Failure Mode Analysis of an Autonomous Guided Robot using JDBC

A thesis submitted to the Division of
Graduate Studies and Advanced Research
Of the University of Cincinnati

in partial fulfillment of the
requirements for the degree of

MASTER OF SCEIENCE

In the Department of Mechanical, Industrial and Nuclear Engineering
Of the College of Engineering
2000

by

Vishnuvardhanaraj Selvaraj
B.E (Mechanical Engineering), Kumaraguru College of Technology,
Bharathiyar University, Coimbatore
India, 1998

Thesis Advisor and Committee Chair: Dr. Ernest L. Hall

Table of Contents

Abstract

Diagnosis of Faults and problems in equipment is a vitally important contributor to throughput and efficiency of the equipment. This work describes the development of Online Fault Diagnosis System for the Bearcat Robot using a cutting edge technology called Java Database Connectivity (JDBC). The approach followed in improving fault diagnosis efficiency is to capture and reuse know-how that exists in the heads of the key individuals who really understand how the Bearcat Robot works. The robot is sub-divided into different functional units as the Power System, Mechanical System, Vision guidance System and Ultrasonic Obstacle Avoidance System called SONAR. This in turn is divided into specific component. For each specific part taken into consideration, the failure mode is analyzed in the form of the possible failure symptom, the reason why the part fails and finally the steps to be taken to rectify the fault are discussed.

The information then obtained is designed as a database table. This is presented by means of a Java Applet, the link of which is available in the UC Robot Web page. The technology used to establish the connection between the database and the Java Applet is JDBC. It is possible to develop database applications using Java and provide a standard interface between user and the database server using JDBC. The JDBC API can be used regardless of what database is being used in the back end and it is supported by a large set of JDBC drivers. The online system could be accessed anytime from any part of the world with a web browser and the connection to the Internet. The provision is made such that user authenticity is obtained before allowing the user to access the data. Thus the system developed helps in cutting down the time required for diagnosing a fault.

Acknowledgements

I am grateful to my advisor Dr. Ernest Hall for giving me an opportunity to work on the cutting edge technology of JDBC. He has been the constant source of support for gathering relevant resources and information. He helped and encouraged me from all perspectives to complete this work.

I am thankful to the team members of the robotic team for providing their valuable suggestions and supporting my work.

I wish to thank my committee members Dr. Richard L. Shell and Dr. Ronald L. Huston for supporting my work. I take this opportunity to thank the entire faculty and staffs of MINE department for helping in pursue my masters at University of Cincinnati.

Last but not least my parents and family should be thanked. They stand behind all my success.

List of Figures

Chapter One

Introduction

The present day requirement for ever-increasing reliability is now more important than ever before and continues to grow constantly. Advances are continually being made in engineering. This means that the detection, location and analysis of faults play a vital role. The need to have efficiency and safety in the design and development of Automated Guided Vehicle (AGV) leads to the development of diagnostic strategies that could cover the major potential faults of the AGV.

A hi-tech product like Robot needs a sophisticated system for analyzing the failures, storing the related information in an integrated repository and retrieving the same via standard user-friendly interface. The Center for Robotics Research at University of Cincinnati has been developing Robot for the past 13 years. The Bearcat robot of UC Robotics team has undergone two major versions till now. The latest version called Bearcat II is an Autonomous Guided Vehicle.

Fault diagnosis system provides a systematic review of the components, assemblies and subsystems of a product to identify single point failures and the causes and effects of such failures. It identifies and tabulates the potential modes by which equipment or a system might fail and the consequences such a failure would have on the equipment or system being studied.

The picture of the Bearcat II robot is shown below. It weighs 450 lbs. and is 2 feet wide and 4 feet long.

Figure 1: BEARCAT Robot

The user-friendly Java applet developed using JDBC technology is shown in the picture below. This applet establishes the connection with the database developed in the back end using Microsoft Access. In the picture, the major system is chosen as Mechanical System from the list one and the sub component is chosen as Servo Motor in the list two. The possible fault symptom for this combination, the reason why it happens and the solution to rectify it has been shown in the text fields placed below the list. The UC logo is used as a button to get the results for the queries posed.

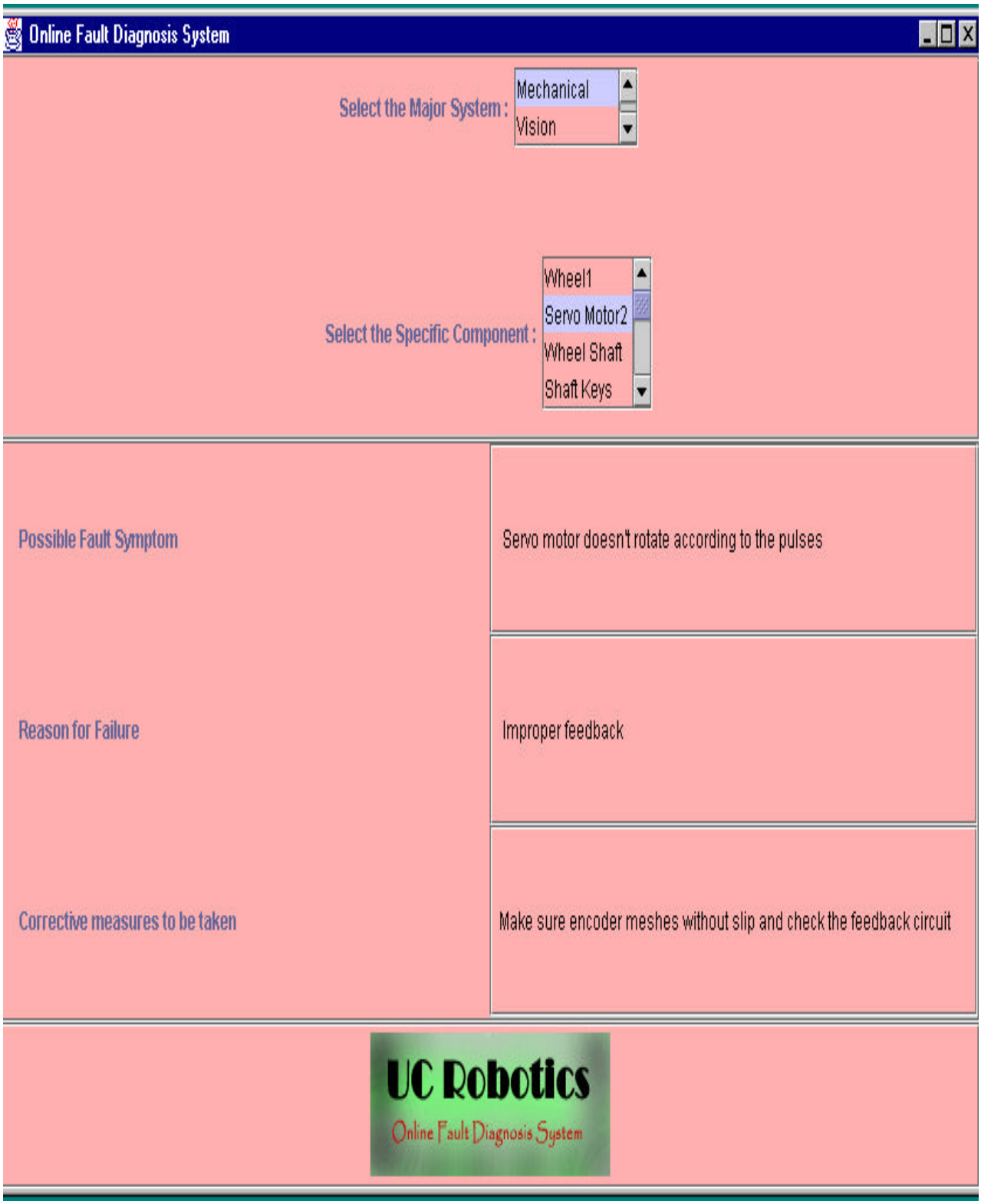


Figure 2: GUI Interface

1 . 1 Objective

The objective of the project is to develop the fault diagnosis system using the latest web technology, JDBC. The Java Applet developed is made available from the UC Robot web page. Presently the system is designed for the team use alone and this is ensured with secure login. The features of the Online Fault Diagnosis are

- A user friendly Java Applet interface to connect to the database
- Functionality to check for user authentication
- Options to choose between major units and its sub-classifications
- A Comprehensive data table showing the information for fault diagnosis
- The online system could be accessed anytime from any part of the world with a web browser and the connection to the Internet

1 . 2 Organization of the Thesis

I have organized my documentation in separate chapters, which differentiates the ideas and technologies that I have used to develop the web application.

Chapter 2 describes the idea for grouping the components of the robot into major units and its sub units. Chapter 3 deals with the database design for storing the information about various faults and their solutions. Chapter 4 deals with the explanation of the platform independent JDBC technology and its key features giving the required flexibility. The working of the application is explained in chapter 5, while the final outputs and future recommendations are discussed in chapter 6.

The appendix A shows the programming codes and appendix B shows database tables used to obtain the results.

Chapter Two

Major Systems and Components of Bearcat robot

In order to obtain the failure modes for individual subsystems as well as the entire robot, a thorough study and grouping of the components is essential. This chapter outlines system description of Bearcat II, by grouping the components into major units and a sub classification of the major units. For efficient operation of the robot not only should the individual subsystems work satisfactorily but they also should work in tandem. This thorough analysis has been tremendously useful in understanding the architectural, functional and behavioral details of the system**.**

According to the functionality the robot has been categorized into major units as

- Power System
- Vision System
- Mechanical System
- SONAR system

Power System

The Power system is further classified into seven finite elements. Power system of the robot consists of the components that help to power the electrical components of the robot. The heart of the power system being the Solenoid acts as a switch, which can be controlled to cut off the power during emergency.

The database consists of the Power System as the attribute value in the major unit column. The other major parts involved in the power system are Battery, Inverter, Fuse, Servo Motor and SONAR Motor. The Schematic diagram showing the major components of the power system is shown below.

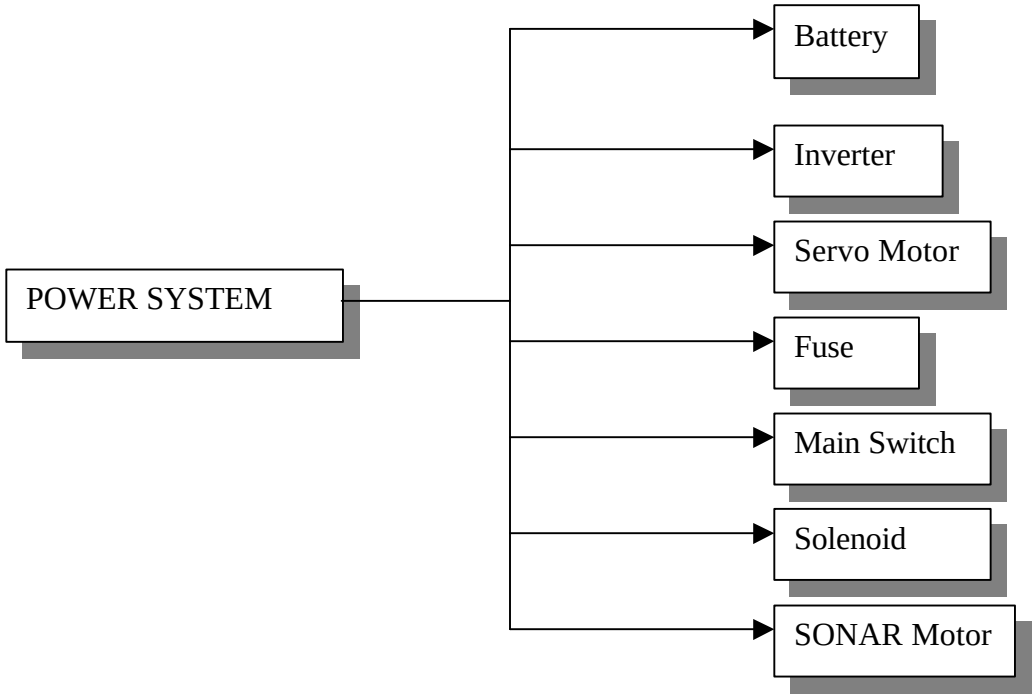


Figure 3: Elements of Power System

Vision System

Vision systems consists of an optical system to collect and focus the light from a finite field view and an optoelectrical system to take focused light and give a computer readable signal. This mapping of the two-dimensional points and reorganization of the three-dimensional image is done by the vision calibration. The robot has to follow a track

of 10 feet width with two lines on either side of it. It tracks either one of the lines and makes its navigation parallel to that line. If there is a break in that line, it searches for the line on the other camera.

The vision system defines the components that assist in the line following of Bearcat II. It consists of two JVC CCD cameras mounted on either side of the robot, such that a clear line tracking can be achieved. The sub classification the vision system yields the following components for fault diagnosis.

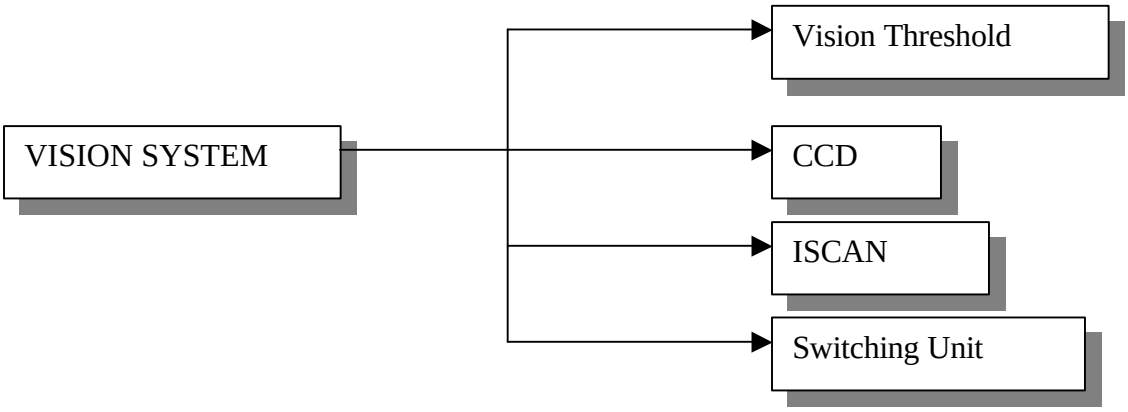


Figure 4: Elements of Vision System

Mechanical System

The mechanical system as a whole serves as steering control for the robot. The components include 40:1 reduction gearbox, two pairs of flexible couplings, two 36 volts servomotors and two sets of wheels with shafts, couplings and keys. The computer

through Galil motion controller controls the servomotors. The schematic diagram showing the components is listed below.

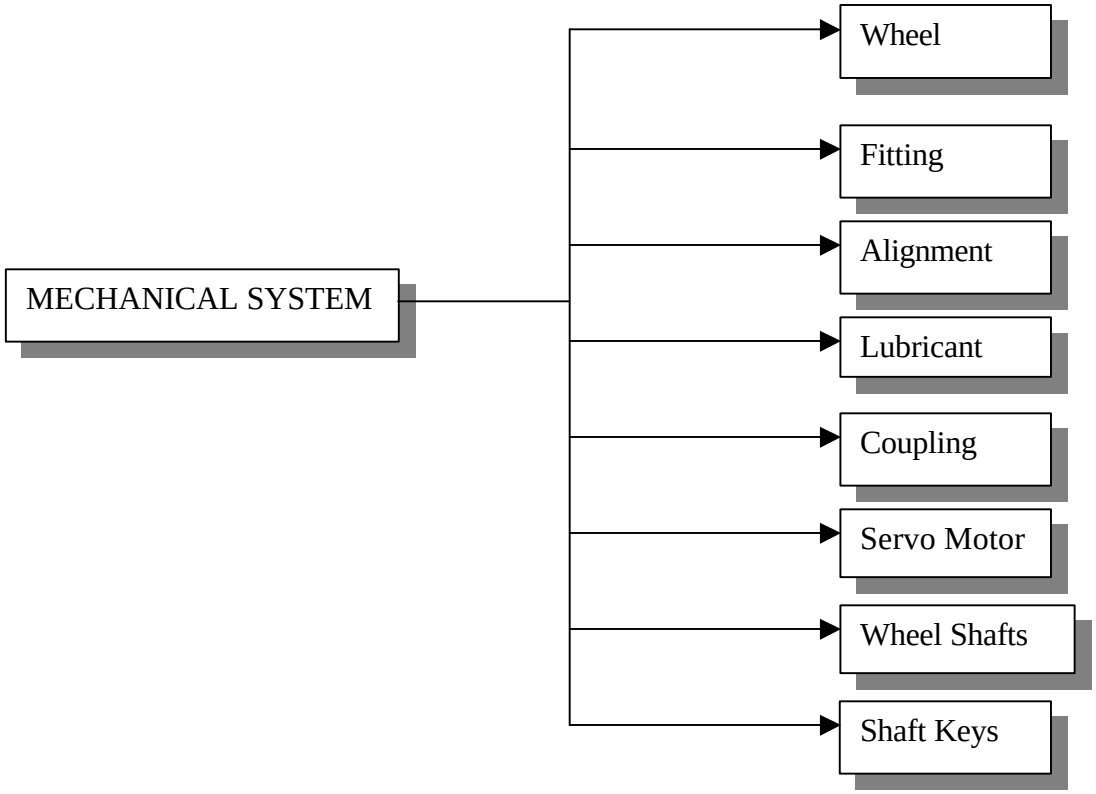


Figure 5: Elements of Mechanical System

SONAR System

SONAR system is used for obstacle avoidance, apart from the vision system. The rotating sonar mounted in front of the robot senses them with the width of space occupied by the obstacle and distance from where robot sensed the obstacle is navigated around.

For accurate path navigation it is essential that the obstacle avoidance system function properly. This system consists of a rotating transducer, which makes mirror stops on either side of the centerline for obstacle avoidance.

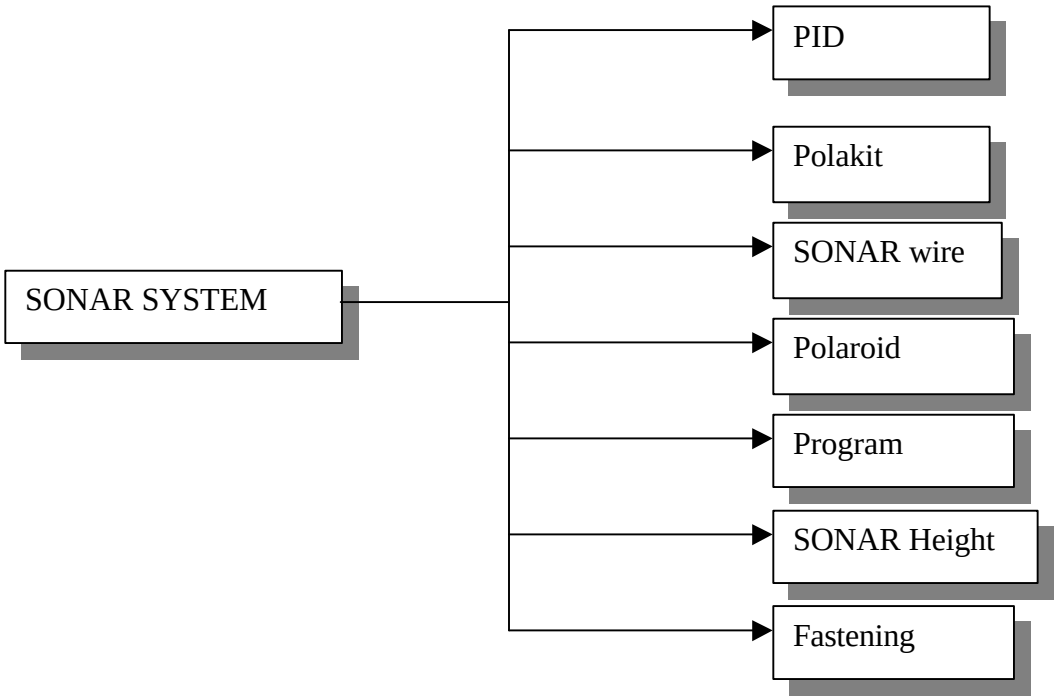


Figure 6: Elements of SONAR System

The major units are then sub classified into specific units for the purpose of building the relational database, which holds the information about the fault analysis of each component. Each specific unit is then analyzed for its failures and the required actions are then recorded which becomes an important source of information during the break down of the robot.

Chapter Three

Database Design

Need for the Database

Database systems are primarily concerned with the creation and maintenance of large and long-lived collections of data. It is a self-describing collection of integrated records and the repository of the data. The three main things you do with data are acquire it, store it and retrieve it. A relational database management system gives you a way of doing these tasks in an understandable and reasonably uncomplicated way. Organizing the various faults analyzed in various systems in a database will be very helpful for future reference and will be a better form of data management.

Relational database management systems currently constitute the most widely used mechanism for storing business information within the industry. It typically provides support for storing data used in traditional business applications such as banking transactions and inventory control. This important fundamental technology provides businesses with a flexible environment within which to store, retrieve, and process the data associated with the core systems they use. The structured query language (SQL) is now a widely accepted standard for both retrieving and updating data.

Advantages of Relational Database

The advantages of a relational database are numerous. Some of them are as follows

- Data entry, updates and deletions will be efficient.

- Data retrieval, summarization and reporting will also be efficient.
- Being well-formulated database, its behavior is predictable
- It is self-documenting.
- Changes to the database schema are easy to make.

Database Development Methodology

The various phases that are involved in the database design methodology of a fault diagnosis system are as follows.

- Understanding System Requirements: Finding out the requirements involved in developing is the first step. They will feed number of entities, attribute names, and types of data stored in each attribute. For example, in the Vision system developed for fault diagnosis, major system is the entity, reason for failure and solutions are the attributes and the type of data stored in each attribute is a string.
- Build Logical Data Model: The logical data model is built iteratively. The first view usually is done at a high level, beginning with a subject area or conceptual data model. Subsequent levels contain more detail like the failure analysis for the various components involved in any major system.
- Build the Physical Data Model: The logical data model is converted to a physical data model based on the specific database that is used. The major system of the robot and the specific component of the major system are used in all the queries used to retrieve data from the database.

- Refine the Data Model: The physical data model is refined continuously as more information becomes available, and the results of stress testing and benchmarking become available to the database development team.

- Populate the Data: After the database structure is established and the database is created, it is necessary to populate the database. This can be done through data scripts, applications, or data conversions.

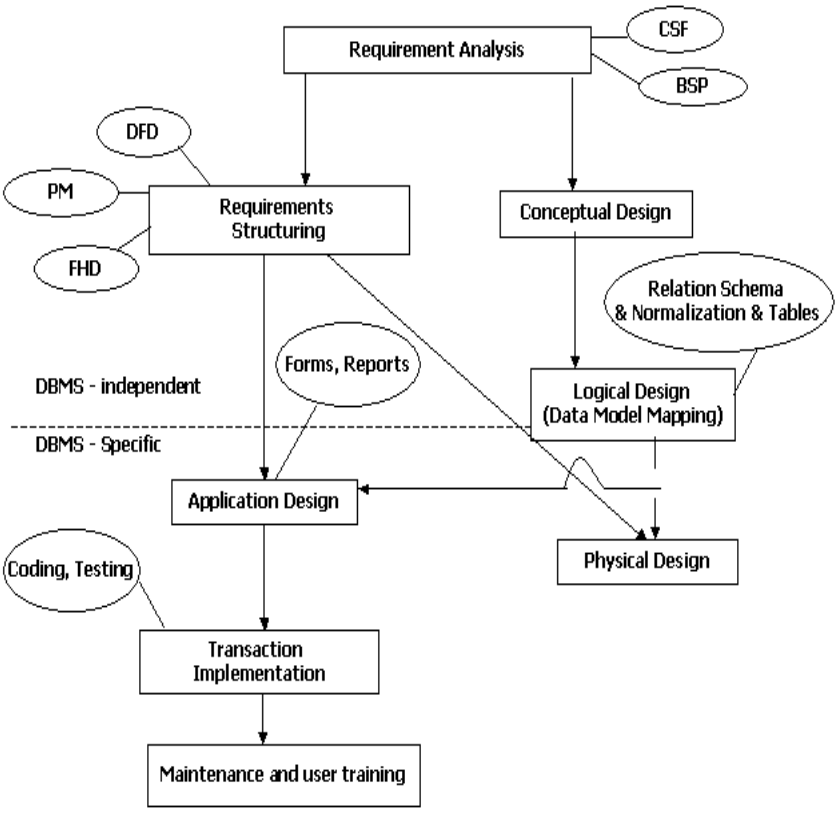


Figure 7: Diagrammatic Representation of Database Design

Design of Failure Mode Analysis Database

The Relational Database was created using Microsoft Access. The file was named dbaccess.mdb. The major attributes of the database developed are

- Major System
- Specific Component
- Fault Symptom
- Reason for Failure
- Solution to the Fault

The major system denotes the system of the robot on which we are looking for the fault. The major systems covered in the database are Mechanical System, Vision System, Power System and the SONAR System. The specific component refers to the sub parts of the major system. For each specific part taken into consideration, the failure mode is analyzed and solution is provided. Then for each failure mode analyzed, the possible failure symptom, the reason why the part fails and the steps to be taken to rectify the part is discussed. Dividing them into major sub systems and then into specific components covers almost the entire system of the Robot.

Each fault symptom is then analyzed depending on how the failed component is related to the rest of the components in the system. It is seen that for the same sub-classified unit there maybe more than one potential mode of failure. Once the fault symptom is known, then the possible reasons are analyzed so that the source of problem could be targeted and concentrated for fixing up the problems. The schematic diagram showing the design of the database is shown below.

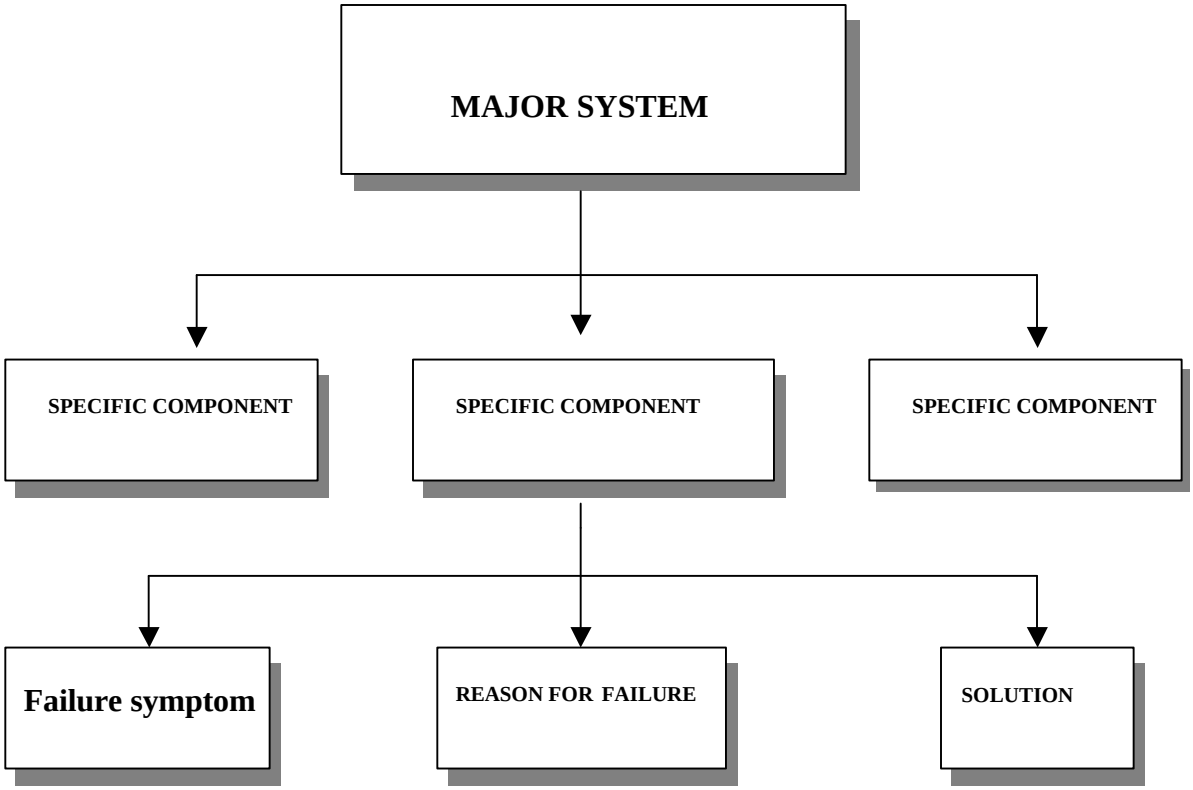


Figure 8: Diagram indicating the Design of Fault Diagnosis Database

The database is queried using SQL (Structured Query Language) and forms the central repository for accessing the information pertaining to the failures for the particular components and their potential effects and causes. The solutions provided for each failure has been obtained over the time with experience and also widely known facts.

An instance of the database table having the various values that it holds for the Mechanical System and the Vision System is shown below. This information could be upgraded to any database system with third party software's available in the markets.

System	Component	Fault Symptoms	Reason for Failure	Solutions to the Fault
Mechanical	Wheel2	Wheel coming off the shaft	Screw coming off the retainer	Check the size of the retainer screw size and secure it tightly
Mechanical	Fitting1	Heating of the main wheel bearings	Misalignment of the bearing and the plumber block	Check the levels with the spirit levels and adjust the plumber block
Mechanical	Fitting2	Wheels jammed	A bulged shaft due to improper fit	Filing or grounding of the shaft and inner hub of the wheel is necessary
Mechanical	Alignment1	Wheels doesn't turn smoothly	Improper lubrication and alignment	greasing on the inner surface of the coupling is necessary
Mechanical	Lubricant	Increase in the gear meshing noise	Lubricant level in the gearbox	The lubricant levels must be checked and filled with specified grade oil
Mechanical	Alignment2	Heating of servo motors	Alignment between servo motor and the gear box shaft	Horizontal and vertical alignment of the shafts to be done
Mechanical	Coupling	Heating of servo motors	Improper coupling alignment	Check the coupling between motor and gearbox and lubricate them frequently.
Mechanical	Servo Motor1	Servo motors turns very slowly	Amplifier output is insufficient	Re-adjust amplifier gain value and PID values
Mechanical	Servo Motor2	Servo motor doesn't rotate according to the pulses	Improper feedback	Make sure encoder meshes without slip and check the feedback circuit
Mechanical	Wheel Shaft	Wheel Shaft slipping out of coupling	Bending of wheel shaft	The wheel shaft must be secured with a collar on the inner side of the bearing
Mechanical	Shaft Keys	Key Moving out of its place	Shaft gets disengaged from the coupling and Robot stops	The shaft keys must be tightened at regular intervals with the Allen screw
Vision	Vision Threshold1	No points picked	Threshold limit not reached	Adjust threshold
Vision	CCD	No points picked	CCD camera not working	Check connections wrt to the schematic diagram
Vision	ISCAN1	Coordinates not available	ISCAN tracker is not functioning	Check the circuit and connections
Vision	Vision Threshold2	Incorrect points picked	Points outside the line are brighter	Readjust the threshold so that image on the line is seen as black image
Vision	Switching Unit1	Camera switch fails	Galil board controller not working	Reset the galil board
Vision	Switching Unit2	Camera doesn't switch properly	Switching unit not working	Check connections wrt to the schematic diagram
Vision	ISCAN2	Cannot calibrate	Image obtained doesn't reflect properly in the video monitors	Adjust the ISCAN settings written as specified on the robot panel

Chapter Four

JDBC Technology

What is JDBC

JDBC technology is an API that lets you access virtually any tabular data source from the Java programming language. It provides cross-DBMS connectivity to a wide range of SQL databases, and now, with the new JDBC API, it also provides access to other tabular data sources, such as spreadsheets or flat files.

The JDBC API allows developers to take advantage of the Java platform's capabilities for industrial strength, cross-platform applications that require access to enterprise data. With a JDBC technology-enabled driver, a developer can easily connect all corporate data even in a heterogeneous environment.

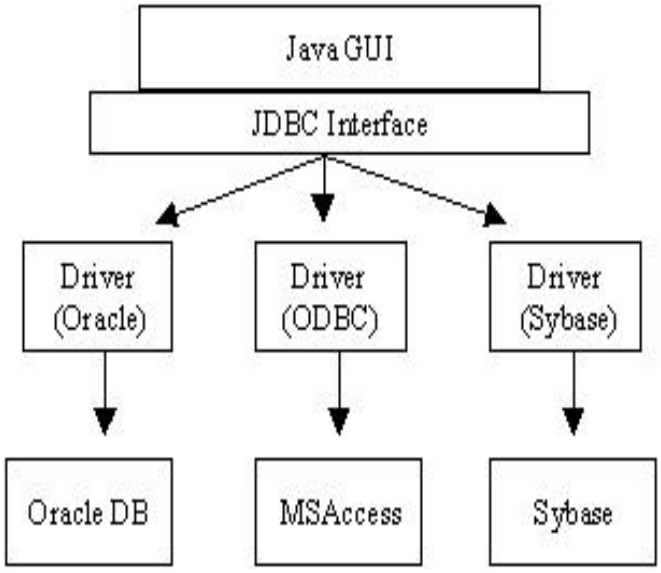


Figure 10: General Functionality of JDBC

JDBC makes it possible to develop database applications using Java and provides a standard interface between you and the database server. The JDBC API can be used regardless of what database is being used in the back end and it is supported by a large set of JDBC drivers. Using JDBC, a developer could execute SQL statements from any relational database. It is not necessary to develop a separate program to access databases from different vendors.

JDBC Architecture

A Java application built on top of JDBC API goes through three different phases:

- Open a connection to a database
- Create a statement objects through which it passes SQL statements to the DBMS
- Retrieve the results

The JDBC API contains two major sets of interfaces: the first is the JDBC API for application writers, and the second is the lower-level JDBC driver API for driver writers. JDBC technology drivers fit into one of four categories. Applications and applets can access databases via the JDBC API using pure Java JDBC technology-based drivers.

- Type 3 - Pure Java Driver for Database Middleware: This style of driver translates JDBC calls into the middleware vendor's protocol, which is then translated to a DBMS protocol by a middleware server. The middleware provides connectivity to many different databases.

- Type 4 - Direct-to-Database Pure Java Driver: This style of driver converts JDBC calls into the network protocol used directly by DBMS's, allowing a direct call from the client machine to the DBMS server and providing a practical solution for intranet access.

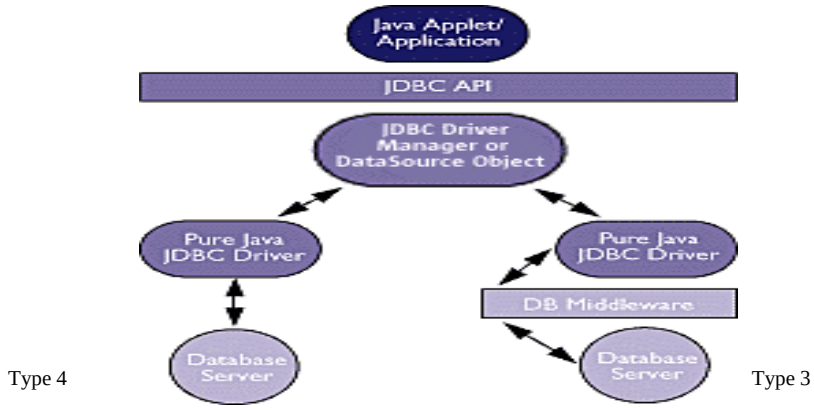


Figure 11: DATABASES VIA THE PURE JAVA JDBC TECHNOLOGY-BASED DRIVERS.

The graphic below illustrates JDBC connectivity using ODBC drivers and existing database client libraries.

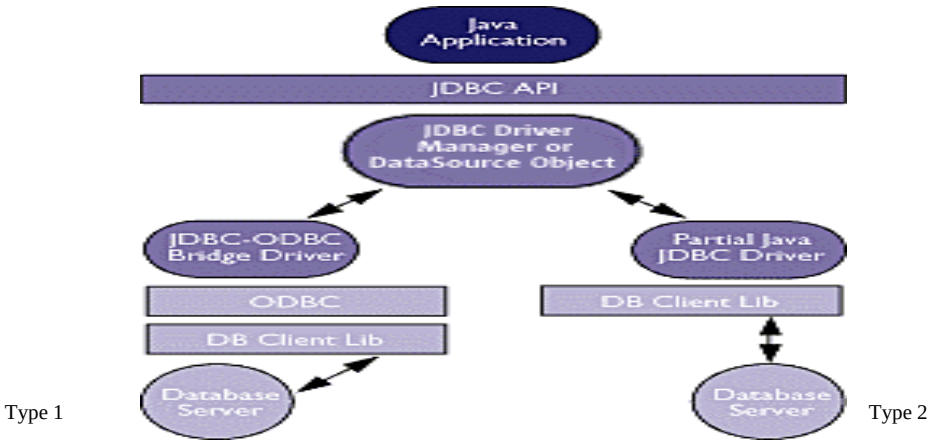


Figure 12: JDBC connectivity using ODBC drivers and existing database client libraries.

- Type 1 - JDBC-ODBC Bridge plus ODBC Driver: This combination provides JDBC access via ODBC drivers. ODBC binary code--and in many cases, database client code-- must be loaded on each client machine that uses a JDBC-ODBC Bridge. Sun provides a JDBC-ODBC Bridge driver, which is appropriate for experimental use and for situations in which no other driver is available.

- Type 2 - A native-API partly Java technology-enabled driver: This type of driver converts JDBC calls into calls on the client API for Oracle, Sybase, Informix, DB2, or other DBMS. Note that, like the bridge driver, this style of driver requires that some binary code be loaded on each client machine.

The Purpose and need for JDBC

- The emerging need of database connectivity for Java Applets will be met.
- Java's focus of cross-platform compatibility will be satisfied. There will be one uniform interface for programmers.
- Internet considerations will be fulfilled because of access to many types data sources (Oracle, Sybase, etc)
- The need to have a better low-level programming interface than ODBC. JDBC is simpler, based on Java (so no pointers like ODBC), and more secure.

How JDBC Works

The steps to accessing a relational database using JDBC are explained in detail below. It is assumed a database already exists.

Establishing a Connection

The first thing you need to do is establish a connection with the DBMS you want to use. This involves two steps: (1) loading the driver and (2) making the connection.

Loading Drivers

Loading the driver or drivers you want to use is very simple and involves just one line of code. If, for example, you want to use the JDBC-ODBC Bridge driver, the following code will load it.

```
Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
```

There is no need to create an instance of the driver and register it with the Driver Manager because calling Class.forName will do that automatically. When a driver is loaded, it is available for making a connection with a DBMS.

Making the Connection

The second step in establishing a connection is to have the appropriate driver connect to the DBMS. The following line of code illustrates the general idea.

```
Connection = DriverManager.getConnection (url);
```

If the JDBC-ODBC Bridge driver is used, the JDBC URL will start with jdbc:odbc: . The rest of the URL is generally the data source name or database system. So, if ODBC is used to access an ODBC data source called " dbaccess " for example, the JDBC URL could be jdbc:odbc: dbaccess. If one of the drivers loaded recognizes the

JDBC URL supplied to the method DriverManager.getConnection, that driver will establish a connection to the DBMS specified in the JDBC URL. The Driver Manager class manages all of the details of establishing the connection behind the scenes. The connection returned by the method DriverManager.getConnection is an open connection that can be used to create JDBC statements that pass SQL statements to the DBMS.

Creating JDBC Statements

A Statement object sends the SQL statement to the DBMS. A Statement object is created and then executed by supplying the appropriate execute method with the SQL statement. For a SELECT statement, the method to use is executeQuery. For statements that create or modify tables, the method to use is executeUpdate. The statement object is created as shown below.

```
Statement stmt = connection.createStatement();
```

The following example shows how the executeUpdate method is supplied along with the SQL code.

```
stmt.executeUpdate("SELECT a,b,c  FROM Table “);
```

Instead of the way it is written above, the whole query could be written in a string called “query” and the same method can be called as follows.

```
stmt.executeUpdate(query);
```

Executing Statements

The method executeUpdate is used because the SQL statement is a DDL (data definition language) statement. Statements that create a table, alter a table, or drop a table are all examples of DDL statements and are executed with the method executeUpdate. The method executeUpdate is also used to execute SQL statements that update a table. In practice, executeUpdate is used far more often to update tables than it is to create them because a table is created once but may be updated many times.

. The method used most often for executing SQL statements is executeQuery. This method is used to execute SELECT statements, which comprise the vast majority of SQL statements.etrieving Values from Result Sets

Retrieving Values from Result Sets

JDBC returns results in a ResultSet object, so an instance of the class ResultSet need to be declared to hold the results. The following code demonstrates declaring the ResultSet object rs and assigning the results of our earlier query to it.

```
ResultSet rs = stmt.executeQuery (“SELECT name.number FROM
addressbook");
```

Using the Method next

The variable rs, which is an instance of ResultSet, contains the rows of name and number. To access hem, each row is to be reached and the values are retrieved according to the type of data. The method next moves what is called a cursor to the next row and

makes that row the current row. Since the cursor is initially positioned just above the first row of a ResultSet object, the first call to the method next moves the cursor to the first row and makes it the current row. Successive invocations of the method next move the cursor down one row at a time from top to bottom.

Using Prepared Statements

Sometimes it is more convenient or more efficient to use a PreparedStatement object for sending SQL statements to the database. This special type of statement is derived from the more general class, Statement. If there is a need to execute a Statement object many times, it will normally reduce execution time to use a PreparedStatement object instead.

The main feature of a PreparedStatement object is that, unlike a Statement object, it is given an SQL statement when it is created. The advantage to this is that in most cases, this SQL statement will be sent to the DBMS right away, where it will be compiled. As a result, the PreparedStatement object contains not just an SQL statement, but also an SQL statement that has been precompiled. This means that when the PreparedStatement is executed, the DBMS can just run the PreparedStatement 's SQL statement without having to compile it first. Although PreparedStatement objects can be used for SQL statements with no parameters, you will probably use them most often for SQL statements that take parameters. The advantage of using SQL statements that take parameters is that, the same statement can be used repeatedly by supplying it with different values each time it is executed.

Creating a PreparedStatement Object

As with Statement objects, PreparedStatement objects are created with a Connection method. An example of creating a PreparedStatement object that takes two input parameters is shown below.

```
PreparedStatement updateSales = con.prepareStatement(
"UPDATE COFFEES SET SALES = ? WHERE COF_NAME LIKE ?");
```

The variable updateSales now contains the SQL statement, "UPDATE COFFEES SET SALES = ? WHERE COF_NAME LIKE ?", which has also, in most cases, been sent to the DBMS and been precompiled.

Supplying Values for PreparedStatement Parameters

Values need to be supplied in place of the question mark placeholders, if there are any, before executing a PreparedStatement object. Calling one of the setXXX methods defined in the class PreparedStatement does this. If the value to be substituted for a question mark is a Java int, the method setInt is called. If the value is a Java String, the method setString is called, and so on. In general, there is a setXXX method for each type in the Java programming language.

Using the PreparedStatement object updateSales, the following line of code sets the first question mark placeholder to a Java int with a value of 75:

```
updateSales.setInt(1, 75);
```

The next line sets the second placeholder parameter to the string " Colombian ":

```
updateSales.setString(2, "Colombian");
```

After these values have been set for its two input parameters, the SQL statement in updateSales will be equivalent to the SQL statement in the String object updateString.

Code Fragment 1:

```
String updateString = "UPDATE COFFEES SET SALES = 75
" + "WHERE COF_NAME LIKE 'Colombian'";
stmt.executeUpdate(updateString);
```

Code Fragment 2:

```
PreparedStatement updateSales = con.prepareStatement(
"UPDATE COFFEES SET SALES = ? WHERE COF_NAME
LIKE ? ");      updateSales.setInt(1, 75);
updateSales.setString(2, "Colombian");
 updateSales.executeUpdate():
```

If the SALES column were to be updated only once or twice, then there would be no need to use an SQL statement with input parameters. But if it is updated often, it might be much easier to use a PreparedStatement object. Once a parameter has been set

with a value, it will retain that value until it is reset to another value or the method clearParameters is called.

Creating the Complete JDBC Application

All the codes that need to be executed should be written inside a class definition. Then all the packages that are used in writing the codes are imported into the class file defined, For the JDBC application, java.sql package must be imported to handle all the SQL queries.

If a class is to be executed, it must contain a static public main method. This method comes right after the line declaring the class and invokes the other methods in the class. The keyword static indicates that this method operates on a class level rather than on individual instances of a class. The keyword public means that members of any class can access this method.

Using try and catch Blocks

Java requires that when a method throws an exception, there be some mechanism to handle it. Generally a catch block will catch the exception and specify what happens. The try block contains the method Class.forName, from the java.lang package. This method throws a ClassNotFoundException, so the catch block immediately following it deals with that exception. The second try block contains JDBC methods, which all throw SQLExceptions, so one catch block at the end of the application can handle all of the rest of the exceptions that might be thrown because they will all be SQLException objects.

Features and Advantages of JDBC

- **Leverage Existing Enterprise Data**

With JDBC technology, businesses are not locked in any proprietary architecture, and can continue to use their installed databases and access information easily - even if it is stored on different database management systems.

- **Simplified Enterprise Development**

The combination of the Java API and the JDBC API makes application development easy and economical. JDBC hides the complexity of many data access tasks, doing most of the "heavy lifting" for the programmer behind the scenes. The JDBC API is simple to learn, easy to deploy, and inexpensive to maintain.

- **Zero Configurations for Network Computers**

With the JDBC API, no configuration is required on the client side. With a driver written in the Java programming language, all the information needed to make a connection is completely defined by the JDBC URL or by a DataSource object registered with a Java Naming and Directory Interface™ (JNDI) naming service. Zero configurations for clients support the network-computing paradigm and centralize software maintenance.

❖ **No Installation**

A Pure JDBC technology-based driver does not require special installation; it is automatically downloaded as part of the applet that makes the JDBC calls.

❖ **Full Access to Metadata**

The JDBC API provides metadata access that enables the development of sophisticated applications that need to understand the underlying facilities and capabilities of a specific database connection.

❖ **Database Connection Identified by URL**

JDBC technology exploits the advantages of Internet-standard URLs to identify database connections.

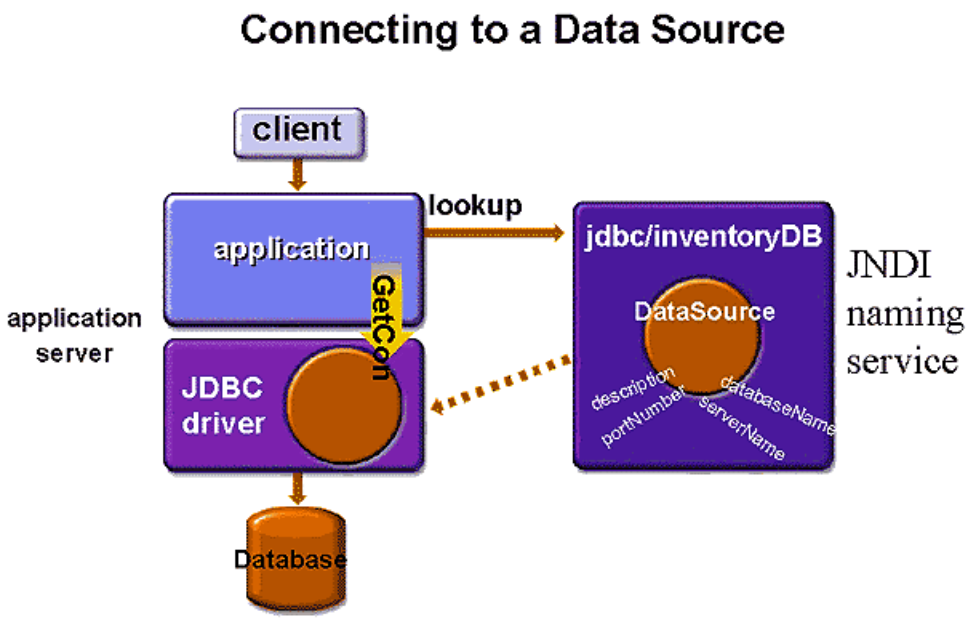


Figure 13: Connecting to a Database System

The new JDBC 2.0 API adds an even better way to identify and connect to a data source, using a Data Source object that makes code even more portable and easier to maintain. In addition to this important advantage, Data Source objects can provide connection pooling and distributed transactions, essential for enterprise database computing. This functionality is provided transparently to the programmer.

Chapter Five

Working of Online fault Diagnosis System

The information collected by grouping the components of the Robot are stored in a database developed using MS Access. The architecture of the Fault Diagnosis system developed using JDBC is shown below. The Java Applet developed is stored on the Robot Server. From the link provided for the fault diagnosis applet, the users from various places can send their choice and get back the desired results from the database. The figure below shows the flow of data from the database to the user interface through the Java database connectivity.

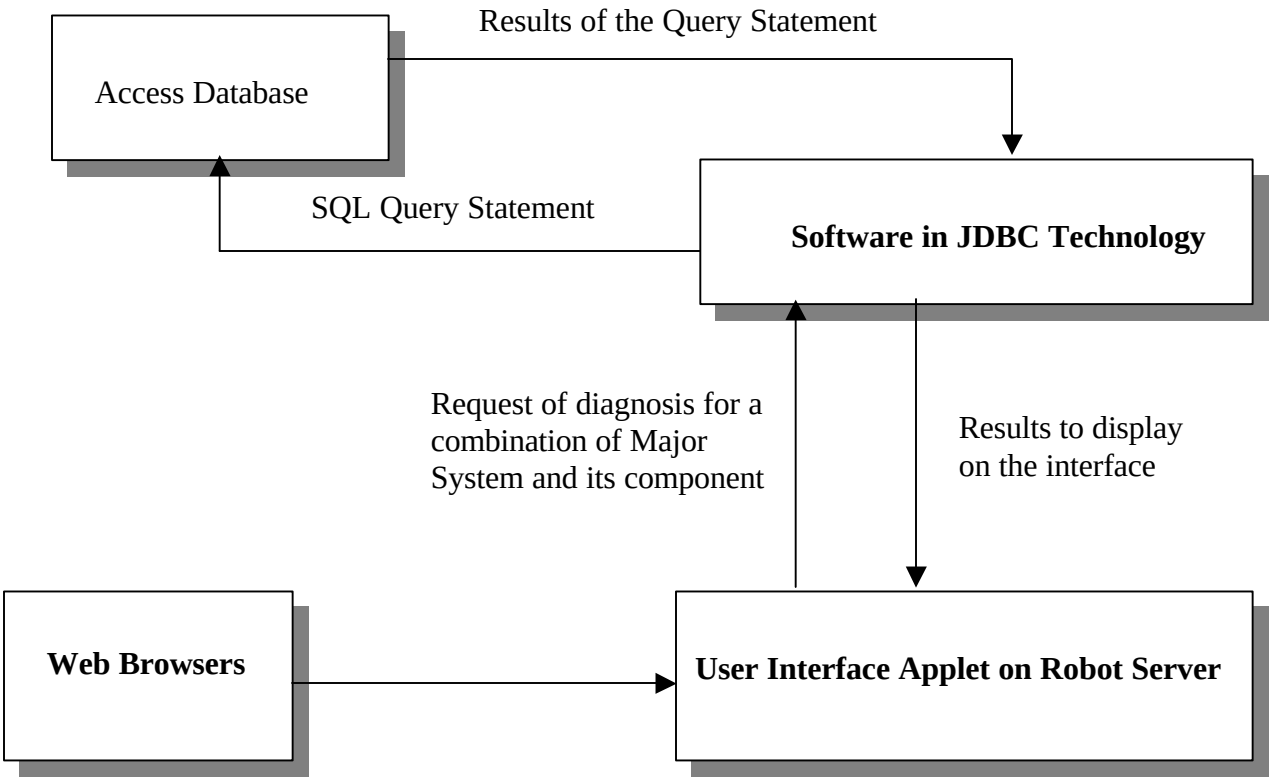


Figure 14: Architecture of Fault Diagnosis System

Development of the user interface

The interface developed is developed in the form of panels. The Java Swing classes are used completely to develop the interface. There are totally three Panels in the interface. The top panel contains the list for the major system and the list for the specific components. There are also two labels used to indicate the presence of two lists to the user. The top panel in itself is divided into two sub panels, each one having a label and a list placed side by side. These two sub panels are then placed in the top panel using the Grid Layout in the form of two rows and one column.

The middle panel contains the text fields displaying the results of the SQL queries from the database. The labels on the left side of the panel indicate to the user, which text field displays what information. The labels are stored in a panel called labelpanel and the text fields are placed in Textfield panel. Again for the whole of the middle panel, a Grid layout containing a row and two columns are applied. The bottom most panel contains the button represented in the form of UC logo. Pressing the logo produces action events, which in turn executes the queries and displays the results in the text fields. The layout of the interface is shown below.

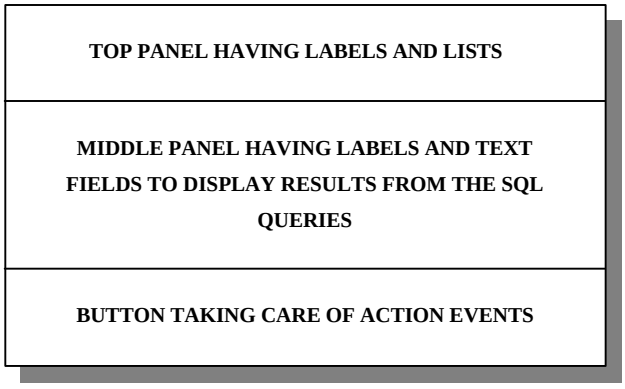


Figure 15: Layout of the User Interface

The whole interface is stored as a Java component. Border layout is used to place the panels in the components. The top panel, middle panel and bottom panel are respectively placed in the north, center and south of the component using the border layout commands.

Handling of the Action Events

On pressing of the UC logo, which acts as the action-handling component, the actionPerformed event of the button is provoked, which in turn executes the SQL query on the database. The user has to first select the major system from the first list. Once an item is selected, its action listeners are automatically invoked. The components of each major system are in turn stored in individual Vectors, which is a dynamic array. Based on the selection of the major system, the components of the system are automatically loaded in the second list using the setlistdata method. Now the user has to select the specific component for which he needs to find fault diagnosis. After selecting the major system and the specific component, the pressing of the button will establish link with the database using the Connection method and the query is automatically executed.

The values obtained after executing the SQL query is stored in a result set method. The getstring method is then used to assign the values stored in the result set object to the corresponding text fields of fault symptom, reason for failure and the solution to the problem. A separate method called display results is used to place the values in the text fields. So in all there are two action events that will occur for each choice made in the system list and component list. First is the automatic loading of

the component list based on the selection of the major system. Second is the display of results that will happen after the button press event. The exceptions that will occur during establishing the connection and executing the queries are caught and printed out in the main window. The flows of action events are shown in the diagram below.

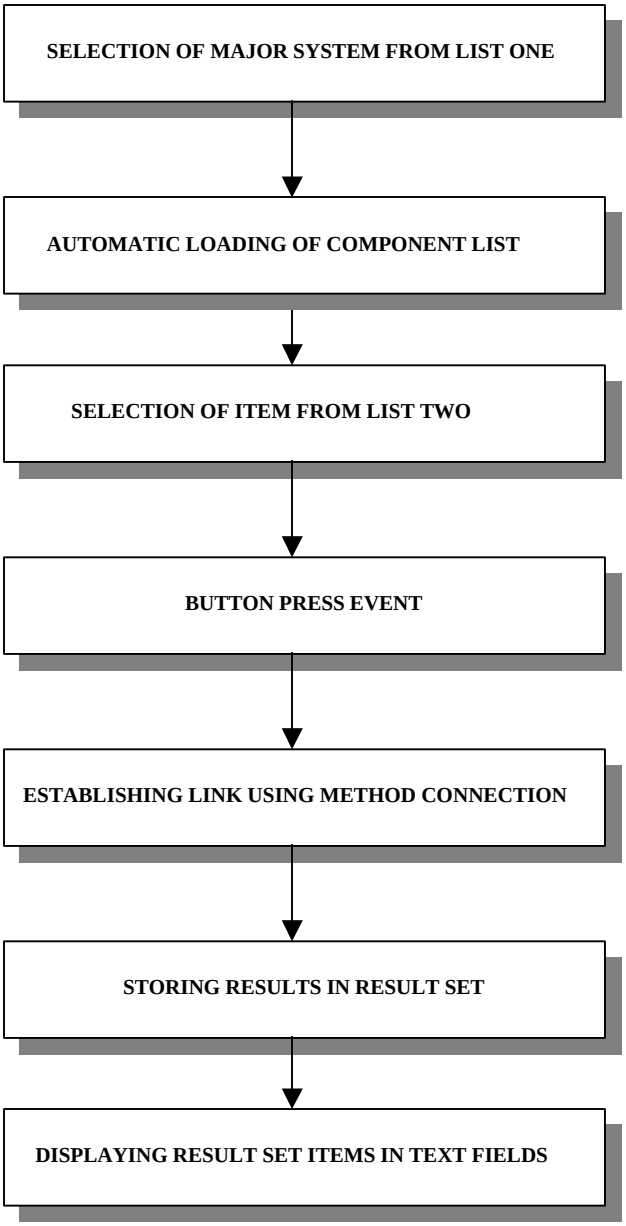


Figure 16: Flow of Action Events

Chapter 6

Conclusion and Recommendations

The online fault diagnosis system is built and it will be implemented into the robotics sun web server to efficiently manage the fault diagnosis and fixing the same.

The online system will be accessible with any popular web browser like Internet Explorer or Netscape Navigator. It will be available on the University of Cincinnati robotics web server.

In future, if the computer used on the Bearcat Robot is enhanced to have a web access and have an IP address on its own, it can be operated from a remote location without even a need to go near the robot. This is possible by using Java Networking classes and Object Streams to develop a Client-Server Application.

Client-Server Application will enable the accessing of the JDBC Fault diagnosis System from the Robot itself, right on the competition site without the need for any other external source of computer. In developing this application, the Robot Sun Server can be used as the server on which the database and the programs are stored and the computer in the Robot can be treated as a client. The user interface developed to access the database can be stored in the client computer, so that during the competition, the Fault Diagnosis system can be accessed directly on the site.

The same concept of Client – Server application can also be used to drive and activate the robot, without even going near the robot, provided there is an IP address for the computer used in the robot. The user interface similar to the one in JDBC application can be developed using Java Swing classes according to the requirements of the system

developed. The computer in the robot is used as the server and the client can be from any part of the world, provided a connection is established between the server and the client using the IP numbers of the respective computers. Thus by running the server program, the commands required to activate the robot can be written and included in the client program and the robot can be activated using the interface from the client program.

References

[1] Gregory Dudek, Michael Jenkin, Computational Principles of Mobile Robotics, ch4, Cambridge University Press, New York, 2000

[2] Nikam, B. Umesh, “A fault diagnostic system for an unmanned autonomous mobile robot”, Masters Thesis, 1997

[3] Sampath Kanakaraju, “Online fault diagnostic system for an unmanned autonomous mobile robot”, Masters Thesis, 2000

[4] Deitel & Deitel, JAVA – How to program, ch 18, Prentice Hall, New Jersey, 1999

[5] Fred R. McFadden, Jeffrey A. Hoffer, Mary B. Prescott, Modern Database Management, ch3, Addison-Wesley, New York, 1999

[6] David Flanagan, Jim Farley, William Crawford, Kris Magnusson, Java Enterprise in a Nutshell, ch5, O’Reilly & Associates, Inc., Sebastopol, CA, 1999

Appendix A

Program Scripts that are used in the Online Fault Diagnosis System

Program to take care of the Button Handling Events

```
import java.util.*;
import javax.swing.event.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.sql.*;



public class buttonevents  implements ActionListener
  {
    private listpanel lp;
    private ltpanel lt;
    private Connection connection;

    JButton submit;


    public buttonevents(listpanel s,ltpanel l, Connection c1)
     {
          lp = s;
          lt = l;
          connection = c1;



       Icon img1 = new ImageIcon("robologo.gif");

       submit = new JButton(img1);
       submit.setBackground(Color.pink);

       submit.addActionListener(this);
     }
```

```
  public void actionPerformed( ActionEvent e)
      {
          try
            {
              String query = " SELECT FaultSymptoms,ReasonforFailure,Solutions FROM
dbaccess " +
                          " WHERE (System ='" + sp.test.getSelectedValue() + "') AND
(Component ='" + sp.quest.getSelectedValue() + "') ";

                    Statement statement = connection.createStatement();

              ResultSet rs = statement.executeQuery(query);
              display(rs);
              statement.close();
            }

           catch( SQLException sqlex)
              {
              sqlex.printStackTrace();
              }
           }

     public void display( ResultSet rs)
       {
        try
         {
          rs.next();

          lt.fsymptom.setText(rs.getString(1) );
          lt.rff.setText(rs.getString(2) );
          lt.solution.setText(rs.getString(3) );
         }

        catch(SQLException sqlex)
         {
          sqlex.printStackTrace();

         }
       }
  }
```

Program Designing the List and Labels Panel

```
import java.util.*;

import javax.swing.event.*;

import java.awt.*;

import javax.swing.*;

import java.awt.event.*;

import java.sql.*;


public class listpanel extends JPanel

  {

    JPanel panelone,paneltwo;

    JList systemlist,componentlist;


    Vector mech = new Vector();

    Vector vision = new Vector();

Vector power = new Vector();

    Vector sonar = new Vector();

    String system[]= { "Mechanical", "Vision" , "Power", "SONAR"};


    public listpanel()

      {

        JLabel main =new JLabel("Select the Major System :");

        JLabel sub =new JLabel("Select the Specific Component :");
```

```
systemlist = new JList ( system);
systemlist.setVisibleRowCount(2);
systemlist.setFixedCellHeight(18);
systemlist.setFixedCellWidth(88);
systemlist.setBackground(Color.pink);
 systemlist.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

componentlist = new JList();
componentlist.setVisibleRowCount(4);
componentlist.setFixedCellHeight(18);
 componentlist.setBackground(Color.pink);
 componentlist.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    mech.addElement("Wheel1 ");
   mech.addElement("Servo Motor2 ");
    mech.addElement("Wheel Shaft");
    mech.addElement("Shaft Keys");
    mech.addElement("Wheel2");
   mech.addElement("Fitting1");
   mech.addElement("Fitting2");
   mech.addElement("Alignment1");
   mech.addElement("Lubricant");
   mech.addElement("Alignment2");
   mech.addElement("Coupling");
```

```
    mech.addElement("Servo Motor1");
    vision.addElement("CCD");
    vision.addElement("ISCAN1");
    vision.addElement("Vision Threshold2");
    vision.addElement("Switching Unit1");
    vision.addElement("Switching Unit2");
    vision.addElement("ISCAN2");

power.addElement("Battery1");
power.addElement("Battery2");
power.addElement("Fuse");
power.addElement("Battery3");
power.addElement("Invertor");
power.addElement("Main Switch");
power.addElement("SONAR Motor");
power.addElement("Servo Motor");
power.addElement("Solenoid");
 sonar.addElement("PID");
sonar.addElement("Polakit1");
sonar.addElement("Polakit2");
sonar.addElement("SONAR Wire1");
sonar.addElement("Polaroid");
sonar.addElement("Program");
```

```
sonar.addElement("SONAR Wire2");
        sonar.addElement("SONAR Height");
        sonar.addElement("Fastening");


      panelone= new JPanel();
      panelone.add(main);
      panelone.add(new JScrollPane(systemlist));
      panelone.setBackground(Color.pink);
paneltwo = new JPanel();
      paneltwo.add(sub);
      paneltwo.add(new JScrollPane(componentlist));
      paneltwo.setBackground(Color.pink);
       setLayout(new GridLayout(2,1));
      add(panelone);
      add(paneltwo);
systemlist.addListSelectionListener(new ListSelectionListener()
        {
          public void valueChanged( ListSelectionEvent e)
              {
                if(systemlist.getSelectedValue() == "Mechanical")
              {
               componentlist.setListData( mech );
                  }
```

```
            else if(systemlist.getSelectedValue() == "Vision")
             {
              componentlist.setListData( vision);
                }
            else if(systemlist.getSelectedValue() == "Power")
             {
              componentlist.setListData( power);
}
            else if(systemlist.getSelectedValue() == "SONAR")
              {
              componentlist.setListData( sonar);
                }
          }
        } );


    }
  }
```

Program Designing the Labels and Text Fields displaying results from the Database

```
import javax.swing.event.*;

import java.awt.*;

import javax.swing.*;

import java.awt.event.*;


public class ltpanel extends JPanel

 {

          JPanel labelpanel,textpanel;


        String labelstring[]= { "       Possible Fault Symptom "," Reason for Failure "," Corrective measures to be taken" };

        JTextField fsymptom,rff,solution;


   public ltpanel()

    {

      labelpanel= new JPanel();

      labelpanel.setLayout(new GridLayout(labelstring.length,1) );


      for(int i=0 ; i< labelstring.length ;i++)

      labelpanel.add(new JLabel( labelstring[i]));

      labelpanel.setBackground(Color.pink);
```

```
        textpanel = new JPanel();
        textpanel.setLayout(new GridLayout(labelstring.length,1) );
        fsymptom = new JTextField(15);
        fsymptom.setBackground(Color.pink);
        textpanel.add(fsymptom);

        rff = new JTextField(15);
        rff.setBackground(Color.pink);
        textpanel.add(rff);

        solution = new JTextField(15);
        solution.setBackground(Color.pink);
        textpanel.add(solution);
        setLayout(new GridLayout(1,2));
        add(labelpanel);
        add(textpanel);
      }
   }
```

Program establishing connection with the database driving other programs

```
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
```

```
import javax.swing.*;
import java.sql.*;
public class driverprogram extends JFrame
  {
    private listpanel lp;
    private ltpanel lt;
    private  buttonevents bev;

    private Connection connection;
    private String url;

   public driverprogram(String name)
    {
      super(name);
      Container c = getContentPane();
      c.setLayout( new BorderLayout() );

      lp = new listpanel();
      c.add(new JScrollPane(lp),BorderLayout.NORTH);
      lt = new ltpanel();
      c.add(new JScrollPane(lt),BorderLayout.CENTER);

      try
```

```
      {
         url = "jdbc:odbc:dbaccess";
         Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver");
connection = DriverManager.getConnection(url)

      }


   catch( ClassNotFoundException cnfex )
   {
          cnfex.printStackTrace()
   }

catch( SQLException sqlex
{
     sqlex.printStackTrace()
 }

catch( Exception ex )
     {
          ex.printStackTrace()
      }
```

```
    bev = new  buttonevents(lp,lt,connection );
    c.add(bev.submit,BorderLayout.SOUTH);

    setSize( 575,390);
    show();
}
  public static void main(String args[])
   {
     driverprogram s = new driverprogram(" Online Fault Diagnosis System ");

     s.addWindowListener( new WindowAdapter()
                          {
                            public void windowClosing(WindowEvent e)
                              {
                                System.exit(0);
                              }
                          }
                       );
   }
 }
```

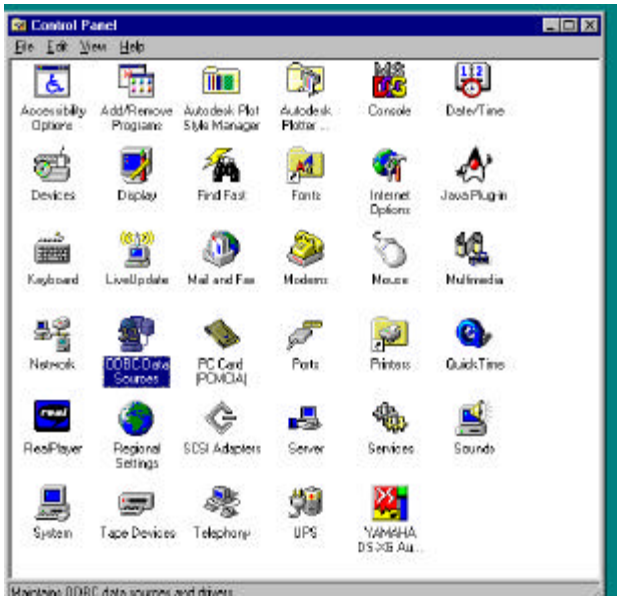
Appendix B - Contents of Database Table

System	Component	Fault Symptoms	Reason for Failure	Solutions to the Fault
Mechanical	Wheel2	Wheel coming off the shaft	Screw coming off the retainer	Check the size of the retainer screw size and secure it tightly
Mechanical	Fitting1	Heating of the main wheel bearings	Misalignment of the bearing and the plumber block	Check the levels with the spirit levels and adjust the plumber block
Mechanical	Fitting2	Wheels jammed	A bulged shaft due to improper fit	Filing or grounding of the shaft and inner hub of the wheel is necessary
Mechanical	Alignment1	Wheels doesn't turn smoothly	Improper lubrication and alignment	greasing on the inner surface of the coupling is necessary
Mechanical	Lubricant	Increase in the gear meshing noise	Lubricant level in the gearbox	The lubricant levels must be checked and filled with specified grade oil
Mechanical	Alignment2	Heating of servo motors	Alignment between servo motor and the gear box shaft	Horizontal and vertical alignment of the shafts to be done
Mechanical	Coupling	Heating of servo motors	Improper coupling alignment	Check the coupling between motor and gearbox and lubricate them frequently.
Mechanical	Servo Motor1	Servo motors turns very slowly	Amplifier output is insufficient	Re-adjust amplifier gain value and PID values
Mechanical	Servo Motor2	Servo motor doesn't rotate according to the pulses	Improper feedback	Make sure encoder meshes without slip and check the feedback circuit
Mechanical	Wheel Shaft	Wheel Shaft slipping out of coupling	Bending of wheel shaft	The wheel shaft must be secured with a collar on the inner side of the bearing
Mechanical	Shaft Keys	Key Moving out of its place	Shaft gets disengaged from the coupling and Robot stops	The shaft keys must be tightened at regular intervals with the Allen screw
Vision	Vision Threshold1	No points picked	Threshold limit not reached	Adjust threshold
Vision	CCD	No points picked	CCD camera not working	Check connections wrt to the schematic diagram
Vision	ISCAN1	Coordinates not available	ISCAN tracker is not functioning	Check the circuit and connections
Vision	Vision Threshold2	Incorrect points picked	Points outside the line are brighter	Readjust the threshold so that image on the line is seen as black image
Vision	Switching Unit1	Camera switch fails	Galil board controller not working	Reset the galil board
Vision	Switching Unit2	Camera doesn't switch properly	Switching unit not working	Check connections wrt to the schematic diagram
Vision	ISCAN2	Cannot calibrate	Image obtained doesn't reflect properly in the video monitors	Adjust the ISCAN settings written as specified on the robot panel

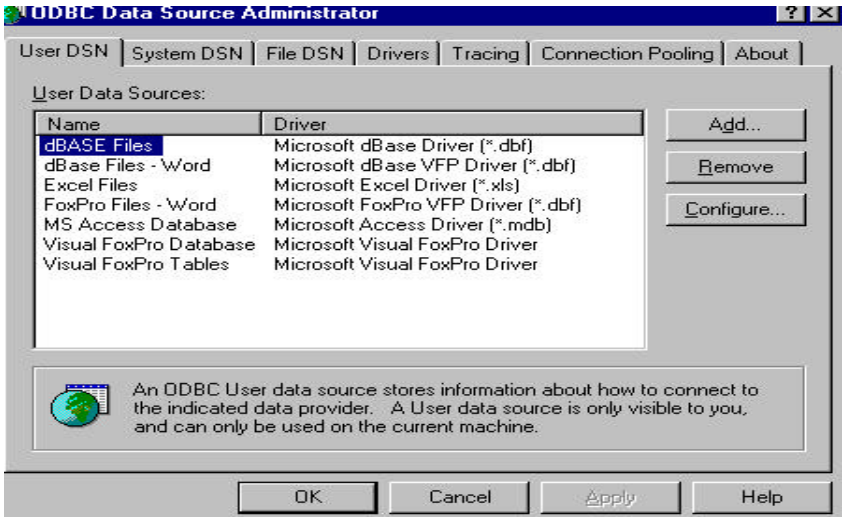
System	Component	Fault Symptoms	Reason for Failure	Solutions to the Fault
Power	Battery1	No Main power	Battery down	Check voltage by the voltmeter which is mounted on the side panel
Power	Fuse	No Power	Fuse blown	Check and replace fuse
Power	Battery3	Low power	Battery down	Check voltage by the voltmeter which is mounted on the side panel
Power	Invertor	No Power	Loose connection	Check connection at the invertor
Power	Main Switch	No power	Main switch in off position	Switch the main on
Power	SONAR Motor	No main Power	Battery down	Check the battery voltage and charge if below the 36V
Power	Servo Motor	low power	Less voltage input due to low battery voltage	Check the voltage for 36V and charge the batteries sufficiently
Power	Solenoid	No power	Solenoid coil is burnt	Check the voltage across the solenoid energizing coil
SONAR	PID	No intermittent stops	Improper SONAR amplifier output	Adjust the PID values
SONAR	Polakit1	Error in calculating the obstacle distance	Sufficient reflections not received from the obstacle	Adjust parameters in Polakit program
SONAR	Polakit2	Error in calculating the obstacle distance	Micro controller malfunction	Adjust parameters in Polakit program
SONAR	SONAR Wire1	Failure to detect obstacle	SONARs not working	Secure connections, replace SONARs
SONAR	Polaroid	Failure to detect obstacle	Polaroid board may no tbe working	Secure connections, refer polaroid manual
SONAR	Program	Failure to detect obstacle	Polakit program values not adjusted	Reset parameters in the program
Mechanical	Wheel1	Wheel rotating freely	Shaft key missing	Remove the wheel and place appropriate shaft key
SONAR	SONAR Wire2	Failure to initiate change in steering	Open in the circuit between CPU and Polaroid board	Secure connections
SONAR	SONAR Height	False feedback pulse	Detects grass as obstacle	Adjust elevation to 18" from ground to Edge of SONAR polaroid
SONAR	Fastening	False feedback pulse	Loose SONAR polaroid	Check if the SONAR is secured to the motor shaft
Power	Battery2	Low battery power	Battery down	Check voltage by the voltmeter which is mounted on the side panel

Appendix C – Establishing Connection between the Database and the Java Program

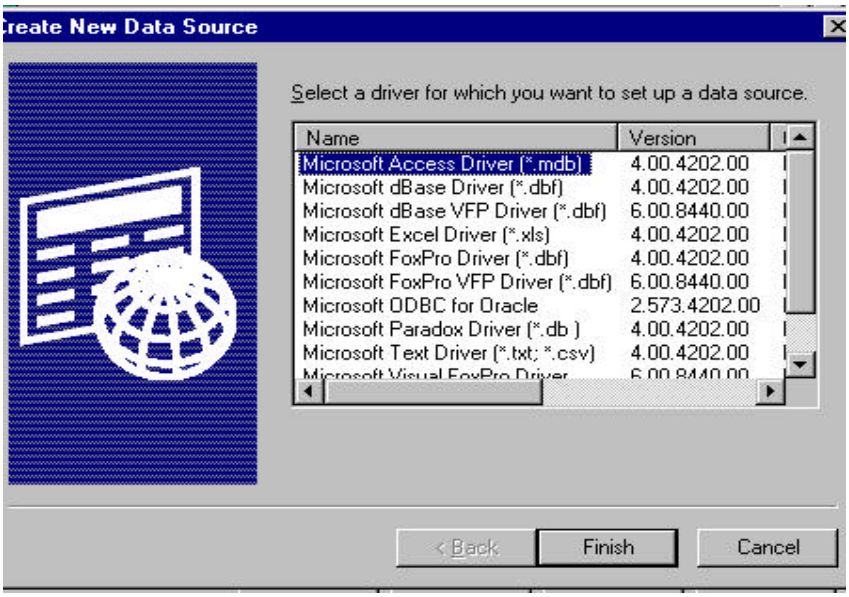
Step One: Open Control Panel and Click on ODBC Data Sources



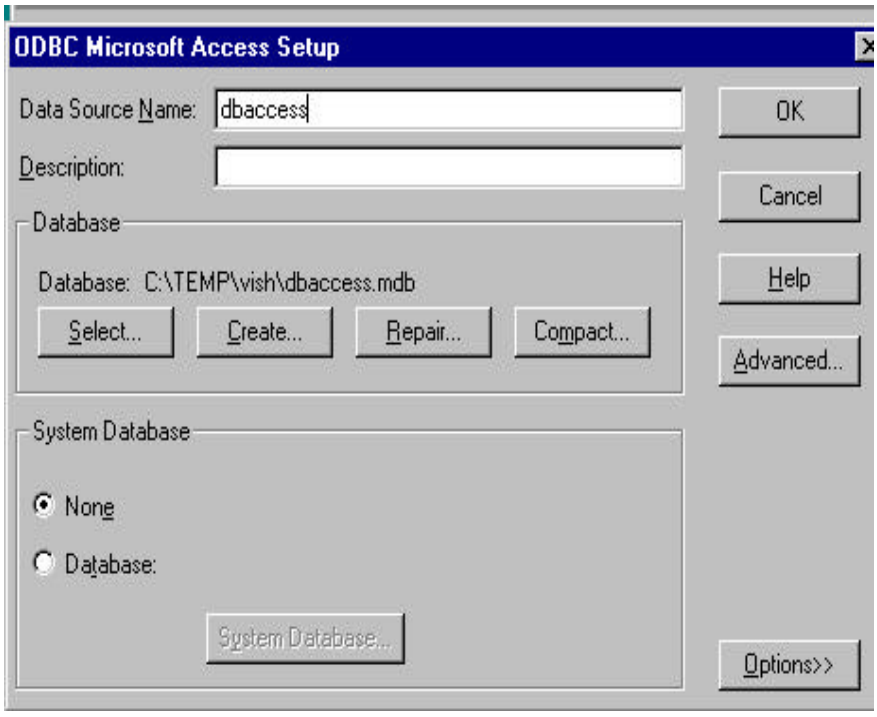
Step Two: Click on Add in the Screen that follows



Step Three: Click on Finish after highlighting the MS Access Driver



Step Four: Select the Access file from the directory and press OK



Highlighted Access file dbaccess confirms the connection established