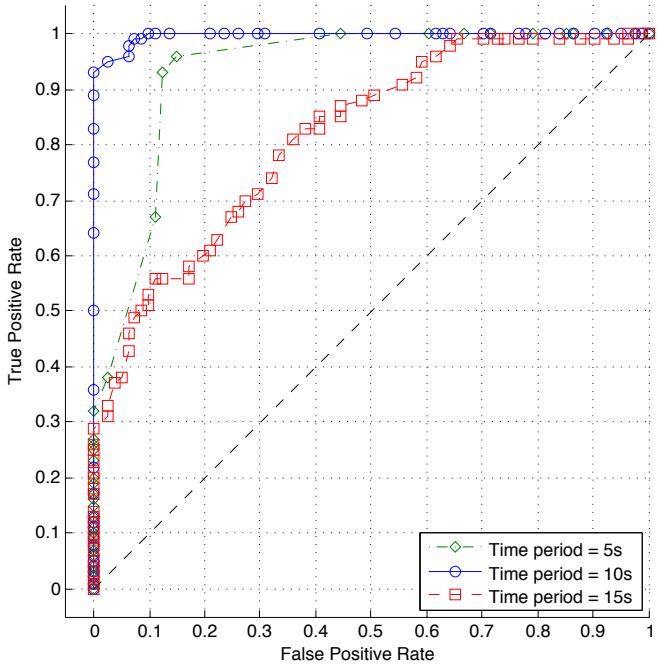

Fig. 6. ROC curves for reinitialisation time periods of 5 seconds, 10 seconds, and 15 seconds. The analysis is performed on the smoothed output of the classifier.

in the data. We have applied the same technique to the raw classifier output to produce the smoothed output shown in Figure 5. This allows the fault detector to ignore brief anomalies in the data, such that it only detects persistently faulty behaviour. The smoothing not only helps to prevent increases in the FPR, but also decreases in the TPR. This is because short drops in the classifier output caused by overlapping classes are smoothed out when the robot is persistently faulty. With smoothing applied, the classes can now be differentiated using the decision boundary shown in Figure 5. However, note that this approach increases the latency of fault detection, and may prevent some intermittent faults from being detected.

A. ROC Analysis

Finally, Receiver Operating Characteristics (ROC) analysis was carried out to assess the performance of the classifier. This was achieved by calculating the TPR and FPR for a range of decision boundary sizes. ROC curves for three different time periods are shown in Figure 6. Each point on a curve represents a particular TPR/FPR trade-off produced by some decision boundary size. The diagonal line represents the performance of a random classifier.

When a reinitialisation time period of 5 seconds is used, the classifier's performance is clearly much better than a random classifier. However, the time period is too short as it does not allow the non-faulty and faulty classes to fully separate before reinitialisation. Using a time period of 10 seconds produces almost perfect results for this particular case study, as shown by the larger area under the curve. The decision boundary that gives the highest TPR with no FPs is a threshold of 57mm from the mean, as shown in Figure 5. A time period of 15 seconds causes the classifier to perform much worse. This is because the longer time period results in a large amount of drift, causing an increase in the FPR.

V. Conclusions & Future Work

To our knowledge, this paper represents the first application of simulation to predicting robot behaviour for the detection of faults in autonomous mobile robots. We have shown that the main issue with such an approach is the selection of an appropriate reinitialisation time period. This is a non-trivial consideration, as a trade-off exists between minimising drift caused by the reality gap, and detecting faulty behaviour.

The experimental results show that there exists an optimal time period that provides the best trade-off characteristic between the TPR and FPR for the partial motor failure used as a case study. However, this may not be the optimal time period for detecting other failure modes. Although this work constitutes only a proof of principle, it is conjectured that similar issues will exist in more complex scenarios.

In future work we intend to extend this approach to implement exogenous fault detection in robot swarms as proposed in [4], with the aim of furthering the development of fault tolerant swarm systems.

References

[1] J. D. Bjerknes and A. F. T. Winfield, "On Fault Tolerance and Scalability of Swarm Robotic Systems," in *Distributed Autonomous Robotic Systems*, ser. Springer Tracts in Advanced Robotics, 2013, vol. 83, pp. 431–444.

[2] A. L. Christensen, R. O'Grady, and M. Dorigo, "From Fireflies to Fault-Tolerant Swarms of Robots," *Evolutionary Computation, IEEE Transactions on*, vol. 13, no. 4, pp. 754–766, 2009.

[3] A. F. T. Winfield and J. Nembrini, "Safety in numbers: fault-tolerance in robot swarms," *International Journal of Modelling, Identification and Control*, vol. 1, no. 1, pp. 30–37, 2006.

[4] A. G. Millard, J. Timmis, and A. F. T. Winfield, "Towards Exogenous Fault Detection in Swarm Robotic Systems," in *Proceedings of the 14th Annual Conference Towards Autonomous Robotic Systems (TAROS)*, 2013, to appear.

[5] R. Isermann, "Model-based fault-detection and diagnosis – status and applications," *Annual Reviews in Control*, vol. 29, no. 1, pp. 71–85, 2005.

[6] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *Advances in Artificial Life*. Springer, 1995, pp. 704–720.

[7] P. O'Dowd, A. F. T. Winfield, and M. Studley, "The distributed co-evolution of an embodied simulator and controller for swarm robot behaviours," in *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, 2011, pp. 4995–5000.

[8] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli, "The e-puck, a Robot Designed for Education in Engineering," in *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, 2009, pp. 59–65.

[9] W. Liu and A. F. T. Winfield, "Open-hardware e-puck Linux extension board for experimental swarm robotics research," *Microprocessors and Microsystems*, vol. 35, no. 1, pp. 60 – 67, 2011.

[10] M. Quinn, L. Smith, G. Mayley, and P. Husbands, "Evolving Formation Movement for a Homogeneous Multi-Robot System: Teamwork and Role-Allocation with Real Robots," 2002, Cognitive Science Research Paper 515, University of Sussex, UK.

[11] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[12] A. L. Christensen, R. O'Grady, M. Birattari, and M. Dorigo, "Automatic Synthesis of Fault Detection Modules for Mobile Robots," in *Adaptive Hardware and Systems, 2007. AHS 2007. Second ASA/ESA Conference on*, 2007, pp. 693–700.