Proc. of IEEE International Conference on Robotics and Automation, Rome, Italy, 2007.

Sensor Analysis for Fault Detection in Tightly-Coupled Multi-Robot Team Tasks

Xingyan Li and Lynne E. Parker
Distributed Intelligence Laboratory, Department of Computer Science
The University of Tennessee, Knoxville, TN 37996-3450
Email: {li, parker}@cs.utk.edu

***Abstract*— This paper presents a sensor analysis based fault detection approach (which we call SAFDetection) that is used to monitor tightly-coupled multi-robot team tasks. Our approach aims at detecting both physical and logic faults of a robot system with little prior knowledge on the system. We do not need the motion model or a priori knowledge of the possible fault types of the monitored system. Our approach treats the monitored robot system as a black box, with only sensor data available. Thus, we believe the approach is general, and can be used in a wide variety of robot systems performing many different kinds of tasks. Our approach combines data clustering techniques with the generation of a probabilistic state diagram to model the normal operation of the multi-robot system. We have implemented this approach on a physical robot team. This paper presents the results of these experiments, which show that sensor data analyzed from a training phase of normal operation can be used to generate a model of normal robot team operation. This model can then be used to detect many types of abnormal behavior of the system, based purely on monitoring the sensor data of the system.**

I. Introduction

Due to the increasing needs for robotic applications in various challenging tasks, the demands for highly reliable robot systems are significantly growing. We know that even the most carefully designed and tested robots may behave abnormally in some situations; therefore, it is necessary for robots to monitor their performance so that the deviation from the expected behavior can be promptly detected. Here, we define a robot *fault* as a deviation from the expected behavior of the robot system, and *fault detection* as the process of automatically determining that a fault has occurred. The reasons for faults vary from mechanical degradation to information insufficiency, and may be due to a large spectrum of components, including robot sensors, actuators and system components. Developing fault models for every possible range of faults and components is a daunting task. Instead, a preferred approach is to build a fault detection capability that is independent of the particular type of fault that occurs. This paper presents such an approach for fault detection that is based only on the monitoring of robot sensor values. Once a fault is detected, it can be used by a higher-level diagnosis and recovery strategy, such as we are developing in [15]. This paper focuses on the initial detection of the fault.

A common approach to detecting faults in robots systems is to make use of a motion model of the robot. In this case, fault detection is relatively straightforward – if the values estimated by the model deviate significantly from the actual measurements, a fault must have occurred. However, in many applications, a motion model of the robot may not be available, or the information we have is only a very rough approximation of how the robot operates. This paper addresses the question of how to detect robot faults in such situations when a motion model is unavailable. Our approach, which we call SAFDetection ("Sensor Analysis for Fault Detection"), is to learn normal robot state transitions from a history of sensor data during the normal operational mode of the robot. Faults are then identified online via a deviation of the sensor data from the model of normal operation. Our approach currently identifies three types of sensor faults: "hard" fault, "logic" fault and "coalition" fault. The hard fault reflects an abnormal robot state in which the sensor data does not match any previously seen before. The logic fault reflects an abnormal state transition, in which a robot is performing an unlikely transition from previously modeled states. Finally, the coalition fault reflects conflicts among team robots, in which the sensor model expectations of each robot differ.

The remainder of the paper is organized as follows. We present related work in Section II. An overview of the overall approach is given in Section III, followed by details of the approach in Section IV. In Section V, we present results of using this approach in a physical robot implementation of a box pushing experiment. Conclusions are discussed in Section VI.

II. Related work

Fault detection for robots is a complex problem, for a number of reasons: the space of possible faults is very large; robot sensors, actuators, and environment models are uncertain; and there is limited computation time and power. Nevertheless, because of its importance, much prior work has been done in this area.

The most popular method for providing fault detection in robot systems is based on motion control [8], [18], [12]. This method compares the values estimated by the motion model and the current measurements to detect a fault. For example, in the Hannibal robot system [5], if the leg sensors do not agree with the set of plausible leg motions that are programmed for the leg, the robot generates a belief that the sensor is not working. This method only works, however, when the motion model of the robot is completely known.

Another widely used computer fault detection method is voting based on modular redundancy [9], [3]. This method is commonly used in highly reliable systems in which more than one module works redundantly to perform the same task given the same input data. If one of the modules is faulty and its result does not agree with the results of the other modules, the faulty module is voted out of the final decision and the correct result is passed on to the rest of the system.

Analytical redundancy is another concept for fault detection that does not need redundant modules [13], [10], [6]. By comparing the histories of sensor outputs versus the actuator inputs, results from dissimilar sensors can be compared at different times in order to check for failures.

In recent years, particle filter techniques for robot fault detection have become popular [7], [17], [2]. This method can estimate the robot and its environmental state from a sequence of noisy, partial sensor measurements. Most particle filter based fault detection methods work with a known set of possible fault types.

Unlike motion model based methods, our SAFDetection approach does not require knowledge of the internal control of the robot system. We also assume no advance knowledge of the possible fault types, unlike the partial filter based approaches. Additionally, no functionally redundant modules are required and no requirement is made for specifying the relationship between the measured variables, unlike the analytical redundancy methods. Instead, our approach treats the robot or robot team as a black box. We learn the probabilistic robot state transition diagram from the histories of robot sensor data during normal operation based on a clustering algorithm, and then use this diagram together with the on-line sensor data to detect faults in a real-time fashion.

Commonly used techniques to extract system knowledge from data include decision trees [21], artificial neural networks [16] and probabilistic networks. The Bayesian network is a popular representation of a probabilistic network [22], [4]. Matsuura [14] shows a Bayesian network based fault detection method which does not require previous information about the dynamic system. In their work, a Bayesian network is learned from a series of data acquired under normal conditions, and faults are detected as low probability values. Another fault detection method similar to our approach constructs a system behavior model in the form of a set of rules generated by applying pattern clustering and association. This approach has been used in some complex systems [20] outside of robotics. Hybrid control systems can be used to identify generic patterns of continuous and discrete event dynamical systems [1]. A generic framework for hybrid systems includes transitions between continuous and discrete states. Our approach is different from this prior work in that it learns a state transition diagram using a data clustering algorithm and is applied to the domain of multi-robot systems.

III. Overall approach

To detect faults in a mobile robot system, we need the ability to distinguish a normal robot state from an anomalous robot state. If we were to use the motion model of the robot, faults can be easily detected by comparing the estimated output of the model to the actual measurement. However, without the motion model or any idea of the expected fault signatures, we can instead learn normal robot states by analyzing the sensor data during normal robot operation. The states learned from the sensor data analysis may differ from the actual robot "inner states" (i.e., the robot's designed programming and control algorithm), since the inner states and the sensor data are not strictly mapped in a one-to-one relationship to each other. However, with sufficient sensor information, most inner states can be partitioned to meaningful sensor states that are useful for fault detection.

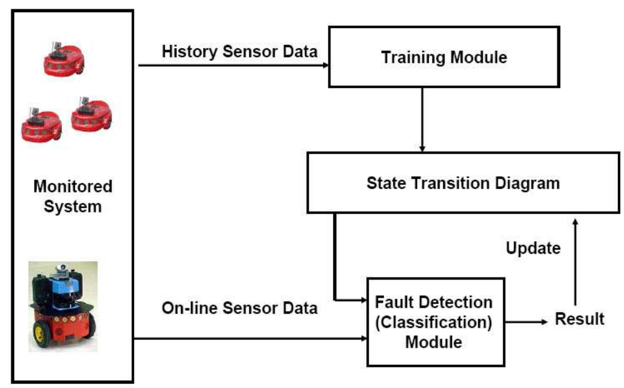

Fig. 1. The structure of SAFDetection approach

Our SAFDetection approach, as shown in Fig. 1, is a training-classification based method. In the training stage, a history of sensor data (i.e., training data) during normal operation is clustered into different states by using the Fuzzy C-Means (FCM) clustering algorithm [11]. A state transition diagram that represents the probabilistic transitions between these states is then learned from the training data. In the classification stage, the on-line sensor data is compared with the state transition model and three types of faults can be detected. If the sensor data does not belong to any of the states we learned from the normal data, a *hard fault* is detected. If the sensor data belongs to one of the states but the observed state transition deviates significantly from the state transitions we learned, a *logic fault* is detected. In a similar manner, when this approach is used in a coalescent multi-robot team, the inconsistency between robots can be detected as a *coalition fault*. If no fault is detected, the sensor data is classified as normal and is used to update the state transition diagram. Our belief is that since our SAFDetection approach regards the monitored system as a black box, it can be used on different robot systems, including single robots and team robots performing a variety of tasks.

IV. Details of Approach

A. Data preprocessing

Since the SAFDetection approach treats the monitored system as a black box, the only information obtained from the system is the robot sensor data. To obtain this data, the robot makes use of the sensors already installed on the robot for its mission-relevant tasks. Obviously, it is impractical to use all the available sensor data for training the system and learning the state transition diagram. Instead, relevant features are selected and monitored. Thus, we choose a p-dimensional essential feature vector f as the first step of our approach. The essential features are primarily determined by the tasks executed by the robot system. The correlation between different sensors must also be considered. In some cases, it is helpful to use correlated sensors, such a slaser and sonar, to provide rational redundancy. In our current work, we regard a complete multi-robot team as one monolithic robot with a unified set of sensors. (We are extending this approach in ongoing work to a distributed monitoring approach.) Therefore, the same types of sensors, when available, can be selected from different robots as essential features to provide some measure of data redundancy.

Selecting the correct components of the feature vector is a non-trivial problem, since the computational demands will be intractable if we make use of every possible feature in the system. Of course, in most cases, the computational requirements of clustering algorithms increase as the dimension p goes up. In addition, we have to deal with the curse of dimensionality, meaning that the clustering becomes less meaningful as the dimensionality of the data increases. We therefore must make use of historical sensor data to select only the most relevant features as components of the feature vector. Many methods can be used to reduce the feature dimension, including Principal Components Analysis (PCA) and Singular Value Decomposition (SVD), which are two of the common linear algebra techniques for continuous data dimensionality reduction. Ultimately, we will make use of these techniques to automatically select the best features. At present, some of this work is performed manually by the designers, based on the variance of the sensor readings during experimental trials. We also make use of the mean and standard deviation values in the online fault detection phase.

B. Data clustering

Many clustering algorithms have been developed to find groups in unlabeled data based on a similarity measurement among the data patterns. In these approaches, similar patterns are placed together in the same cluster. For this work, we use the Fuzzy C-Means clustering algorithm (FCM) [11] to cluster the sensor data into groups, which we call states. Since FCM is a fuzzy algorithm, a single data pattern may belong to several clusters, having different membership values in each cluster. This property of fuzziness is advantageous when dealing with the noisy or partial data of typical robot applications.

In the FCM algorithm, given a set of n data patterns, $X = x_1, \cdots, x_k, \cdots, x_{n,}$, the algorithm minimizes the objective function, $J(U, V)$:

$$J(U, V) = \sum_{k=1}^{n} \sum_{i=1}^{c} u_{ik}^{m} d^2(x_k, v_i) \tag{1}$$

where x_k is the k-th p-dimensional data vector, v_i is the prototype of the center of cluster i, u_{ik} is the degree of membership of x_k in the i-th cluster, m is a weighting exponent on each fuzzy membership, $d(x_k, v_i)$ is a distance measurement between data pattern x_k and cluster center v_i, n is the number of data patterns, and c is the number of clusters. The objective function $J(U, V)$ is minimized via an iterative process in which the degree of membership, u_{ik}, and the cluster centers, v_i, are updated, as follows:

$$u_{ik} = \frac{1}{1 + \sum_{j=1}^{c} (d_{ik}/d_{ij})^{\frac{2}{m-1}}} \tag{2}$$

$$v_i = \frac{\sum_{k=1}^{n} u_{ik}^{m} x_k}{\sum_{k=1}^{n} u_{ik}^{m}} \tag{3}$$

where $\forall i \; u_{ik}$ satisfies: $u_{ik} \in [0, 1]$, $\forall k \sum_{i=1}^{c} u_{ik} = 1$ and $0 < \sum_{k=1}^{n} u_{ik} < n$.

One limitation of FCM is that it requires knowledge of the number of clusters c. In our case, this number of clusters is unknown before clustering is performed. Thus, we iteratively run the FCM algorithm over several trials with varying cluster numbers and select the number of clusters that gives the best clustering quality. Xie defined a clustering quality measurement [19] to measure the overall quality of a fuzzy c-partition. This validity function, S, is given by:

$$S = \frac{\sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij} (v_i - x_j)^2}{n \times min_{ij} (v_i - v_j)^2} \tag{4}$$

Since a smaller S value means a more compact and separate c-partition, we choose the cluster number with the minimum S value as the final result. By using the validity function, we can optimally cluster the n data patterns into c states in a fuzzy way.

C. State transition diagram

Our state transition diagram is similar to a Markov model, in that it records states and the transition probabilities between each pair of states. (In ongoing work, we are dealing with hidden variables, and thus our subsequent model will be similar to a Hidden Markov Model.) In addition, our state transition diagram also includes the mean and standard deviation value of the time duration of the system in each state (i.e., before the system transits to another state).

The algorithm for building the state transition diagram is sketched in Algorithm 1. At the beginning, we need to categorize all the sensor data into different states. As a result of the fuzzy clustering algorithm, the data is not crisply labeled into only one state. Equation 2 shows that the u value is used to represent the degree of membership of the data x in different clusters. We then use the u value to assign data x into states with different probabilities. Variables t_1 and t_2 are threshold values. By arranging the labeled data in sequence, we can determine the transition between states and easily build the state transition diagram.

Algorithm 1 Sketch of building the state transition diagram

for each data x and its membership value u **do**
  find the max(u)=u_i and max (u-u_i)=u_j.
  if u_i-u_j $> t_1$ **then**
    x is set to cluster i with weight 1.
  else
    if $u_j > t_2$ **then**
      x is set to cluster i with weight u_i and set to cluster j with weight u_j.
    else
      x is invalid and discarded.
    end if
  end if
end for
for each time step **do**
  if the state is the same as the last time step **then**
    record the time length the system remains in this state.
  else
    record the state transition.
  end if
end for
for each state c **do**
  find the mean m_c and standard deviation value σ_c of the time the system remains in state c.
  find the probability p_{cd} for each possible state d.
end for

D. Fault detection

After we have built the state transition diagram, we can use it to detect faults online. The algorithm for detecting faults is sketched in Algorithm 2. The on-line sensor data x and its membership value u are sent to the fault detection module and three types of faults can be detected. If the membership u value does not show clearly which cluster x belongs to, a hard fault is detected because the robot enters an unknown state. If x belongs to state c and the robot has stayed in that state c for an unusually long time, a logic fault is detected because the robot is stuck in that state. If a state transition occurs and the observed state transition does not exist in the learned state transition diagram, a logic fault is detected because the robot has changed its state in an unknown manner. If one robot team member detects a hard or logic fault, but this fault is not detected by another team member, a coalition fault is detected indicating inconsistency between team robots (i.e., in their perception of the current team state). The following section shows how these three types of faults can be detected in a physical robot experiment.

V. Experimental results

We implemented the SAFDetection approach on a physical robot team performing a cooperative box pushing task. In this section, we first describe the task, and then the features used for sensor analysis. We present the clustering results, the state transition diagram that was learned, and the results of using this model for fault detection for both a single robot and a multi-robot team.

Algorithm 2 Sketch of fault detection module

for each on-line data x **do**
  x's membership value u is obtained from equation 2.
  if max(u) $< t_1$ **then**
    x does not belong to any known state, and a hard fault is detected.
  else
    x is set to the state c with the highest membership value.
  end if
  find the state d of the last time step.
  if c = d **then**
    calculate the time t of the duration in state c.
    if $t > m_c$+3σ_c **then**
      system is stuck in state c, and a logic fault is detected.
    end if
  else
    if p_{dc} = 0 in the state transition diagram **then**
      system performed an abnormal state transition, and a logic fault is detected.
    end if
  end if
end for

A. Box pushing task

The box pushing task has been used by many researchers as a canonical problem in multi-robot systems. In this task, two or more robots work together to push a box to a goal position in a cooperative manner. In our experiments, the goal was indicated by a red object clearly visible to the robots. During the pushing, the robot needs to adjust its speed and pushing direction to avoid losing the box; it also may need either to change the pushing direction fairly significantly, or to idle for a while to coordinate with the teammate's pushing activities. For these experiments, we made use of no additional knowledge of the motion model of the robots. We applied our approach for monitoring both a single robot and a robot team performing this pushing task. In our experiments, the box pushing task is repeated 15 times, resulting in more than 600 data entries being collected for training the SAFDetection system.

B. Feature selection

Two Pioneer mobile robots are used in our experiments. Both robots are equipped with sensory devices that include laser, sonar ring, color camera and motors with encoders. Each of these devices provides sensor data that can be monitored for our application. Clearly, many possible features could have been used for this task. We explored the usefulness of many features for this task. Some features were determined by understanding the characteristics of the the task; for example, the red block information is relevant for indicating the goal position, and thus its size in the camera image is important for the task. Some of the features were chosen to provide rational redundancy (e.g., laser and sonar). We also chose the robot battery charge value as one essential feature because it affects all robot tasks if there is no backup power supply. In our implementation, this feature is used as a check value; if it deviates significantly from the mean value that was learned, a hard fault is detected. To select from the candidate features, we ran multiple trials of the box pushing experiment. We monitored the feature values and filtered out those that are irrelevant, in order to reduce the dimensionality of the feature vector. Based on this analysis, we selected a subset of the initial features to be used in building the sensor model. The list of features explored and the eight that were selected as a result of our analysis are as follows:

- Minimum laser range (SELECTED)
- Laser index with minimum laser range (SELECTED)
- Maximum laser range
- Laser index with maximum laser range
- Minimum sonar range (SELECTED)
- Sonar index with minimum sonar range (SELECTED)
- Maximum sonar range
- Sonar index with maximum sonar range
- Robot speed (SELECTED)
- Robot turn rate (SELECTED)
- Robot current position
- Red block's center in the camera image
- Red block's area in the camera image
- Red block's width in the camera image
- Red block's height in the camera image (SELECTED)
- Battery charge (SELECTED)

TABLE I

MEAN AND STANDARD DEVIATION OF FEATURES

Feature	mean	standard deviation	std to mean
min laser range	0.132	0.027	20.45%
min laser index	90.304	9.223	10.21%
min sonar range	0.177	0.017	9.42%
min sonar index	3.424	1.089	31.83%
speed	0.056	0.048	82.14%
turn rate	0.0016	0.011	687.5%
blob height change	1.281	9.725	759.2%
battery charge	12.208	0.214	1.75%

From table I, we find that the mean values of the selected features vary significantly. To keep these values near the same magnitude in clustering, we weight them so that the resulting values lie within the same range.

C. State transition diagram result

To detect the fault of a single robot, more than 600 sensor data entries (collected from 15 trials of the experiment) are used to train the system. Five clusters are grouped using the FCM clustering algorithm and validity function. Table II shows the centers of these clusters.

The center results show that it is reasonable to consider the clusters as states that represent different status conditions of a robot. Examining the cluster center vector in the table

TABLE II

CENTERS OF CLUSTERS WITH SINGLE ROBOT DETECTION

Feature	C1	C2	C3	C4	C5
min laser range	90.11	99.73	84.85	80.43	90.95
min laser index	0.201	0.137	0.095	0.094	0.139
min sonar range	3.26	3.27	3.82	3.96	2.50
min sonar index	0.196	0.17	0.17	0.17	0.17
speed	0.001	-0.038	0.088	0.002	-0.374
turn rate	0.001	-0.112	0.002	0	0.109
blob height change	0	47.03	2.17	0.54	50.12

II, we can roughly interpret the states represented by the clusters. For example, according to the speed and the turn rate features, we know that clusters C2 and C5 correspond to "alignment" actions of the robot (with opposite turning directions). C3 means that the robot is moving forward while clusters C1 and C4 stand for the "waiting" or "idle" status of the robot.

After we obtain the states of the robot system, Algorithm 1 is used to build the state transition diagram. Fig. 2 shows the resulting transition diagram with the five states in our single robot fault detection tests. The states "start" and "end" are added manually. Table III shows the mean and standard deviation values of the time the robot remained in each state (i.e, before transiting to a different state).

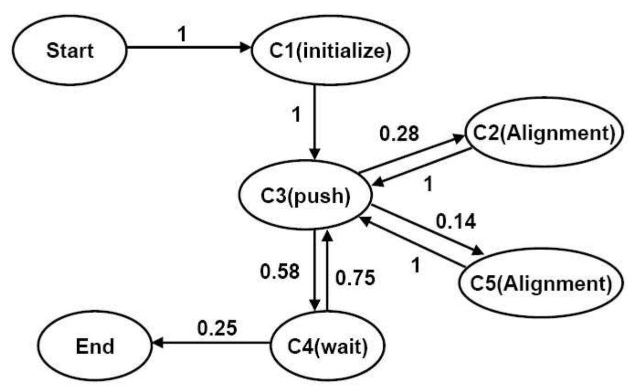

Fig. 2. Learned state transition diagram for a single robot.

TABLE III

STATE REMAINING TIME IN EACH STATE

State	C1	C2	C3	C4	C5
Mean time (s)	2.36	4.22	27.8	4.13	4.11
Standard derivation (s)	0.41	0.43	8.22	0.58	0.39

For the purposes of comparison, we show in Fig. 3 the state transition diagram representing the actual robot control code for this task. We can see that it has a structure similar to that learned by our approach (i.e., in Fig. 2). The two "Alignment" states in Fig. 2 are the representation of the single "Alignment" state in Fig. 3, but with different alignment directions. We note, unlike Fig. 3, there is no transition from 'initialize' to "alignment" or "wait" in Fig. 2 because the motions "alignment" and "wait" always happen after "push" in the real experiments. We can also regard those transitions as having probability zero.

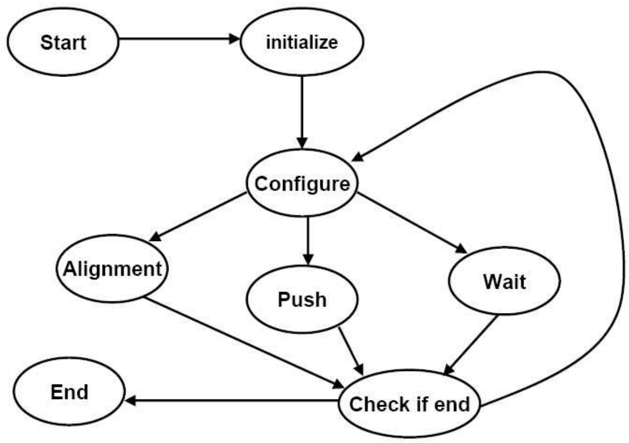

Fig. 3. Actual algorithmic control code for a single robot.

The diagrams also show that most robot control states can be partitioned into meaningful sensor states. However, in more complex situations, we recognize that operator commands or internal variables cannot be directly observed from sensor data. Addressing the issue of such "hidden" variables is an area of our future work.

D. Results for fault detection with single robot

Using the learned state transition diagram, single robot faults can be detected online. To evaluate the SAFDetection approach, two simulated faults were generated manually. The learned state transition diagram shown in Fig. 2 is used for performing fault detection.

In Test 1, we jammed the robot against an obstacle so that it could not move; in this case, our system was able to detect the fault in 6 seconds. Fig. 4 shows the minimum laser index, speed and turn rate features of the single robot in this test. At time 31, we jammed the robot. The consequence was that the robot entered state C4 which is a normal "idle" state of the robot, and then remained in that state. At time 37, the SAFDetection approach noticed that the robot had remained in state C4 for 6 seconds, which is an unusually long time compared to the learned data. In normal operation, the robot would stay in state 4 for 4.13 seconds, with a standard deviation of 0.58. After comparing these values, SAFDetection made a judgment that the robot was stuck in state C4, yielding the detection of a logic fault at time 37.

Of course, our SAFDetection is only practical if it can achieve good results without requiring a large number of learning trials. To study this issue, we used our system to develop sensor models for fault detection using a variable number of learning trials. Fig. 5 shows the results for the experimental setup reported here. In this figure, the time -1 represents a false alarm; as can be seen, with 3 or fewer learning trials, the robot expects every new situation to be a fault. However, with only a few additional trials, the fault detection time becomes more and more accurate, with correct and timely fault detection occurring after about 6 trials.

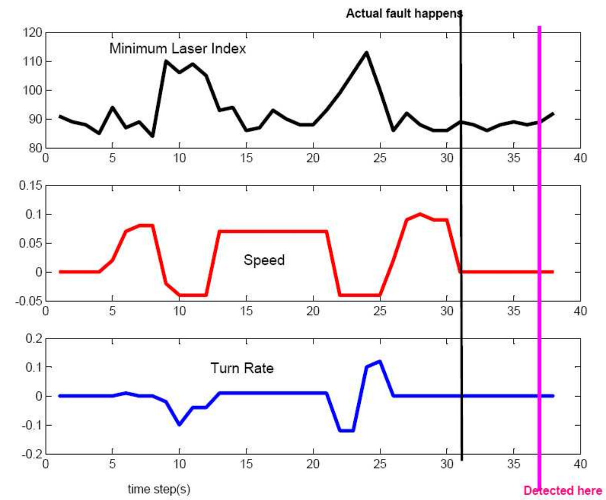

Fig. 4. Sensor data when single robot stuck.

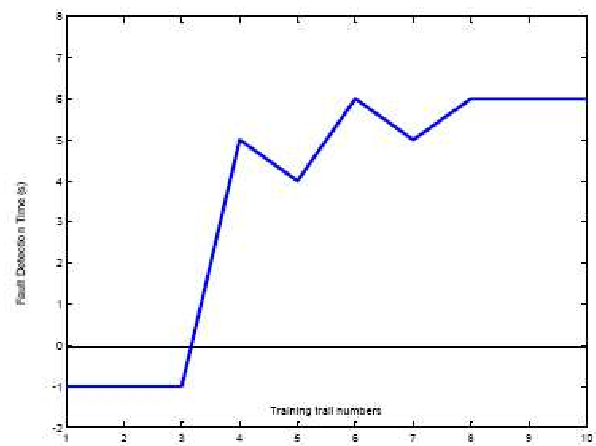

Fig. 5. Fault detection time (i.e., time to detect fault after it occurs) as a function of number of training trials.

In Test 2, while the robot was pushing, we removed the red block that indicated the goal position. The result was that the robot began wandering because it lost the goal. In this case, the fault was detected in 2 seconds. Fig. 6 shows the minimum laser index, minimum laser range, and turn rate features of the single robot in this test. At time 27, we removed the red block and the robot started to wander. At time 29, the SAFDetection approach noticed that the on-line sensor data did not belong to any of the known states. It made a judgment that the robot entered some abnormal situation and a hard fault was detected.

In most circumstances, hard faults can be detected earlier than logic faults. For example, in Test 1, at any time before 37 seconds, we cannot detect a fault because we cannot immediately distinguish between a robot being in normal idle state and the robot being stuck. Only after more time passes is the system able to make this distinction.

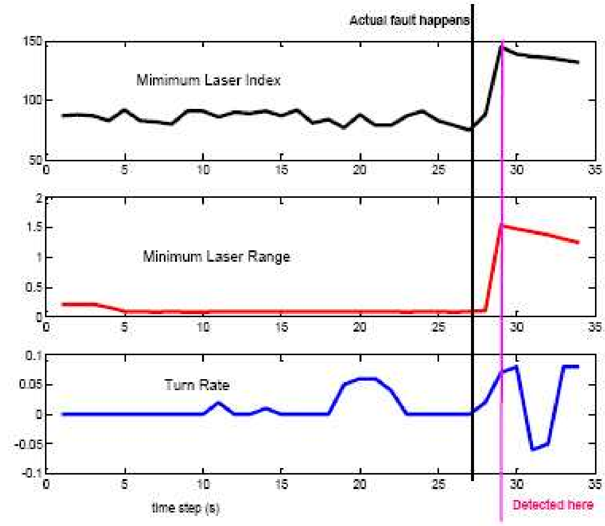

Fig. 6. Sensor data when single robot wanders.

E. Results for fault detection with tightly-coupled multi-robot team

Using the SAFDetection approach with a multi-robot team is quite similar to the single robot situation, since at present, we treat a multi-robot team as one monolithic robot with more sensors than an usual robot. (As noted previously, in ongoing work, we are extending this approach to distribute the detection across multiple team members, and thus no longer treat the robots as a single monolithic system.) Because of this, we have higher dimensional data vectors to be grouped and a larger state transition diagram to be built. For example, in our box pushing experiment, the same eight sensor features are collected from the second robot and a fourteen dimensional sensor data vector is formed for clustering. If we monitor the two robots separately, both robots can be grouped into 5 states, leading to at most 25 states for the robot team as a whole. The actual number of states learned is 7, because the robots work in a cooperative manner. For example, when one of the robots is aligning its direction, its teammate will wait until it has finished alignment. Fig 7 shows the state transition diagram of the robot team.

We simulated a coalition fault by disturbing the communication between the two robots. The state transition diagram shown in Fig. 7 was used to perform the fault detection with the robot team. In this experiment, the fault was detected in 1 second. Fig. 8 shows the speed and turn rate features from both robots in this test. At time 22, robot B started performing alignment while robot A continued pushing the box, due to lacking communication. At time 23, the SAFDetection approach noticed that the on-line sensor data did not belong to any of the known states and a fault was detected.

This test shows that, by monitoring the robot team as a monolithic robot, we can detect more faults than monitoring team robots separately. For example, if we monitor the two

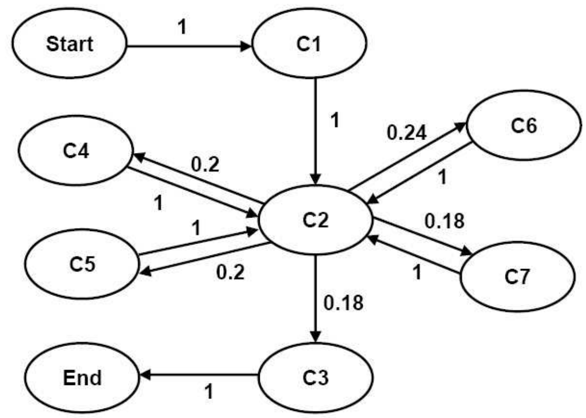

Fig. 7. Learned robot team state transition diagram.

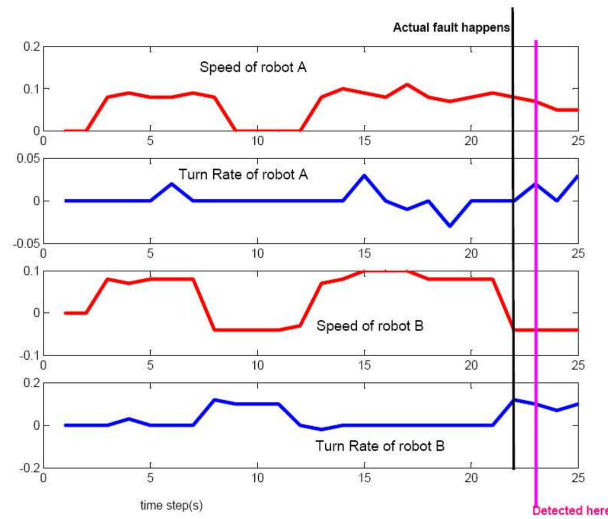

Fig. 8. Sensor data with robot team coalition fault.

robots separately in this test, we will find that robot A is pushing, robot B is doing alignment and both of them are working in normal states. Also, in Test 1, if only one robot is jammed by an obstacle and the robot team is monitored as a whole, the SAFDetection approach will discover that robot A is pushing while robot B is idling. Thereby, an unknown state can be found and the fault could be detected earlier than time 37.

F. Results analysis and future work

We believe our results validate the SAFDetection approach, by illustrating its ability to detect faults based solely on sensor data analysis. No prior knowledge is needed of the motion models of the robots, and no prior knowledge is needed of the possible failure states of the robots. Because of the general nature of the approach, and because we treat the robot system as a black box, we believe SAFDetection has applicability to a wide range of applications.

However, more work is needed to refine the system, and there are some shortcomings we are addressing in ongoing work. Our tests revealed some problems that must be addressed. One issue is the asynchronous nature of the sensor data. To monitor the robot team, we need to collect the sensor data from all the robots in one time step. In addition, different sensors on the same robot also have synchronization problems, due to different response speeds. Noisy and partial sensor data is another issue that needs to be considered. Currently, false alarms often happen because of the noisy or partial sensor data we read from the robots. A sliding window or partial filter may be introduced in our future work to correct this problem. Further study on the training data should also be explored in our future work. At present, we only use sensor data during normal robot operation. However, we should also pay attention to the sensor data during anomalous robot operation. These data provide the fault information of the robot system and can help us to understand the faulty state of system. In a future extension, we will include both normal and anomalous data in our fault detection model.

Another limitation of our approach is that the features we use should remain stable while in the same state for the purposes of clustering. Obviously, some features do not satisfy this requirement; thus, we intend to extend our approach to deal with this issue. Another future work we plan to undertake is to implement the SAFDetection approach in a distributed manner, so that any robot of a team can monitor the whole team. We need to further study the issue of scalability, and to what extent a single monitoring process can be used for larger and larger teams. Clearly, as the team size gets very large, a distributed approach that can break down the monitoring problem, combined with communication, is the most likely solution for very large team applications. Nevertheless, we believe the results shown in this paper illustrate the viability of this approach.

VI. Conclusion

In this paper, we have defined the SAFDetection approach and illustrated how it can be implemented on a single robot, and in a tightly-coupled multi-robot team to successfully detect faults. We focused on the unsupervised FCM clustering algorithm and how to learn a probabilistic state transition diagram for fault detection using the analyzed sensor data. We presented results from physical robot implementations that illustrated the effectiveness of this approach for the online detection of hard faults, logic faults and coalition faults. The experiments showed that the Fuzzy C-Means algorithm provided sufficient speed and reliability to achieve the desired fault detection capabilities. Additionally, the clustered sensor data and learned state transition diagram are feasible for representing the action states of robot system, and for use in detecting robot system faults. We believe that this approach is general enough to apply to a wide range of single- and multi-robot system applications, and can thus achieve online fault detection based on sensor analysis.

References

[1] M. Branicky, V. Borkar, and S. Mitter. A unified framework for hybrid control: Model and optimal control theory. *IEEE Transactions on Automatic Control*, 43(1):31–45, 1998.

[2] Z. Cai and Z. Duan. A multiple particle filters method for fault diagnosis of mobile robot dead-reckoning system. In *IEEE International Conference on Intelligent Robots and Systems*, pages 481–486, 2005.

[3] W. Chen, R. Gong, and K. Dai. Two new space-time triple modular redundancy techniques for improving fault tolerance of computer systems. In *IEEE International Conference on Computer and Information Technology*, 2006.

[4] E. Delage, H. Lee, and A. Y. Ng. A dynamic Bayesian network model for autonomous 3D reconstruction from a single indoor image. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2418–2428, 2006.

[5] C. Ferrell. Failure recognition and fault tolerance of an autonomous robot. *Adaptive behavior*, 2(4):375, 1994.

[6] F. J. Garcła, L. J. Miguel, and J. R. Pern. Fault-diagnostic system using analytical fuzzy redundancy. *Engineering Applications of Artificial Intelligence*, 13:441–450, 2000.

[7] P. Goel, G. Dedeoglu, S. Roumeliotis, and G. Sukhatme. Fault detection and identification in a mobile robot using multiple model estimation and neural network. In *IEEE International Conference on Robotics and Automation*, pages 2302–2309, 2000.

[8] M. Hashimoto, H. Kawashima, and F. Oba. A multi-model based fault detection and diagnosis of internal sensor for mobile robot. In *IEEE International Conference on Robotics and Automation*, pages 3787–3792, 2003.

[9] A. H. Jackson, R. Canham, and A. M. Tyrrell. Robot fault-tolerance using an embryonic array. In *NASA/DoD Conference on Evolvable Hardware*, pages 91–100, 2003.

[10] B. P. Jeppesen and D. Cebon. Analytical redundancy techniques for fault detection in an active heavy vehicle suspension. *Vehicle System Dynamics*, 42:75–88, 2004.

[11] F. Klawonn and A. Keller. Fuzzy clustering and fuzzy rules. In *7th International Fuzzy Systems Association World Congress*, pages 193–198, 1997.

[12] I. S. Lee, J. T. Kim, and J. W. Lee. Model-based fault detection and isolation method using ART2 neural network. *International Journal of Intelligent Systems*, 18:1087–1100, 2003.

[13] M. L. Leuschen, J. R. Cavallaro, and I. D. Walker. Robotic fault detection using nonlinear analytical redundancy. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 456–463, 2002.

[14] J. P. Matsuura and T. Yoneyama. Learning Bayesian networks for fault detection. In *Proceedings of the IEEE Signal Processing Society Workshop*, pages 133–142, 2004.

[15] L. E. Parker and B. Kannan. Adaptive causal models for fault diagnosis and recovery in multi-robot teams. In *IEEE International Conference on Intelligent Robots and Systems*, 2006.

[16] M. H. Sadeghi, J. Raflee, and F. Arvani. A fault detection and identification system for gearboxes using neural networks. In *International Conference on Neural Networks and Brain*, pages 964– 969, 2006.

[17] V. Verma and Simmons. Scalable robot fault detection and identification. *Robotics and Autonomous Systems*, 54:184–191, 2006.

[18] M. L. Visinsky, J. R. Cavallaro, and I. D. Walker. Robotic fault detection and fault tolerance: A survey. *Reliability Engineering and System Safety*, 46:139–158, 1994.

[19] X. L. Xie and G. Beni. A validity measure for fuzzy clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(8):841–847, 1991.

[20] T. Yairi, Y. Kato, and K. Hori. Fault detection by mining association rules from house-keeping data. In *International Symposium on Artificial Intelligence,Robotics and Automation in Space*, 2001.

[21] Q. Yang, J. Yin, and C. Ling. Postprocessing decision trees to extract actionable knowledge. In *IEEE International Conference on Data Mining*, pages 685–688, 2001.

[22] H. Zhou and S. Sakane. Sensor planning for mobile robot localization using Bayesian network inference. *Advanced Robotics*, 16(8):751–771, 2002.