Exogenous Fault Detection in a Collective Robotic Task

Anders Lyhne Christensen, Rehan O'Grady, Mauro Birattari and Marco Dorigo

IRIDIA, CoDE, Université Libre de Bruxelles
50, Av. Franklin Roosevelt, CP 194/6
1050 Brussels, Belgium
alyhne@iridia.ulb.ac.be,{rogrady,mbiro,mdorigo}@ulb.ac.be

Abstract. In robotics, exogenous fault detection is the process through which one robot detects faults that occur in other, physically separate robots. In this paper, we study exogenous fault detection in a collective leader-follower task for autonomous robots. We record sensory inputs from the robots while they are operating normally and after simulated faults have been injected. Given that faults are simulated, we can correlate the flow of sensory inputs with the fault state of the robots. We use back-propagation neural networks to synthesize fault detection components. We show that the flow of sensory inputs is sufficient information for performing exogenous fault detection, that is, we show that the leader robot is capable of detecting faults in the follower robot. All results are based on experiments with real robots.

1 Introduction

Some faults are hard to detect in the robot in which they occur. These faults include software bugs that cause a robot to hang, sensor failures that prevent a robot from detecting that something is wrong, and mechanical faults such as a loose connection to a battery. In this study, we show how fault injection and learning can be applied in order to synthesize software components for exogenous fault detection in autonomous robots. We present a concrete method for obtaining components that let one robot detect faults in collaborating robots. The ability to perform exogenous fault detection can improve the reliability of multi-robot systems. If one of the constituent robots fails, other robots in the system can take corrective actions even if the failed robot is unable to detect or communicate that it has experienced a fault.

The method relies on recording sensory data, firstly over a period of time when the robots are operating as intended, and secondly over a period of time when simulated hardware faults have been injected. After the data collection phase, fault detection components in the form of back-propagation neural networks are synthesized through supervised learning. The method requires no special fault detection hardware and relatively few computational resources on the robots. The work presented in this paper is an extension of previous studies

in which we have shown that the proposed method can be used to synthesize endogenous fault detection components for autonomous robots [1].

Technically, a fault is an unexpected change in system function which hampers or disturbs normal operation, causing unacceptable deterioration in performance. Fault detection is a binary decision process confirming whether or not a fault has occurred in a system. Fault detection is a single facet of the larger objective of ensuring fault tolerance. Other aspects of achieving fault tolerance for a system might include *fault diagnosis*, namely determining the type and location of faults, and *protection* which comprises any steps necessary to ensure continued safe operation of the system [2].

2 Related Work

Fault detection is based on observations of a system's behavior. Deviations from normal behavior can be interpreted as symptoms of a fault in the system. Several model-based approaches have been proposed [2, 3]. In model-based fault detection some model of the system or of how it is supposed to behave is constructed. The actual behavior is then compared to the predicted behavior and deviations can be interpreted as symptoms of faults. A deviation is called a *residual*, that is, the difference between the estimated and the observed value. As uncertainty and noise play a significant role, techniques such as artificial neural networks and radial basis function networks for residuals-based fault detection have been proposed [4–6].

Alternative approaches relying on multi-model estimation have been studied, see for instance [7, 8]. In these studies, banks of Kalman filters are applied to track multiple models with embedded fault states. Recently, computationally efficient approaches for approximating Bayesian belief using *particle filters*[1] have been suggested as a means for fault detection and identification [9–11].

Systems of multiple collaborating robots have the potential to achieve a high degree of fault tolerance. If one robot in such a system fails while performing a task, another one can take over and complete the task. In some cases, fault tolerance is an inherent property of the system and not handled explicitly. Lewis and Tan [12] have, for instance, shown that their control algorithm for maintaining geometric formations exhibits correct behavior even if one of the robots fails. However, the fault tolerance is a consequence of the adaptive nature of their controller design (robots attempt to maintain a virtual structure and do not let any robots fall behind) and not of explicit fault detection, diagnosis and protection. Implicit fault tolerance by design is, however, not generally feasible. In most tasks, faults must be detected and handled explicitly for the system to be fault tolerant.

Parker [13] has demonstrated that cooperating teams of robots based on the ALLIANCE software architecture can achieve a high degree of fault tolerance. Fault tolerance is obtained by modelling "motivations" mathematically and by

[1] Particle filters are Monte Carlo methods capable of tracking hybrid state spaces of continuous noisy sensor data and discrete operation states.

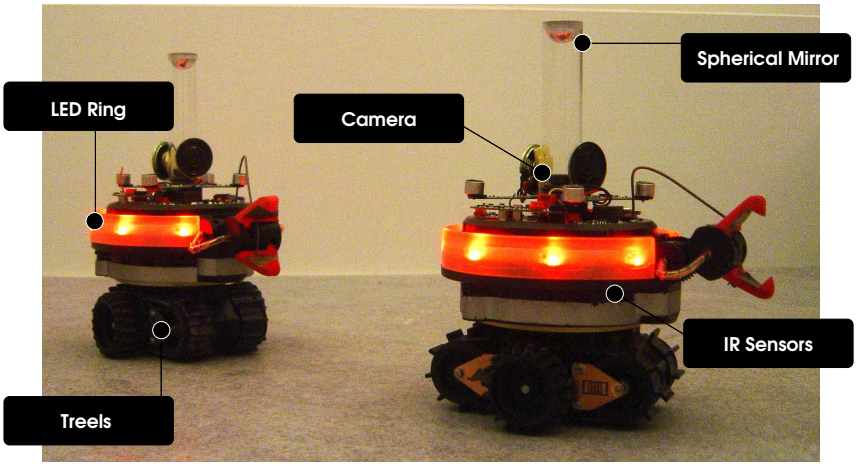

Fig. 1: Two *s-bots*, sensors and actuators. An *s-bot* has a diameter of 120 mm, a height of 190 mm, and weighs approximately 700 g.

adaptive task selection based on these motivations. If a robot experiences a fault in one or more of its components and if the fault degrades performance, the robot's motivation for performing its current task decreases. Eventually, the robot will switch to another task that it may still be able to perform. Alternatively, another robot will discover that there is limited or no progress in the task undertaken by the failed robot, and take over.

Other approaches, such as MURDOCH [14, 15] and TraderBots [16], use explicit communication and negotiation of task allocation. In these cases fault detection and tolerance are built into the negotiation process.

In this paper, we address the issue of how a robot can determine if a collaborating robot is not operating correctly. Exogenous fault detection is only based on sensory inputs. There is no explicit communication, nor is there any pre-specified behavior used to aid fault detection. We take techniques that we have successfully used in the past for endogenous fault detection and apply them to exogenous fault detection [1]. We apply a technique known as *software implemented fault injection* (SWIFI) used in dependable systems research. The technique is usually applied to measure the robustness and fault tolerance of software systems [17]. In our case, we inject faults to discover how sensory readings and the control signals sent to the actuators by the control program change when faults occur.

3 The Robots and The Task

For the experiments, we use real robots known as *s-bots* [18] (see Fig. 1). The *s-bot* platform has been used for several studies in swarm intelligence and collective

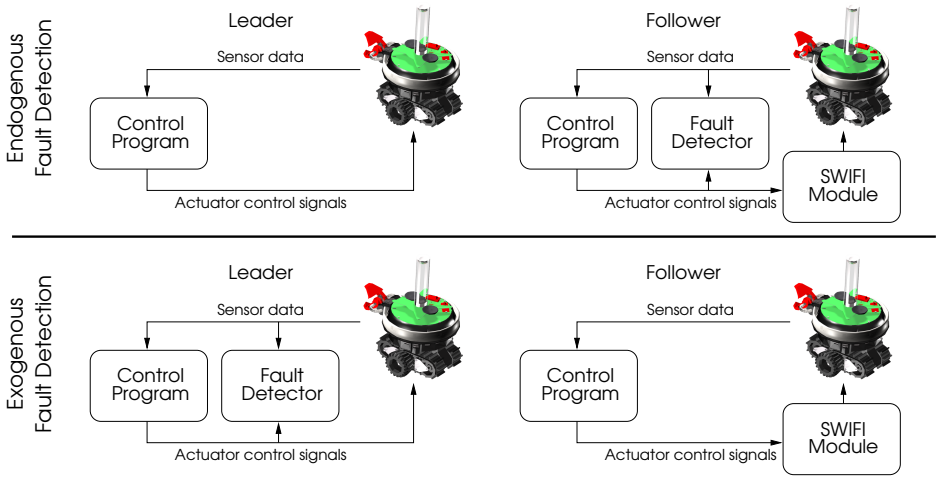

Fig. 2: The software architecture.

robotics [19–21]. Each *s-bot* is equipped with an Xscale CPU running at 400 MHz and several sensors including an omni-directional camera and infrared proximity sensors. Each *s-bot* also has a number of actuators. These include differential *treels* (combined tracks and wheels) and 8 sets of RGB colored LEDs distributed around the circumference of the *s-bot* body. Using the omni-directional camera, one *s-bot* can see the colored LEDs on other *s-bots* up to 50 cm away depending on the light conditions.

We have chosen a simple *follow the leader* task in which two robots are placed in a 180 cm by 180 cm walled arena. One of the robots has been preassigned the *leader* role, while the other has been preassigned the *follower* role. The *leader* moves around in the environment. The *follower* tails the *leader* and tries to stay at a distance of 35 cm. If the *follower* falls behind, the *leader* waits. During experiments, we inject faults in the *follower* robot.

4 Software Architecture

An overview of the software architectures is shown in Fig. 2. The *Control Programs* are responsible for steering the robots. They read sensory inputs and send control signals to the robots' actuators. The *Fault Detectors* passively monitor the flow of sensory inputs and control signals that passes to and from the *Control Programs*. Faults are simulated by the *SWIFI Module* in the *follower*. When the *follower*'s *Control Program* sends actuator control signals, these commands pass through the *SWIFI Module*. If no fault is currently being simulated the *SWIFI Module* forwards all actuator control signals to the robot hardware. If a fault has been injected, control signals to the hardware affected by the fault are discarded.

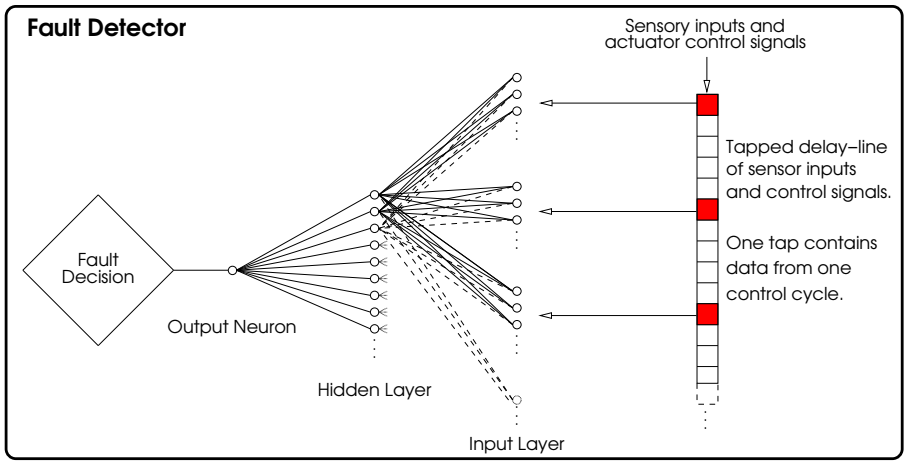

Fig. 3: An overview of the fault detection component, consisting of an artificial neural network and a tapped delay-line of observations. The input neurons in the neural network are logically organized into groups. In each group, the neurons encode observations from one tap.

The *Fault Detector* component consists of a time-delay artificial neural network (TDNN). A TDNN is a feed-forward network that allows for reasoning based on time-varying inputs without the use of recurrent connections [22, 23]. In a TDNN, the input layer is logically organized into a number of groups. In each group, the activations of the neurons are set based on observations from a fixed distance into the past. TDNNs have been extensively used for time-series prediction due to their ability to make predictions based on data distributed in time. The TDNNs used in this study are normal multilayer perceptrons for which the inputs are taken from multiple, equally spaced points in a delay-line of past observations. Fig. 3 illustrates this concept. The current sensor inputs and control signals are stored in a tapped delay-line, and the activations of the neurons in the input layer are set based on data from the delay-line. The neurons from all input groups are fully connected to the neurons in the hidden layer. The term *input group distance* refers to the distance in time between adjacent input groups. In this study, we use an input group distance of 5 control cycles (as illustrated in Fig. 3), and TDNNs with hidden layer of 5 neurons and an input layer with a total of 10 input groups. Each input group consists of 15 neurons corresponding to the 15 infrared proximity sensors and 16 neurons encoding data extracted from the omni-directional camera. Images from the camera are partitioned into 16 sections and each of the 16 neurons are assigned values inversely proportional to the distance of the closest object perceived in the corresponding section. The image processor has been configured to detect colored LEDs. This configuration means that the camera only detects other *s-bots* and not objects like walls.

5 Experimental Setup

A total of 60 runs on real *s-bots* are performed. In each run, the robots start in perfect condition, and at some point during the run a fault is injected in the *follower*. The fault is injected at a random point in time after the first 5 seconds of the run and before the final 5 seconds of the run according to a uniform distribution. There is a 50% probability that a fault affects both treels instead of only one of the treels, and faults of the type *stuck-at-zero* and *stuck-at-constant* are equally likely to occur. A fault of the type *stuck-at-zero* means that the affected treel effectively blocks and stops to move. A *stuck-at-constant* fault means that the speed of the motor controlling a treel is set to a random value and the motor ceases to respond to control signals sent from the control program.

Each run consists of 1000 control cycles (equivalent to 150 seconds) and for each control cycle the sensory inputs, control signals, and the current fault state are recorded. The data sets are partitioned into two subsets, one consisting of data from 40 runs, which is used for training; and one consisting of the data from the remaining 20 runs, which is used for performance evaluation. After a network has been trained by a batch learning back-propagation algorithm, it is evaluated on data from one evaluation run at a time. The output of the network is recorded and compared to the fault state.

Fault detection is a binary classification problem and, in this study, we present results for threshold-based classification. If the TDNN outputs a value lower than the threshold we interpret it as a *no fault* classification, whereas we interpret outputs equal to or above the threshold as a *fault* classification. The interpreted output is compared to the correct output and fault detectors are scored on their *latency* and the number of *false positives* they produce.

Latency refers to the span of time from the moment a fault is injected until it is detected. False positives refers to the number of control cycles for which a fault detector is wrongly classifying the state as a *fault*. For this study, we have chosen the threshold to be 0.75. If we were to choose a lower threshold, we would expect a lower latency but more false positives. Similarly, if we were to choose a higher threshold we would expect a higher latency and fewer false positives.

6 Results

Box-plots of the latency and the false positive results from the 20 evaluation runs are shown in Fig. 4. Each box comprises observations ranging from the first to the third quartile. The median is indicated by a bar, dividing the box into the upper and lower part. The whiskers extend to the farthest data points that are within 1.5 times the interquartile range. Outliers are shown as dots. Each sample point corresponds to the results of a single evaluation run. In the figure we have plotted results for the *follower* performing endogenous fault detection and for the *leader* performing exogenous fault detection during the same runs. The median latency for the *follower* performing endogenous fault detection is

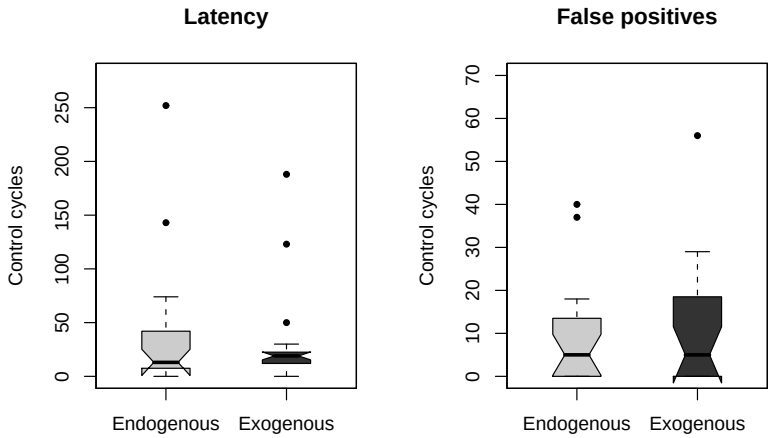

Fig. 4: Box-plots of the performance results using threshold-based classification for the *follower* performing endogenous fault detection and for the *leader* performing exogenous fault detection during the same runs.

14 control cycles, while the median latency for the *leader* performing exogenous fault detection is 19 control cycles. This difference of 5 control cycles corresponds to 750 ms. The median number of false positives is 5 control cycles for both the endogenous and exogenous detection. In every trial, the fault injected was detected.

The results indicate that the *leader* robot is capable of detecting faults injected in the *follower*. Furthermore, the performance of the exogenous fault detection is comparable to the performance of the endogenous fault detection performed by the *follower*. The latency is, however, slightly higher for exogenous fault detection.

In order to obtain a fault detector that produces fewer false positives, we implemented a threshold-based classification scheme based on the moving average of the TDNN output value. Hence, instead of interpreting the output of the TDNN directly, as above, a number of past values are stored, and the fault classification is based on the average of those values. Fig. 5 shows the false positive results for the different lengths of the moving average window. For moving average windows up to 10 control cycles, false positives occur in several trials. For longer windows, false positives are only observed in one or two trials. When a moving average window of length 50 is used, the exogenous fault detector produces no false positives.

For window lengths of 20-50, false positives for the *follower* performing endogenous fault detection occurred in one of the 20 trials. The endogenous fault

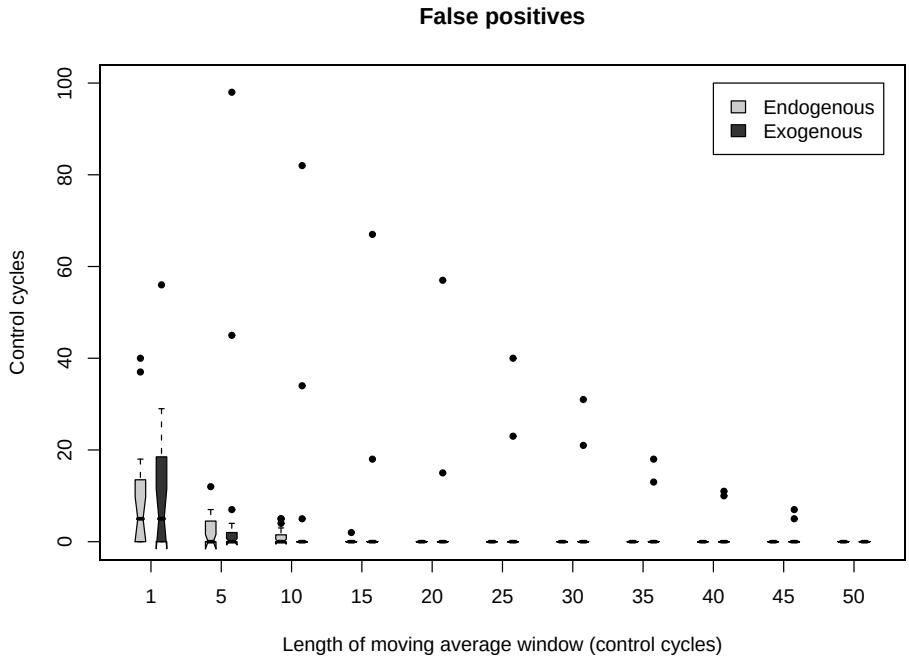

Fig. 5: False positives results for moving average threshold-based classification.

detector produced 173 false positives with a window length of 20 control cycles and 128 false positives with a window length of 50 cycles for the trial in question. These results are not shown in Fig. 5, since they are outside the scale of the figure.

When the moving average is used, the latencies are increased by the length of the moving average window (results are not shown). Thus, introducing a threshold-based classification scheme based on the moving average of the TDNN output value can remove nearly all false positives, but this comes at the cost of a higher latency.

7 Conclusion

In this study, we synthesized fault detection components for exogenous faults using fault injection and supervised learning. Our synthesizing technique can be used for both endogenous and exogenous fault detection. We achieved this transparency between endogenous and exogenous faults by avoiding the use of any explicit modelling. The method's only requirement is that a fault must create a detectable change in the detecting robot's flow of sensory input. Our method

proved effective on real robots; one robot was able to detect faults injected in another robot without the use of explicit communication or any specific behavior written to aid fault detection.

Many faults cannot be detected endogenously. We therefore believe that the use of exogenous fault detection in combination with endogenous fault detection has the potential to significantly enhance the reliability and robustness of multi-robot systems. We are currently investigating the scalability of the proposed approach. One consideration is how to apply exogenous fault detection to systems with larger numbers of robots. When multiple robots with limited sensory range interact, it is unclear how a set of observations from one robot should be correlated with the fault state of other robots that may or may not be within sensory range. We are also investigating how larger numbers and a broader range of faults can be detected.

Acknowledgements: This work was supported by the *SWARMANOID* project, funded by the Future and Emerging Technologies programme (IST-FET) of the European Commission, under grant IST-022888. Anders Christensen acknowledges support from COMP2SYS, a Marie Curie Early Stage Research Training Site funded by the European Community's Sixth Framework Programme (grant MEST-CT-2004-505079). The information provided is the sole responsibility of the authors and does not reflect the European Commission's opinion. The European Commission is not responsible for any use that might be made of data appearing in this publication. Marco Dorigo acknowledges support from the Belgian FNRS, of which he is a Research Director. This research was supported by the ANTS project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium.

References

1. Christensen, A.L., O'Grady, R., Birattari, M., Dorigo, M.: Automatic synthesis of fault detection modules for mobile robots. In: Proc. of the NASA/ESA Conf. on Adaptive Hardware and Systems (AHS-2007), MIT Press, Cambridge, MA (2007) In press.
2. Isermann, R., Ballé, P.: Trends in the application of model-based fault detection and diagnosis of technical processes. Control Engineering Practice **5**(5) (1997) 709–719
3. Gertler, J.J.: Survey of model-based failure detection and isolation in complex plants. IEEE Control Systems Magazine **8** (1988) 3–11
4. Vemuri, A., Polycarpou, M.: Neural-network-based robust fault diagnosis in robotic systems. IEEE Trans. on Neural Networks **8**(6) (1997) 1410–1420
5. Terra, M., Tinos, R.: Fault detection and isolation in robotic manipulators via neural networks: A comparison among three architectures for residual analysis. Journal of Robotic Systems **18**(7) (2001) 357–374
6. Patton, R., Uppal, F., Lopez-Toribio, C.: Soft computing approaches to fault diagnosis for dynamic systems: A survey. In: Proc. of 4th IFAC Symposium on Fault Detection supervision and Safety for Technical Processes. Volume 1., Elsevier, Oxford, UK (2000) 298–311

7. Roumeliotis, S., Sukhatme, G., Bekey, G.: Sensor fault detection and identification in a mobile robot. In: Proc. of IEEE/RSJ Inter. Conf. on Intelligent Robots and Systems. Volume 3., IEEE Press, Los Alamitos, CA (1998) 1383–1388
8. Goel, P., Dedeoglu, G., Roumeliotis, S., Sukhatme, G.: Fault detection and identification in a mobile robot using multiple model estimation and neural network. In: Proc. of IEEE Inter. Conf. on Robotics and Automation, ICRA'00. Volume 3., IEEE Computer Society Press, Los Alamitos, CA (2000) 2302–2309
9. Dearden, R., Hutter, F., Simmons, R., Thrun, S., Verma, V., Willeke, T.: Real-time fault detection and situational awareness for rovers: Report on the Mars technology program task. In: Proc. of IEEE Aerospace Conf. Volume 2., IEEE Computer Society Press, Los Alamitos, CA (2004) 826–840
10. Verma, V., Gordon, G., Simmons, R., Thrun, S.: Real-time fault diagnosis. IEEE Robotics & Automation Magazine **11**(2) (2004) 56–66
11. Li, P., Kadirkamanathan, V.: Particle filtering based likelihood ratio approach to fault diagnosis in nonlinear stochastic systems. IEEE Trans. on Systems, Man and Cybernetics, Part C **31**(3) (2001) 337–343
12. Lewis, M.A., Tan, K.H.: High precision formation control of mobile robots using virtual structures. Autonomous Robots **4**(4) (1997) 387–403
13. Parker, L.E.: ALLIANCE: an architecture for fault tolerant multirobot cooperation. IEEE Trans. on Robotics and Automation **14**(2) (1998) 220–240
14. Gerkey, B.P., Matarić, M.J.: Sold!: auction methods for multirobot coordination. IEEE Trans. on Robotics and Automation **18**(5) (2002) 758–768
15. Gerkey, B., Matarić, M.J.: Pusher-watcher: An approach to fault-tolerant tightly-coupled robot coordination. In: Proc. of IEEE Inter. Conf. on Robotics and Automation, ICRA'02, IEEE Press, Piscataway, NJ (2002) 464–469
16. Dias, M.B., Zinck, M.B., Zlot, R.M., Stentz, A.: Robust multirobot coordination in dynamic environments. In: Proc. of IEEE Inter. Conf. on Robotics and Automation, ICRA'04. Volume 4., IEEE Press, Piscataway, NJ (2004) 3435–3442
17. Arlat, J., Aguera, M., Amat, L., Crouzet, Y., Fabre, J., Laprie, J., Martins, E., Powell, D.: Fault injection for dependability validation: A methodology and some applications. IEEE Trans. on Software Engineering **16**(2) (1990) 166–182
18. Mondada, F., Gambardella, L.M., Floreano, D., Nolfi, S., Deneubourg, J.L., Dorigo, M.: The cooperation of swarm-bots: Physical interactions in collective robotics. IEEE Robotics & Automation Magazine **12**(2) (2005) 21–28
19. Groß, R., Bonani, M., Mondada, F., Dorigo, M.: Autonomous self-assembly in swarm-bots. IEEE Trans. on Robotics **22**(6) (2006) 1115–1130
20. O'Grady, R., Groß, R., Mondada, F., Bonani, M., Dorigo, M.: Self-assembly on demand in a group of physical autonomous mobile robots navigating rough terrain. In: Advances in Artificial Life: 8th European Conf., ECAL 2005. Proc. Volume 3630 of LNAI., Springer Verlag, Berlin, Germany (2005) 272–281
21. Dorigo, M., Trianni, V., Şahin, E., Groß, R., Labella, T.H., Baldassarre, G., Nolfi, S., Deneubourg, J.L., Mondada, F., Floreano, D., Gambardella, L.M.: Evolving self-organizing behaviors for a swarm-bot. Autonomous Robots **17**(2–3) (2004) 223–245
22. Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., Lang, K.: Phoneme recognition using time-delay neural networks. IEEE Trans. on Acoustics, Speech, and Signal Processing **37** (1989) 328–339
23. Clouse, D., Giles, C., Horne, B., Cottrell, G.: Time-delay neural networks: Representation and induction of finite-state machines. IEEE Trans. on Neural Networks **8** (1997) 1065–1070