12-2008

SAFDetection:Sensor Analysis based Fault Detection in Tightly-CoupledMulti-Robot Team Tasks

Xingyan Li
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a dissertation written by Xingyan Li entitled "SAFDetection:Sensor Analysis based Fault Detection in Tightly-CoupledMulti-Robot Team Tasks." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

Lynne E. Parker, Major Professor

We have read this dissertation and recommend its acceptance:

Michael W. Berry, Michael G. Thomason, J. Wesley Hines

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:
I am submitting herewith a dissertation written by Xingyan Li entitled "SAFDetection: Sensor Analysis based Fault Detection in Tightly-Coupled Multi-Robot Team Tasks." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

Lynne E. Parker, Major Professor

We have read this dissertation
and recommend its acceptance:

Michael W. Berry

Michael G. Thomason

J. Wesley Hines

Accepted for the Council:

Carolyn R. Hodges,
Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

SAFDetection: Sensor Analysis based Fault Detection in Tightly-Coupled Multi-Robot Team Tasks

A Dissertation
Presented for the
Doctor of Philosophy
Degree
The University of Tennessee, Knoxville

Xingyan Li
December 2008

Dedication

This dissertation is dedicated to my family who gave me tremendous encouragement and support in my work for all these years, with love and gratitude.

Acknowledgements

This dissertation could not have been written without the inspiration and support from many people. First and foremost, I express my gratitude for my parents Kuanyuan Li and Youju Li and my sister, Keyan Li, who encouraged me to pursue a Ph.D. degree and supported me unconditionally, with great patience and love. I also give my heartfelt thanks to my husband Feiqiong Liu, who shared both the joyful and the difficult times with me.

I additionally would like to express my deepest thanks to my advisor, Dr. Lynne Parker. Her willingness to support my work and her inspiration and guidance throughout my studies have helped me to step into the interesting world of robotics. I thank her for giving me the instructions and suggestions that are indispensable to the completion of this dissertation. Also, I would like to thank the rest of my thesis committee, Dr. Michael Berry, Dr. Michael Thomason and Dr. Wesley Hines, for their invaluable advice that help to improve this work. I greatly appreciate their time and input to this dissertation.

Within the Distributed Intelligence Laboratory (DILab), I really enjoyed working with Arno, Blitz, Cappy, and many other robots. I would like to give my thanks to my fellow graduate students in the lab, Yuanyuan Li, Yifan Tang, Daisy Tang, Balajee Kannan, Rasko Pjesivac and Chris Reardon, for their constant encouragement and helpful suggestions on my work.

Finally, I must express my appreciation to the many friends who have helped to relieve the stressful solitude of studying thousands of miles away from home. I enjoyed the friendly environment of our Computer Science Department and the friendship with Yuanyuan, Yifan, Yun, Huadong, Chang, Teng and many others. Your encouragement through both good times and hard times have been a source of strength for me.

Abstract

This dissertation addresses the problem of detecting faults based on sensor analysis for tightly-coupled multi-robot team tasks. The approach I developed is called SAFDetection, which stands for Sensor Analysis based Fault Detection, pronounced "Safe Detection". When dealing with robot teams, it is challenging to detect all types of faults because of the complicated environment they operate in and the large spectrum of components used in the robot system. The SAFDetection approach provides a novel methodology for detecting robot faults in situations when motion models and models of multi-robot dynamic interactions are unavailable. The fundamental idea of SAFDetection is to build the robots' normal behavior model based on the robots' sensor data. This normal behavior model not only describes the motion pattern for the single robot, but also indicates the interaction among the robots in the same team. Inspired by data mining theory, it combines data clustering techniques with the generation of a probabilistic state transition diagram to model the normal operation of the multi-robot system.

The contributions of the SAFDetection approach include: (1) providing a way for a robot system to automatically generate a normal behavior model with little prior knowledge; (2) enabling a robot system to detect physical, logic and interactive faults online; (3) providing a way to build a fault detection capability that is independent of the particular type of fault that occurs; and (4) providing a way for a robot team to generate a normal behavior model for the team based the individual robot's normal behavior models. SAFDetection has two different versions of implementation on multi-robot teams: the centralized approach and the distributed approach; the preferred approach depends on the size of the robot team, the robot computational capability and the network environment.

The SAFDetection approach has been successfully implemented and tested in three robot task scenarios: box pushing (with two robots) and follow-the-leader (implemented with two- and five-robot teams). These experiments have validated the SAFDetection approach and demonstrated its robustness, scalability, and applicability to a wide range of tightly-coupled multi-robot applications.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 The problem

This dissertation addresses the problem of detecting faults based on sensor analysis for tightly-coupled multi-robot team tasks. It is known 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 states so that the deviation from the expected behavior can be promptly detected. Here, I 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 in robot systems vary from mechanical degradation to information insufficiency, and may be due to a large spectrum of components in the system, including robot sensors, actuators and system components. Furthermore, insufficient collaboration or asynchronous communication between robots can also induce faults in multi-robot teams. A common approach to detect faults in robot systems is comparing estimated values with the actual measurements; a fault is detected if the estimated values deviate significantly from the actual measurements. Typically, the estimated values are predicted by the motion model of the robots, which is preprogrammed by the designer. However, there are situations in which the motion model is unknown or difficult to build. In addition, these motion model based fault detection approaches are not suitable for multi-robot team tasks since it is difficult to build models that capture the close dynamic interactions between robots. For example, consider a simple multi-robot following application where a team of robots needs to move from a starting position to a goal position, one followed by another. The motion model of the robot is not only dependent on itself, but is also related to the behavior of the robot that leads it.

My approach focuses on detecting faults in tightly-coupled team tasks accomplished by a robot system. **Team task** here refers to a task that is performed by several robots, with each accomplishing a part but all working together for efficient completion of the team work. It provides a novel methodology for detecting robot faults in situations when motion models and models of multi-robot dynamic interactions are unavailable. As in many other fault detection approaches, in SAFDetection, the estimated robot sensor values and the

actual measurements are compared to detect faults. Instead of using the preprogrammed robot motion model, the estimated values are predicted by a robot's normal behavior model, which is generated from the history of the robot's sensor data during normal operations. Furthermore, the robot team's normal behavior model also describes the interactions between robots so that interactive faults can be detected.

1.2 The approach

To address the problem and challenges, I present the SAFDetection approach, which stands for Sensor Analysis based Fault Detection, pronounced "Safe Detection". The fundamental idea of SAFDetection is to build the robot system's normal behavior model based on the robot system's sensor data. The whole fault detection process can be divided into two stages: training stage and detection stage. In the training stage, normal robot state transitions are learned from a history of sensor data during the normal operational mode of the robot. In the detection stage, online faults are then identified via a deviation of the sensor data from the model of normal behavior.

This approach can be applied to a single robot or a tightly-coupled multi-robot system with a set of pre-defined and pre-programmed tasks. To provide the data for learning, the tasks are repeated in advance as necessary. The normal behavior of the tightly-coupled task is learned through several trials before performing the online detection. Once a fault is detected in real time, it can be used by a higher-level diagnosis and recovery strategy, such as [Parker and Kannan, 2006]. The most difficult aspect of the problem is distinguishing between normality and abnormality with no knowledge of the robots' motion models or models of dynamic interactions between robots and limited prior knowledge of the robot system. I solved this problem by learning the normal behavior of the multi-robot system through sensor analysis using a combination of data clustering techniques and a probabilistic state transition diagram. I have also implemented a distributed version of the SAFDetection approach to distribute the computational load and improve the reliability and scalability of the system.

SAFDetection handles three types of sensor faults: "hard" fault, "logic" fault and "interactive" 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 interactive fault reflects conflicts among team robots, in which the sensor model expectations of each robot differ.

1.3 Preview of results and contributions

The major contribution of this dissertation is the development of SAFDetection — a novel general approach that autonomously detects the faults of a robot or tightly-coupled multi-robot team task according to the sensor data collected from the robot(s). It is the first approach that builds the robot's normal behavior model from historical sensor data using

data clustering and probabilistic state transition diagram techniques. The development of SAFDetection involves the design and construction of the following sub-systems:

- A centralized SAFDetection approach that autonomously detects the faults of a robot or robot team task according to the sensor data collected from the robot(s), which I first introduced in [Li and Parker, 2007]. Principal Components Analysis methods are used to improve the automaticity of the approach. I have validated the approach through physical multi-robot team tasks and compared the experimental results of the performance with other data driven fault detection methods [Li and Parker, 2008].

- A distributed SAFDetection approach that autonomously detects the faults of a robot team task according to the sensor data collected from the robots. This approach uses the combination of individual robot states to represent the robot team's states and share the robot team's states through message communication. The distributed approach offers a tradeoff of increased robustness versus detection quality compared to the centralized SAFDetection approach. I have validated the distributed approach through physical and simulation robot team tasks. Experimental results have also been analyzed to compare the centralized approach with the distributed approach.

Compared to other fault detection approaches in the robotics area, the SAFDetection approach is different in that it addresses the detection of interactive faults with limited a prior knowledge of the robot system. This dissertation emphasizes how to build the robot's normal behavior model which represents the interactions between robots in a general and autonomous manner.

1.4 Organization of the dissertation

The organization of this dissertation is as follows:

Chapter 2 Related Work. This chapter presents a review of the related work in robot fault detection and discusses the differences between the SAFDetection approach and other works.

Chapter 3 Centralized SAFDetection. This chapter presents the centralized SAFDetection approach, which regards the entire robot team as a monolithic single robot and uses the robots' sensor data from the entire team to build the robot team's normal behavior model.

Chapter 4 Distributed SAFDetection. This chapter presents the distributed SAFDetection approach, a distributed system that enables robots to make use of the individual robot's normal behavior models and monitor the robot team through message communication.

Chapter 5 Experimental Validation. This chapter presents the experiments that I have implemented to validate the SAFDetection approach for both the centralized and distributed approaches. Experiments are performed both in simulation and on physical robots in two applications.

Chapter 6 Summary and Conclusions. This chapter summarizes the main contributions of this dissertation and describes potential extensions of the SAFDetection approach.

Chapter 2

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.

2.1 Fault detection methods in robotics

The most popular method for providing fault detection in robot systems is based on motion control [Hashimoto et al., 2003] [Visinsky et al., 1994] [Luo and Wang, 2005] [Lee et al., 2003]. This method compares the values of actuators estimated by the motion model and the current measurements to detect a fault. Usually, the motion model is preprogrammed by the designers. For example, in the Hannibal robot system [Ferrell, 1994], 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. This requirement may become inconvenient especially when shareware or applications implemented by the third party are used. Furthermore, this method does not address the dynamic interactions of multiple robots. SAFDetection addresses situations where the motion model is not applicable, the robot motion model is not totally known or the robots are working on tightly-coupled tasks.

Voting based on modular redundancy is another robot fault detection method [Jackson et al., 2003] [Sundvall and Jensfelt, 2006] [Chen et al., 2006]. 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. For example, in one work of the American Institute of Aeronautics and Astronautics [Marmon, 1979], three processors are used for fault tolerance. This fault detection method is not commonly used in mobile robot applications since the redundant modules increase the cost and energy consumption. Also, it is unrealistic to have completely redundant components on a small mobile robot and this approach does not address interactive failures.

Analytical redundancy is a fault detection method that does not need redundant modules [Chow and Willsky, 1984] [Leuschen et al., 2002] [Jeppesen and Cebon, 2004] [Staroswiecki and Comtet-Varga, 2001] [Garca et al., 2000]. It essentially takes two forms: direct redundancy and temporal redundancy. The direct redundancy exists among sensors whose outputs are algebraically related, which means the sensor outputs are related in such a way that the variable one sensor measures can be determined by the instantaneous outputs of the other sensors. The temporal redundancy approach relates sensor outputs and actuator inputs and presents the relationship among the histories of sensor outputs and actuator inputs. According to these relationships, the outputs of dissimilar sensors at different times can be compared. The residuals resulting from these comparisons are then measured as the discrepancy between the behavior of observed sensor outputs and the behavior that should result under normal conditions. Analytical redundancy provides a useful mathematical approach for determining the various redundancies that are relevant to the failures under consideration. For example, if the robot used in the analysis contained a position sensor and an engine force actuator per joint, the relationship between current position value, previous position value and engine force value can be examined by the temporal redundancy approach to detect the fault. The analytical redundancy approach is not suitable for the problem I addresses since the cooperative relationship among robots is not sufficiently precise and synchronous to be represented by the redundancy.

In recent years, particle filter techniques for robot fault detection have become popular [Goel et al., 2000] [Verma and Simmons, 2006] [Cai and Duan, 2005] [Cheng et al., 2005] [Kadirkamanathan et al., 2002]. This method can estimate the robot and its environmental states from a sequence of noisy, partial sensor measurements. For example, in the work of Planetary Rovers [Dearden and Clancy, 2002], particle filters are used to distinguish a change in the battery current drawn due to a fault in the wheel. Many particle filter based fault detection methods work with a known set of possible fault types and each particle represents a fault state. I do not use particle filters as the frame work of my approach since possible fault types are unknown in advance in my problem.

Neural networks have also been used in the robotics area recently. Skoundrianos [Skoundrianos and Tzafestas, 2004] trains multi-layer perceptron neural networks to capture the input-output relationship of the robot components to detect faults, especially focusing on the relationship between the motor voltage and speed. Marslands [Marslanda et al., 2005] detects faults with a neural network based novelty filter which is trained to ignore sensory data similar to previously perceived data and can be used to detect environmental differences. Christensen [Christensen et al., 2007] assumes that hardware faults change the flow of sensory data and the actions performed by the control program. Thus, the presence of faults can be inferred by detecting the changes. In their work, the sensory data from the robots are collected while they are operating normally and after a fault has been injected, then a back-propagation neural network is used to synthesize fault detection components based on the data collected in the training runs.

2.2 Data driven fault detection methods

Data driven fault detection methods, especially the data mining method, have been used in many software fault diagnosis approaches [Kawabata et al., 2004] [Singhal and Jajodia, 2006] [Teoh et al., 2004]. In general, there are two types of fault detection methods based on data extracted from the system: proximity-based and density-based techniques. In the proximity-based approach, anomalous objects are those that are distant from most of the other objects. Many of the techniques in this area are based on distances and are referred to as distance based outlier detection techniques. The density-based approach is relatively straightforwards – objects that are in regions of low density, or have a local density significantly less than that of most its neighbors, are considered anomalous. The SAFDetection approach belongs to the proximity-based approach since the deviation of the faults from the normal behavior is determined by the distance between clusters.

Commonly used techniques to extract system knowledge from data include decision trees [Yang et al., 2001] [Sheng and Rovnyak, 2004], artificial neural networks [Sadeghi et al., 2005] [Manikopoulos and Papavassiliou, 2002] and probabilistic networks. The Bayesian network is a popular representation of a probabilistic network [Zhou and Sakane, 2002] [Delage et al., 2006] [Krishnamachari and Iyengar, 2004] [Lerner et al., 2000]. Matsuura [Matsuura and Yoneyama, 2004] describes 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.

One fault detection method similar to my approach is to construct 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 [Yairi et al., 2001] [Bhunia and Roy, 2002] outside of robotics such as spacecraft and digital circuits. Their approach is not directly applicable to robotics since their patterns are sensitive to time, while strict time limit is not required in our problem. Hybrid control systems can be used to identify generic patterns of continuous and discrete event dynamic systems [Branicky et al., 1998]. A generic framework for hybrid systems includes transitions between continuous and discrete states.

In the work of [Fatta et al., 2006], a method to enhance fault localization for software systems based on a frequent pattern mining algorithm is presented. Function call trees and test oracles are used to classify successful and failing tests. A frequent pattern mining algorithm is used to extract patterns from the database of execution traces that carry discriminative information between correct executions and executions that lead to a failure. The functions are ranked according to the probability that they contain a fault and present a final report to the programmer. The SAFDetection approach is different in that it learns a state transition diagram rather than 'if-then' rules or frequent patterns. The state transition diagram has the advantage of representing the probability of the transitions between states. Furthermore, I applied it to the domain of multi-robot systems, which has not previously been done.

2.3 Team work

Fault detection in a multi-robot system is challenging. This dissertation focuses on tightly-coupled multi-robot team tasks. In these tasks, robots form a team to work together to accomplish a common goal. The cooperation among robots is tightly-coupled and necessary for accomplishing the task at hand. The box pushing task [Donald et al., 1997] is a typical tightly-coupled multi-robot team task. In this task, a box is so heavy that it needs two or more robots working together to push the box to a goal position in a cooperative manner (as shown in Figure 2.1). The robot following task is another tightly-coupled multi-robot team task. In this task, a team of robots moves in a line from the starting position to the goal position, one followed by another (as shown in Figure 2.2). Fault detection for such tasks not only requires fault detection for each individual robot, but also requires monitoring the consistency among the team members.

Previous work on detecting faults occurring among tightly-coupled multi-robot team members is very limited while some related topics, such as fault tolerance for multi-robot teams [Gerkey and Mataric, 2002] [Sun and McCartney, 2001], have been explored. Most of these works are based on understanding the structure of the team task. For example, in Goldberg's foraging task, robots are assigned to different hierarchies and perform different parts of the work. More than one robot works in the same hierarchy to reduce the possible loss caused by one robot. Robots can switch between hierarchies to accomplish fault tolerance in a higher level.

2.4 Summary

In summary, unlike motion model based methods, the SAFDetection approach does not require knowledge of the "inner control" of the robot system. I also assume no advance knowledge of the possible fault types, unlike several other approaches, such as particle filters. 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, the SAFDetection approach treats the robot or robot team system as a black box. It also deals with the detection of faults in the dynamic interactions of multiple robots, unlike the other approaches which do not address this issue.

Figure 2.1: Box pushing task with two robots.

Figure 2.2: Following task with two robots.

Chapter 3

Centralized SAFDetection

This chapter describes a centralized approach that I have designed and implemented for a robot system to autonomously detect faults based on sensor data analysis. This centralized approach can be applied on a single robot or a robot team. However, since all the computation is performed in a centralized manner on the "server" of the robot team, the centralized approach may have difficulty scaling to large team sizes. As discussed in Chapter 4, the distributed SAFDetection overcomes this problem by distributing the clustering process across every robot. In a real application, the designer needs to take into account the trade-off between scalability and fault detection quality (see Chapter 4 for details). This work was presented at IEEE International Conference on Robotics and Automation (ICRA'07) [Li and Parker, 2007] and IEEE SoutheastCon 2008 [Li and Parker, 2008].

3.1 Overview of approach

To detect faults in a mobile robot system, the detection approach needs the ability to distinguish a normal robot behavior from an anomalous robot behavior. With the motion model of the robot, individual faults can be easily detected by comparing the estimated output of the preprogrammed motion model to the actual measurement. However, faults in tightly-coupled multi-robot team tasks, especially faults caused by dynamic interactions between robots, usually cannot be detected using this method. Even in single robot tasks, there are situations in which the motion model or any idea of the expected fault signature is unknown or difficult to determine. Instead, by analyzing the sensor data during a robot's normal operations, the SAFDetection approach can build the robot's normal behavior model as a probabilistic state transition diagram. States in the diagram are learned from the sensor data and 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. The centralized SAFDetection approach, (shown in Figure 3.1), is a

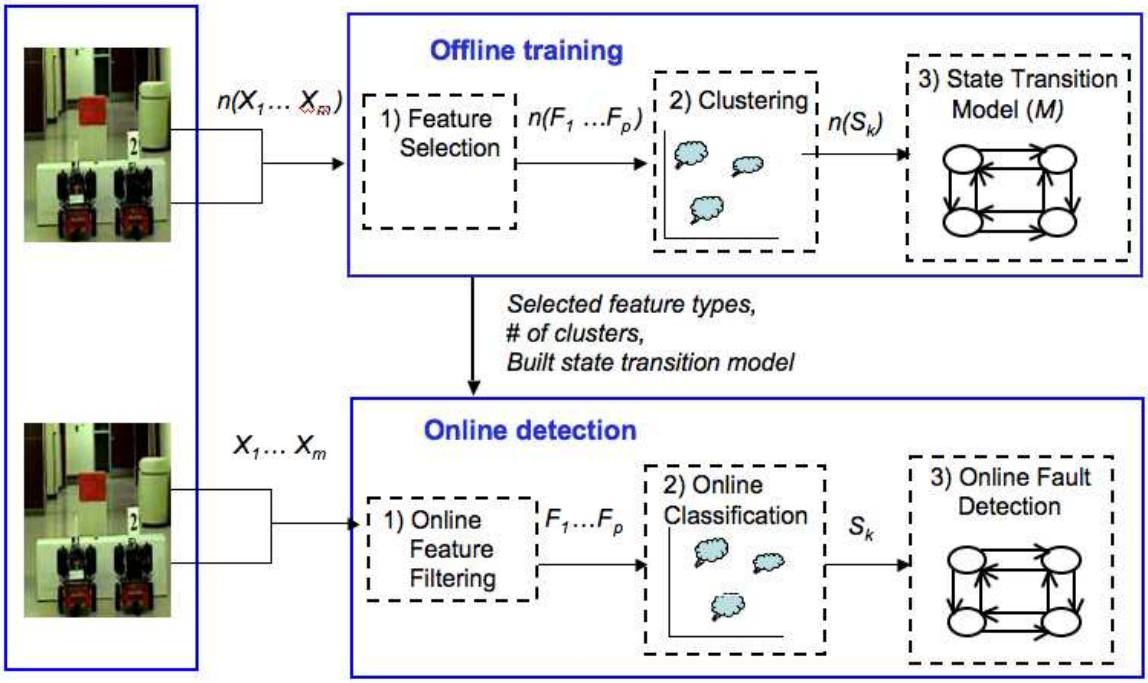


Figure 3.1: The structure of centralized SAFDetection approach.

training-classification based method. In the training stage, the following steps are performed to build the robot system's normal behavior model:

Original data collection. A history of robots' sensor data (i.e., training data) during normal operation is collected as $\mathbf{X}$. Here, $\mathbf{X}$ is a sequence of n m-dimensional data vectors, where n is the number of training samples and m is the number of sensor data elements. For example, if for one minute, the robot speed and orientation values are collected every second, then n is sixty (60) and m is two (2).

Essential features selection. Since some of the robots' sensor data is not related to the task, only essential features are selected for the next step. Thus, $\mathbf{X}$ is compressed to $\mathbf{F}$, an n-size sequence of p-dimensional feature vectors:

$$\mathbf{F} = FeatureSelection(\mathbf{X}) \tag{3.1}$$

This feature selection can be accomplished manually or based on statistical analysis. One statistical analysis method, PCA, is used in this dissertation and is discussed in Section 3.2.1.

Data clustering. The sequence of feature vectors, $\mathbf{F}$, is then mapped to $\mathbf{S}$, an n-size index array that maps each feature vector to its best cluster:

$$\mathbf{S} = Clustering(\mathbf{F}) \tag{3.2}$$

There are k clusters in total and each cluster centroid is a p-dimensional feature vector, which stands for a robot state. Three different clustering algorithms are explored in this dissertation, as discussed in Section 3.2.2.

Build probabilistic state transition diagram. Using the index array, $\mathbf{S}$, the robot system's normal behavior model is represented as a probabilistic state transition diagram, the $k \times k$ size matrix M:

$$\mathbf{M} = BuildingDiagram(\mathbf{S}) \tag{3.3}$$

where M_{ij} is the transition probability from cluster (state) C_i to cluster (state) C_j. How to incorporate time factors into this diagram is explored in this dissertation, as discussed in Section 3.2.3.

In the online fault detection stage, the online sensor data is collected, filtered, classified and compared with the state transition model $\mathbf{M}$ for detecting three types of faults. If the sensor data does not belong to any of the states learned from the training data, a *hard fault* is detected. If the sensor data belongs to one of the learned states but the observed state transition deviates significantly from the state transitions 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 an *interactive fault*. If no fault is detected, the sensor data is classified as normal. Since the SAFDetection approach regards the monitored system as a black box, it uses the sensory data read from the robots, no matter what task is executed by the robot system or the type of the robot. This approach thus can be used on different robot systems, including single robot and team robots performing a variety of tasks. In the centralized approach, a small team of robots could be regarded as one monolithic robot with the combined set of sensors.

3.2 Training stage

The training stage of the centralized SAFDetection approach is made up of three modules: feature selection is performed on the history of sensor data (i.e., training data) during normal operation to select the essential features; a data clustering algorithm is applied on the selected feature data to map it to the appropriate robot system's state; and the state transition diagram is built with the sequence of robot system's states to describe the normal behavior of the robot system.

3.2.1 Feature selection

Since the SAFDetection approach treats the monitored system as a black box, the only information obtained from the system is the robot sensor data. This approach makes use of the sensory resources already installed on the robots that enable them to perform their tasks. Examples of these task relevant sensors include a laser ranging device, which can provide the range to the nearest object in a 180 degree field of view, and a camera, which can record the image in its field of view. Obviously, it is impractical to use all these data for training the SAFDetection system and learning the state transition diagram. Instead, relevant sensor features are selected and monitored. The essential features are primarily

determined by the tasks executed by the robot system. For instance, in the robot team navigation task [Parker et al., 2004], each robot in the team has a red and black marker as identity so that the robot can determine the relative position of its teammates from the marker image through the camera. Thus, the red and black marker blob captured by the camera is more important than the other objects in the camera's field of view since it provides information on the state of the robot. The correlation between different sensors is another factor to consider. For example, both the laser and the sonar can measure the distance to nearby objects around the robot, and are highly correlated; it is therefore reasonable to choose both of these sensors as essential features to take advantage of the system's available rational redundancy. For a multi-robot system, the centralized approach regards the entire team as one monolithic robot with a unified set of sensors. 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. For example, in one of my experiments, two robots are working together to push a long box to a goal position, which is indicted by a colored blob. The robot sensors used in this task are: laser, sonar, camera, motor and battery. Thus, possible features include 180 degrees of laser measurement, 16 degrees of sonar measurement, the colored blob's border positions in the camera image, a 320×180 pixel color image, speed, orientation and battery charge value from both robots. Even if only the colored blob's border positions are counted as the camera image features (instead of all the 57,600 pixels), there are more than 400 possible features for this two-robot system. It is impractical for the SAFDetection approach to use every possible feature in the system. First, the curse of dimensionality, meaning that the clustering becomes less meaningful as the dimensionality of the data increases, makes it difficult to find compact clusters based on data density when the data dimension is large. Secondly, the time complexity of the clustering algorithm, $O(nkp)$ (n is the amount of data, k is the number of clusters and p is the dimension of the data), increases as the data dimension p increases. Thirdly, transmitting high-dimension data from client robots to the server also increases the risk of network traffic congestion. Thus, historical sensor data is used to select the most relevant features to include in the feature vector. In this dissertation, two feature selection methods are implemented and compared (see Chapter 5 for applicaton-specific details): the manual feature selection approach and statistical analysis-based feature selection.

Manual feature selection

Manual feature selection is performed by the human designers, based on the designers' knowledge of the robot task and the variance of the history sensor data. To reduce the feature vector dimension without affecting the final clustering result, the "almost constant" features that do not provide any information are eliminated. Therefore, for each feature, the mean and standard deviation values are calculated from the training data. If the standard deviation value is relatively small, the feature is considered as a constant value when the robots operate normally and it is filtered out during data clustering. However, if the online sensor data shows that this filtered out feature deviates largely from the learned mean value,

some fault may have occurred. Here, the "filtered out" features are not useless features; they are filtered out in order to reduce the complexity and improve the accuracy of the clustering algorithm. The mean and standard deviation values of the filtered out features are still used to test the on-line data during the fault detection stage.

Statistical analysis-based feature selection

An experienced human system designer who is highly familiar with the robot tasks could likely select the relevant sensor features manually. Unfortunately, in most robot tasks, it will be difficult to determine the relevant sensor features manually. Thus, techniques that can automatically select the best sensor features for the robot team task are preferred. In this dissertation, statistical analysis-based feature selection is performed based on Principal Component Analysis (PCA) and correlation analysis on the robot sensor data.

Principal Component Analysis (PCA) [Jolliffe, 2002] is a common method used to reduce the dimensionality of an input space without losing a significant amount of information (variation). The transformation to principal component space can be thought of as projecting the m-dimensional data set $\mathbf{X}$ onto the new lower p-dimensional coordinate system $\mathbf{P}$, resulting in p-dimensional scores $\mathbf{F}$ and residue $\mathbf{E}$. The new p variables, called principal components or features, are globally uncorrelated, while most of the variation of the original data set $\mathbf{X}$ is preserved:

$$\mathbf{F} = \mathbf{X} * \mathbf{P} + \mathbf{E} \tag{3.4}$$

Various guidelines [Moore, 1981] have been developed to find the proper reduced space dimension, p. In PCA, the principal components are the eigenvectors of the covariance matrix of $\mathbf{X}$; the first eigenvector corresponds to the largest eigenvalue. Since the eigenvalues are proportional to the amount of variance (information) represented by the corresponding principal component, the value p is limited by selecting the first few principal components that represent most (for example, over 80%) of the information.

In SAFDetection, I use PCA to automatically determine the best features for representing the state of a robot system during the performing of a repetitive task. PCA is performed on different subsets of the robot sensor data to select useful and important features: (1) on sensor data values from the same sensor on the same robot, for example, the colored blob's left, right, upper and bottom edge position in the camera image; (2) on sensor data values from different sensors on the same robot that reflect similar information, for example, the laser and sonar data that both measure the distance from objects around the robot; and (3) on sensor data values from the same sensor on different robots, for example, the speed from both robots on the team.

In addition, correlation analysis is performed on the PCA results to select the essential features. The "actuator sensor" data, for example, the speed and turn rate, are regarded as the essential features since they reflect the robot behavior directly. On the other hand, the "sensory sensor" data, for example, the laser or sonar data, does not directly reflect the robot behavior and may not relate to the robot state. Thus, correlation analysis between the actuator sensor data and sensory sensor data is performed to select only sensor data that is related to the robot state.

3.2.2 Data clustering

A clustering algorithm is used in SAFDetection to map the robot feature data to a robot state. Data clustering is a common technique to classify similar objects into different groups, or more precisely, to partition a data set into subsets (clusters), so that the data in each subset (ideally) shares some common trait. In SAFDetection, a data clustering method is used to create subsets that represent different states of the robot system. The robot sensor data in the same subset has similar values. Many clustering algorithms have been developed to find groups in unlabeled data based on a similarity measurement among the data patterns. An open question is the degree to which the particular clustering algorithm used affects the ability of the system to accurately detect faults. In this dissertation, performance of crisp (K-means), probabilistic (soft K-means) and fuzzy (Fuzzy C-means) clustering methods within the SAFDetection approach are compared, determining the affect on the fault detection rate when using different clustering methods (see Chapter 5 for application-specific details).

K-means clustering [Hartigan and Wong, 1979] is one of the simplest unsupervised learning algorithms. It aims at minimizing the following objective function:

$$J = \sum_{k=1}^{c} \sum_{x_i \in C_k} (x_i - c_k)^2 \tag{3.5}$$

where c is the total number of clusters, x_i is the i^{th} data point and c_k is the centroid of the k^{th} cluster, C_k. K-means is a fast, simple and efficient clustering algorithm. However, it is crisp, meaning that one data point is assigned to exactly one cluster and all points assigned to a cluster are equal in that cluster. This clustering method may encounter problems when dealing with clusters that have some overlap. Additionally, when applied to robot applications, it is sensitive to noisy or partial data.

Soft K-means clustering [Kim, 2007] is a modified version of the K-means clustering method with a stiffness parameter, β. In soft K-means clustering, each data point x_i is given a soft degree of membership to each of the centroids. This characteristic of soft clustering helps deal with overlapping clusters, and is advantageous when dealing with noisy or partial data that is typical of robot applications. The soft degree of assignment of data x_i to k^{th} cluster is labeled r_i^k, and defined as:

$$r_i^k = \frac{\exp(-\beta * d(x_i, c_k))}{\sum_{j=1}^{c} \exp(-\beta * d(x_i, c_j))} \tag{3.6}$$

where c is the total number of clusters, c_k is the centroid of the k^{th} cluster and $d(c_k, x_i)$ is the distance measurement (in SAFDetection, Euclidean distance is used) between data point x_i and centroid c_k, which is calculated as:

$$c_k = \frac{\sum_{i=1}^{n} r_i^k x_i}{\sum_{i=1}^{n} r_i^k} \tag{3.7}$$

Fuzzy C-means clustering (FCM) [Bezdek et al., 1984] has been used in many areas. The advantage of FCM is due to its fuzziness, in which a single data point may belong to several clusters, having different membership values in each cluster. This property is advantageous when dealing with the noisy or partial data of typical robot applications. In the FCM algorithm, the algorithm minimizes the objective function J:

$$J = \sum_{i=1}^{n} \sum_{k=1}^{c} (r_i^k)^m d^2(x_i, c_k) \tag{3.8}$$

where x_i is the i^{th} data point, c_k is the centroid of the k^{th} cluster, r_i^k is the degree of membership of x_i in the k^{th} cluster, m is a weighting exponent on each fuzzy membership, $d(x_i, c_k)$ is the distance measurement (in SAFDetection, Euclidean distance is used) between data point x_i and cluster centroid c_k, n is the number of data, and c is the number of clusters. This objective function J is minimized via an iterative process in which the degree of membership, r_i^k, and the cluster centers, c_k, are updated, as follows:

$$r_i^k = \frac{1}{1 + \sum_{j=1}^{c} (d(x_i, c_k)/d(c_j, c_k)^{\frac{2}{m-1}}} \tag{3.9}$$

$$c_k = \frac{\sum_{i=1}^{n} (r_i^k)^m x_i}{\sum_{i=1}^{n} (r_i^k)^m} \tag{3.10}$$

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

One limitation of the above three clustering algorithms is that the number of clusters, c, is needed for calculation. However, without the knowledge of the true model, this number of clusters is unknown before clustering is performed. To solve this problem, the clustering algorithms are run iteratively over several trials with varying cluster numbers and the one giving the best clustering quality is selected. Xie defined a clustering quality measurement in [Xie and Beni, 1991] to measure the overall quality of a clustering algorithm. This validity function, V, is given by:

$$V = \frac{\sum_{k=1}^{c} \sum_{i=1}^{n} r_i^k (x_i - c_k)^2}{n \times \min_{j \neq k} (c_j - c_k)^2} \tag{3.11}$$

where r_i^k is the degree of membership for data x_i in cluster c_k. Since a smaller V value means a more compact and separate partition, the cluster number with the minimum V value is chosen as the final result.

In SAFDetection, our results (in Chapter 5) show that Fuzzy C-means and Soft K-means clustering result in better fault detection rates with noisy and partial data than K-means clustering. I chose Fuzzy C-means as the clustering algorithm in SAFDetection because it is more popular than the Soft K-means algorithm. According to the curse of dimensionality [Richard, 1957], adding extra dimensions to a (mathematical) space will cause an exponential increase in volume. Therefore, when the data dimension is large, the clustering becomes less meaningful since it is difficult to find compact clusters based on data density given scattered data in the vast space. In centralized SAFDetection, since data

clustering is performed on the essential features from the entire robot team, the performance of data clustering will degrade as the robot team size increases. To overcome this limitation, a distributed approach is implemented in Chapter 4.

3.2.3 Probabilistic state transition diagram

In centralized SAFDetection, the robot team is regarded as a monolithic robot and a probabilistic state transition diagram is built to describe the normal behavior of the entire robot team. The interaction between the robots is presented as the existing team robot state. A probabilistic state transition diagram is similar to a Markov chain [Rabiner and Juang, 1986], in that it records states and the transition probabilities between each pair of states.

A Markov chain is a useful tool to recognize the statistical pattern of the transitions among the states. The probabilistic state transition diagram is a finite set of states and transitions among the states are governed by a set of probabilities called transition probabilities. A Markov chain requires that the modeled system satisfy the Markov property, which means the conditional probability distribution of future states of the process is conditionally independent of the past states (the path of the process) given the present state. However, most of the robot systems do not behave according to the Markov process in reality. Simplifying the real robot's continuous motion to a motion sequence with the Markov property may induce negative effects in the model. I have explored several methods to record "past information" into the probabilistic state transition diagram.

The first method is to keep the mean and standard deviation values of the time duration of the system in each state (i.e., before the system transits to another state). Thus, with the time duration information for each state, robots being abnormally stuck in one state can be detected. In such cases, given the same online sensor data, the robot may actually be in different states, depending on the "past states". For example, if the time the robot remains in the "pause" state during normal operation averages five (5) seconds with a standard deviation of one (1) second, then the robot is in a "faulty (stuck)" state if it remains in the "pause" state for ten (10) seconds or more.

The second method is to record the transition probabilities between states as a time vector instead of an average value. Thus, the transition probabilities will vary as the time duration in the state varies. For example, the transition probability from state "pause" to state "move" changes as the time length the robot remains in state "pause" increases. Like the first method, the "stuck" fault can also be detected under this situation. Compared to the first method, this method gives a more precise description for states that have relatively stable time duration lengths. However, for the state whose time duration length may change greatly, this method can result in false alarms given insufficient training data.

The third method is to expand the state to a sliding window which includes the history states of the robot. In this way, the robot state is dependent on not only the current sensor data, but also the historical sensor data. For example, the state sequence <move, move, pause> is different from state sequence <pause, pause, pause> even though the online sensor data shows that the robot is in "pause" currently. Compared to the other two methods, this method records much more history information and the "stuck" fault can be detected given a proper window size. However, the state expansion increases the

complexity of the state transition diagram; it is difficult to decide the size of the sliding window if states have diverse time duration lengths.

In my experiments, as in many other robot applications, the time duration of one state varies greatly and the transition between states is not very complicated (see details in Chapter 5). Therefore, I chose the first method to build the probabilistic state transition diagram, which not only records states and the transition probabilities between each pair of states but also keeps the record of the robot behavior history – 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, all the sensor data should be categorized into different states. As a result of the fuzzy clustering algorithm, the data is not crisply labeled to only one state. Equation 3.9 shows that the r value is used to represent the degree of membership of the data x in different clusters. Thus, the r value is used to assign data x into states with different probabilities. Variables t_1 and t_2 are threshold values to deal with the fuzzy sensor data. By arranging the labeled data in sequence, the transition between states can be determined and the state transition diagram is built easily.

3.3 Detection stage

In the training stage, the team state transition diagram is generated on the "server" of the robot team. In the detection stage, the online team sensor data is collected by the "server" and the learned essential feature values are selected. The clustering algorithm with the learned cluster centers is performed on the selected online feature values to determine the online robot state. Finally, the online robot state is compared with the learned team state transition diagram to detect faults. The algorithm for detecting faults is sketched in Algorithm 2.

The on-line sensor data x and its membership value r are sent to the fault detection module and different types of faults can be detected. If no online data is received for time threshold value t_s, then a communication fault between the client robots and the server is detected. The value of t_s is determined by the data sampling frequency and the network environment, as shown in Equation 3.12:

$$t_s = \sum_{n=1}^{m} (1-p) \times p^{n-1} \times [t_m + (n-1)/f] \tag{3.12}$$

where f is the data sampling frequency, t_m is the message transmission time from client robots to the server and p is the average dropped message rate of the network. The value m counts how many times the sensor data is sent before the server receives it. In theory, m can be infinity for any network whose average dropped message rate p is greater than 0 since there is always a non-zero probability that the data is dropped after m times. In real applications, I assume that data are transmitted when the successfully transmitted probability is $\geq 99.99\%$. In addition, if the membership r value does not show clearly

Algorithm 1 Sketch of building the state transition diagram in centralized SAFDetection.

for each data x and its membership value r **do**
  find the max(r)=r_i and max (r-r_i)=r_j.
  if r_i-$r_j > t_1$ **then**
    x is set to cluster i with weight 1.
  else
    if $r_j > t_2$ **then**
      x is set to cluster i with weight r_i and set to cluster j with weight r_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

Algorithm 2 Sketch of fault detection module in centralized SAFDetection.

```
for each sampling time do
    while no online data received do
        collect online sensor data x from the robot system.
        if no online data is received for time t_s then
            communication fault between clients and server is detected.
        end if
    end while
    x's membership value r is obtained from equation 3.9.
    if max(r) < 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
```

which cluster x belongs to, a hard fault or interactive fault is detected because the robot team enters an unknown state. If x belongs to state c and the robot has remained 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.

3.4 Summary

This chapter has presented centralized SAFDetection as an applicable approach to detect both local and interactive faults in tightly-coupled multi-robot team tasks. However, the "server" of the robot team centralizes all the work, which results in potential problems with reliability and scalability. To detect faults in a robot team with a large number of members, a distributed approach is developed, presented and evaluated in Chapter 4. Results of the centralized SAFDetection are given in Chapter 5.

Chapter 4

Distributed SAFDetection

4.1 Overview of approach

The centralized approach I introduced in Chapter 3 has two potential shortcomings: robustness and scalability. Centralized SAFDetection uses a client/server model with message passing communication. The client robots send the original sensor data to the server robot and wait for the fault detection results. All the data is stored centralized and all the computation work is performed on the server. It is well known that this client/server model lacks reliability, especially when the critical server is one robot in the team. Secondly, the curse of dimensionality in clustering restricts the scalability of the centralized approach. Although PCA (Principal Components Analysis) can be used to reduce the dimension of robot sensor data, the feature dimension for the entire robot team still becomes large as the robot team size increases, resulting in decreased clustering performance. In addition, traffic congestion on the network and limited computation capabilities on the server robot also become issues when more clients join the team. A distributed approach that overcomes these problems is, therefore, introduced in this chapter. It offers a more reliable and extensible mechanism to make SAFDetection scalable for applications with increasing robot team size.

Distributed SAFDetection shares the theoretic foundation with centralized SAFDetection, which maps selected robot sensor data to robot state using a clustering algorithm, and builds state transition diagrams to describe the normal behavior of the robot system. However, rather than building the robot team's state directly from multiple robots' sensor data, the clustering algorithm is performed on individual robots to obtain the individual robot's states and the robot team's state is represented as a vector of individual robot states. Thus, the data dimension for clustering is limited to the number of features on a single robot and the curse of dimensionality is broken. In distributed SAFDetection, I use a peer-to-peer and client/server mixture model to distribute the computation work and failure risk: building of the individual robot state transition diagram and local fault detection are accomplished locally by each of the robots in the team; the team robot state transition diagram is built in a server robot and then distributed to all the client robots; the team fault is detected using a peer-to-peer mechanism as each robot keeps a copy of the team robot state transition diagram. To detect team faults in distributed SAFDetection, each robot should know

the latest state information of its teammates. A periodic and on-request mixed message transmission protocol is used to provide a balance between reliable information sharing and unnecessary message communication. With this mixed model, the original sensor data is stored locally and only robot states are transmitted among robots. Therefore, the network traffic is reduced. For example, consider the two-robot box pushing (see Chapter 5 for application-specific details): in the centralized SAFDetection, at the training stage, more than four hundred (400) original sensor data are collected and transmitted from the client robots to the server in each time step to build the robot team's normal behavior model; however, in the distributed SAFDetection, at the training stage, original sensor data is stored locally and mapped into individual robot's state. Thus, only two (2) cluster index values are transmitted to the "server" in each time step to build the robot team's normal behavior model. Similarly, in the centralized SAFDetection, more than four hundred (400) original real time sensor data are collected and transmitted from the client robots to the server to detect real time faults, while in the distributed SAFDetection, the local faults are detected locally by individual robots and only simple cluster index values are transmitted between the robots to update the entire team's real time state for detecting team faults. Thus, with the distributed approach, the network traffic load can be reduced significantly.

Since there is no specific "server" in the team, each robot can detect local faults by its local data and detect the interactive faults by message communication. Thus, the distributed SAFDetection is more robust compared to the centralized approach. However, since the data storage is distributed, data updating and information sharing are more difficult and critical. These issues are discussed further in the following sections.

4.2 Training stage

Similar to centralized SAFDetection, the distributed approach is also a training-classification based method. In the training stage, PCA (Principal Component Analysis) is performed on the history of each robot's local sensor data (i.e., training data) during normal operation to automatically select the essential features for individual robots. The selected feature data set is then mapped into a sequence of single robot's states using the Fuzzy C-means clustering algorithm (Euclidean distance is used). After that, the state transition diagrams are built with the sequence of robot states. In centralized SAFDetection, the team robot is regarded as a single monolithic robot and only one state transition diagram is generated to describe the normal behavior of the entire robot team. Distributed SAFDetection, on the other hand, learns two kinds of state transition diagrams in the training stage: the individual (local) one and the team (global) one (shown in Figure 4.1).

Compared to Figure 3.1, Figure 4.1 shows that in distributed SAFDetection, feature selection, clustering and local state transition diagram building are all accomplished on the client side locally, while only the team state transition diagram is built in the "server". Thus, the computational load is distributed efficiently. For example, the time complexity of clustering is $O(nkp)$ where n is the size of the data, k is the number of clusters and p is the data dimension. With the distributed approach, each robot R_i has p_i-dimensional features, giving the clustering complexity for each robot as $O(nkp_i)$, while the clustering

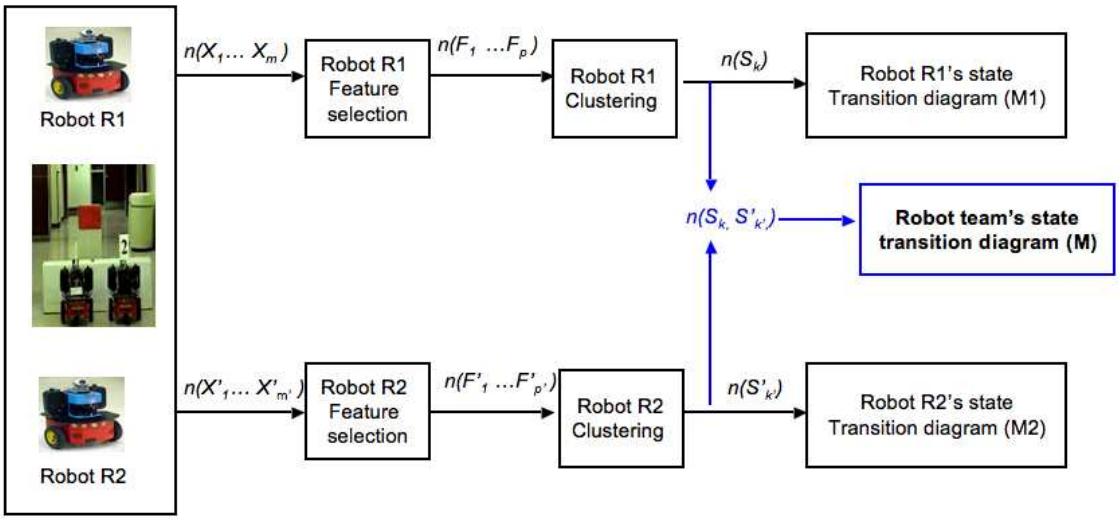


Figure 4.1: The training stage in distributed SAFDetection approach.

complexity for the server in the centralized approach is $O(nk(p_1 + p_2 + ... + p_n))$. More importantly, since the client robots perform clustering only with their local sensor data, the data dimension is limited and the curse of dimensionality is broken (see Chapter 5 for application-specific details).

The individual state transition diagram is built to describe the normal behavior of the single robot with its local sensor data. The steps to build the individual state transition diagram (Algorithm 3) are quite similar to Algorithm 1, except that only local sensor data is used.

The team state transition diagram is generated with a client/server model. The server (which can be either a robot or a PC) collects the individual states (achieved in Algorithm 3, step 3) from each client robot. The vector of those individual states is used to present the team state and the team state transition diagram is generated with the sequence of team robot states in the same way. The steps for building the team state transition diagram are shown in Algorithm 4.

Algorithm 4 shows that the team robot state is a vector of size n, where n is the size of the robot team. Suppose that client robot r_i has number c_i individual states. The possible number of team robot states is $c_1 \times c_2 \times ... \times c_i \times ... \times c_n$, which is of exponential magnitude ${c_{max}}^n$ (c_{max} is the maximum value of c_i). However, since only robot team states that exist when the robot team is in the normal tightly-coupled behavior are created, the final number of robot team states is much less than the theoretical possible number in real applications. In addition, given the increasing memory that one robot can have nowadays, even a very large matrix can be stored easily.

Algorithm 3 Steps for building the individual state transition diagram in distributed SAFDetection.

For every robot in the team
for each time step **do**
    Collect data from the robot local sensors during normal operation.
end for
Perform PCA on the history data and select features that cover sufficient (user defined) information.
Perform clustering algorithm on selected features to create individual robot states.
Build individual state transition diagram with the sequence of individual robot states.

Algorithm 4 Steps for building the team state transition diagram in distributed SAFDetection

For the server robot in the team
for each time step **do**
    Collect individual state index (S_{r_i}) from all n client robots in the team.
    Create vector $< S_{r_1}, S_{r_2}, ...S_{r_i}, ...S_{r_n} >$ as the team robot state.
end for
Build team state transition diagram with the sequence of team robot states.
Send the team state transition diagram to all n client robots.

For the client robot in the team
for each time step **do**
    Send individual state index (S_{r_i}) to the server robot.
end for
Save the team state transition diagram created by the server robot.

One issue about using a combined vector to represent the team state is that it results in some transient states because of the asynchronization among robots, which can cause false alarms during detection. For example, in one task, two robots in the same team are designed to move and stop at the same time. But in reality, one robot is a little bit later than the other robot. In the centralized SAFDetection, all robot sensor data is collected to perform clustering; and the small difference of speed is ignored given other correlated features are correct. However, with the distributed approach, the clustering is performed locally on the individual robot and the difference in speed is regarded as two states. When constructing the robot team state from the individual robot state, transient states such as <move, stop> and <stop, move> are generated. Two rules are used to reduce this side-effect: first, when building the team state transition diagram, time durations in each team state (combined from individual states) are checked and those that have a very short time duration (for example, only 1 second) are removed; secondly, in the detection stage, a two-layer alarm alert is used to report the final alarm. That is, only when the fault detected by SAFDetection lasts for some time will the final alarm be reported.

4.3 Detection stage

In the training stage, all individual state transition diagrams and the team state transition diagram are generated. Each robot in the team will keep its own individual transition diagram and the team state transition diagram as well. Therefore, each robot in the team knows the normal behavior pattern of itself and the normal behavior pattern of the entire team. In the detection stage, the individual state transition diagrams are used to detect local faults for the individual robot, while the team state transition diagram is used to detect interactive faults among robots (Figure 4.2).

Compared to Figure 3.1, Figure 4.2 shows that in the distributed SAFDetection, both local and team faults are detected by individual robots.

4.3.1 Information sharing and updating

To detect interactive faults using the team state transition diagram, each robot should know each teammate's state in real time. The critical information sharing and updating are implemented with a peer-to-peer model by passing messages. Initially, each robot broadcasts its current state to its teammates and every robot that receives the broadcast keeps a copy of the state. Each time one robot detects that its state has changed according to the sensor data, it sends the new state to its teammates again and all robots receiving the message will update their local copy of that state. For example, assume robots A and B are in one team. Robot A decides A's real time state with its own sensor data and keeps a local copy of robot B's state by receiving a message from B. When robot A detects a local state change, it sends the new state to robot B to update B's local copy of robot A's state. In the same way, robot B has a local copy of robot A's state and sends message to A when its own state has changed. Ideally, the local copy of robot B's state in robot A should be synchronized with robot B's real time state.

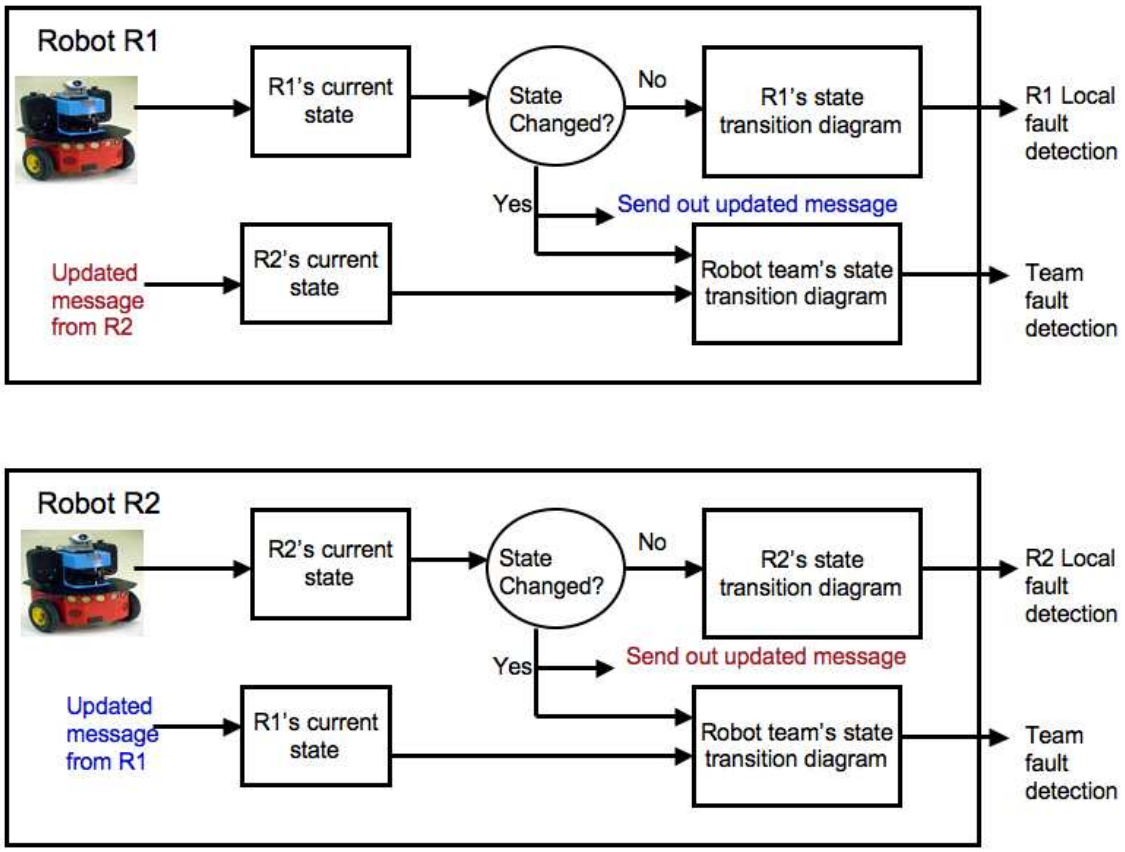


Figure 4.2: The detection stage of distributed SAFDetection approach.

To reduce the asynchronization caused by message transition time delay, broadcasting is used without acknowledgment or other hand-shaking protocols. The message format I use is <Update, sender id, receiver id, current state of sender>. Since no acknowledgment is required in this message passing protocol, the update message may be lost during transmission and the robot may have an expired state copy from its teammates. To make sure that the information is shared and updated correctly and efficiently, a periodic message will be sent. Therefore, two kinds of messages (with the same format) are transmitted among robots to update team information so that every team member knows the state of the entire team.

Each robot in the team broadcasts its current state to all other members in the same team with a particular frequency, which can be defined by the user according to different network situations. This broadcast message is called *periodic message*. When one robot detects a change of its own individual state, it broadcasts its new state to all other members in the same team. This broadcast message is called *update message*. When one robot receives a periodic or update message from its teammates, it compares the newly arrived state with the local copy. If they are different, the local copy is updated and the team state transition diagram is checked to detect possible interactive faults.

This message passing strategy provides a solution to the information sharing and updating problem in distributed SAFDetection that balances on reliability and efficiency. The periodic message reduces the message transmission loss risk; the update message reduces unnecessary communications among robots; and, the broadcast without acknowledgment cuts down on the message transmission time between robots. However, with a very unreliable network, the message loss will greatly affect the performance of fault detection. Suppose the average dropped message rate of the network is p, the periodic message is sent with frequency f and the message transmission time from robot A to its teammate robot B is t. Then the average time between when robot A detects a state change and robot B receives this information update is T, as given by Equation 4.1:

$$T = \sum_{n=1}^{m} (1-p) \times p^{n-1} \times [t + (n-1)/f] \tag{4.1}$$

where m counts how many times the update message is sent before robot B receives the update information. In theory, m can be infinity for any network whose average dropped message rate p is greater than 0 since there is always a non-zero probability that the message is dropped after m times. In real applications, I assume that messages are transmitted when the successfully transmitted probability is $\geq 99.99\%$. In most cases, the message transmission time t for a particular network is fixed. If the network is not reliable and the dropped message rate p is high, then a higher periodic frequency f is needed to obtain an acceptable T. On the other hand, the user can choose a lower periodic frequency with a very stable and reliable network. Table 4.1 shows some anticipated timings given particular dropped message rates and message delay requirements. Typically, an ad hoc network's message transmission time is from fifteen (15) to thirty (30) microseconds; a reliable network

Table 4.1: Periodic frequency in different situation.

Message transmission time t (ms)	**Average dropped message rate p**	**Periodic frequency f (Hz)**	**Message delay time T (ms)**
30	1%	0.2	79.49
30	1%	1	39.89
30	10%	1	140.69
30	10%	5	52.13
30	30%	1	458.02
30	30%	5	115.62
30	30%	10	72.79

has an average dropped message rate of less than 1%. If the average dropped message rate is greater than 30%, the network is very unstable.

4.3.2 Fault detection

Fault detection in distributed SAFDetection is quite similar to the centralized approach. The online sensor data is collected in a real time manner and filtered to select useful features according to the selected features in the learning stage. After that, the feature values are classified to a real time robot state according to the clustering information achieved during the learning stage. Then, this real time robot state will be checked with the state transition diagram to detect faults.

As previously described, distributed SAFDetection has two kinds of state transition diagrams: the individual one and the team one. Two kinds of faults can be detected correspondingly: the local faults and the interactive faults. For every time step, the individual robot maps its real time sensor data to the local state and compares it with the individual state transition diagram to detect local faults. The team state transition diagram is checked to detect interactive faults when the team state changes, which means either the local state has changed or it has received a state change message from its teammate. When checking the team state transition diagram, the team robot is regarded as a monolithic robot so the fault detection method is the same as for the individual robot. In addition, the periodic update message is checked to detect the communication problem between robots. The steps to detect faults are shown in Algorithm 5.

As previously noted, unlike the centralized SAFDetection, this distributed approach uses a peer-to-peer model to detect faults. Algorithm 5 shows that there is no server robot in the detection stage. The data collection, feature filtering, classification and local fault detection are completed on the local robot using its own data. The computational load is distributed efficiently, which improves the scalability and real time property of the system. In addition, only very simple data (i.e., state) is transmitted and the interactive faults are detected on the individual robots. Ideally, the state copy of robots in the same team should be updated synchronously and the interactive faults can be detected by all robots in the same team, thus improving the reliability of the system. In real applications, due to the

Algorithm 5 Steps for detecting faults with distributed SAFDetection

```
1: Every robot in the team needs to run this program
2: Initialize the local copy of teammates' state.
3: for each time step do
4:   if no update message received from teammate for time T (from Equation 4.1) then
5:     Communication fault happens and is detected.
6:   end if
7:   Collect real time data from the robot local sensors.
8:   Map the original sensor data to features that were selected in the learning stage.
9:   Classify the features into an individual robot state based on cluster centers achieved
     in the learning stage.
10:  Check the individual state transition diagram to detect local faults.
11:  if the individual state is different from the state in the last time step then
12:    Send update message to teammates.
13:    Check the team state transition diagram to detect team faults.
14:  else
15:    if it is the periodic time then
16:      Send periodic message to teammates.
17:    end if
18:  end if
19:  if receive message from teammate then
20:    Compare the newly arrived teammate's state to the local copy.
21:    if there is a change then
22:      Update the local copy.
23:      Check the team state transition diagram to detect team faults.
24:    end if
25:  end if
26: end for
```

network environment, the robots' states are not strictly synchronized, thus causing a higher false alarm rate than the centralized approach. Thus, there is a tradeoff between robustness, scalability and detection quality.

4.4 Summary

This chapter has presented distributed SAFDetection as an efficient approach to detect both local and interactive faults in tightly-coupled multi-robot team tasks. The motivation for building a distributed system is to increase the scalability by breaking the curse of dimensionality and by cutting down the network traffic. In addition, a peer-to-peer model is used to replace the client/server model in real time detection, which improves the reliability of the system. With the increasing scalability and reliability, there is a tradeoff in other issues caused by data sharing, data updating and transient state resulting from the asynchronous property of the robot interaction. A peer-to-peer message passing mechanism is designed to improve the reliable and efficient data synchronization and proper rules are used to reduce the side-effect caused by transient states. In Chapter 5, I compare and analyze the various characteristics of the centralized SAFDetection approach and the distributed SAFDetection approach.

Chapter 5

Experimental Validation

I have designed and implemented both the centralized and distributed version of SAFDetection. These approaches have been tested and evaluated with several multi-robot cooperative tasks on physical robots and in simulation. In this chapter, a cooperative multi-robot box pushing task and a multi-robot following task are tested to demonstrate the feasibility of both centralized and distributed SAFDetection. Simulation of the multi-robot following task is used to validate the robustness and scalability of the distributed approach. Performance is evaluated based on fault detection rate, false alarm rate and detection delay.

5.1 Experimental environment

The physical experiments are implemented on the ActivMedia Pioneer 3DX mobile robot (Figure 5.1) using a robot control server called Player [Gerkey et al., 2001]. The robot simulation is built with the multi-robot simulator Stage [Gerkey et al., 2003]. An ad hoc network provides the communication channel for the robots.

5.1.1 Pioneer 3DX

Pioneer 3DX is a commercial mobile robot that contains all of the basic components for sensing and navigation in a real-world environment. The CPU of the Pioneer robot is an Intel Pentium III processor with 850MHz speed. In my experiments, robot sensor data is collected and processed to build the normal robot behavior model. Here, the robot sensors include not only the environmental sensors, but also robot actuators and components that provide information about the robot system. The Pioneer robot sensors I use are shown in Table 5.1.

5.1.2 Player and Stage

Player is a network server that provides an interface to a collection of sensors and actuators constituting a robot. Many proxies are provided by Player to read data from sensors or to write commands to actuators. The proxies I use to collect robot sensor data in my experiments are shown in Table 5.2.

Figure 5.1: Pioneer 3DX mobile robot.

Table 5.1: Sensors on Pioneer 3DX mobile robots.

Sensors	**Description**
Laser	Range-finding laser (SICK LMS-200) in 180 degrees
Sonar	8 sonars on the front and 8 sonars on the back
Camera	Pan-Tilt-Zoom Canon VC-C4 communication camera
Wheel	Two-wheel differential drive and one caster
Battery	Three batteries provide 12-volt power

Table 5.2: Player proxies used in my experiments.

Proxy	**Data provided**
LaserProxy	Laser range in 180 degrees
SonarProxy	Sonar range in 16 directions
BlobfinderProxy	Blob position in the camera image
PositionProxy	Robot's speed and orientation
PowerProxy	Robot's current battery charge

The simulation experiments are performed using the Stage simulation environment, which can simulate a population of mobile robots, sensors and objects in a two-dimensional bit-mapped environment. Player is also used in the simulations to provide sensor data readings or to generate actuator commands for the client programs. In my experiments, Player version 1.6.5 and Stage version 1.6.2 are used.

5.1.3 Programming environment

The client side of Player supports C++, Tcl, Java, and Python. Both centralized and distributed SAFDetection are implemented in C++. To be compatible with the simulation, g++ version 3.4.6 is used as the compiler. Template matrix in Boost C++ Libraries is used to store the sensor data and state transition diagrams.

5.1.4 Experimental evaluations

In Chapter 3, I have shown that SAFDetection uses a state transition diagram with time factors to describe the normal behavior of the robot system. Thus, if a fault occurs on the robot, my approach will definitely detect it given enough time. To evaluate the performance reasonably, a successful detection should be a detection that is made within a limited time delay after the real fault happens. Given the various multi-robot tasks, it is very difficult to define the ground truth of a normal condition and fault. In my experiments, the "real faults" are decided by a person observing the experiments. If the observer finds a fault and SAFDetection does not detect it within a limited time, I interpret this to mean that the fault is missed by SAFDetection. On the other hand, if SAFDetection detects a fault while the observer determines that the robot system is running under a normal condition, I interpret this to mean that a false alarm is made by SAFDetection.

5.2 Cooperative multi-robot box pushing task

I have implemented both centralized and distributed SAFDetection on a physical Pioneer robot team performing a cooperative box pushing task. The box pushing task has been used by many researchers as a canonical problem in multi-robot systems [Donald et al., 1997] [Kube and Zhang, 1996] [Sen et al., 1994] [Mataric et al., 1995]. In this section, I illustrate how SAFDetection is used on this particular task to generate the normal behavior model and how to use the model to detect faults.

5.2.1 Box pushing task description

The cooperative multi-robot box pushing task requires multiple (two or more) robots to push a long box from a starting position to a goal position in a cooperative manner. There are several implementations to solve this problem. In my experiments, two Pioneer robots are used and the solution is made based on the following assumptions and strategies:

- The box is long enough for multiple team members to work on, and they are properly arranged on one side of the box at the beginning of the task.

- There is an obstacle-free path between the starting position and the goal position that is wide enough for the box and the robots to pass.
- The goal position is indicated by a red blob clearly visible to the robots.
- The robots use the laser to determine the distance and angle between the pushed box and themselves.
- The robots use a camera to track the goal position and guide the pushing direction.
- The robots adjust their speed and pushing directions to avoid losing the goal position.
- The robots communicate with each other to work in a cooperative way. The pushing speed and direction are adjusted to coordinate with the teammate's pushing activities.

The above assumptions and strategies provide prior knowledge to SAFDetection for feature selection. However, this knowledge is not required; feature selection can be made based on sensor data statistical analysis.

Figure 5.2 shows a series of snapshots taken during one run of the box pushing task. During this particular trial, both the left and right robots perform alignment to adjust their pushing directions. However, the alignment action is not required for the box pushing task; there are also successful trials during which none of the robots perform alignment. To train SAFDetection, twenty (20) normal box pushing trials with different situations (with or without alignment) are performed to collect more than one thousand (1000) training data entries.

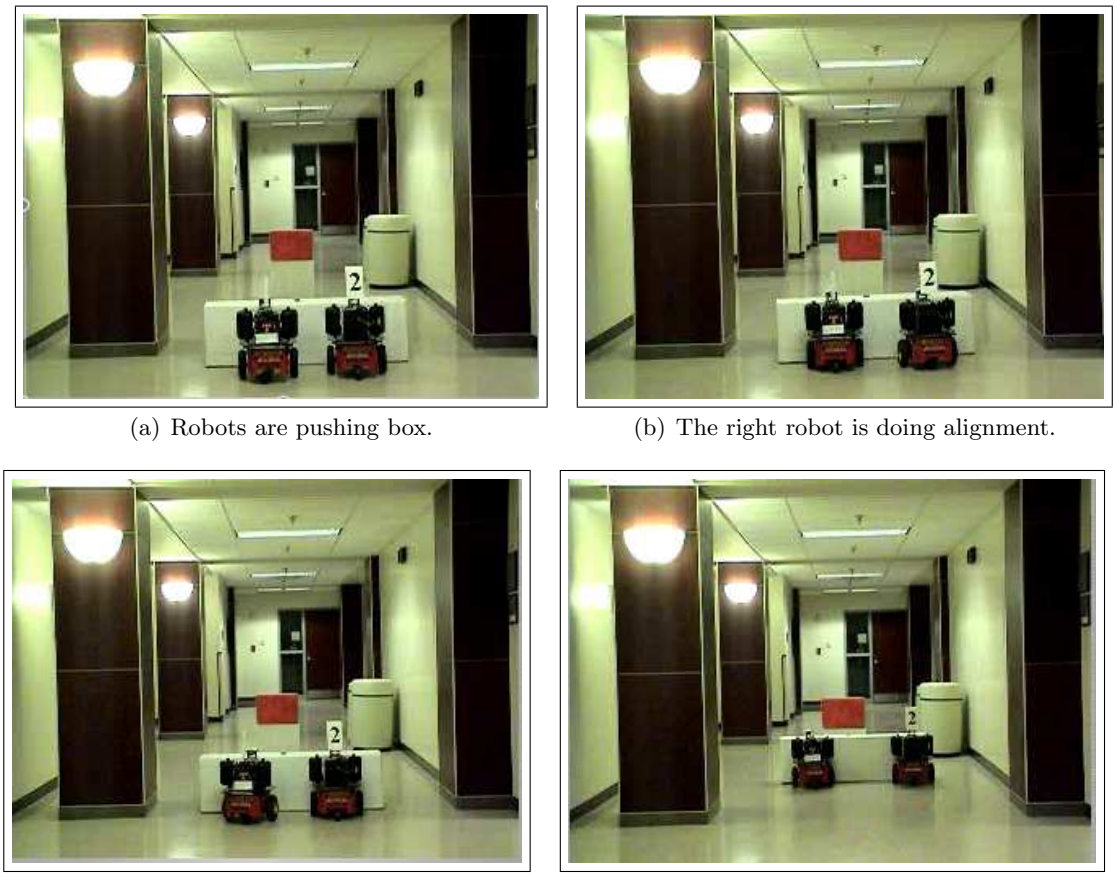
5.2.2 Box pushing model built by centralized SAFDetection

This sub-section illustrates how centralized SAFDetection is used to build the normal behavior model (state transition diagram) of the robot team performing the box pushing task. Manual feature selection and statistical based feature selection are compared. Three different clustering algorithms are also used in data clustering for comparison.

Feature selection

Both Pioneer mobile robots used in the experiment 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 in my approach. Clearly, there are many obviously unrelated pieces of information. For example, the color camera provides 320×180 (pixels) color pictures while only the red blob (goal position) in the picture is the useful information in this task. After discarding those obviously irrelevant features, the available sensor features to choose from are listed as follows:

- Laser range in 180 directions
- Sonar range in 16 directions
- Red blob's left, right, upper, bottom edge position in the camera image



(a) Robots are pushing box.

(b) The right robot is doing alignment.

(c) The right robot is doing alignment.

(d) Robots reach goal position.

Figure 5.2: A series of snapshots taken during the normal box pushing task. Two alignments occur during the pushing.

- Robot speed
- Robot turn rate
- Battery charge level

Manual feature selection is made dependent on the human supervisor's decision of how important or useful the features are for the task. Some features are determined by understanding the characteristics of the task; for example, the red blob information is relevant for indicating the goal position, and thus its size in the camera image is important for the task (the bigger red blob size means that the robot is getting closer to the goal position). Some features are chosen to provide rational redundancy; for example, both laser and sonar measure the distance between the robot and the objects around it. Some important features that can predicate faults are also selected; for example, the robot battery charge value is one essential feature because it affects all robot tasks if there is no backup power supply. Although some general rules (as listed above) can be defined for the human to select features, the result of manual feature selection still varies, depending on the supervisor's experience and knowledge of the robot system. Here, two manual selection results are made and their results are compared with statistical-selected features given twenty (20) normal test trials and twenty (20) test trials in which faults occur.

The first manual selection makes use of the following sixteen (16) features (sixteen (16) for each robot, with thirty-two (32) features for the robot team in total):

- Minimum laser range
- Laser index with minimum laser range
- Maximum laser range
- Laser index with maximum laser range
- Minimum sonar range
- Sonar index with minimum sonar range
- Maximum sonar range
- Sonar index with maximum sonar range
- Robot speed
- Robot turn rate
- 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
- Battery charge

The second manual selection makes use of the following eight (8) features (eight (8) for each robot, with sixteen (16) features for the robot team in total):

- Minimum laser range
- Laser index with minimum laser range
- Minimum sonar range
- Sonar index with minimum sonar range
- Robot speed
- Robot turn rate
- Red block's height in the camera image
- Battery charge

An alternative way to select feature uses statistical analysis methods, including PCA (Principal Components Analysis) and correlation analysis. This method assumes that only highly correlated data is considered as important and useful for the tightly-coupled task. In this method, PCA is used to compress the features returned from sensors that provide similar information and correlation analysis is used to select features returned from different sensors that are highly correlated with each other.

In the cooperative box pushing task, at first, PCA is performed on features returned from the same sensor to figure out the information provided by that sensor. For example, the "left", "right", "top" and "bottom" edge positions of the red blob in the camera image provide the basic information of the goal position. Figure 5.3 shows the original data of these four features. Table 5.3 shows the correlation among those four features. It is shown that the "left" edge position is highly correlated with the "right" edge position and the "top" edge position is highly correlated with the "bottom" edge position. Therefore, PCA can be performed to compress those four features.

Table 5.3: The correlation coefficients for the red blob's four features in the box pushing task.

Correlation Coefficient	**Left**	**Right**	**Top**	**Bottom**
Left	1	0.925	-0.013	-0.170
Right	0.925	1	-0.225	-0.078
Top	-0.013	-0.225	1	0.586
Bottom	-0.170	-0.078	0.586	1

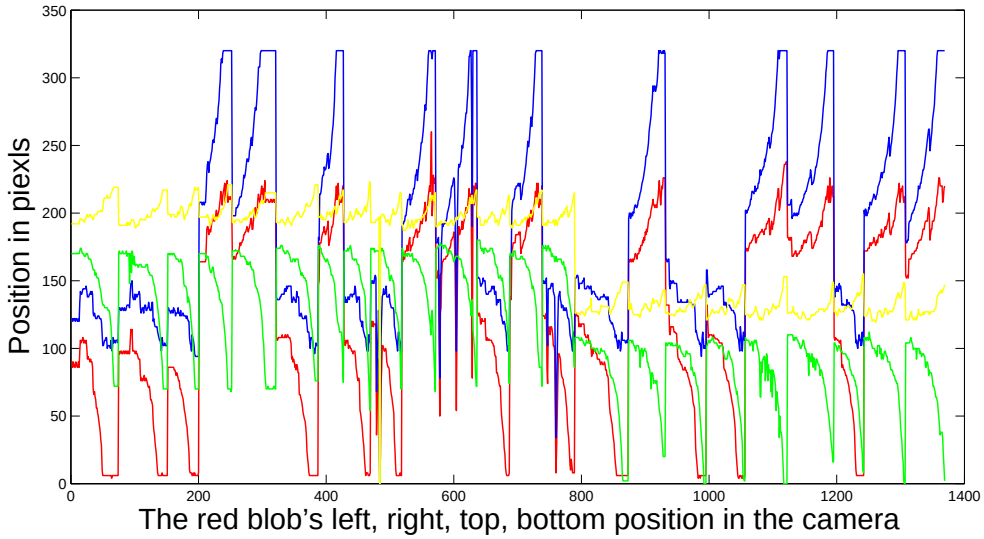


Figure 5.3: The red blob's edge position in the camera in box pushing task. Red line for left edge, Blue line for right edge, Green line for the top edge, Yellow line for the bottom edge.

The PCA results on these four features are shown in Figure 5.4. In this figure, the x axis refers to the number of the principal components, while the y axis represents the cumulative amount of information (variation) represented by the principal components from 1 through x. This figure illustrates that the first two principal components contain most of the information (over 80%); therefore, these two principal components are selected as the features to represent the camera sensor instead of the original four (4) features.

Secondly, PCA is performed on features returned from sensors that provide similar information. For example, the laser and sonar sensors both reflect the distance between the robot and the objects around it. Figure 5.5 shows the Principal Components of the 180 laser data and 16 sonar data values. Similar to Figure 5.4, the x axis refers to the number of the principal components and the y axis represents the cumulative amount of information (variation) represented by the principal components from 1 through x. It can be seen that the first eight (8) features include most of the information (over 80%) and can thus be selected as the features for representing the laser and sonar sensors.

In addition, correlation analysis is used to select related features between different sensors in one robot. For example, these eight (8) laser-sonar features represent the information about all objects around the robot. However, not all of these eight (8) features are related to the task; for example, in this task, only objects in front of the robot are important. In SAFDetection, I assume that only if the feature is highly correlated to the actuator sensor data (speed and turn-rate in this experiment), it is essential to the task and selected.

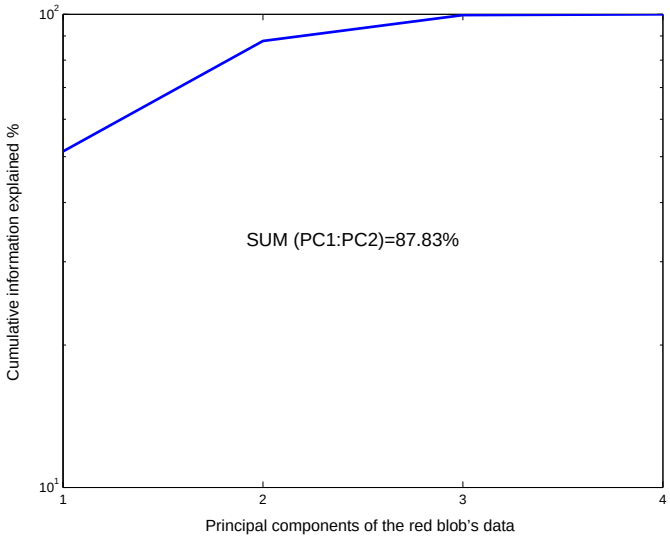


Figure 5.4: The PCA results of red blob data in box pushing task.

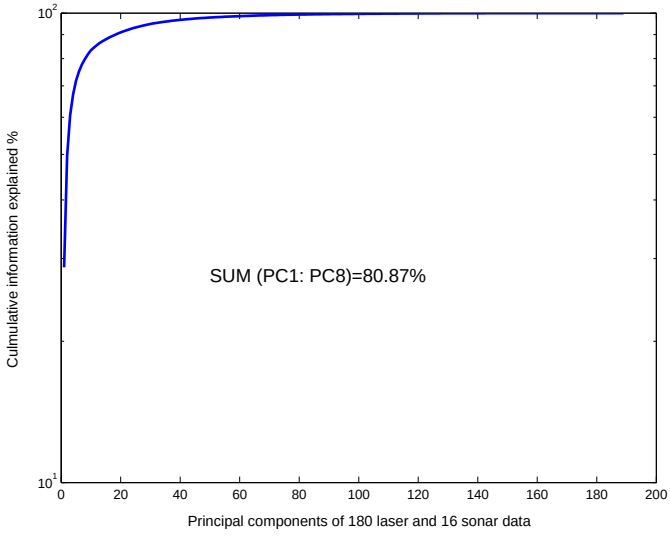


Figure 5.5: The PCA results of laser and sonar data in box pushing task.

Table 5.4 shows the correlation between the eight (8) features and the robot's speed and turn-rate. In this experiment, only feature PC1 has high correlation with the robot's speed feature and is therefore selected.

Finally, PCA is performed on the same sensors from different robots in the team to find the interactive relationship between the robots. Table 5.5 shows the correlation coefficient between the speed and turn-rate from both robots (R_1 and R_2) in the team. It can be seen that the speed from the Left (R_1) and Right (R_2) robots are highly correlated with each other.

Figure 5.6 shows the PCA results on those four features; the x axis refers to the number of the principal component and the y axis represents the cumulative amount of information (variation) represented by the principal components from 1 through x. This figure shows that the first three principal components contain most of the information (over 95%) and are therefore selected as the features that represent the speed and turn-rate of both robots. With the above analysis, ten (10) features are selected by PCA as follows:

- Two principal components of laser and sonar data (1 for each robot)
- Four principal components of red blob data (2 for each robot)
- Three principal components of speed and turn-rate for robot team
- Battery charge

Although the battery charge level is not related to other features, it is still selected since it is one essential feature and affects all robot tasks. In my experiments, this feature is used

Table 5.4: The correlation coefficient for laser-sonar PCs and actuator features in the box pushing task.

Correlation Coefficient	**PC1**	**PC2**	**PC3**	**PC4**	**PC5**	**PC6**	**PC7**	**PC8**
Speed	0.5277	-0.0537	0.0146	0.0234	0.1301	-0.0955	-0.0181	-0.0009
Turn-rate	0.0659	0.1039	0.0625	0.0842	0.0072	-0.0059	0.0302	0.0383

Table 5.5: The correlation coefficient for speed and turn-rate from both robots.

Correlation Coefficient	**R_1 speed**	**R_1 turn-rate**	**R_2 speed**	**R_2 turn-rate**
R_1 speed	1.0000	0.1814	0.8320	0.1053
R_1 turn-rate	0.1814	-1.0000	0.0676	-0.0262
R_2 speed	0.8320	0.0676	1.0000	0.0662
R_2 turn-rate	0.1053	–0.0262	0.0662	1.0000

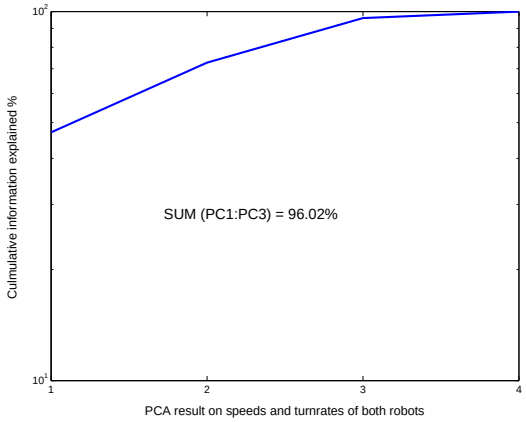


Figure 5.6: The PCA results of speed sensor in box pushing task.

as a check value instead of a clustering feature; if it deviates significantly from the mean value that was learned, a hard fault is detected.

Twenty (20) normal box pushing trials and twenty (20) trials in which faults occur are used to compare the fault detection rate of the two manual selection results and the statistically-selected results. Table 5.6 shows that statistical based selection achieves an almost equivalent fault detection rate compared to a good manual selection (i.e., the sixteen (16) manually selected features). It is also shown that the thirty-two (32) manually selected features is a bad manual selection that causes a decrease in performance. Comparing the two manually selected feature sets, it can be seen that the thirty-two (32) feature set includes some irrelevant features, such as the maximum laser range and laser index with maximum laser range (i.e., the robot does not care about far away objects in this task). These irrelevant features degrade the performance of clustering, resulting in a low fault detection rate (true positive) and a high false alarm rate (false positive). In SAFDetection, therefore, the statistical analysis based feature selection is used.

Data clustering

I have implemented three clustering algorithms — K-means, Soft K-means and Fuzzy C-Means — on the sensor features obtained based on statistical selection. In this experiment, nine of the ten statistically selected features (except for the battery charge level) are used in the clustering. As noted before, the battery charge value is almost a constant in the training data set and is used as a checking value instead of the clustering feature. Different cluster numbers (from two (2) to ten (10)) are tested with Equation 3.11 and five is chosen as the best cluster number for all three clustering methods. Table 5.7 shows the clustering centers resulting from three clustering algorithms.

When detecting faults, the online sensor data is classified into different clusters using these three algorithms. Table 5.8 shows the fault detection rates using the above three clustering algorithms based on twenty (20) normal trials and twenty (20) abnormal trials. It can be seen that the Soft K-means and Fuzzy C-means are effectively equivalent for this

Table 5.6: Fault detection rate for three feature selection methods.

	Manually selected (32)	**Manually selected (16)**	**Statistically selected (10)**
True positive	40%	85%	85%
True negative	55%	90%	95%
False positive	60%	15%	15%
False negative	45%	10%	5%

Table 5.7: Cluster centers resulting from three different clustering algorithms.

	Cluster	**PC1**	**PC2**	**PC3**	**PC4**	**PC5**	**PC6**	**PC7**	**PC8**	**PC9**
K-means	C1	0.478	-0.560	-1.455	0.507	-1.375	-0.320	-2.751	-0.641	0.223
	C2	-4.537	4.646	-0.738	0.854	-1.067	-0.567	-2.135	-1.134	0.798
	C3	0.397	-0.461	2.334	-0.487	2.281	0.109	4.562	0.217	-1.035
	C4	0.311	-0.425	0.485	1.512	0.303	-1.651	0.606	-3.302	-0.059
	C5	0.430	-0.472	-0.487	-1.634	-0.392	1.593	-0.785	3.187	0.068
Soft	C1	0.392	-0.323	1.741	-0.591	1.892	0.434	3.785	0.868	0.162
K-means	C2	0.447	-0.359	-0.341	-1.720	-0.311	1.700	-0.623	3.401	0.163
	C3	-4.392	4.672	-0.624	0.597	-0.905	-0.373	-1.810	-0.747	0.911
	C4	0.068	-0.089	0.702	1.520	0.496	-1.700	0.993	-3.400	-0.430
	C5	0.393	-0.555	-1.577	0.317	-1.509	-0.104	-3.019	-0.208	-0.146
Fuzzy	C1	-4.270	4.481	-0.641	0.805	-0.949	-0.525	-1.899	-1.050	0.757
C-means	C2	0.573	-0.554	1.676	-0.609	1.678	0.207	3.357	0.414	0.042
	C3	0.256	-0.388	0.569	1.491	0.385	-1.633	0.771	-3.267	-0.172
	C4	0.428	-0.446	-0.393	-1.573	-0.300	1.536	-0.601	3.073	0.153
	C5	0.464	-0.548	-1.446	0.488	-1.367	-0.299	-2.734	-0.599	0.196

Table 5.8: Fault detection rates of three clustering methods.

	K-means	**Soft K-means**	**FCM**
True positive	45%	80%	85%
True negative	90%	95%	95%
False positive	55%	20%	15%
False negative	10%	5%	5%

application, with both giving good results. However, the crisp K-means clustering algorithm results in the most false positive errors, and in a lower accuracy for true positives. These errors are primarily caused by the misclassification of noisy and partial data. In SAFDetection, therefore, the Fuzzy C-means clustering algorithm is used.

Building state transition diagram

With the five clusters in Table 5.7, each standing for one robot team state, SAFDetection builds the state transition diagram of the robots as shown in Table 5.9. The "start" and "end" states in the diagram are added manually to indicate the beginning and ending of the task. By examining the cluster centroid value, these five clusters are labeled as "push" and four "alignment" states (alignment may happen in two different directions –left and right for both robots).The diagram shows that the robot primarily switches between the "push" state and "alignment" state during this task, the average state transition probabilities between these states and the mean and standard deviation values of the time the robots remain in each state (i.e, before transiting to a different state). It can be seen that for most of the time, the robot team moves forward, while occasionally different robots realign in different directions.

5.2.3 Box pushing model built by distributed SAFDetection

This section shows how distributed SAFDetection is used to build the normal behavior model (state transition diagram) in the box pushing task. Results are compared with the centralized approach.

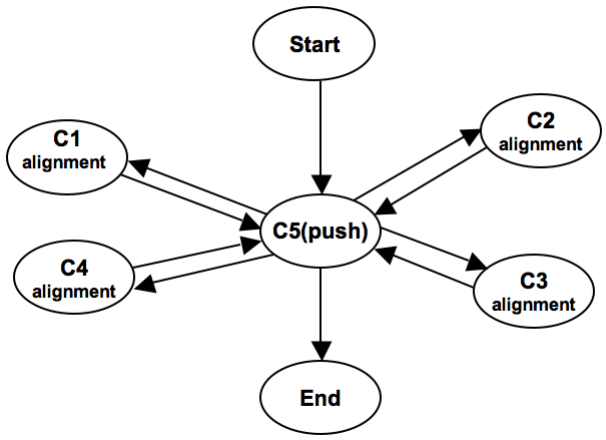
Feature selection

Instead of collecting and selecting features from the entire team's sensor data as the centralized approach does, in distributed SAFDetection statistical based feature selection is performed on individual robots' sensor data in the team to determine the useful features, meaning that PCA is not performed on sensor data from multiple robots. Similar to the centralized approach, sensor data from the same or different sensor(s) in one robot are analyzed to determine the essential features. According to the results shown in Figure 5.4, Figure 5.5 and Table 5.4, six (6) features are selected for each robot in the team as follows:

- One principal component of laser and sonar data
- Two principal components of red blob data
- Robot speed
- Robot turn rate
- Battery charge

Table 5.9: Normal behavior model for the entire robot team in box pushing task.

(a) State transition diagram.



(b) State transition probabilities between states.

State	Start	C1	C2	C3	C4	C5	End
Start	0	0	0	0	0	1	0
C1	0	0	0	0	0	1	0
C2	0	0	0	0	0	1	0
C3	0	0	0	0	0	1	0
C4	0	0	0	0	0	1	0
C5	0	0.16	0.22	0.18	0.26	0	0.18
End	0	0	0	0	0	0	0

(c) Duration time in each state.

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

Data clustering

Fuzzy C-Means clustering is performed on the sensor features obtained using PCA for the individual robot. Different cluster numbers (from two (2) to ten (10)) are tested using Equation 3.11 and four is chosen as the best cluster number for both robots in the team.

Table 5.10 shows the resulting cluster centers for the left robot. Here, state C1 stands for moving forward; state C2 stands for alignment left; state C3 stands for alignment right and state C4 stands for pause.

Table 5.11 shows the cluster centers for the right robot. Here, state C1 stands for moving forward; state C2 stands for alignment right; state C3 stands for alignment left, and state C4 stands for pause.

The robot team state is represented by a vector combination of the individual robot states, which is <left robot's state, right robot's state> in this box pushing task with two robots. Table 5.15 shows the robot team states and their meanings for this task. It does not show all possible combinations, but only the combinations that exist in the training data. In theory, there exist sixteen (16, as 4×4) combinations; however, only five (5) of these exist in the training data (two transient states, (C1, C2) and (C3, C1), are removed by checking the state time duration as noted in Chapter 4).

Building state transition diagram

Two kinds of state transition diagram are built with the distributed SAFDetection – one is the local state transition diagram for the individual robot, and the other is the global state transition diagram for the robot team. The individual state transition diagram of the left robot in the box pushing task is shown in Table 5.12. It shows the average state transition

Table 5.10: Cluster centers of the left robot in the box pushing task.

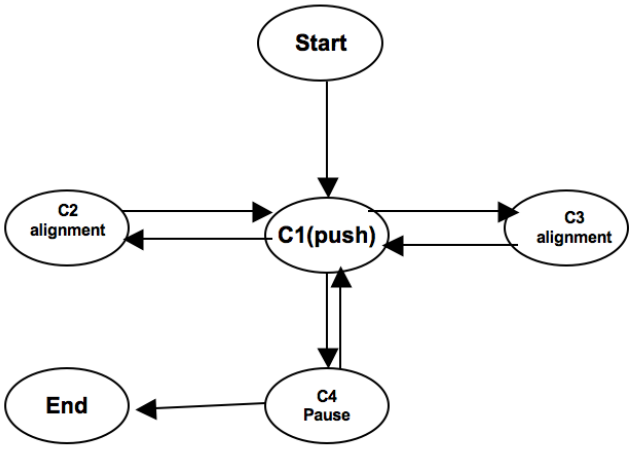
	PC1	**PC2**	**PC3**	**Speed**	**Turn-rate**	**Meaning**
C1	-3.9223	-0.5442	0.6510	0.0934	-0.0042	Moving forward
C2	0.4563	-0.4021	-1.7926	-0.0418	-0.0552	Alignment left
C3	0.3987	-1.5719	0.3304	-0.0531	0.0743	Alignment right
C4	0.2352	0.7483	1.4905	0.0063	-0.0026	Pause

Table 5.11: Cluster centers of the right robot in the box pushing task.

	PC1	**PC2**	**PC3**	**Speed**	**Turn-rate**	**Meaning**
C1	-4.2696	-0.5486	0.6325	0.1005	0.0049	Moving forward
C2	0.3284	-1.1421	0.6704	-0.0640	0.0527	Alignment right
C3	0.4407	-0.5440	-1.7055	-0.0521	-0.0846	Alignment left
C4	0.2701	1.6402	0.0737	0.0041	0.0080	Pause

Table 5.12: Normal behavior model for the left robot in box pushing task.

(a) State transition diagram.



(b) State transition probabilities between states.

State	Start	C1	C2	C3	C4	End
Start	1	0	0	0	0	0
C1	0	0	0.14	0.28	0.58	0
C2	0	1	0	0	0	0
C3	0	1	0	0	0	0
C4	0	0.75	0	0	0	0.25
End	0	0	0	0	0	0

(c) Duration time in each state.

State	C1	C2	C3	C4
Mean time (s)	28.2	3.56	4.12	4.50
Standard derivation (s)	7.97	0.40	0.23	0.38

probabilities between states and the mean and standard deviation values of the time the robot remained in each state (i.e, before transiting to a different state). The diagram and tables show that for most of the time, the left robot moves forward, while occasionally performing alignment in different directions or pausing.

The state transition diagram of the right robot in the box pushing task is shown in Table 5.13. It is very similar to the state transition diagram of the left robot. It also shows the average state transition probabilities between states and the mean and standard deviation values of the time robot remained in each state (i.e, before transiting to a different state). The diagram and tables show that for most of the time, the right robot moves forward, while occasionally performing alignment in different directions or pausing. It can be seen that the individual robots' behaviors are quite similar in this task, which is consistent with the fact that the two robots are interchangeable in this task.

For the purposes of comparison, Figure 5.7 shows the state transition diagram representing the actual robot control code for this task, which has a structure similar to the state transition diagram learned for the individual robot. The two "Alignment" states in Table 5.12 and Table 5.13 are the representation of the single "Alignment" state in Figure 5.7, but with different alignment directions. Unlike Figure 5.7, there is no transition from 'initialize (start)" to "alignment" or "wait" in Table 5.12 or Table 5.13. That is because the motions "alignment" and "wait" always happen after "push" in the real experiments.

In distributed SAFDetection, the robot team state is obtained by combining the individual robot states in the team. The state transition diagram of the robot team in the box pushing task is built with the team states in Table 5.15, as shown in Table 5.14. It also shows the average state transition probabilities between states for the robot team and the mean and standard deviation values of the time the robot team remained in each state (i.e., before transiting to a different state). The diagram and tables show that for most of the time, the team robot moves forward, while occasionally different robots realign in different directions. This normal team behavior model generated by distributed SAFDetection is consistent with the normal team behavior model generated by the centralized approach.

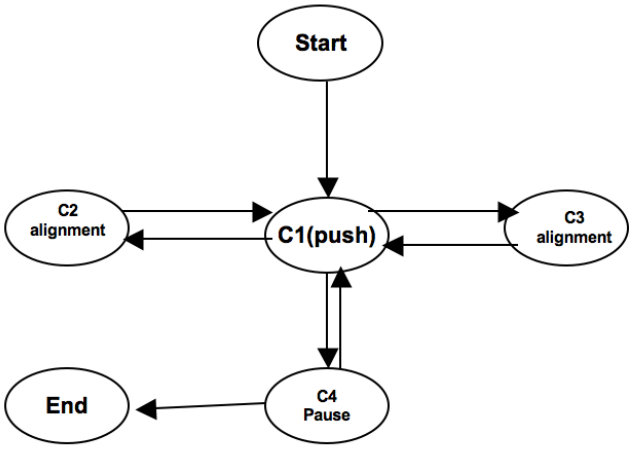
5.2.4 Box pushing task monitored by SAFDetection

These results show that the state transition diagram generated by centralized SAFDetection (Table 5.9) and the robot team state transition diagram generated by distributed SAFDetection (Table 5.14) are quite similar. Both centralized and distributed SAFDetection detect faults efficiently but for different reasons. An ad hoc network provides the communication channel for the robots in our laboratory. The network's message transmission time is thirty (30) microseconds and the average dropped message rate is less than 1%. Thus, according to Table 4.1, a periodic frequency of 0.2 Hz is used in distributed SAFDetection to keep the average message delay time within 100 microseconds.

Figure 5.8 illustrates how the centralized and distributed SAFDetection detect the fault when one of the robots becomes stuck. In this scenario, the right robot gets stuck and the left robot keeps moving forward as shown in the pictures. The centralized SAFDetection detects the "stuck" fault because the server detects an unknown state; with FCM clustering, this means that the clustering result of the robot team's sensor data does not have a high

Table 5.13: Normal behavior model for the right robot in box pushing task.

(a) State transition diagram.



(b) State transition probabilities between states.

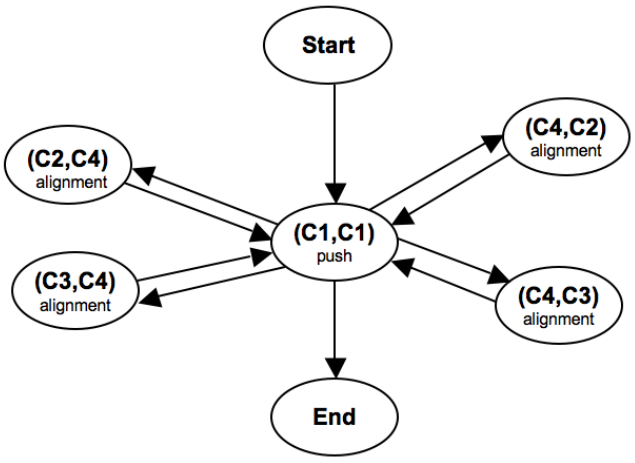
State	Start	C1	C2	C3	C4	End
Start	1	0	0	0	0	0
C1	0	0	0.26	0.24	0.40	0
C2	0	1	0	0	0	0
C3	0	1	0	0	0	0
C4	0	0.70	0	0	0	0.30
End	0	0	0	0	0	0

(c) Duration time in each state.

State	C1	C2	C3	C4
Mean time (s)	28.4	4.56	3.82	3.90
Standard derivation (s)	7.64	0.22	0.43	0.36

Table 5.14: Normal behavior model for the entire robot team in box pushing task with distributed approach.

(a) State transition diagram.



(b) State transition probabilities between states.

State	Start	(C1,C1)	(C2,C4)	(C3,C4)	(C4,C2)	(C4,C3)	End
Start	0	1	0	0	0	0	0
(C1,C1)	0	0	0.20	0.18	0.28	0.18	0.16
(C2,C4)	0	1	0	0	0	0	0
(C3,C4)	0	1	0	0	0	0	0
(C4,C2)	0	1	0	0	0	0	0
(C4,C3)	0	1	0	0	0	0	0
End	0	0	0	0	0	0	0

(c) Duration time in each state.

State	(C1,C1)	(C2,C4)	(C3,C4)	(C4,C2)	(C4,C3)
Mean time (s)	28.8	2.56	4.42	4.53	3.91
Standard derivation (s)	7.92	0.72	0.23	0.68	0.79

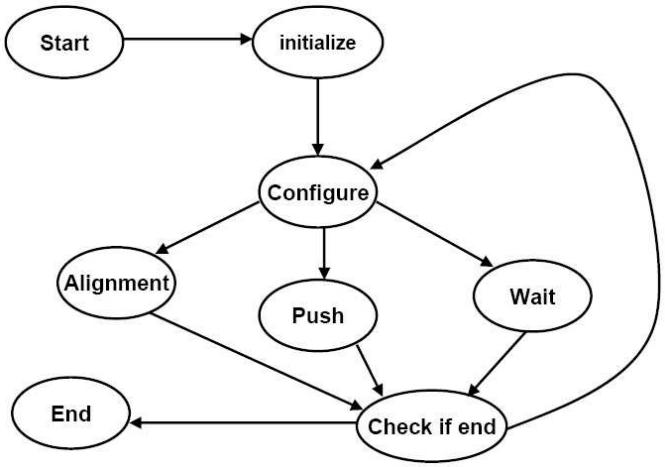
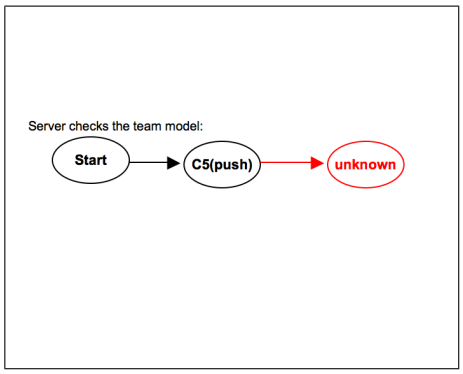


Figure 5.7: Actual algorithmic control code for individual robot in the box pushing.

Table 5.15: Robot team states and meanings in the box pushing task with distributed approach.

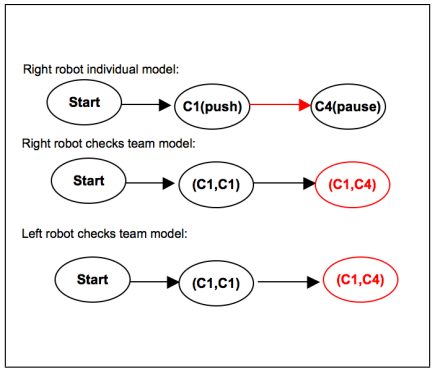
Team state	**Meaning**
(C1,C1)	Both robots are moving forward
(C2,C4)	The left robot is performing alignment in left direction
(C3,C4)	The left robot is performing alignment in right direction
(C4,C2)	The right robot is performing alignment in right direction
(C4,C3)	The right robot is performing alignment in left direction

(a) The right robot gets stuck during pushing.



(b) Centralized approach: the server robot detects the fault: unknown robot team state found.

(c) The right robot gets stuck during pushing.



(d) Distributed approach: both robots detect the fault: conflict with global state model.

Figure 5.8: A series of snapshots taken during the box pushing task. The right robot gets stuck during pushing.

probability of being in any of the known clusters. In the distributed SAFDetection approach, when the right robot gets stuck, it detects this state change (from "push" to "pause") by clustering the local sensor data and comparing the current robot team state (state (C1, C4)) with the team state transition diagram. Then it checks the robot team's normal behavior model and the "stuck" fault is detected since the current robot team's state is unknown in the team state transition diagram. In addition, the right robot also broadcasts its state change to its teammate, causing the left robot to also detect the "stuck" fault by checking the team state transition diagram.

Another fault example, "blob missing" is shown in Figure 5.9. In this scenario, the red blob that indicates the goal position is taken away during the pushing process. The centralized SAFDetection detects the fault by observing an unknown team state by clustering the team robots' sensor data. In the distributed SAFDetection, both the left and right robots detect the fault by clustering its local sensor data.

Figure 5.10 shows how the centralized and distributed SAFDetection detect the fault when the network communication meets a problem. In this scenario, the communication between the robots is disabled, as well as the communication between the client robots and the server. Thus, when the left robot performs alignment, the cooperative request to its teammate is blocked and the right robot still moves forward. The centralized approach detects the fault because the server does not receive the sensor data from client robots. In the distributed approach, when the left robot changes its state from "push" to "alignment", it checks the team state transition diagram and finds an unknown team state (C2, C1); thus, the communication problem is detected.

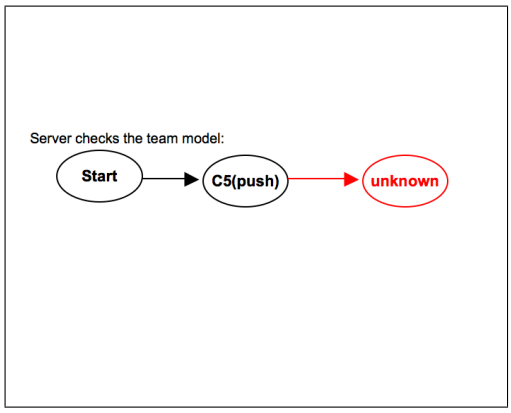
The individual state transition diagram can be used to detect local faults. For example, in the box pushing process, if the red blob is removed, both robots begin wandering because of losing the goal. In this case, the fault can be detected by individual robots in a short time. Figure 5.11 shows the minimum laser index, minimum laser range, and turn rate features of one robot in this test.

At time 27, the red blob was removed and the robot started to wander. At time 29, the individual robot 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.

Another issue of SAFDetection is whether the training data is sufficient. Of course, the SAFDetection approach is only practical if it can achieve good results without requiring a large number of learning trials. To study this issue, SAFDetection builds the individual robot's normal behavior model to detect the "stuck" faults using a variable number of learning trials. Figure 5.12 shows the experimental results. 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.

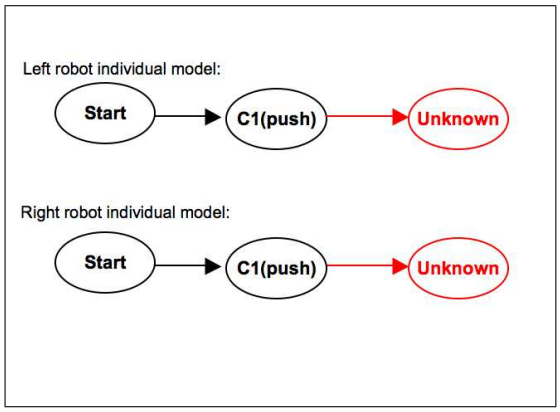
However, in many situations, more faults can be detected or be detected earlier by monitoring the entire robot team than monitoring individual robots separately. For example, in the box pushing experiment, when one of the robots gets stuck, the individual robot

(a) The red blob is missing.



(b) Centralized approach: the server robot detects the fault: unkown robot team state found.

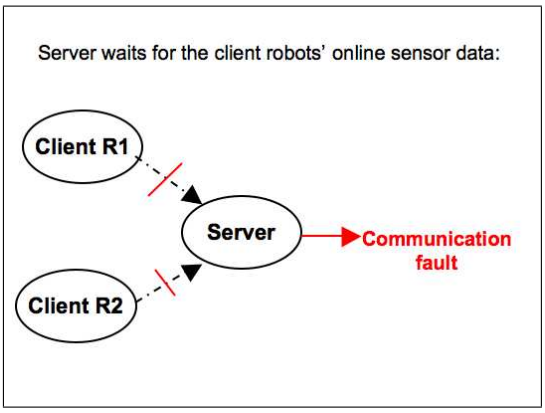
(c) The red blob is missing.



(d) Distributed approach: Both the left and right robots detect the fault: individual new state found.

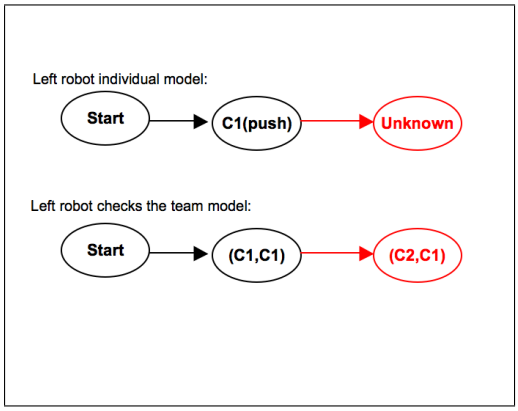
Figure 5.9: A series of snapshots taken during the box pushing task. The red blob is missing.

(a) Communication problem.



(b) Centralized approach: the server robot detects the fault: no online sensor data is received from the client robots.

(c) Communication problem.



(d) Distributed approach: Left robot changes individual state and detects the fault: conflict with global state model.

Figure 5.10: A series of snapshots taken during the box pushing task. Communication between robots is disabled.

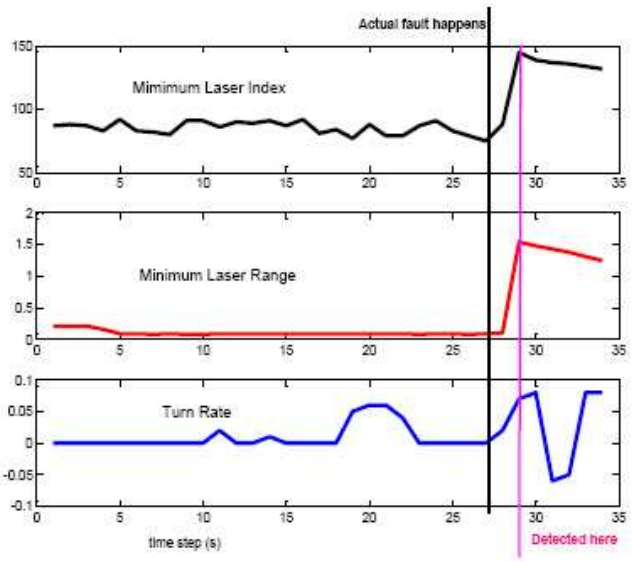


Figure 5.11: Single robot's sensor data when losing the goal during box pushing task.

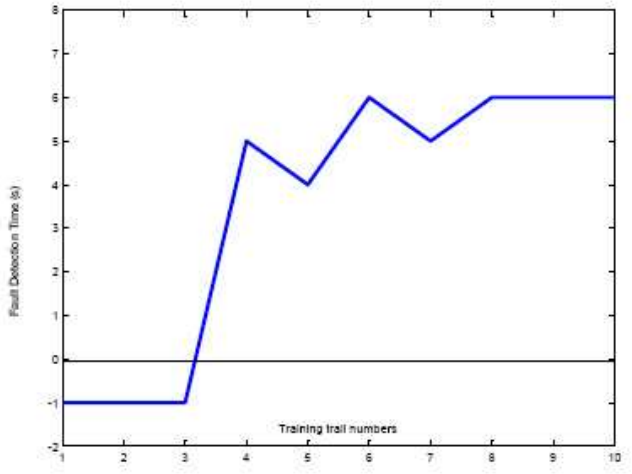


Figure 5.12: Stuck fault detection time (i.e., time to detect fault after it occurs) as a function of number of training trials.

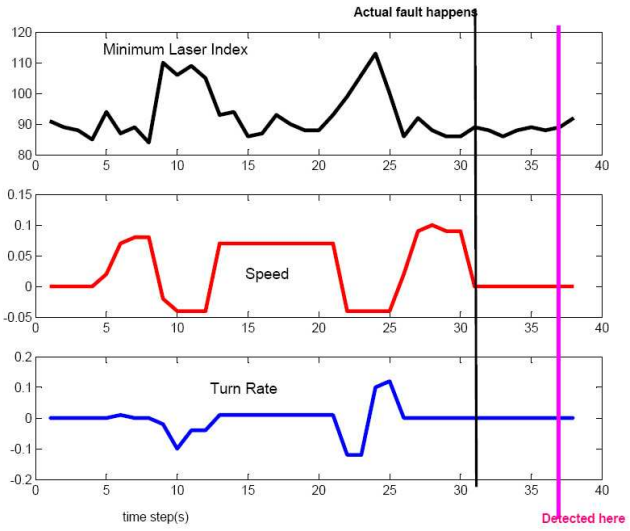


Figure 5.13: Individual robot's sensor data when the robot gets stuck.

state transition model can detect the fault in 6 seconds. Figure 5.13 shows the minimum laser index, speed and turn rate features of the stuck robot in this test. At time 31, the robot is jammed. The consequence was that the robot entered the "pause" state and then remained in that state. At time 37, the SAFDetection approach noticed that the robot had remained in that state for 6 seconds, which is an unusually long time compared to the learned data. Thus, SAFDetection made a judgment that the robot was stuck in the "pause" state, yielding the detection of a logic fault at time 37.

Another example is the communication disabled fault in the box pushing task. By monitoring the entire team, the fault can be detected immediately. Figure 5.14 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.

These three examples show that both the centralized and distributed approach can detect faults efficiently. The centralized approach detects fault using the clustering result of the team robots' sensor data. The distributed approach distributes the computational load and sometimes can detect faults with only local information (e.g., the blob missing case). Another advantage of the distributed approach is that faults can be detected by more than one robot in the team, instead of only the server. Table 5.16 compares the fault detection rates of centralized and distributed SAFDetection with twenty (20) normal trials and twenty (20) abnormal trials. It can be seen that distributed SAFDetection has a little higher false alarm rate since the transient state, as a result of the asynchronism between the robots, is detected as a fault. Therefore, the choice of which SAFDetection approach to use is dependent on the user's requirement. In general, there is no preferred choice when

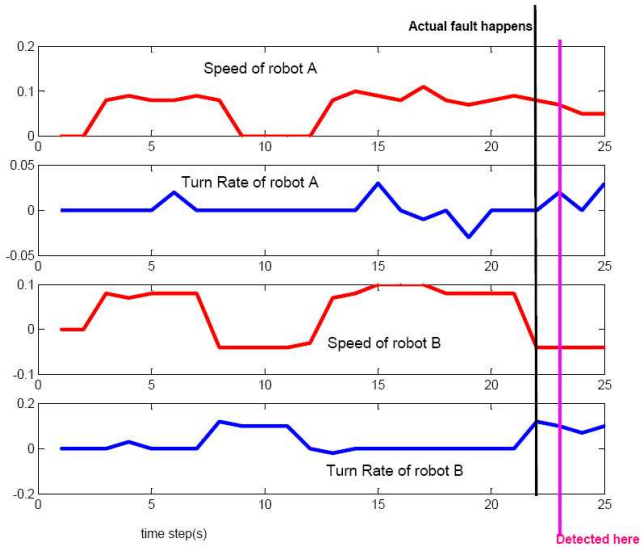


Figure 5.14: Team sensor data with robot team interactive fault.

Table 5.16: Fault detection rates by centralized and distributed SAFDetection

	Centralized SAFDetection	**Distributed SAFDetection**
True positive	85%	80%
True negative	95%	95%
False positive	15%	20%
False negative	5%	5%

the team size is small (two in this case). Experiments with a larger robot team size are discussed in Section 5.3.5.

5.3 Multi-robot following task

I have implemented both centralized and distributed SAFDetection on a physical Pioneer robot team performing a multi-robot following task. In this section, I illustrate how SAFDetection is used in this particular task to generate the normal behavior model and how the model is used to detect faults.

5.3.1 Task description

In the multi-robot following task, multiple (two or more) robot team members move from a starting position to a goal position, one followed by another. There are several implementations to solve this problem. In my experiments, two Pioneer robots are used and the solution is made based on the following assumptions and strategies:

- The start and goal positions are defined in a global coordinate reference frame by the leading robot.
- The leading robot has the ability to navigate from the starting position to the goal position.
- There is an obstacle-free path between the starting position and the goal position that is wide enough for the robots to pass.
- The leading robot has a red blob which is clearly visible for the following robot.
- The following robots use a camera to track the red blob of the previous robot.
- The following robot adjusts its speed to avoid losing the red blob.

The above assumptions and strategies provide prior knowledge to SAFDetection that is useful for feature selection. However, this knowledge is not required, in which case feature selection can be made based on sensor data statistical analysis.

Figure 5.15 shows a series of snapshots taken during one run of the robot following task. During this particular trial, the robot team needs to turn right to pass the corner. However, the turning action (left or right) is not a requirement for the following task; there are also successful trials during which the robots only move straight ahead. To train SAFDetection, twenty (20) normal following trials with different situations (with or without turning) are performed to collect more than one thousand (1000) training data entries.

5.3.2 Robots following model built by centralized SAFDetection

This section illustrates how centralized SAFDetection is used to build the normal behavior model (state transition diagram) of the robot following task.

(a) Robots are moving to the goal position.

(b) The leading robot is turning around the corner.

(c) The following robot is turning around the corner.

(d) Robots reach goal position.

Figure 5.15: A series of snapshots taken during the normal robots following task.

Feature selection

Both Pioneer mobile robots used in the experiment 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, there is much obviously unrelated information. For example, the color camera of the leading robot provides a 320×180 pixel color picture, but all this information is not useful information in this task. After discarding the obviously irrelevant features, the available sensor features chosen are listed as follows:

- Laser range in 180 degrees (both robots)
- Sonar range in 16 degrees (both robots)
- Red blob's left, right, upper, bottom edge position in the follower robot's camera image
- Robot speed (both robots)
- Robot turn rate (both robots)
- Battery charge level (both robots)

The statistical analysis based method, with PCA (Principal Components Analysis) and correlation analysis, is performed to select essential features in this task. First, PCA is performed on the red blob's data returned from the follower robot's camera. Figure 5.16 shows the original data of four edge positions and Table 5.17 shows the correlation among those four features. It can be seen that the "left" edge position is highly correlated with the "right" edge position, so as "top" edge and "bottom" edge position. Therefore, PCA can be used to selected compress those four (4) features.

The PCA results on those four features are shown in Figure 5.17. In this figure, the x axis refers to the number of the principal component, while the y axis represents the cumulative amount of information (variation) represented by the principal components from 1 through x. From this figure, we can see that the first two principal components contain most of the information (over 95%); therefore, these two principal components are selected as the features to represent the camera sensor instead of the original four (4) features.

Table 5.17: The correlation coefficient for red blob's four features in the robot following task.

Correlation Coefficient	**Left**	**Right**	**Top**	**Bottom**
Left	1	0.7074	0.1704	-0.0914
Right	0.7074	1	-0.5054	-0.5795
Top	0.1704	-0.5054	1	-0.8411
Bottom	-0.0914	-0.5795	-0.8411	1

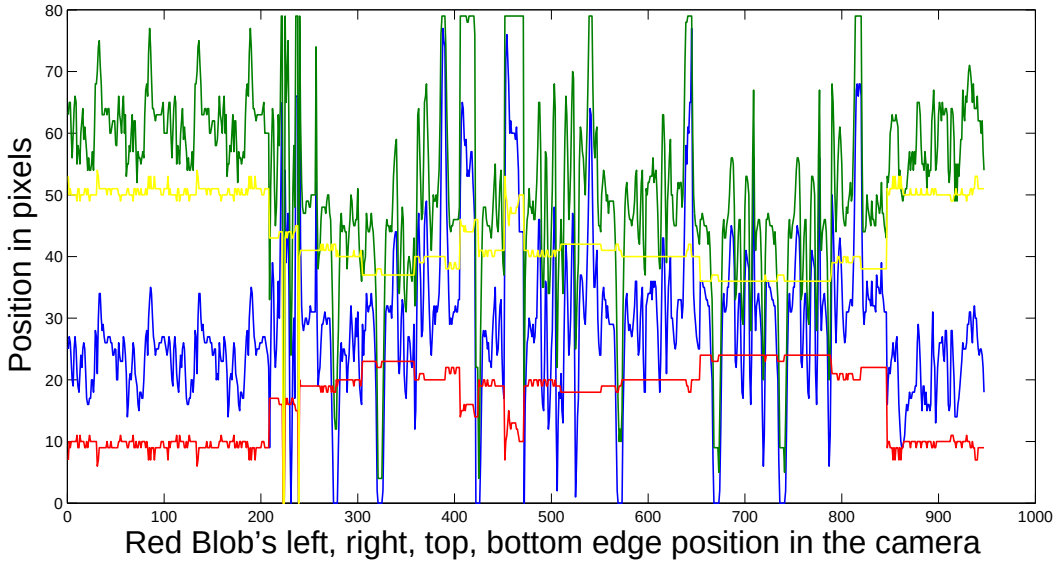


Figure 5.16: The red blob's edge position in the camera for the following robot. Blue line for left edge, Green line for right edge, Red line for the top edge, Yellow line for the bottom edge.

Secondly, PCA is performed on the the robots' laser and sonar sensors that both reflect the distance between the robot and the objects around it. Figure 5.18 shows the Principal Components of the leader robot's 180 laser data and 16 sonar data values; the x axis refers to the number of the principal components and the y axis represents the cumulative amount of information (variation) represented by the principal components from 1 through x. It can be seen that the first eight (8) features include most of the information (over 80%) and can thus be selected as the features for representing the laser and sonar sensors of the leader robot. In the same way, Figure 5.19 shows the Principal Components of the follower robot's 180 laser data and 16 sonar data values. It can be seen that the first twelve (12) features include most of the information (over 80%) and can thus be selected as the features for representing the laser and sonar sensors of the follower robot.

To determine if the laser and sonar features is related to the task, correlation analysis is then performed on the features with robot's speed and turn-rate. Table 5.18 shows the correlation among the leader robot's eight (8) features and its speed and turn-rate. For the leader robot, only feature PC2 has high correlation with the robot's turn-rate feature, and is therefore selected. Table 5.19 shows the correlation between the follower robot's twelve (12) features and its speed and turn-rate. For the follower robot, only feature PC3 has high correlation with the robot's turn-rate feature, and is therefore selected.

In summary, nine features are selected for the following task with the centralized approach, as follows:

- Leader robot's one principal component of laser and sonar data

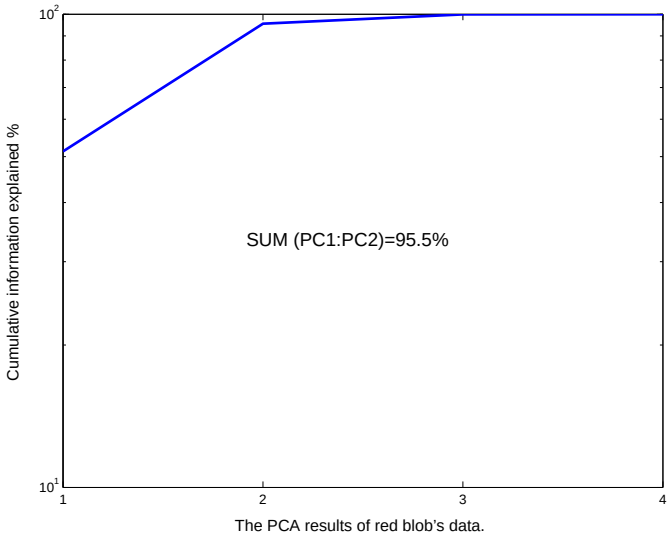


Figure 5.17: The PCA results of red blob data.

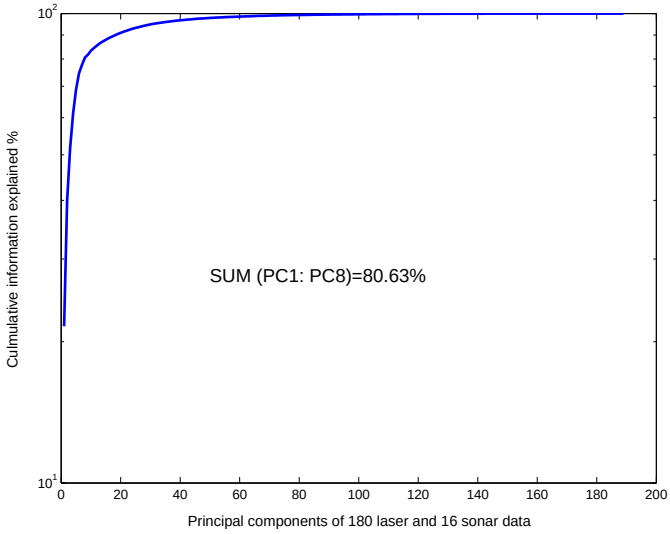


Figure 5.18: The PCA results of leader robot's laser and sonar data.

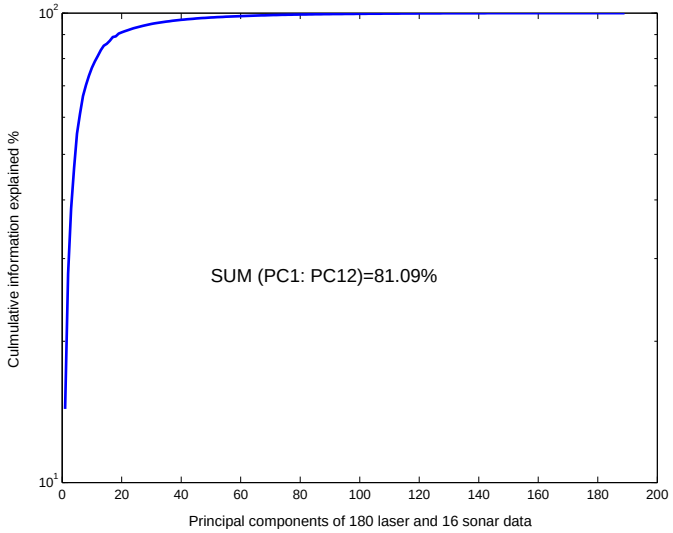


Figure 5.19: The PCA results of follower robot's laser and sonar data.

Table 5.18: The correlation coefficient for the leader robot's laser-sonar PCs and motor features.

Correlation Coefficient	**Speed**	**Turn-rate**
Laser-sonar-PC1	0.1095	0.2372
Laser-sonar-PC2	-0.0164	0.4094
Laser-sonar-PC3	-0.0499	-0.1555
Laser-sonar-PC4	0.0096	-0.0738
Laser-sonar-PC5	0.0521	0.2588
Laser-sonar-PC6	0.0482	-0.0592
Laser-sonar-PC7	-0.0626	-0.0177
Laser-sonar-PC8	-0.0587	-0.0885

Table 5.19: The correlation coefficient for the follower robot's laser-sonar PCs and motor features.

Correlation Coefficient	Speed	Turn-rate
Laser-sonar-PC1	-0.1745	-0.1121
Laser-sonar-PC2	0.0328	0.0851
Laser-sonar-PC3	0.1122	0.4031
Laser-sonar-PC4	-0.1024	0.0327
Laser-sonar-PC5	0.0072	0.0199
Laser-sonar-PC6	0.0525	0.1117
Laser-sonar-PC7	0.0310	0.0197
Laser-sonar-PC8	0.0202	0.1009
Laser-sonar-PC9	-0.0307	0.0037
Laser-sonar-PC10	-0.0087	0.0758
Laser-sonar-PC11	-0.0159	0.0241
Laser-sonar-PC12	-0.0721	-0.0196

- Follower robot's one principal component of laser and sonar data
- Follower robot's two principal components of red blob data
- Leader robot speed
- Follower robot speed
- Leader robot turn-rate
- Follower robot turn-rate
- Battery charge

Data clustering

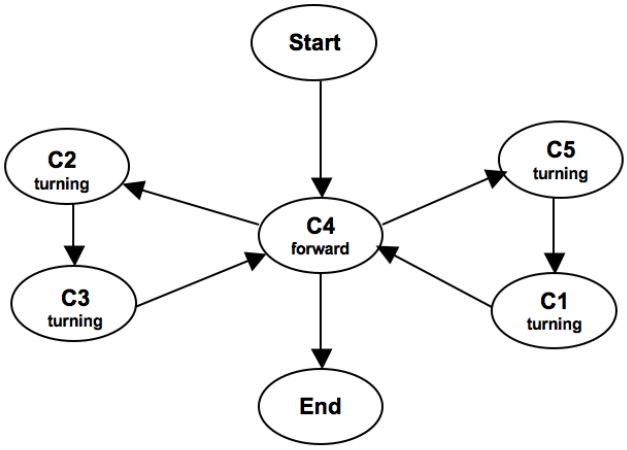
Fuzzy C-Means clustering is performed on the statistically selected sensor features to determine the robot team state. Different cluster numbers (from two (2) to ten (10)) are tested with Equation 3.11 and five is chosen as the best cluster number. Table 5.21 shows the clustering centers resulting for the robot team.

Building state transition diagram

With the five clusters (Table 5.21), each standing for one robot team state, SAFDetection builds the state transition diagram of the robots as shown in Table 5.20. The "start" and "end" states in the diagram are added manually to indicate the beginning and ending of the task. By examining the cluster centroid value, these five clusters are labeled as "forward"

Table 5.20: Normal behavior model for the entire robot team in robot following task.

(a) State transition diagram.



(b) State transition probabilities between states.

State	Start	C1	C2	C3	C4	C5	End
Start	0	0	0	0	1	0	0
C1	0	0	0	0	1	0	0
C2	0	0	0	1	0	0	0
C3	0	0	0	0	1	0	0
C4	0	0	0.38	0	0	0.42	0.2
C5	0	1	0	0	0	0	0
End	0	0	0	0	0	0	0

(c) Duration time in each state.

State	C1	C2	C3	C4	C5
Mean time (s)	5.48	6.02	4.85	20.71	5.24
Standard derivation (s)	0.33	0.62	0.59	8.11	0.47

Table 5.21: Cluster centers of the robot team in the following task

	PC1	PC2	PC3	PC4	Speed1	Turn-rate1	Speed2	Turn-rate2
C1	0.111	0.036	0.166	-0.024	0.199	-0.001	0.199	-0.004
C2	-1.014	-0.020	0.238	-0.336	0.176	-0.173	0.189	-0.033
C3	-0.110	-0.267	0.599	-1.193	0.195	-0.010	0.155	-0.202
C4	0.080	-0.085	-1.300	0.989	0.200	0.008	0.197	0.117
C5	-0.365	-0.097	-0.763	0.119	0.194	0.099	0.199	0.009

and four "turning" states (turning may happen in two different directions – left and right for both robots). This table also shows the average state transition probabilities between these states and the mean and standard deviation values of the time the robots remain in each state (i.e, before transiting to a different state). It can be seen that for most of the time, the robot team moves forward, while occasionally turning occurs in different directions; the follower robot's turning occurs after the leader robot's turning.

5.3.3 Robot following model built by distributed SAFDetection

This section shows how distributed SAFDetection is used to build the normal behavior model (state transition diagram) of the robot following task. Results are compared with the centralized approach.

Feature selection

In the distributed approach, statistical based feature selection is performed on individual robots in the team to select the essential features. According to the results shown in Figure 5.18, and Table 5.18, four (4) features are selected for the leader robot in the following task with distributed SAFDetection:

- One principal component of laser and sonar data
- Robot speed
- Robot turn-rate
- Battery charge

According to the results shown in Figure 5.17, Figure 5.19 and Table 5.19, six (6) features are selected for the follower robot in the following task with distributed SAFDetection:

- One principal component of laser and sonar data
- Two principal components of red blob data
- Robot speed

- Robot turn-rate
- Battery charge

Data clustering

Fuzzy C-Means clustering is performed on the sensor features obtained using PCA for the individual robot. Different cluster numbers (from two (2) to ten (10)) are tested with Equation 3.11 and three is chosen as the best cluster number for both robots in the team.

Table 5.22 shows the resulting cluster centers for the leader robot. By interpreting the cluster center S, we see that state C1 stands for turning left; state C2 stands for moving straight forward, and state C3 stands for turning right.

Table 5.23 shows the resulting cluster centers for the follower robot. By interpreting the cluster center S, we see that state C1 stands for turning right; state C2 stands for moving straight forward, and state C3 stands for turning left.

While these tables show the cluster centers (states) for each individual robot in the team, the robot team state is represented by the combination of the individual robot states. Table 5.24 shows the robot team states and their meanings for this task. It does not show all possible combinations, but only the combinations that exist in the training data. In theory, there exist nine (9, as 3×3) combinations; however, only five (5) of them exist in the training data (two transient states, (C1, C2) and (C3, C3), are removed by checking the state time duration as noted in Chapter 4).

Building state transition diagram

Two kinds of state transition diagram are built with the distributed SAFDetection – one is the local state transition diagram for the individual robot, the other is the global state

Table 5.22: Cluster centers of the leader robot in the following task.

State	**PC1**	**Speed**	**Turn-rate**
C1	-0.8025	0.18696	-0.1305
C2	0.1296	0.19912	-0.0036
C3	-0.0804	0.19773	0.0654

Table 5.23: Cluster centers of the follower robot in the following task.

	PC1	**PC2**	**PC3**	**Speed**	**Turn-rate**
C1	-0.0868	-1.1717	1.0389	0.1965	0.1138
C2	0.0223	0.0770	-0.0363	0.1987	-0.0044
C3	-0.2311	0.6909	-1.1560	0.1597	-0.1941

Table 5.24: Robot team states and meanings in the following task with the distributed approach.

Team state	**Meaning**
(C1,C2)	Leader robot is turning left
(C2,C1)	Follower robot is turning right
(C2,C2)	Robot team is moving straightly forward
(C2,C3)	Follower robot is turning left
(C3,C2)	Leader robot is turning right

transition diagram for the robot team. The individual state transition diagram generated for the leader robot in the following task is shown in Table 5.25. This table also shows the average state transition probabilities between these states and the mean and standard deviation values of the time the leader robot remained in each state (i.e, before transiting to a different state). The diagram and tables show that for most of the time, the leader robot moves forward, while occasionally turning in different directions.

The individual state transition diagram generated for the follower robot in the following task is shown in Table 5.26. This table also shows the average state transition probabilities between these states and the mean and standard deviation values of the time the follower robot remained in each state (i.e, before transiting to a different state). The diagram and tables show that for most of the time, the follower robot moves forward, while occasionally turning in different directions.

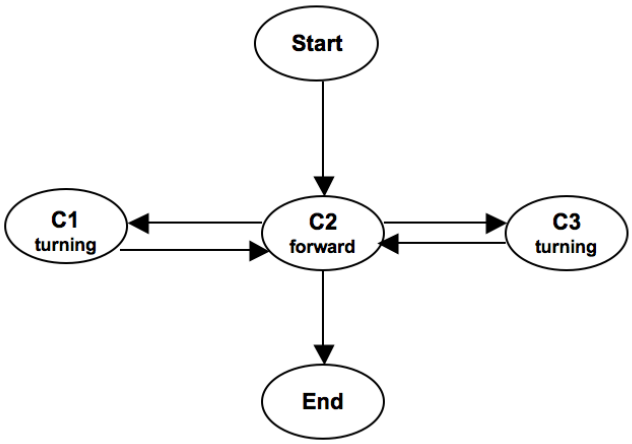
Combining the individual robot states to form the robot team state, the state transition diagram of the robot team in the following task is shown in Table 5.27. This table also shows the average state transition probabilities between these states and the mean and standard deviation values of the time the robot remained in each state (i.e, before transiting to a different state). The diagram and tables show that for most of the time, the robot team moves straight forward, while occasionally robots turn in different directions. Also, the follower robot turns after the leader robot turns. This normal team behavior model generated by distributed SAFDetection is consistent with the normal team behavior model generated by the centralized approach.

5.3.4 Robots following task monitored by SAFDetection

These results show that the state transition diagram generated by centralized SAFDetection (Table 5.20) and the robot team state transition diagram generated by distributed SAFDetection (Table 5.27) are quite similar. Both centralized and distributed SAFDetection detect faults efficiently but for different reasons.

Table 5.25: Normal behavior model for the leader robot in robot following task.

(a) State transition diagram.



(b) State transition probabilities between states.

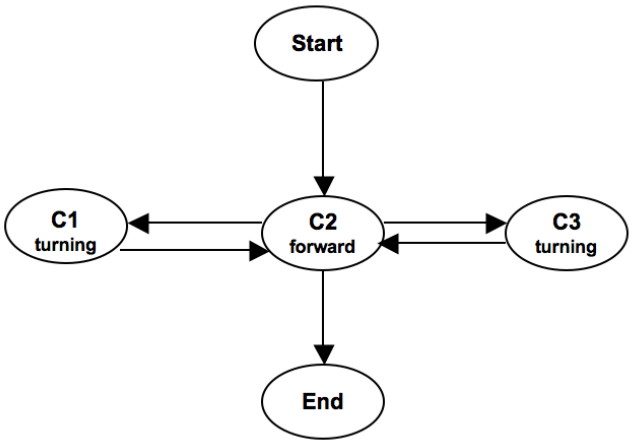
State	Start	C1	C2	C3	End
Start	0	0	1	0	0
C1	0	0	1	0	0
C2	0	0.36	0	0.44	0.2
C3	0	0	1	0	0
End	0	0	0	0	0

(c) Duration time in each state.

State	C1	C2	C3
Mean time (s)	5.24	23.16	6.00
Standard derivation (s)	0.47	8.40	0.23

Table 5.26: Normal behavior model for the follower robot team in robot following task.

(a) State transition diagram.



(b) State transition probabilities between states.

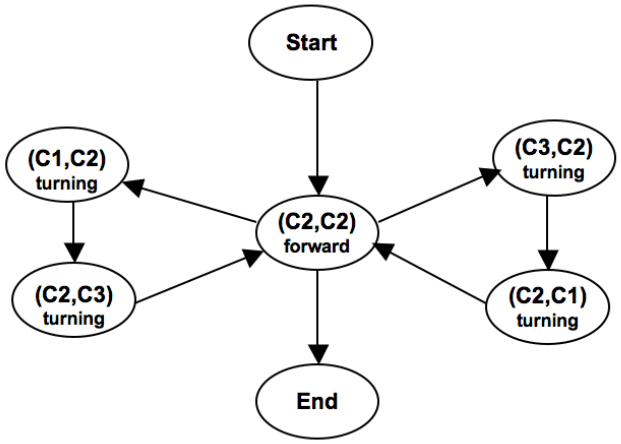
State	Start	C1	C2	C3	End
Start	0	0	1	0	0
C1	0	0	1	0	0
C2	0	0.42	0	0.38	0.2
C3	0	0	1	0	0
End	0	0	0	0	0

(c) Duration time in each state.

State	C1	C2	C3
Mean time (s)	4.91	20.1	5.47
Standard derivation (s)	0.81	9.22	0.34

Table 5.27: Normal behavior model for the entire robot team in robot following task with distributed approach.

(a) State transition diagram.



(b) State transition probabilities between states.

State	Start	(C1,C2)	(C2,C1)	(C2,C2)	(C2,C3)	(C3,C2)	End
Start	0	0	0	1	0	0	0
(C1,C2)	0	0	0	0	0	1	0
(C2,C1)	0	0	0	1	0	0	0
(C2,C2)	0	0.42	0	0	0	0.38	0.2
(C2,C3)	0	0	0	1	0	0	0
(C3,C2)	0	0	1	0	0	0	0
End	0	0	0	0	0	0	0

(c) Duration time in each state.

State	(C1,C2)	(C2,C1)	(C2,C2)	(C2,C3)	(C3,C2)
Mean time (s)	5.18	5.92	24.85	6.11	4.54
Standard derivation (s)	0.53	0.72	9.59	0.41	0.97

Figure 5.20 shows how the centralized and distributed SAFDetection detect the fault when the follower robot loses the red blob. In this scenario, the leader robot successfully turns around the corner but the follower robot turns to the wall and loses the red blob. The centralized SAFDetection detects the "blob missing" fault because the server detects an unknown state by clustering the team robots' sensor data. In the distributed SAFDetection approach, when the follower robot turns to the wall, it detects the fault by clustering its local sensor data.

Another fault example, "confusing blob" is shown in Figure 5.21. In this scenario, another red blob is placed on the path to the goal position and the follower robot becomes confused and follows the wrong red blob. When the leader robot continues moving, the follower robot stops since it is close enough to the confusing blob. The centralized SAFDetection detects the fault by observing an unknown team state by clustering the team robots' sensor data. In the distributed SAFDetection, when the follower robot changes its state, it checks the team state transition diagram and detects an unknown team state.

Table 5.28 shows the fault detection results for the robot following task by centralized and distributed SAFDetection with ten (10) normal trials and ten (10) abnormal trials. Consistent with the box pushing task, the distributed SAFDetection approach has a little higher false alarm rate in the following task.

5.3.5 Multi-robot following task in simulation

With the box pushing and following experiments, both centralized and distributed SAFDetection are efficient for detecting faults when the robot team size is small. In this section, a larger sized robot team performing the following task is used to test the scalability of the SAFDetection approach.

A team of five robots performing the following task is implemented with the robot simulation, Stage. Figure 5.22 shows a series of snapshots taken during one run of the robot following task. One leading robot (R_1) navigates from the starting position to the goal position with laser localization. Four other robots (R_2, R_3, R_4 and R_5) follow the path one by one, using a camera to track the robot in front of itself.

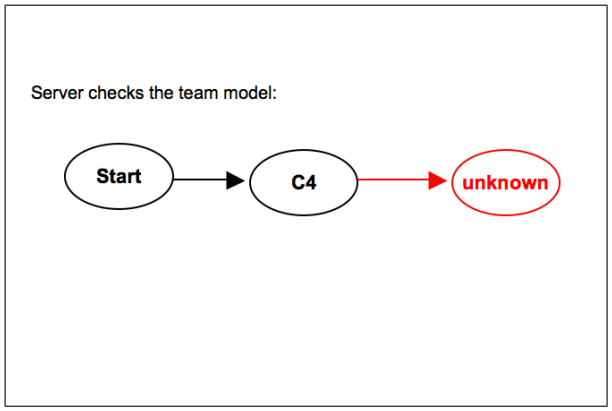
First, centralized SAFDetection is used to build the robot team behavior model. Similar to the two-robot team, PCA is performed on the robot team's sensor data and the following twenty-four (24) features are selected as essential features:

Table 5.28: Fault detection rates by centralized and distributed SAFDetection the following task.

	Centralized SAFDetection	**Distributed SAFDetection**
True positive	80%	70%
True negative	90%	90%
False positive	20%	30%
False negative	10%	10%

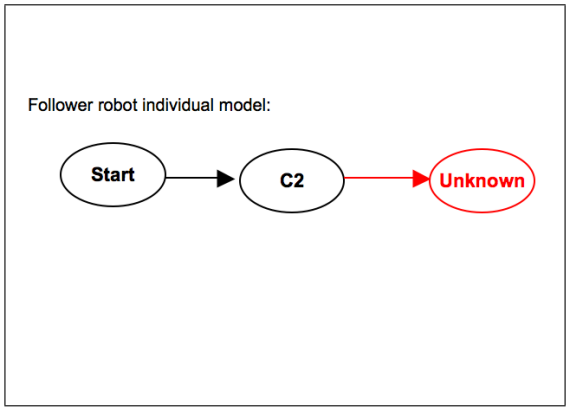
(a) Leader robot turns around the corner.

(b) Follower robot turns to the wall and loses the red blob.



(c) Centralized SAFDetection approach: the server robot detects fault: unknown robot team state found.

(d) Follower robot turns to the wall and loses the red blob.

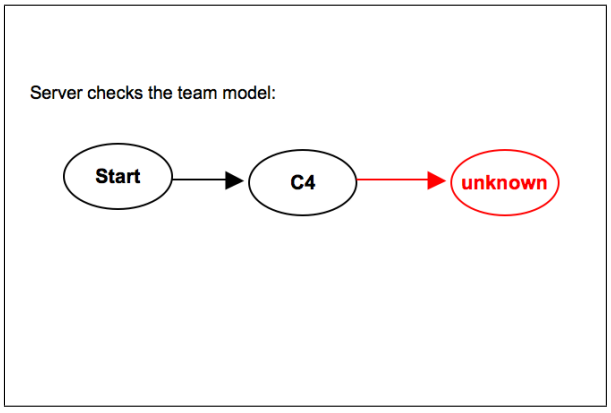


(e) Distributed SAFDetection approach: the follower robot detects fault: unknown robot state found.

Figure 5.20: A series of snapshots taken during the following task. Follower robot turns to the wall instead of to the hallway.

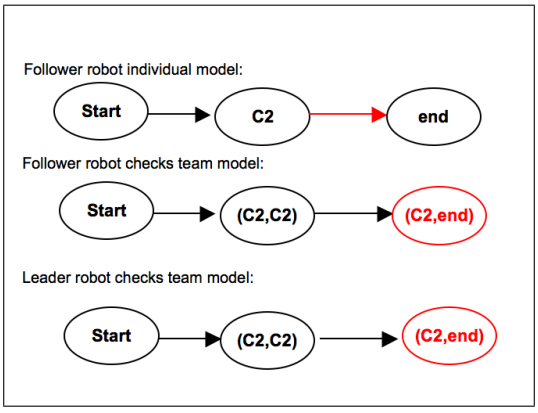
(a) Leader robot turns around the corner.

(b) Follower robot follows the red blob in the door and stops.



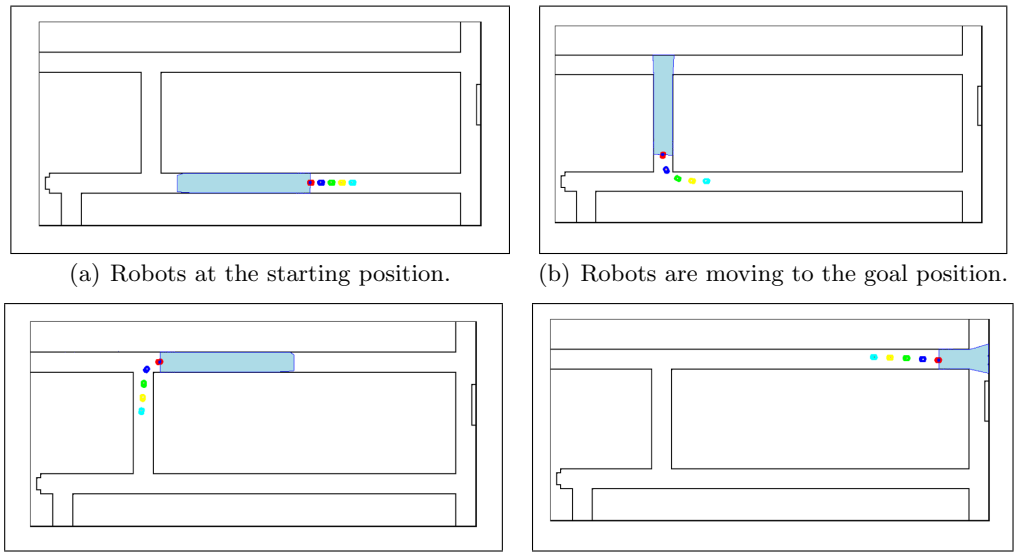
(c) Centralized SAFDetection approach: the server robot detects fault: unknown robot team state found.

(d) Follower robot follows the red blob in the door and stops.



(e) Distributed SAFDetection approach: the follower robot changes individual state and detects the fault: conflict with global state model.

Figure 5.21: A series of snapshots taken during the following task. The follower robot becomes confused with another red blob.



(a) Robots at the starting position.

(b) Robots are moving to the goal position.

(c) Robots are moving to the goal position.

(d) Robots reach the goal position.

Figure 5.22: A series of snapshots taken during the normal robot following task.

- Robot R_1's one principal component of laser and sonar data
- Robot R_1's speed
- Robot R_1's turn-rate
- Robot R_2's one principal component of laser and sonar data
- Robot R_2's two principal components of color blob data
- Robot R_2's speed
- Robot R_2's turn-rate
- Robot R_3's one principal component of laser and sonar data
- Robot R_3's two principal components of color blob data
- Robot R_3's speed
- Robot R_3's turn-rate
- Robot R_4's one principal component of laser and sonar data
- Robot R_4's two principal components of color blob data
- Robot R_4's speed
- Robot R_4's turn-rate

- Robot R_5's one principal component of laser and sonar data
- Robot R_5's two principal components of color blob data
- Robot R_5's speed
- Robot R_5's turn-rate
- Battery charge

FCM clustering is performed on these twenty-three (23) essential features (except the battery charge value) to determine the team robot state. Different cluster numbers (from two (2) to twenty (20)) are tested with Equation 3.11 to find the best cluster number. However, the result shows that clustering with two (2) clusters has the highest quality, which actually reflects the failure of clustering since there are obviously more than two team robot states in this task. This failure is a result of the curse of dimensionality. According to [Richard, 1957], adding extra dimensions to a (mathematical) space will cause an exponential increase in volume. Therefore, given a increasing data dimension, the training data number need to be increased exponentially to gain the same clustering performance. Therefore, centralized SAFDetection is not appropriate for this task, for the same size of input data. However, since the sensor data for individual robots is limited, distributed SAFDetection can be used.

With distributed SAFDetection, similar to the two-robot team, only six (6) essential features are selected for each individual robot in the team:

- One principal component of laser and sonar data
- Two principal components of color blob data (not needed for the leading robot R_1)
- Robot speed
- Robot turn-rate
- Battery charge

FCM clustering maps the local sensor data of each robot to three states with six (6) essential features: C1, which stands for moving straight forward; C2, which stands for turning left; and C3, which stands for turning right. The individual robot state transition diagram for each of the robots in the team can be built the same way as illustrated in the last section (Table 5.25 and Table 5.26).

After setting up the individual robot model, the robot team behavior model can be built with Algorithm 4. Table 5.29 shows the robot team states resulting from the combination of individual robots and their meanings for this task.

Figure 5.23 shows the state transition diagram of the robot team in 5-robot following task. It can be seen that for most of the time, the team robot moves straight forward, while occasionally robots turn in different directions. During the turning process, the robots turn one by one.

Table 5.29: Robot team states and meanings in the 5-robot following task using the distributed approach.

Team state	**Combination**	**Meaning**
S1	(C1,C1,C1,C1,C1)	The robot team is moving straight forward
S2	(C2,C1,C1,C1,C1)	Robot R_1 is turning left
S3	(C1,C2,C1,C1,C1)	Robot R_2 is turning left
S4	(C1,C1,C2,C1,C1)	Robot R_3 is turning left
S5	(C1,C1,C1,C2,C1)	Robot R_4 is turning left
S6	(C1,C1,C1,C1,C2)	Robot R_5 is turning left
S7	(C3,C1,C1,C1,C1)	Robot R_1 is turning right
S8	(C1,C3,C1,C1,C1)	Robot R_2 is turning right
S9	(C1,C1,C3,C1,C1)	Robot R_3 is turning right
S10	(C1,C1,C1,C3,C1)	Robot R_4 is turning right
S11	(C1,C1,C1,C1,C3)	Robot R_5 is turning right

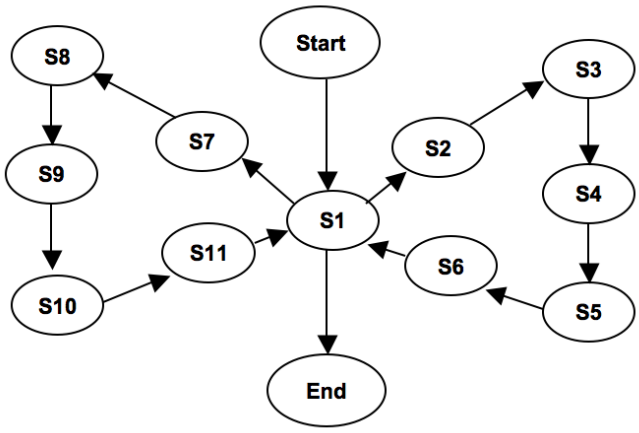


Figure 5.23: Team state transition diagram for the 5-robot following task using the distributed approach.

Figure 5.24 shows how the distributed SAFDetection can be used to detect faults for this multi-robot following task.

For the 5-robot team, since each individual robot has three states, there are 243 (3^5) possible team robot states in total. In this task, only eleven (11) states exist in the training data set. However, with a more complicated map (such as Figure 5.25), the number of robot team states can still be large.

Building the robot team behavior model for the entire team is sometimes unnecessary. For example, in the 5-robot following task, robot R_1 is not directly related to robot R_5 and R_5 can detect its own interactive faults by only checking the state of robot R_4. Therefore, the group state transition diagram can be used to replace the entire team state transition diagram. In these simulations, since each robot only follows the robot in front of it, it is possible to divide the entire team into four compacted groups:

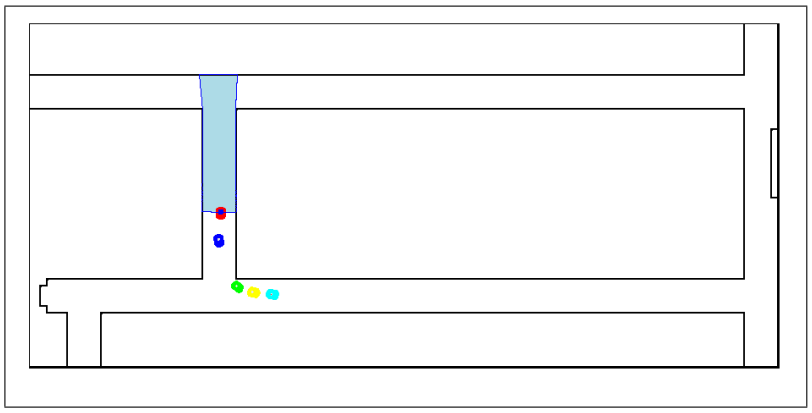
- Group 1: R_1 and R_2
- Group 2: R_2 and R_3
- Group 3: R_3 and R_4
- Group 4: R_4 and R_5

In addition, using a group state transition diagram to detect faults reduces the complexity to maintain and update the team state for the entire team.

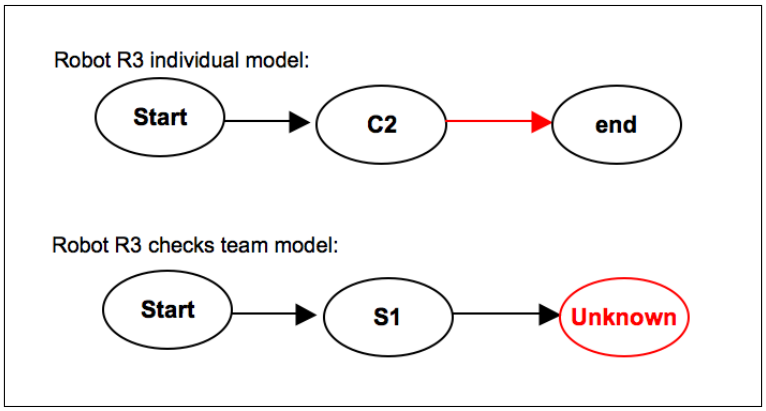
Following tasks with different robot team sizes have been tested with centralized SAFDetection and distributed SAFDetection (Table 5.30). It can be seen that the centralized approach meets the cures of dimensionality when the robot team size is greater than three (3).

5.4 Summary

This chapter has presents the experiments that I have implemented for validating the SAFDetection approach for both the centralized and distributed versions. Experiments are performed both in simulation and on physical robots in two applications: box pushing and following tasks. The results show that both centralized and distributed SAFDetection are efficient for detecting faults when the robot team size is small (≤ 3) while the distributed approach results in slightly higher false alarm rate because of the asynchronism between the robots. The centralized approach has been shown to be not appropriate for larger sized robot teams. In general, if the robot team size is small, and the "server" of the team is reliable and capable in computation, centralized SAFDetection is preferred. Otherwise, distributed SAFDetection is preferred.



(a) Robot 3 stuck while turning the corner.



(b) Distributed SAFDetection: Robot 3's individual state changes, the team state transition diagram is checked and the fault is detected

Figure 5.24: A series of snapshots taken during the robot following task; one robot gets stuck.

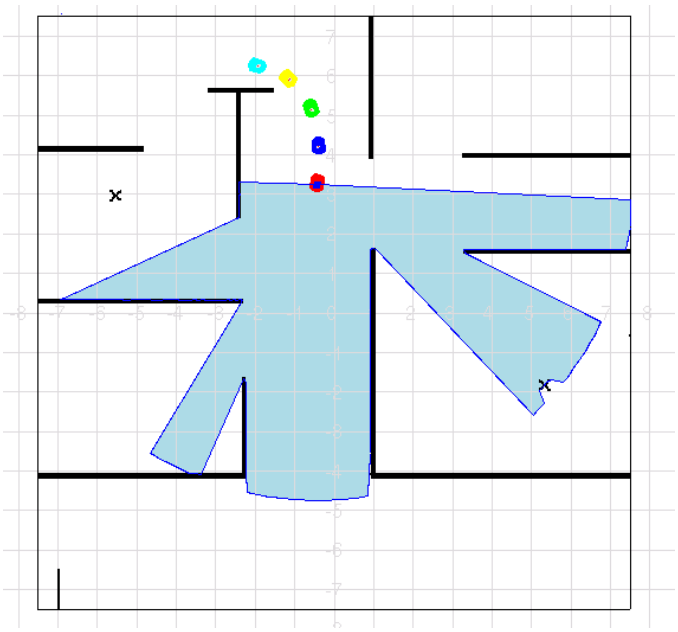


Figure 5.25: A possible complicated map.

Table 5.30: Various robot following tasks with the SAFDetection approach.

Team size	**# of features Centralized**	**# of features Distributed**	**# of team states Centralized**	**# of team states Distributed**
2	8	5	5	5
3	13	5	6	7
4	18	5	2	9
5	23	5	2	11

Chapter 6

Summary and Conclusions

6.1 Summary of contributions

This dissertation makes several contributions to fault detection in multi-robot tasks. The most important is the design and development of SAFDetection – a novel approach that detects the faults for a multi-robot system performing tightly-coupled tasks based on sensor data analysis. It is particularly suitable when the motion models and models of multi-robot interaction dynamics are unavailable. This approach has been shown to have the following characteristics:

- It generates a normal behavior model for the robot system based on sensor data analysis.
- It is applicable to tightly-coupled multi-robot team tasks with flexible team composition (e.g., size, heterogeneity)
- It provides a distributed or centralized implementation, depending on the robot team size and user preference.
- It can detect a wide range of faults in a real-time manner with reasonable accuracy. In particular, it can detect coalition faults caused by the cooperation among tightly-coupled multi-robot team members.
- It can detect faults without prior knowledge of the robot application or the possible fault types.

The SAFDetection approach has been implemented on both simulated and physical robot teams performing a variety of tasks. Particularly, there are two distinct implementation versions: centralized SAFDetection and distributed SAFDetection. The centralized SAFDetection treats the entire robot team as a single monolithic robot and provides an applicable way to build the model of the robot team's normal behavior for detecting faults. However, like many other centralized systems, centralized SAFDetection faces the problem of single point failure, heavy computational load on the server, and a limited robot team

size requirement. To overcome these problems, the distributed SAFDetection approach was developed and implemented, which distributes the computational load, breaks the curse of dimensionality, and cuts down on the network traffic. However, there is a tradeoff in issues caused by data sharing, data updating and transient states resulting from the asynchronous property. Usually, when the robot team size is small (less than or equal to three (3)), the communication bandwidth is sufficient, and the "server" of the robot team is reliable, the centralized approach is preferred. On the other hand, the distributed approach is more appropriate if the robot team size is large, the communication bandwidth is limited or there is no reliable "server".

The SAFDetection approach has been applied to a variety of physical and simulated robot applications to validate and demonstrate its characteristics. These applications include:

- Two robot cooperative pushing
- Two robot follow the leader
- Five robot follow the leader

6.2 Future work

There are several promising research directions that follow directly from my research on SAFDetection:

- *False alarms.* In the current SAFDetection approach, any significant deviation from the robot normal behavior model is regarded as a fault. This can cause a high false alarm rate especially when the training data does not cover all possible normal situations. Currently, I use a two-layer fault alarm structure to reduce false alarms. Other approaches can be investigated to reduce this sensitivity.

- *Online learning.* In the current SAFDetection approach, the learning stage is offline; an extension to online learning can be considered. With the feedback from a supervisor, the fault detected by SAFDetection can be classified as a "real" fault or a normal state which does not appear in the training data set. Thus, online learning can reduce the learning trials and enable the robot system to adapt to new situations, especially to changing environments.

- *Fault learning.* At present, the SAFDetection approach only builds and maintains a robot's (or team's) normal behavior model. Additional attention could be paid to the sensor data during anomalous robot operations. These data provide fault information of the robot system, which could allow the team to become knowledgeable about frequently occurring faults, thus lowering the false positive rate.

- *Incorporating prior knowledge.* Compared to other fault detection approaches, SAFDetection has the advantage of detecting faults without prior knowledge of the robot motion model or possible fault types. However, if some prior knowledge of the robot

motion model or possible fault type is available, it should be used to help to build a more accurate normal behavior model. How to incorporate prior knowledge into my current approach can be investigated.

- *Training issues.* Training data directly affects the performance of the SAFDetection approach. Further investigation can be made on problems including: testing robots with the same training robot task in different environments; testing robots with insufficient training data, etc.

- *Asynchronous issue.* To monitor the robot team, sensor data is collected from all the robots in one time step. However, the data collection is not necessarily synchronized in time across robots. In addition, different sensors on the same robot also have this synchronization problem, due to different response speeds. For example, the laser device responds faster than the motor device. Currently, the correlation operation is applied to find the time delay between different sensor data. Other approaches such as the use of a sliding window or data delay can be investigated to correct this problem. In addition, as the robot team size increases, it becomes more difficult to synchronize the team robot state for the entire team. Further investigation can be made on the asynchronization problem with large team sizes.

- *Incorporate other techniques.* In the current SAFDetection, PCA based dimension reduction, FCM based clustering algorithm and a Markov chain based state transition diagram are used to build the normal behavior model of the robot system. These techniques may not be the best combination to detect fault with SAFDetection structure and other techniques can be explored and tested. For example, multi-dimension PCA is an alternative way to reduce data dimension; clustering with an actual data point instead of using the centroid as a cluster center is another option; high dimension clustering algorithm can also be explored for comparison.

Bibliography

Bibliography

[Kim, 2007] (2007). Soft geodesic kernel k-means. In *IEEE International Conference on Acoustics, Speech and Signal*, volume 2, pages 429–432.

[Bezdek et al., 1984] Bezdek, J. C., Ehrlich, R., and Full, W. (1984). FCM: The fuzzy c-means clustering algorithm. *Computers and Geosciences*, 10(2-3):191–203.

[Bhunia and Roy, 2002] Bhunia, S. and Roy, K. (2002). Fault detection and diagnosis using wavelet based transient current analysis. In *Proceedings of the conference on Design, Automation and Test in Europe table of contents*, page 1118.

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

[Cai and Duan, 2005] Cai, Z. and Duan, Z. (2005). 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.

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

[Cheng et al., 2005] Cheng, Q., Varshney, P. K., and Michels, J. (2005). Distributed fault detection via particle filtering and decision fusion. In *International Conference on Information Fusion*, pages 1239–1246.

[Chow and Willsky, 1984] Chow, E. and Willsky, A. (1984). Analytical redundancy and the design of robust failure detection systems. *IEEE Transactions on Automatic Control*, 29(7):603–614.

[Christensen et al., 2007] Christensen, A. L., O'Grady, R., Birattari, M., and Dorigo, M. (2007). Fault detection in autonomous robots based on fault injection and learning. *Auton. Robots*, 24(1):49–67.

[Dearden and Clancy, 2002] Dearden, R. and Clancy, D. (2002). Particle filters for real-time fault detection in planetary rovers. In *Proceedings of the Thirteenth International Workshop on Principles of Diagnosis*.

[Delage et al., 2006] Delage, E., Lee, H., and Ng, A. Y. (2006). 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.

[Donald et al., 1997] Donald, B. R., Jennings, J., and Rus, D. (1997). Information invariants for distributed manipulation. *International Journal of Robotics Research*, 16(5):673–702.

[Fatta et al., 2006] Fatta, G. D., Leue, S., and Stegantova, E. (2006). Discriminative pattern mining in software fault detection. In *Proceedings of the 3rd International Workshop on Software quality assurance*, pages 62–69.

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

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

[Gerkey et al., 2003] Gerkey, B., Vaughan, R., and Howard, A. (2003). The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the International Conference on Advanced Robotics*, pages 317–323.

[Gerkey et al., 2001] Gerkey, B., Vaughan, R., Stoey, K., Howard, A., Sukhatme, G., and Mataric, M. (2001). Most valuable player: A robot device server for distributed control. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1226 – 1231.

[Gerkey and Mataric, 2002] Gerkey, B. P. and Mataric, M. J. (2002). Pusher-watcher: An approach to fault-tolerant tightly-coupled robot coordination. In *IEEE International Conference on Robotics and Automation*, pages 464–469.

[Goel et al., 2000] Goel, P., Dedeoglu, G., Roumeliotis, S., and Sukhatme, G. (2000). 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.

[Hartigan and Wong, 1979] Hartigan, J. A. and Wong, M. A. (1979). A K-means clustering algorithm. *Applied Statistics*, 28:100–108.

[Hashimoto et al., 2003] Hashimoto, M., Kawashima, H., and Oba, F. (2003). 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.

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

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

[Jolliffe, 2002] Jolliffe, I. T. (2002). *Principal Component Analysis*. Springer-Verlag.

[Kadirkamanathan et al., 2002] Kadirkamanathan, V., Li, P., Jaward, M. H., and Fabri, S. G. (2002). Particle filtering-based fault detection in non-linear stochastic systems. *International Journal of Systems Science*, 33:259–265.

[Kawabata et al., 2004] Kawabata, K., Okina, S., Fujii, T., and Asama, H. (2004). A system for self-diagnosis of an autonomous mobile robot using an internal state sensory system: fault detection and coping with the internal condition. *Advanced Robotics*, 17:925–950.

[Krishnamachari and Iyengar, 2004] Krishnamachari, B. and Iyengar, S. (2004). Distributed Bayesian algorithms for fault-tolerant event region detection in wireless sensor networks. *IEEE Transactions on Computers*, 53(3):241–250.

[Kube and Zhang, 1996] Kube, C. R. and Zhang, H. (1996). The use of perceptual cues in multi-robot box-pushing. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2085–2090.

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

[Lerner et al., 2000] Lerner, U., Parr, R., Koller, D., and Biswas, G. (2000). Bayesian fault detection and diagnosis in dynamic systems. In *Proceedings of the 17th National Conference on Artificial Intelligence*, pages 531–537.

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

[Li and Parker, 2007] Li, X. and Parker, L. E. (2007). Sensor analysis for fault detection in tightly-coupled multi-robot team tasks. In *IEEE International Conference on Robotics and Automation*, pages 3269–3276.

[Li and Parker, 2008] Li, X. and Parker, L. E. (2008). Design and performance improvements for fault detection in tightly-coupled multi-robot team tasks. In *IEEE Southeast-Con*.

[Luo and Wang, 2005] Luo, M. and Wang, D. (2005). Model-based fault diagnosis/prognosis for wheeled mobile robots: A review. In *Industrial Electronics Society, 2005. 32nd Annual Conference of IEEE*, pages 2267–2272.

[Manikopoulos and Papavassiliou, 2002] Manikopoulos, C. and Papavassiliou, S. (2002). Network intrusion and fault detection: a statistical anomaly approach. *IEEE Communications Magazine*, 40:76–82.

[Marmon, 1979] Marmon, W. C. (1979). A distributed fault-tolerant switch for use in modular redundancy. In *Computers in Aerospace Conference*, pages 182–188.

[Marslanda et al., 2005] Marslanda, S., Nehmzowb, U., and Shapiroc, J. (2005). On-line novelty detection for autonomous mobile robots. *Robotics and Autonomous Systems*, 51:191–206.

[Mataric et al., 1995] Mataric, M. J., Nilsson, M., and Simsarian, K. T. (1995). Cooperative multi-robot box-pushing. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 556–561.

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

[Moore, 1981] Moore, B. (1981). Principal component analysis in linear systems: Controllability, observability, and model reduction. *IEEE Transactions on Automatic Control*, 26:17–32.

[Parker et al., 2004] Parker, L. E., B. Kannan, F. T., and Bailey, M. (2004). Tightly-coupled navigation assistance in heterogeneous multi-robot teams. In *IEEE International Conference on Intelligent Robots and Systems*, volume 1, pages 1016–1022.

[Parker and Kannan, 2006] Parker, L. E. and Kannan, B. (2006). Adaptive causal models for fault diagnosis and recovery in multi-robot teams. In *IEEE International Conference on Intelligent Robots and Systems*, pages 2703–2710.

[Rabiner and Juang, 1986] Rabiner, L. and Juang, B. (1986). An introduction to Hidden Markov Models. *IEEE Signal Processing Magazine*, 3(1):4–16.

[Richard, 1957] Richard, B. (1957). *Dynamic programming.* Princeton University Press.

[Sadeghi et al., 2005] Sadeghi, M. H., Raflee, J., and Arvani, F. (2005). A fault detection and identification system for gearboxes using neural networks. In *International Conference on Neural Networks and Brain*, pages 964– 969.

[Sen et al., 1994] Sen, S., Sekaran, M., and Hale, J. (1994). Learning to coordinate without sharing information. In *Proceedings of AAAI*, pages 426–431.

[Sheng and Rovnyak, 2004] Sheng, Y. and Rovnyak, S. (2004). Decision tree-based methodology for high impedance fault detection. *IEEE Transactions on Power Delivery*, 19:533–536.

[Singhal and Jajodia, 2006] Singhal, A. and Jajodia, S. (2006). Data warehousing and data mining techniques for intrusion detection systems. *Distributed and Parallel Databases*, 20:149–166.

[Skoundrianos and Tzafestas, 2004] Skoundrianos, E. N. and Tzafestas, S. G. (2004). Modelling and fdi of dynamic discrete time systems using a mlp with a new sigmoidal activation function. *Intelligent and Robotic Systems*, 41.

[Staroswiecki and Comtet-Varga, 2001] Staroswiecki, M. and Comtet-Varga, G. (2001). Analytical redundancy relations for fault detection and isolation in algebraic dynamic systems. *Automatica*, 37:687–699.

[Sun and McCartney, 2001] Sun, H. and McCartney, R. (2001). A binary tree based approach for the design of fault-tolerant robot team. In *Proceedings of the Fourteenth International Florida Artificial Intelligence Research Society Conference*, pages 536–540.

[Sundvall and Jensfelt, 2006] Sundvall, P. and Jensfelt, P. (2006). Fault detection for mobile robots using redundant positioning systems. In *IEEE International Conference on Robotics and Automation*, pages 3781–3786.

[Teoh et al., 2004] Teoh, S. T., Zhang, K., and Tseng, S. M. (2004). Combining visual and automated data mining for near-real-time anomaly detection and analysis in BGP. In *Conference on Computer and Communications Security*, pages 35–44.

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

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

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

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

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

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

Vita

Xingyan Li joined the Department of Computer Science at the University of Tennessee-Knoxville (UTK) as Graduate Student in January 2003. She received the M.S. degree in Computer Science from UTK in 2004, and the B.S. degree in Computer Engineering from Beijing University of Posts and Telecommunications, China, in 2001.

She has been conducting research in the Distributed Intelligent Laboratory at UTK since 2005. Her current research focuses on developing mechanisms to enable robots to detect faults automatically based on sensor data analysis. Her research interests also include multi-agent systems, human-robot cooperation, machine learning, and data mining.

She is a student member of IEEE and ACM. After graduation, she is looking forward to joining the Robotics Research group at University of Auckland, New Zealand as a research fellow.