University of Southern Queensland

Faculty of Engineering & Surveying

Modular Robot Communication Interface

A dissertation submitted by

Kevin Stark

in fulfilment of the requirements of

ENG4112 Research Project

towards the degree of

Bachelor of Engineering (Software)

Submitted: November, 2006

Abstract

The field of robotics is a relatively new technology in comparison with other engineering technologies. The USQ Modular Robot is a continually evolving robot development project which brings together a wide variety of engineering disciplines such as mechanical, electrical and software together. The past research projects on the Modular Robot have produced a set of mechanical components to build a robot structure with, and a set of distributed controllers on a CAN network that can provide control over motors, actuators and sensors throughout the robot. This project aims to extend functionality of the distributed controllers by researching and developing a control system for the Modular Robot.

While it has not been possible to achieve full positional control of the robot due to the lack of software functionality in the distributed modules, this project has successfully produced a highly configurable real-time communication interface to the CAN bus. This interface will provide the groundwork required for further research in the field of automation and control of the Modular Robot.

University of Southern Queensland

Faculty of Engineering and Surveying

ENG4111/2 *Research Project*

Limitations of Use

Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.
I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Kevin Stark

0050009783

Signature

Date

Acknowledgments

I would like to acknowledge the support I have received from my supervisor Mr. Mark Phythian and also the assistance of Dr. Wei Xiang during this project. Their help and guidance have been invaluable over the last year.

I would also like to thank the support of enumerable friends and family who have kept me sane and motivated over the last four years. In the words of Matthew Reilly - *Never underestimate the power of your encouragement.*

KEVIN STARK

University of Southern Queensland

November 2006

Contents

List of Figures

List of Abbreviations

Abbreviation	Explanation
CAN	Controller Area Network.
PWM	Pulse Width Modulation
XML	Extensible Markup Language
DLL	Dynamically-Linked Library
GUI	Graphical User Interface

Chapter 1

Introduction

1.1 Project Outline

The USQ Modular Robot is a continually evolving project which aims to develop a complete robot system from a set of basic components. Currently, the main components of the Modular Robot system are a set of basic mechanical components, and a set of distributed controllers linked by a Controller Area Network (CAN) bus.

The aim of this project has been revised due to functionality issues with the distributed controllers and is now based around providing a real-time communication link to the CAN bus. The specific objectives are to develop guidelines for the CAN message protocol used by the Modular Robot and to develop a PC based message handling program to simplify the control of the robot. This software project is an extension of research undertaken by Francois Hoffman (2005) in the area of low cost distributed control. Much of the content in this report is based on the analysis and understanding of his work.

1.2 Overview of the Dissertation

This dissertation is organized as follows:

Chapter 2 describes the general background of robotics and control systems.

Chapter 3 describes the distributed control system of the Modular Robot and its current operation.

Chapter 4 details the application of the CAN bus to this project and its current limitations.

Chapter 5 describes the design of the software interface as developed by this project.

Chapter 6 shows the testing procedures used on the program and the results.

Chapter 7 details the operation of the communication interface and how it can be integrated into future projects.

Chapter 8 concludes the dissertation and suggests further work in the kinematics and human interface areas of the Modular Robot software.

Chapter 2

Background

2.1 Chapter Overview

This chapter introduces the general concepts and designs of robots in industrial environments, and the history of the USQ Modular Robotdevelopment. A brief introduction to the concept of a CAN bus is also described, which forms the backbone of this project's development.

2.2 Robot Applications and Designs

There are a number of basic robot manipulator design configurations in production today which can be mixed to achieve almost any type of manipulator imaginable. The two key designs are Cartesian and articulated manipulators with the minor ones being the SCARA configuration, the spherical configuration and the cylindrical manipulator. The two key designs are described below and each type of manipulator is shown in Figures 2.1 to 2.5. While these designs are primarily for industrial robot situations, they also make up the basic joints and connections for almost any type of robot application.

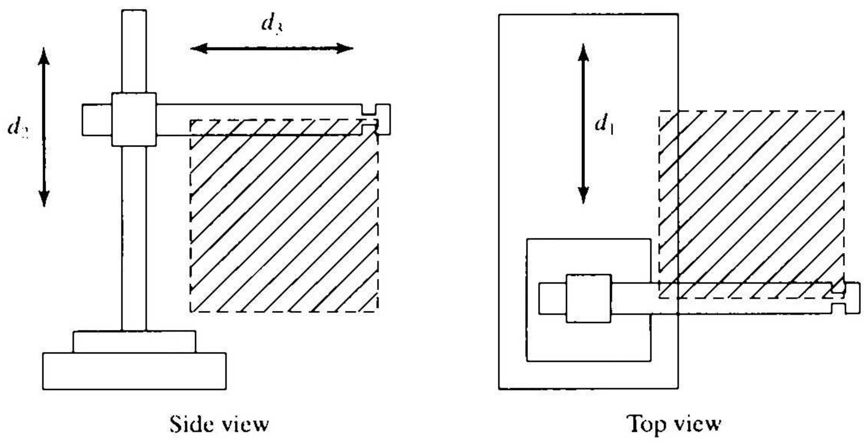


Figure 2.1: A Cartesian manipulator.

Source: (Craig 2005, p. 234)

2.2.1 Cartesian manipulators

Cartesian manipulators are one of the simplest robotic configurations to design and control because their three primary joints are mutually orthogonal (see Figure 2.1). They are most often seen in large scale industrial situations which require heavy loads and/or fine precision. The major limits of Cartesian manipulators however is that the entire work area of the robot must be inside the physical robot structure.

2.2.2 Articulated manipulators

Articulated manipulator are very similar to a human arm in that they use elbow and wrist joints to reach out from a center base structure (see Figure 2.2). They have the advantage over Cartesian manipulators in that they can be mounted at a central location in the workplace and 'reach out' to the surrounding areas. They are best suited for small work areas that do not contain extremely heavy loads.

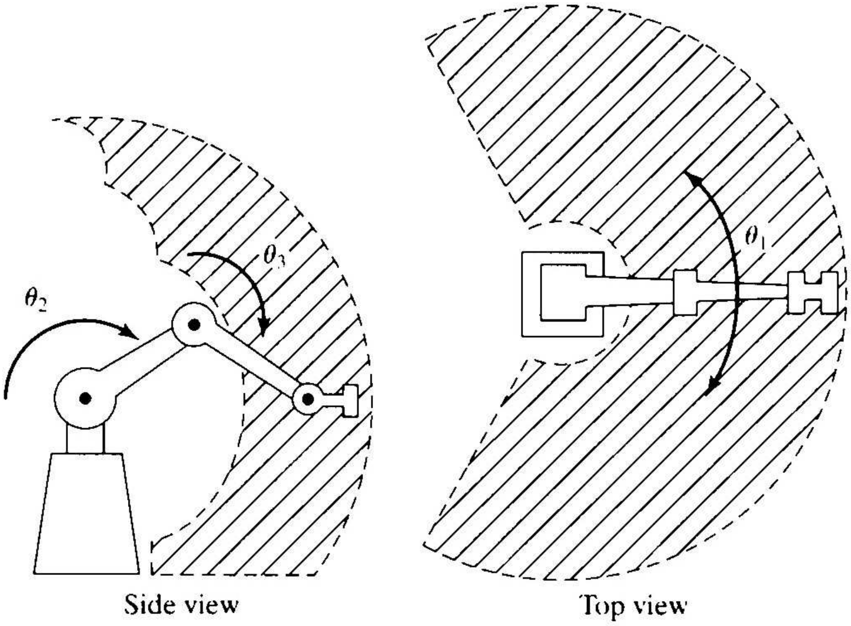


Figure 2.2: An articulated manipulator.

Source: (Craig 2005, p. 235)

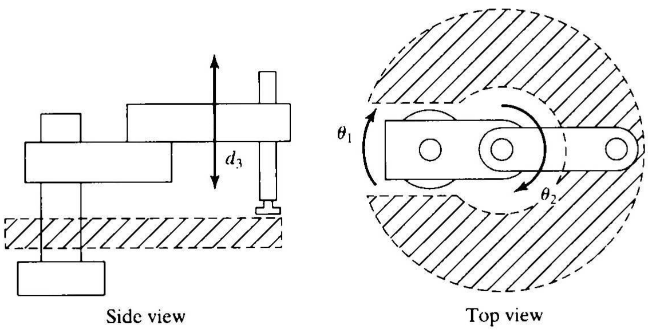


Figure 2.3: The selectively compliant assembly robot arm (SCARA) manipulator.

Source: (Craig 2005, p. 236)

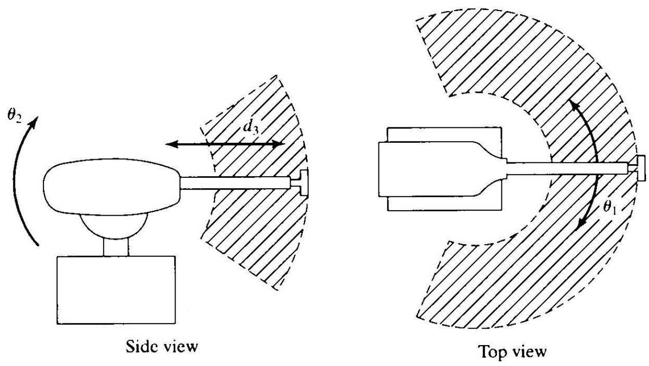


Figure 2.4: A spherical manipulator.

Source: (Craig 2005, p. 236)

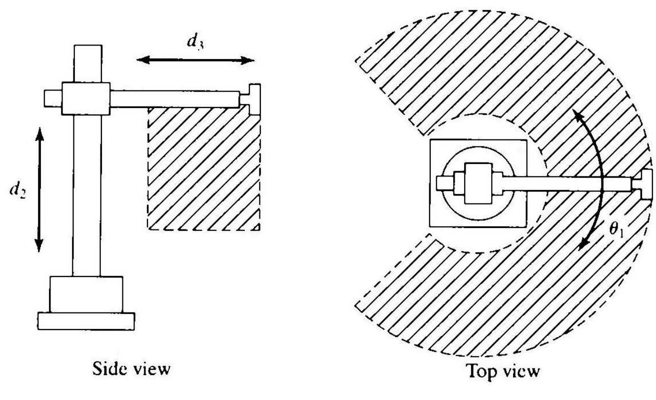


Figure 2.5: A cylindrical manipulator.

Source: (Craig 2005, p. 237)

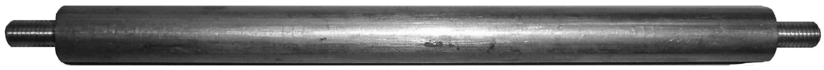


Figure 2.6: Rod component of the Modular Robot's mechanical structure.

2.3 USQ Modular Robot History

The original Modular Robot development was proposed by Mr. Mark Phythian in 2001 and undertaken by Lake Teoh at USQ (Teoh 2001). The requirements at that stage were to develop a minimal set of mechanical components that could be connected together to form a simple robotic structure. The key mechanical components developed by Lake Teoh are the rod and connector shown in Figure 2.6 and Figure 2.7.

A centralised control network was also developed for the Modular Robot by Markus Billerwell (2001). This used a single master processor that could control various motors, sensors and actuators using a number of pluggable "cards". To make this system more user-friendly, a 3D graphical interface was developed that would allow the user to manipulate a virtual model of the robot which would send commands to the on-board computer and physically replicate the movements in real-time (Scouller 2002). While both of these systems were working, the centralised controller was a very complicated system to set up as there were often large numbers of wires required to fully connect the external motors and actuators to the master processor.

Recent work by Francios Hoffman on the Modular Robot has resulted in a distributed control network which consists of a set of basic motor, actuator and sensor modules. The modules can be attached to any part of the robot structure which requires control. They can then be connected together via the CAN bus which completes the distributed network.

Concurrent work to this project on the Modular Robot has been in the area of developing a real-world application for the Modular Robot system. It is expected that this development will result in a configurable walking robot structure that can be used to further extend the research possibilities of the Modular Robot development.

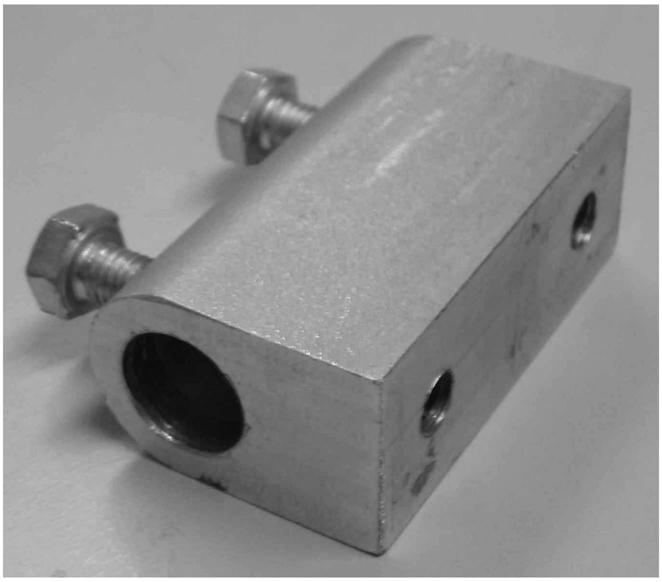


Figure 2.7: Connector component of the Modular Robot's mechanical structure.

2.4 Controller Area Network (CAN)

2.4.1 Features

The CAN system was designed by Robert Bosch GmbH in the late 1980s as a reliable high-speed communication network for use in motor vehicles. It is essentially an advanced serial bus system that efficiently supports distributed control systems (MicroController.com 1999). The main advantages of the CAN protocol in automotive and robotic applications as defined by MicroController.com (1999) are as follows:

- Low cost - It is a fast serial bus with only two wires which gives it a good price/performance ratio. There are also a large number of controllers and transceivers available, mainly driven by high volume production in the automotive market.
- Reliable - Sophisticated error detection and error handling mechanisms results in high reliability transmission. Erroneous messages are detected and repeated while system-wide data consistency is maintained as every bus node is informed about an error. Faulty nodes automatically withdraw from bus communication and the standard twisted pair wires give the physical bus high immunity to electromagnetic interference.

- Real time communication - Maximum data rate of 1MBits/s on a 40m bus length, while still capable of maintaining about 40kBits/s on a 1000m bus. Low latency between transmission request and actual start of transmission. Priority based messages to ensure that the most important message will win arbitration without losing any bus time.

- Flexible operation - Every node is able to access the bus individually and without delay. There are no physical addresses assigned to the nodes which means that any number of nodes can be added or removed without affecting the communication.

- Multicast / Broadcast capable - Messages are not identified by address or destination, but rather by priority and data contents. Messages are received by all nodes on the bus and can be used by none, one, many or all nodes on the network simultaneously.

- ISO standard - The CAN protocol has been accepted by the International Organization for Standardization (ISO) who have published two versions of CAN standards. `ISO-11898` for high speed applications and `ISO-11519-2` for low speed applications.

2.4.2 Physical Layer and Hardware

Implementing a CAN bus involves a number of layers which may be dependent on the individual application at hand. At the lowest level is the physical layer medium which must be chosen so that it is able to transmit the “dominant” and “recessive” bit states. The next layer involves a CAN transceiver which drives the physical layer using data supplied by the next higher layer again - the CAN controller.

The most simplistic and common implementation is shown in Figure 2.8 from Micro-Controller.com (1999). The physical layer in this case is a twisted pair of wires. The push-pull voltage system on a twisted pair of wires is an extremely effective prevention against electromagnetic interference on the bus. There are a large number of cheap and effective CAN controller and transceiver chips on the market today as found by Hoffman (2005). Upper market PIC modules and other embedded processors usually offer a built-in CAN controller which requires only the use of a CAN transceiver.

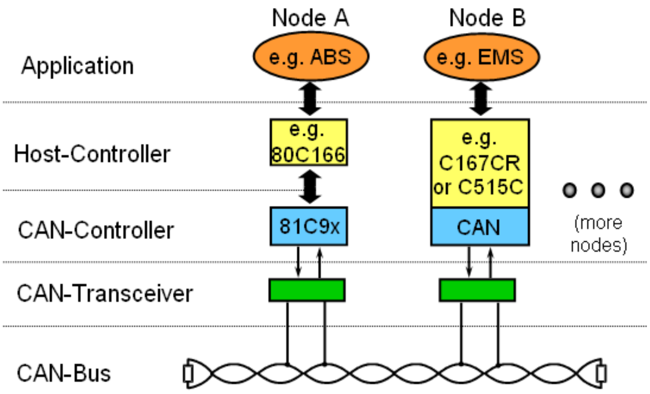


Figure 2.8: Typical embedded CAN bus configuration showing layer segmentation. source: MicroController.com (1999)

2.4.3 CAN Message Types

The CAN protocol specifies 4 different message types or "frames":

Data Frame: The data frame is the most common message on the CAN bus as it is used by nodes to broadcast new information.

Remote Frame: The remote frame is used to request specific information from the CAN bus.

Error Frame: An error frame is generated by all nodes when a bus timing error is detected.

Overload Frame: The overload frame is largely redundant on modern CAN systems as the majority of CAN controllers are more than capable of handling the bus traffic.

The data and remote frames have very similar format as can be seen in Figures 2.9 and 2.10. The main components in the message are the arbitration field, control field, data field, and CRC field. The arbitration field defines the message contents, and is used to determine the priority of a message when multiple nodes are accessing the bus. The control field contains the data length property, which specifies the number of bytes to follow in the data field. The data field is where the data and remote frames differ in specification. In the data frame, the data field contains zero to eight bytes of data as

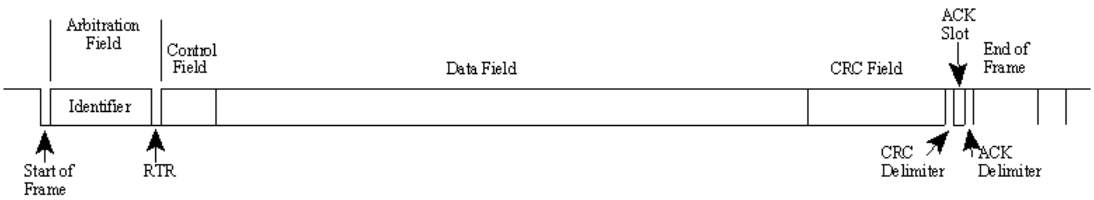


Figure 2.9: Standard CAN data frame. source: (kvaser, 2005)

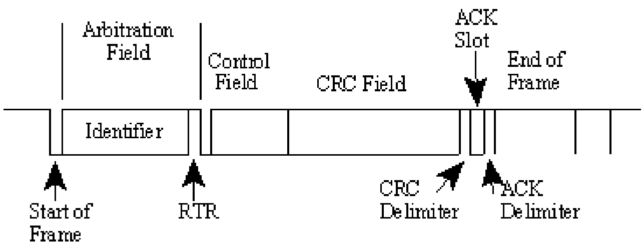


Figure 2.10: Remote CAN request frame. source: (kvaser, 2005)

specified by the control field. In the remote frame however, there is no data field and the control field specifies the number of bytes it expects in response. Also set in the remote frame it the RTR bit after the arbitration field. This bit is set as passive in a remote frame to ensure that if a node broadcasts the requested information at the same time as the request, the data frame will win arbitration. The CRC field is a 15-bit checksum calculated on most parts of the message to ensure the message is received properly.

2.5 Chapter Overview

Although the mechanical components developed by Lake Teoh are very basic and primitive, they do form the basic mechanical components required to construct any or all of the robot designs in Section 2.2. The movement and control for each of the robot joints is made possible by the distributed control modules developed by Francois Hoffman, and the communication throughout the robot is based on the CAN protocol. This leads to the focus of this project which is in the area of software control.

Chapter 3

USQ Modular Robot

3.1 Chapter Overview

This chapter outlines the distributed control system of the Modular Robot and examines the current operational capabilities of the distributed modules.

3.2 Distributed Control Modules

The six modules developed by Francois Hoffman are designed to be completely independent from each other and are able to control any type of motor, actuator or sensor commonly used in robotics. Each module has its own microprocessor and communication interface, and also has a number of specialised I/O ports depending on its type and purpose. The six modules are as follows and are described in more detail in the following sections.

1. DC motor control module;
2. Stepper motor control module;
3. Pneumatic proportional control module;
4. Pneumatic two-way valve controller;

5. Sensor module;
6. Master module;

The microprocessor chosen by Francois Hoffman for each of the modules was the PIC16F88. This processor, coupled with the MCP2510 CAN controller provided all of the I/O and communication requirements while still being a low cost setup (Hoffman 2005, p. 10).

Currently, the PIC code running in each of the modules is only in the very basic prototype stage. Due to the lack of software documentation provided by Francois Hoffman, a walk-through of the original assembly code was performed for each of the modules to determine the current capabilities.

3.2.1 DC Motor Control Module

The DC motor control module is designed to be able to control a standard Direct Current (DC) motor using Pulse Width Modulation (PWM). The generic prototype circuit board is shown in Figure 3.1. The hardware features of this module are as follows:

- PWM to allow variable speed control of the motor;
- H-bridge circuit design that can operate in forward, reverse, brake or free spin mode.
- Two analogue inputs for position or speed measurement which can also be configured as digital I/O if such position or speed measurements are not required;
- Two digital I/O ports for home or end stop sensors;

Analysis of the assembly code for the DC motor control module has found that Francois Hoffman outlined two main modes of operation. In SPEED mode, the module could be given a set speed and direction. It would maintain that SPEED until the next command was received. In POSITION mode, the module could be given a position in

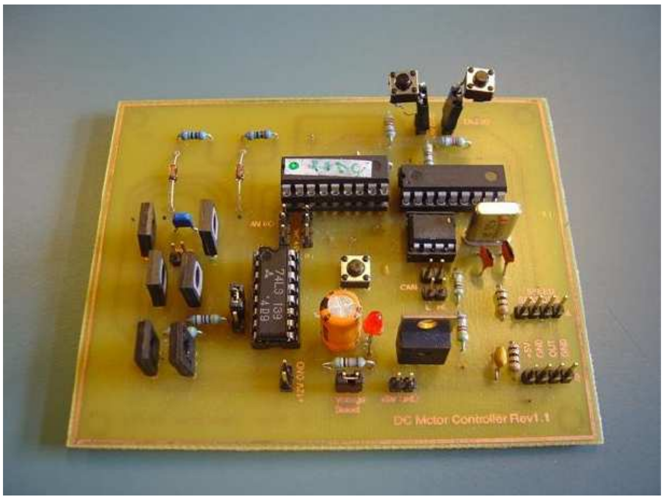


Figure 3.1: DC motor control module.

which to move to and a maximum speed and it would determine the direction necessary to achieve that position.

3.2.2 Stepper Motor Control Module

The stepper motor control module has similar purpose and features to that of the DC motor control module. The circuitry and output control however, are designed specifically for a stepper motor. Speed and direction of the motor are controlled using a half-stepping bit pattern (Hoffman 2005, p. 36). The generic prototype circuit board for the stepper motor controller is shown in Figure 3.2 and the hardware features of this module are as follows:

- Bit stepping pattern to precisely control speed and direction of the motor;
- Two analogue inputs for position or speed measurement which can also be configured as digital I/O if such position or speed measurements are not required;
- Two digital I/O ports for home or end stop sensors;

The software operation of the stepper motor control module is very similar to that of the DC motor. While the same POSITION and SPEED modes are defined in the

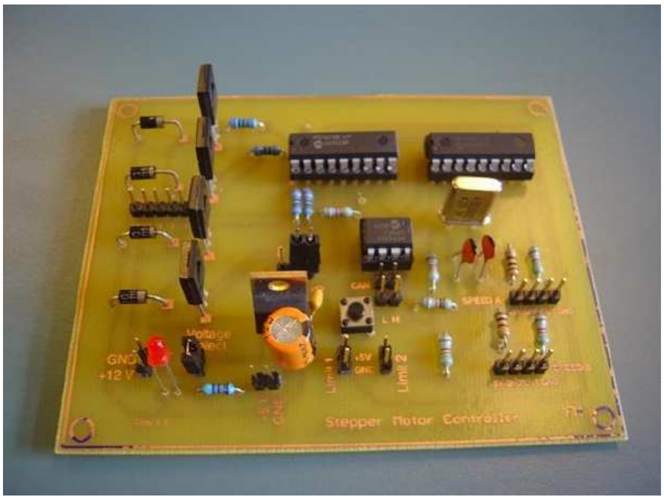


Figure 3.2: Stepper motor control module.

assembly code, current testing using a range of different values has not been able to produce movement from the stepper motor.

3.2.3 Pneumatic Proportional Control Module

The pneumatic proportional control module is designed to allow the control of four valves to control two actuators. This allows the two pneumatic actuators to be controlled either fully in or out, or with the appropriate feedback the actuators can be adjusted to any position in between. The generic prototype circuit board for the pneumatic proportional controller is shown in Figure 3.3 and the hardware features of this module are as follows:

- Two analogue ports available for linear transducers as positional feedback;
- Two digital I/O ports for home or end stop sensors;

The software currently implemented in the proportional pneumatic module only allows allows control over each of the valve outputs. While it is possible to set and query individual valves, the software does not broadcast the analogue value of either positional ports or the end stop sensors.

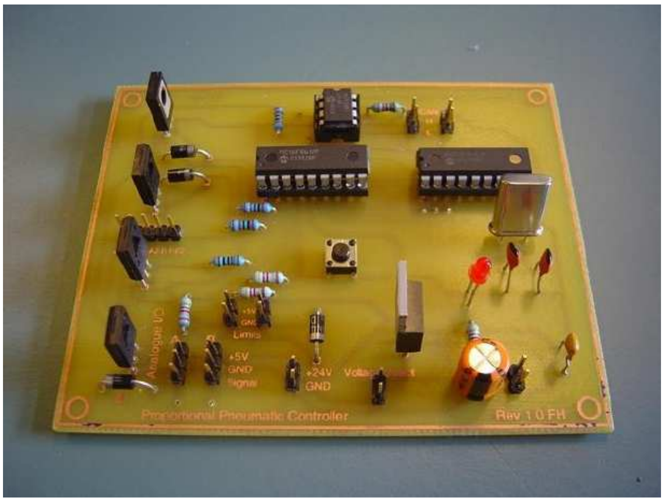


Figure 3.3: Pneumatic proportional control module.

3.2.4 Pneumatic Two-Way Valve Controller

The pneumatic two-way valve controller is designed to allow the control of up to four pneumatic valves. This module differs from the proportional controller by only having four end limit switches, and thus can only operate the attached actuators in fully open or fully closed mode. The generic prototype circuit board for the pneumatic two-way valve controller is shown in Figure 3.4 and the hardware features of this module are as follows:

- Four digital I/O ports for home or end stop sensors;
- One analogue input for system pressure if required.

The control and functionality of the two-way valve controller is the same as that of the proportional controller at this stage. The individual valve controls can be set and queried by the CAN bus, but none of the digital I/O ports or the analogue input can controlled or queried by external sources.

Figure 3.4: Pneumatic two-way valve controller.

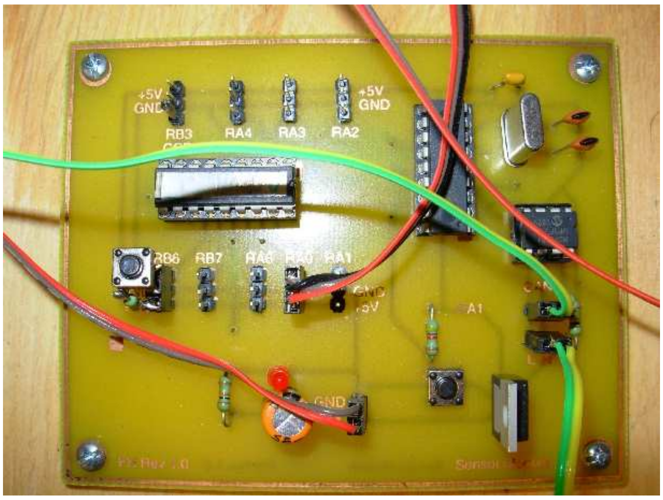


Figure 3.5: Sensor module.

3.2.5 Sensor Module

The sensor module is designed to allow the input or output of up to 9 signals. The ports can be configured individually as required, giving versatility for any situation. The generic prototype circuit board for the sensor module is shown in Figure 3.5.

Due to the wide range of I/O configurations capable of the sensor module, Francois Hoffman has only implemented a single analogue port which can be queried by the CAN bus. A second digital port has also been implemented in software to send a command to the DC motor module when it is triggered, but cannot be queried by a

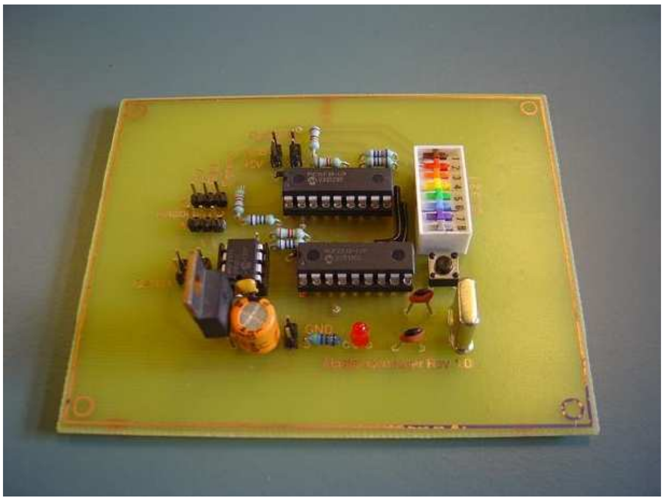


Figure 3.6: Master module.

status request message from the CAN bus.

3.2.6 Master Module

The master module is designed to act as a local monitor of the system to ensure that the nodes are operating properly in the current environment. The master module features five DIP switches that can be used to select modes of operation or other settings. Two analogue and two digital inputs are available to measure environment settings or provide emergency interrupts. The prototype circuit board for the master module is shown in Figure 3.6.

3.2.7 Prototype Testing and Simulation

To aid in the display, simulation and testing of the distributed modules, Francois Hoffman attached them to a sheet of plywood. The individual modules were then connected to a variety of motors, sensors and LEDs to simulate outputs. A photo of the setup is shown in Figure 3.7. The only external connections required are the CAN bus and a power supply. Power is distributed through the centre PCB of the testbed, and can accommodate up to three different voltages.

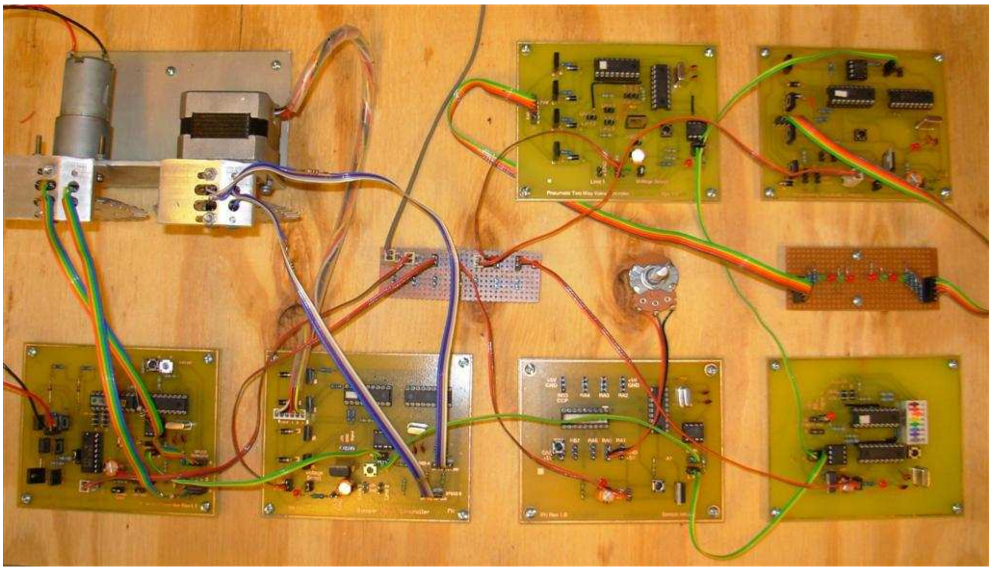


Figure 3.7: Prototype testbed for the distributed control modules.

When $5V$ power was connected to the modules as specified in the report and marked on the modules (Hoffman 2005, p. 22); the CAN bus operation was very unreliable and often reported errors without an apparent cause. Testing the V_{CC} pin of the PIC16F88 processor and the MPC2551 CAN transceiver found that the input voltage was only $3.4V$ rather than the required $5.0V$. This voltage drop was caused by the power regulator on each of the modules. The LM340 is not able to supply a regulated $5.0V$ output when its input voltage is only $5.0V$. The data sheet for the LM340 specifies that the minimum input voltage required to maintain regulation is $7.5V$ (National Semiconductor 2003).

For the purpose of this project, the sensor, master, proportional and two-way pneumatic modules were run on a voltage source of $8.0V - 10.0V$ to minimise heat. The two motor control modules were attached to a minimum $15.0V$ supply to ensure that the motors were running at sufficient speed for testing.

3.2.8 Effect on Project Objectives

The lack of software capabilities in the distributed controllers was the key factor that required the change of specifications for this project. The original objectives for this

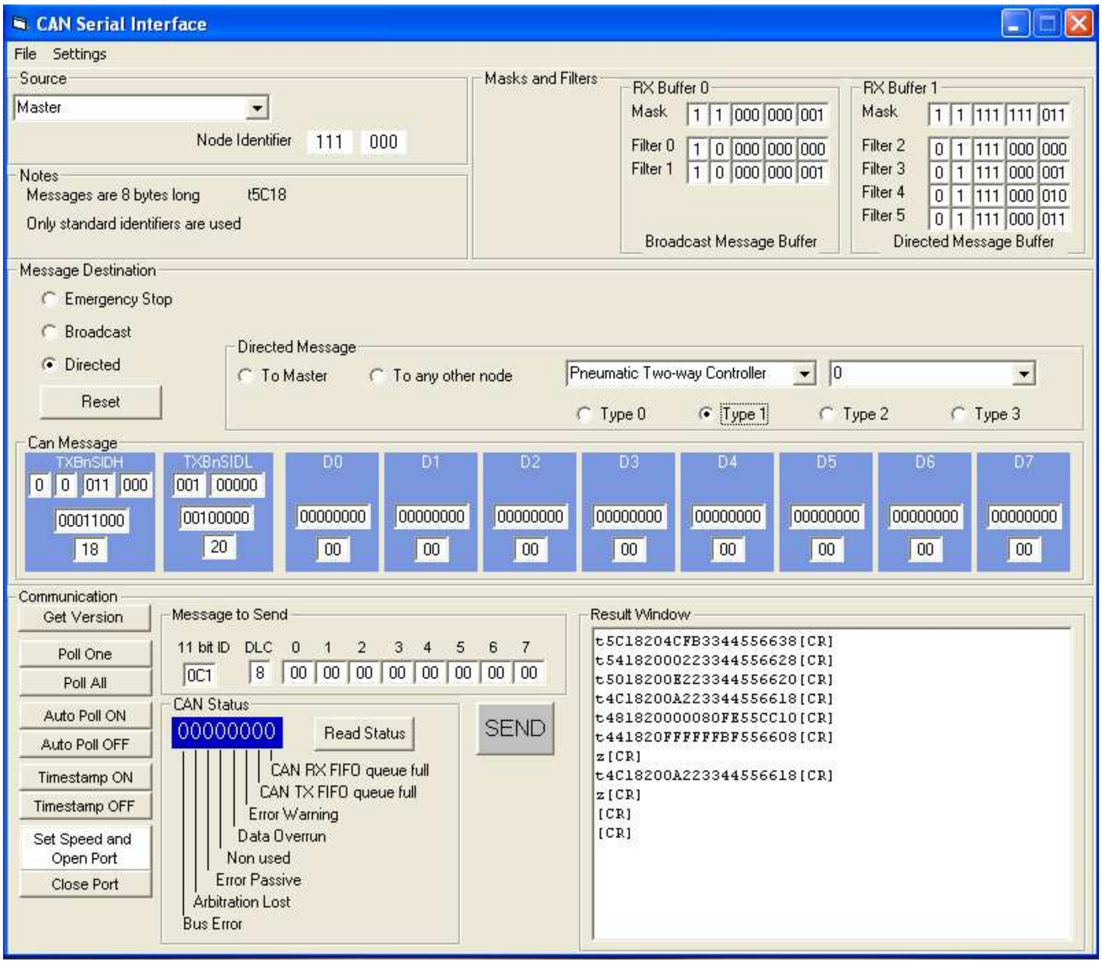


Figure 3.8: Prototype communication interface developed by Francois Hoffman (2005).

project were to develop positional control capabilities for an entire robot structure. However, that depended on each of the motor and pneumatic modules being able to achieve positional control over the individual joints and linkages that they would be attached to. The level of kinematic control and automation originally planned for the project required a large amount of feedback from the modules which has not been implemented.

This lack of complete software has restricted the level of control and therefor testing and demonstration attainable by this project. The current project objectives are centered around developing a generic communication interface that can be expanded upon at a later stage to achieve kinematic control. The primary objective of this project is to develop a method of talking to the robot with a simpler interface than the program developed by Francois Hoffman in Figure 3.8

3.3 Summary

The overall hardware features present in the distributed control modules is very diverse. The hardware should be able to manipulate the various types of robot structures capable of the connector and rod components. The software currently present in the modules is very basic and poorly documented however and it is expected that a large amount of work will be required to make the assembly code as modular and generic as the hardware itself. This has caused the project objectives to be revised and refocus on providing a service that can be used by future developments.

Chapter 4

CAN Protocol

4.1 Chapter Overview

This chapter analyses the current CAN bus specifications and capabilities provided by Francois Hoffman and proposes a standard CAN data frame for each node as a starting point for future implementations.

4.2 Current CAN Implementation

4.2.1 CAN Packet Structure

The general structure of a CAN message is shown in Figure 4.1. This packet example shows a message directed at a proportional pneumatic node and how the individual ports are mapped into the data field.

The 11-bit identifier is partitioned into the following fields as specified by Francois Hoffman (2005, p. 67):

Bit 10	9	8	7	6	5	4	3	2	1	Bit 0
BRD	DTM	NTN2	NTN1	NTN0	NIN2	NIN1	NIN0	MT2	MT1	MT0

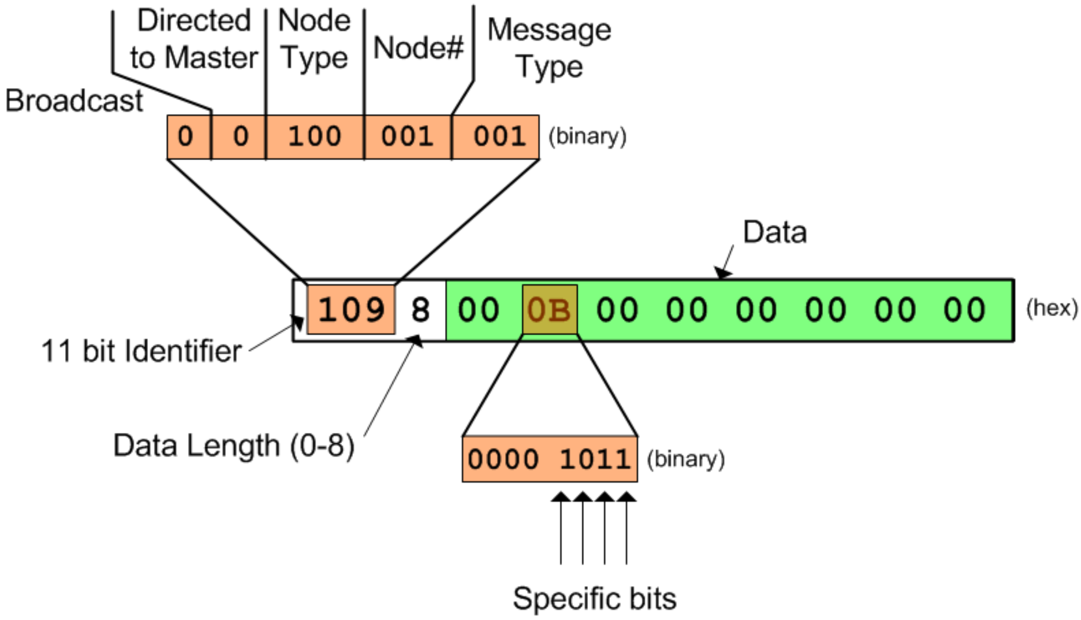


Figure 4.1: Breakdown of a Proportional Pneumatic CAN packet message.

BRD - Broadcast bit.

This bit is set to signal a broadcast message. When BRD is set, DTM must be cleared.

DTM - Directed to Master

This bit signals that the message is directed to the master node. When DTM is set, the NTN[2:0] and NIN[2:0] bits specify the source of the message rather than the destination.

NTN[2:0] - Node Type Number

The NTN bits specify the destination node type number. The Table 4.1 shows the bit and node configurations for this system:

NIN[2:0] - Node ID Number

The node ID number is the numerical index of the node. Each node of a specific type has a unique NIN otherwise the message will be accepted by multiple nodes.

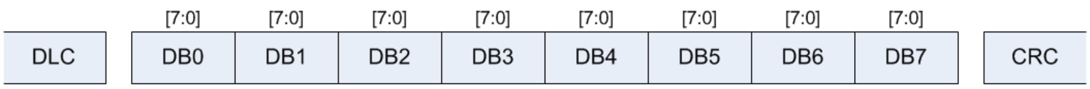
MT[2:0] - Message Type.

Application specific message type depending on the programming of the individual node. All nodes recognise a message type of zero to be a status request.

NTN2	NTN1	NTN0	Node Type
0	0	0	*Not used*
0	0	1	DC Motor
0	1	0	Stepper Motor
0	1	1	Pneumatic Two-Way
1	0	0	Pneumatic Proportional
1	0	1	Sensor
1	1	0	*Not used*
1	1	1	Master

Table 4.1: Allocation of Node Type Numbers to nodes.

The current layout of the data field as specified by Francois Hoffman is as follows:



DB0 - Data byte 0

The high nibble of this data byte is occupied by extra message type bits as specified by Francois Hoffman (2005, p. 67).

DB[1:7] - Data bytes 1 to 7

These bytes are node specific and are further explained in the following sections.

4.3 Proposed Data Frames

4.3.1 Common Elements

The design of the data bytes should be to ensure that similar types of information are in the same data bytes for all of the nodes. For example, any node that operates in a specific state or mode should put this information in the low nibble of data byte 0 (`DB0[3:0]`). Where possible, node specific values have been placed in data bytes 1 to

3, Digital I/O ports have been placed in data byte 4, and generic analogue I/O has been placed after the digital I/O (`DB[5:7]`).

4.3.2 DC Motor Control Module Data Frame

[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]
Mode	Speed	Direction	Position	Digital IO	Analoge1	Analogue2	---
DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7

DB0 - Data byte 0, low nibble:

Current state or mode.

DB1 - Data byte 1.

SPEED value for the motor.

DB2 - Data byte 2

DIRECTION for the motor.

DB3 - Data byte 3

POSITION for the motor to travel to or the current position of the motor.

DB4 - Data byte 4

Digital I/O configured as home or end stop sensors.

DB5 - Data byte 5

Analogue port 1 value if not configured for speed or position.

DB6 - Data byte 6

Analogue port 2 value if not configured for speed or position.

4.3.3 Stepper Motor Control Module Data Frame

[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]
Mode	Speed	Direction	Position	Digital IO	Analoge1	Analogue2	---
DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7

The stepper motor should have the same data byte interface to simplify programming and communication between nodes.

4.3.4 Pneumatic Proportional Control Module Data Frame

[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]
Mode	Valves	Position1	Position2	Digital IO	---	---	---
DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7

DB0 - Data byte 0, low nibble:

Current state or mode.

DB1 - Data byte 1, low nibble:

Pneumatic VALVE settings for the node.

DB2 - Data byte 2

POSITION value from analogue input 1.

DB3 - Data byte 3

POSITION value from analogue input 2.

DB4 - Data byte 4

Digital I/O configured as home or end stop sensors.

4.3.5 Pneumatic Two-Way Control Module Data Frame

[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]
Mode	Valves	---	---	Digital IO	Analogue1	---	---
DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7

DB0 - Data byte 0, low nibble:

Current state or mode.

DB1 - Data byte 1, low nibble:

Pneumatic VALVE settings for the node.

DB4 - Data byte 4

Digital I/O configured as home or end stop sensors.

DB5 - Data byte 5

Analogue input when configured as pressure sensor or other analogue input.

4.3.6 Sensor Module Data Frame

[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]
Mode	Digital IO	Analogue1	Analogue2	Analogue3	Analogue4	Analogue5	Analogue6
DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7

DB0 - Data byte 0, low nibble:

Current state or mode.

DB1 - Data byte 1.

Digital I/O ports as configured by developer.

DB[2:7] - Data bytes 2 - 7

Analogue ports as configured by developer.

4.3.7 Master Module Data Frame

[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]	[7:0]
Mode	DIP	Analogue1	Analogue2	Digital IO	---	---	---
DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7

DB0 - Data byte 0, low nibble:

Current state or mode.

DB1 - Data byte 1.

DIP switch settings as configured on the module.

DB2 - Data byte 2

Analogue port 1

DB3 - Data byte 3

Analogue port 2

DB4 - Data byte 4

Digital I/O configured as required.

4.4 Summary

This chapter has outlined the basic CAN protocol that has been developed by Francois Hoffman. While the design of the identifier does not match the general idea of the CAN network (message rather than node based identifiers), Francois Hoffman has proven the design works (Hoffman 2005, p. 69).

The data frames specified are not final designs that must be used in every CAN module developed for the robot, but rather general recommendations to help the inter-node communication. If the general design of the packets are followed, the embedded code in each of the nodes can be made much more modular.

Chapter 5

Project Methodology

5.1 Chapter Overview

This chapter examines the various approaches that may be used to develop each layer of the interface and the implementation options available to the entire project.

5.2 General Requirements

The key requirements for this Modular Robot communication interface are as follows:

Speed: The interface must be able to operate to some degree as a real time system.

Simplicity of use: The interface must reduce the complexity of the CAN bus operation and communication so that the potential of the Modular Robot project can be fully realised.

Configurable: The programming interface must be able to reflect the structure of the robot it's controlling. A simple method of configuring the modules connected to the CAN bus is required. A method of exporting and importing configuration information is required also required.

Expandable: New features and changes in the Modular Robot CAN bus specifications must be easily replicated in the code. This requires a simple and modular code structure with appropriate documentation.

5.3 Implementation Considerations

5.3.1 Target Platform

The key considerations when determining the target platform for this project were usability, availability and hardware support. The available development platforms were one of many Linux distributions and Windows versions. Taking into account that the majority of the work on the Modular Robot will not be done by software engineering majors, the selected platform was Windows XP. XP has the benefit of widespread use and compatibility with other engineering software.

5.3.2 Programming Language

The original specifications for this project (Appendix A.1) were to develop a very generic and expandable program that could control the movements of a robot. The implementation language to achieve this also needs to be modern and advanced with easy interfacing to the communication hardware. The three main language requirements considered for this project were Object-Oriented, simple Graphical User Interface (GUI) development, and an advanced Integrated Development Environment (IDE) to aid in documentation and project management.

Given the language requirements above, there were a small number of development languages available. The relative merits of each are as follows:

C++: The C++ language is a very efficient and stable O-O language with a number of compilers available for different operating systems. Hardware interfacing is dependent on the operating system which can make platform changes difficult in the future. Developing GUIs can be very tedious as a lot of code is required to achieve full user interface functionality. No IDEs are readily available, however the `MAKE` utility provided under Linux and Cygwin can be used to simplify source code management to some degree.

Java: The java language is a platform independent O-O language which runs in a virtual machine environment above the operating system. Low level hardware interfacing can be difficult due to the virtual machine environment. The development of GUIs are simplified by the extensive libraries provided. There is a free IDE available from Netbeans (Netbeans 2006) which would help in documentation and project structure.

.NET: The .NET framework is an multi-language capable development and execution environment which integrates into the Microsoft Windows operating system.(Microsoft Developer Network 2006). Hardware interfacing and communication is guaranteed. GUI development is very quick and easy due to the drag-and-drop style of the Visual Studio IDE. While the .NET framework is free and can be downloaded from www.microsoft.com, the Visual Studio IDE can be very expensive for a once-off project development.

Although there are some purchasing costs involved with the Visual Studio.NET IDE, this option was chosen on the merits of flexibility, functionality and future expansion for later projects. The .NET environment offers a number of languages which can be used interchangeably throughout a single project. The .NET language chosen for this project is C# (C-sharp) as it closely resembles a combination of C++ and Java. It is expected that the .NET environment will be expanded to provide compatibility with other operating systems (Deitel et al. 2003) which will give a wide range of platform and language choices to future developers of the Modular Robot.

5.3.3 PC — CAN Hardware Interface

There were two types of CAN bus adapters available for this project; the Lawicel CAN232, and the Lawicel CANUSB. The original Modular Robot CAN bus was developed using the CAN232 adapter which connected to the PCs COM port. Due to the lack of serial ports on new laptops, the CANUSB adapter was chosen as it has the

Figure 5.1: The Lawicel CAN232 adapter (left) and the Lawicel CANUSB adapter (right). Source: www.can232.com

benefit of being able to communicate with a virtual COM port interface for backward compatibility with older programs.

5.4 CAN Interface Layer

5.4.1 Purpose

The purpose of the CAN interface layer is to provide a packet buffering and transmission service to a higher level program. This layer handles the CANUSB communication and status checking while the CAN messages are being transmitted.

The general structure of the interface layer is shown in Figure 5.2. While this diagram shows the use of the `COM` port for data transfer, the structure of the interface using the USB DLL is very similar and the differences have been explained in Section 5.4.3. This layer was originally designed to be only one type of communication with the robot and so the programming interface contains only the very basic commands. Using a layered approach it is possible to simply swap this CAN interface with another, such as:

- A simulation interface that can be used for debugging, demonstration or other development purposes.

- A wireless interface to the CAN bus, via a technology such as Bluetooth (Fredriksson 1999).

5.4.2 Initial Design: Virtual COM Port Interface

The initial design of the CAN interface layer used the virtual COM port drivers provided by Lawicel. These drivers were downloaded from the CANUSB website[1] and were installed using default settings. This serial type interface was originally chosen over the USB driver interface due to its simpler programming style and also to enable the use of Hoffman's interface program for comparison during debugging.

The design of the class centered around two threads; a transmit thread (TX) and a receive thread (RX). The design of the TX thread was a simple loop which continually polled the TX queue. If a message was waiting to be sent, it would aquire a lock on the COM port and begin the transmission. The RX thread performed a similar action but in reverse; it would continually poll the COM port until data was available, then read the message and add it to the RX queue. The thread design was complicated however by the operation of the Lawicel COM interface itself which would return a 'z' character when a message had been successfully transmitted. This required the TX and RX threads to be synchronous which increased the dependency on each other to avoid deadlock situations. The final complication to the design was the need for occasional status checking of the Lawicel CANUSB device. This required further communication between the threads to ensure that device was operating properly at regular intervals.

The code that was written for the threads worked reliably and would transfer all messages with error detection. The code required a large amount of CPU time however, as the TX and RX threads used `while` loops when acquiring locks and the `Thread.Sleep()` function only to provide delay between polls of the `COM` port. The performance was rather erratic with the bulk of messages being transferred with about a $10 - 200ms$ delay while others were suffered delays of up to 4 seconds as can be seen in Figure 5.3.

[1] Lawicel virtual COM port drivers: http://www.canusb.com/cdm/cdm_lawicel.zip

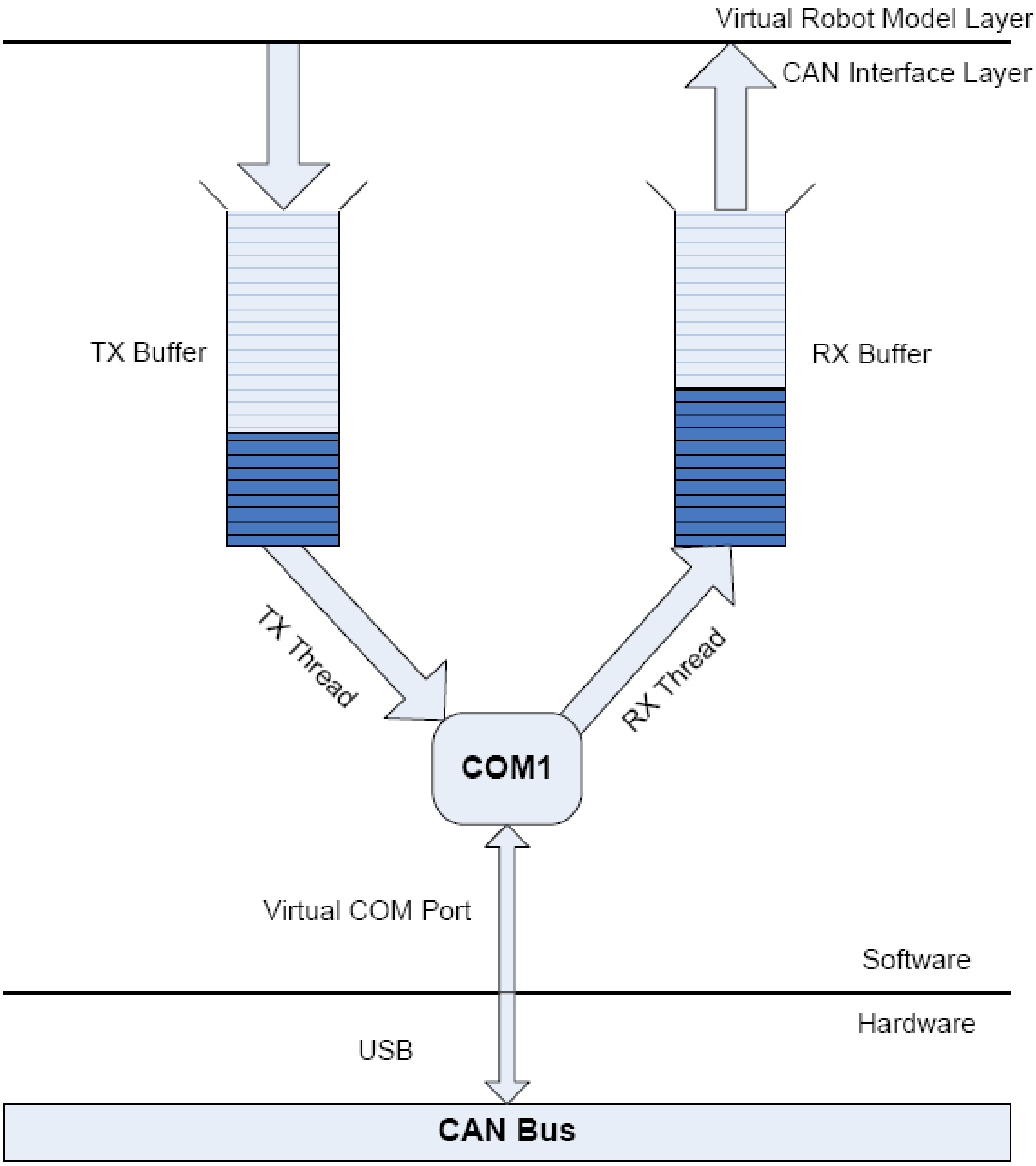


Figure 5.2: CAN Interface layer design using the `COM` port.

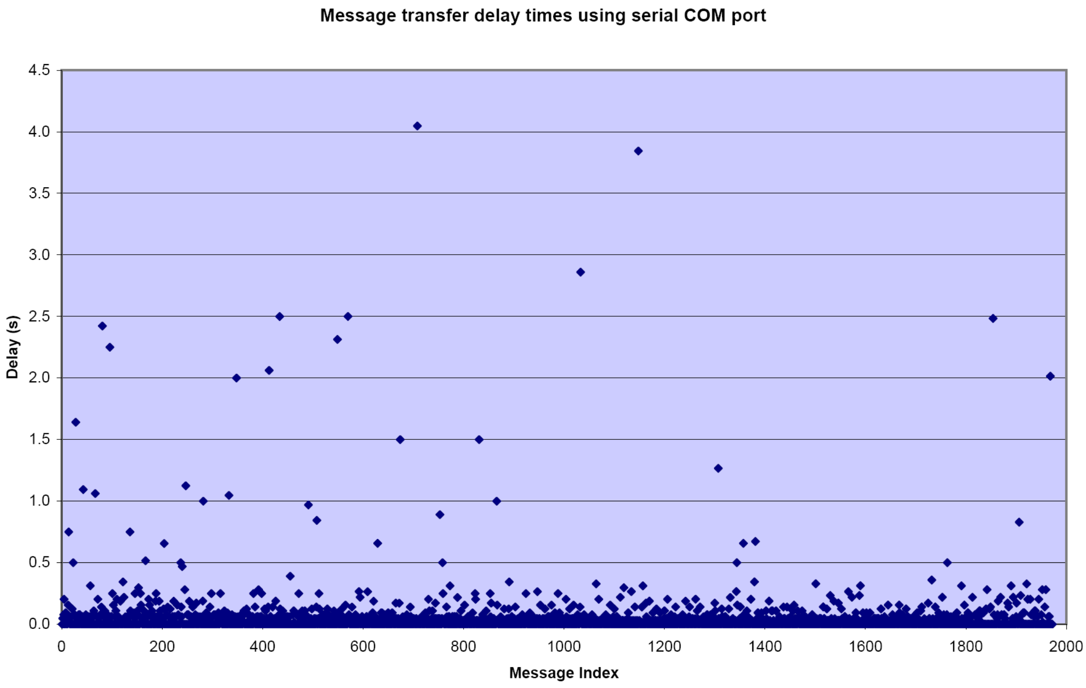


Figure 5.3: Message delay times using the serial `COM` port interface.

5.4.3 Final Design: USB 2.0 Interface

The decision to convert the interface layer to use the USB DLL was primarily to improve the speed and latency statistics of the message transfer. The redesign of the code also helped to reduce the CPU intensive tight `while` loops in the TX and RX threads.

The majority of code had already been written at this time for the virtual robot model layer. The programming interface for the communication layer could not be changed because it would also require rewriting large amount of code in other sections of the program. The original design of the interface already included the TX and RX buffers which effectively made the two layers very independent and so only the underlying code in the CANUSB interface required rewriting.

The TX and RX threads, which were essentially separate programs running tight loops, were replaced with the `System.Threading.Timer` class. The `Timer` class could be configured with a trigger time interval which would call the appropriate function to send or receive data. The same was done for the error checking operations which now meant that the read, write and error checking operations were completely independent of each other. This removed the need for the complicated synchronisation and communication between functions. Each `Timer` callback function simply used the corresponding DLL function call provided by the Lawicel USB driver.

The decision to use an events and callback functions for the RX buffer also improved the latency for the entire system. Rather than have a higher layer continually polling the RX buffer, whenever a message is received by the communication interface it will place the message in the RX buffer and raise an operating system event. This allows the higher layer to continue with its normal activity and only check the RX buffer when a message has definitely arrived.

The final improvement to the CAN interface layer involved adding an emergency or priority message function that would clear the TX buffer and send the emergency message directly to the CAN bus. This has particular importance in a real-time situation when a critical event occurs. Francois Hoffman implemented a emergency message that all nodes would respond to in the definition of the CAN bus (Hoffman 2005, p. 63).

As a result of the redesign, the `CANUSB` class is a very fast and reliable layer as can be seen in Figure 5.4. The scatter chart now shows that instead of messages taking a very random time to be sent and received (Figure 5.3), they are now guaranteed to be completed in less than 50ms 5.4. The scatter plot shows the messages being transferred in specific time intervals, however this is a side effect of the Windows operating system only updating the clock every 15ms. The actual values may be dispersed throughout the time interval. This low latency is very acceptable for real-time applications and the use of callbacks and events has reduced the CPU load by a factor of 5 on the development computer[2].

5.4.4 `CANPacket` Class

The `CANPacket` class is the only common element between all of the layers developed for this project. It was originally developed from scratch to suit the serial COM port interface. The underlying structure was redesigned to wrap around the given CAN

[2] Asus L3 laptop, 2.4Ghz Pentium 4, Windows XP service pack 2.

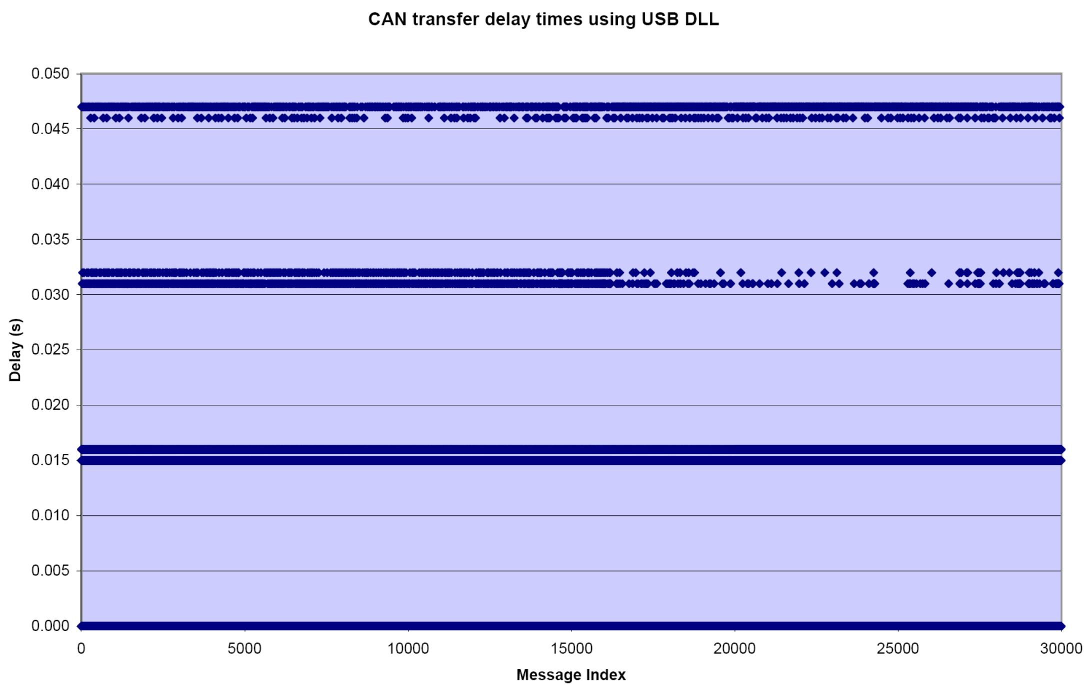


Figure 5.4: Message delay times using the USB DLL interface.

Listing 5.1: The Lawicel CAN message structure

```
public struct CANMsg
{
    public uint id;          // 11/29 bit Identifier
    public uint timestamp;   // Hardware Timestamp (0-9999mS)
    public byte flags;       // Message Flags
    public byte len;         // Number of data bytes 0-8
    public ulong data;       // Data Bytes 0..7
}
```

structure accepted by the Lawicel USB DLL (Listing 5.1). This greatly improved the simplicity of the code as the data storage mechanism is handled by the Lawicel structure and the `CANPacket` class simply handles the interface and data extraction methods. The `CANPacket` class performs a large amount of data checking whenever changes occur to minimise the chance of sending invalid messages to the CAN bus. If the broadcast or directed to master bits are set, the class automatically clears the other. As bytes of data are added to the message, the class automatically updates the data length property to ensure that all significant bytes are sent.

Listing 5.2: Creating nodes from the XML data file. (`MRModel.Open()` method)

```
foreach ( XmlNode node in root.SelectNodes( "node" ) )
{
    try
    {
        Node newNode = new Node( ( XmlElement )( node ) );
        nodes.Add( newNode.Name, newNode );
    }
    catch ( Exception e )
    {
        //MessageBox.Show( "Exception Thrown" );
        MessageBox.Show( e.Message );
        return -3;
    }
}
```

5.5 Virtual Robot Model (VRM) Layer

The VRM layer is essentially the `MRModel` class interface. The layer maintains a list of nodes configures by the robot, and provides access to them using the `Get()` and `Set()` commands. These commands were chosen to appear familiar to the MatLab commands of the same name, improving both the simplicity and usability of the interface for engineers from many disciplines.

An emergency stop function is also provided by the interface which will send the emergency stop command defined in the CAN protocol (Hoffman 2005, p. 67). This has uses in safety critical application where the program is monitoring the position or environment of the nodes and needs interrupt the immediate operations.

The structure of the VRM layer is shown in Figure 5.5. The layer is centred around the `MRModel` class which maintains a list of nodes. The each of the nodes then maintains its own list of ports as configured by the XML data file. The code is very modular, with each class having its own configuration and access methods. For example, when reading in the XML document, the `MRModel`layer will select a `<node>` element from the document and call the `Node` constructor passing in the entire XML element as can be seen in Listing 5.2.

The `Node` class constructor then takes that XML element and uses the attributes defined to configure its name, type, ID etc. Then, for each `<port>` sub-element, it calls the `Port` constructor and passes in the XML string. This separates the configuration code into each of the classes which makes the code very modular and robust. Any changes to the XML specifications only causes changes in the corresponding `MRModel`, `Node` or

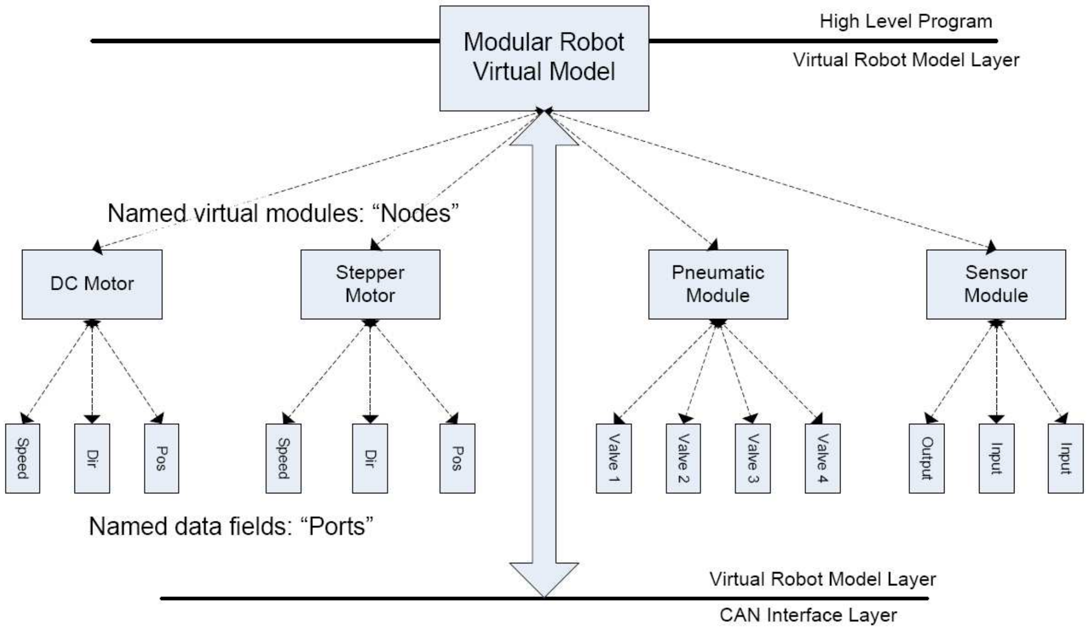


Figure 5.5: Virtual Robot Model (RVM) layer structure.

`Port` class configuration method.

The same theory is applied to the message reception algorithm. When a new status message is received by the `MRModel` callback function, it is immediately passed to any node that has the same type and ID. When the `Node` class receives the message, it immediately passes it to all of its ports who decide whether or not it affects them or not. This follows the principle of object-oriented programming where every object handles its own data and functionality.

5.5.1 XML configuration

The decision to use XML as a data file was primarily to reduce the amount of coding required for the configuration of the robot. The original specifications for this project required a description of more advanced robot kinematics in the data file and so XML was chosen to simplify the data input code. As XML is a integral part of the .NET framework, there are large amounts of documentation and support available for XML input and manipulation in C#. XML has the benefit of being both human and machine readable as it is a text based language. New elements can be added or removed without affecting the operation of the `MRModel` configuration method.

The main component in the `<port>` element is the `filter` attribute. This specifies where in the CAN status reply message to extract the port value from and also where to add the value when sending a message. The `filter` string consists of three parts: Byte Number, Start Bit and Length of Field. These three properties are formed into a `B:S:L` string which the `BitFilter` class can interpret. There are no limits to how many ports can be defined for a node, and the `filter` values can overlap or be the same as other ports as long as the port name is unique.

5.6 Summary

The program developed by this project achieves simplicity of use objectives from Section 5.2 by removing the complicated CAN message structure from the user and offering simple named ports and nodes instead. The performance objectives were met after the `CANUSB` layer was re-designed using the USB DLL driver interface. The configurability and expandability criteria have been achieved through the use of standard XML documents as a data file and configuration tool.

Chapter 6

Performance Analysis and Testing

6.1 Chapter Overview

This chapter evaluates the performance of the interface program as a real-time system and identifies the weak spots in the code by subjecting the individual classes and layers to various user inputs.

6.2 Testing Procedures

The testing procedures undertaken here are unit and integration testing. They are used to ensure that the individual components themselves are operational and that the entire program operates as a functions system. While there are far too many possible data inputs and test conditions to be completely displayed in this report, the main component and operations are included in the following sections. The complete white-box testing of the program modules has been undertaken during the code development to ensure that the system communicates properly with the distributed modules.

Unit testing has been performed on the more independent classes while integration

Listing 6.1: `HexConverter` class declaration.

```
public class HexConverter
{
    // Converts a HEX based character (0:F) to its integer value (0x00:0x0F)
    public static UInt16 ConvertToUInt16( char hexVal );

    // Inverts a given string of HEX values.  Inverts full byte values only
    public static String InvertHexString( String str );
}
```

testing has been performed on the entire system. The tests performed are as follows:

Unit Test :

- `HexConverter`
- `CANPacket`
- `BitFilter`

Integration Test :

- `MRModel` (Covering the `Node` and `Port` classes)

6.3 Unit Testing

6.3.1 `HexConverter` Unit Test

Requirements

The `HexConverter` class is a simple tool containing two static methods which manipulate or convert a string of hex characters. The class declaration can be seen in Listing 6.1. The purpose of the `ConvertToUInt16( char )` function is to help decode a string formatted `CANPacket` into a numerical format. The purpose of the `InvertHexString( string )` function is to convert the text representation of a `CANPacket` data field into the textual version of the Lawicel CAN packet data field and vice versa.

Listing 6.2: HexConverter test driver.

```
// ConvertToUInt16() function test
Console.WriteLine(
        "Char = '0'  Num = " +
        HexConverter.ConvertToUInt16( '0' ).ToString() );
Console.WriteLine(
        "Char = '9'  Num = " +
        HexConverter.ConvertToUInt16( '9' ) );
Console.WriteLine(
        "Char = 'A'  Num = " +
        HexConverter.ConvertToUInt16( 'A' ) );
Console.WriteLine(
        "Char = 'F'  Num = " +
        HexConverter.ConvertToUInt16( 'F' ) );
Console.WriteLine(
        "Char = 'f'  Num = " +
        HexConverter.ConvertToUInt16( 'f' ) );
Console.WriteLine(
        "Char = 'G'  Num = " +
        HexConverter.ConvertToUInt16( 'G' ) );  // Invalid
Console.WriteLine(
        "Char = '#'  Num = " +
        HexConverter.ConvertToUInt16( '#' ) );  // Invalid

// InvertHexString() function test
try
{
    Console.WriteLine(
        "Input: '1234567890ABCDEF'  Output: '" +
        HexConverter.InvertHexString( "1234567890ABCDEF" ) + "'" );
    Console.WriteLine(
        "Input: 'qwerty'  Output: '" +
        HexConverter.InvertHexString( "qwerty" ) + "'" ); // Valid
    Console.WriteLine(
        "Input: '123'  Output: '" +
        HexConverter.InvertHexString( "123" ) + "'" );  // Invalid
}
catch ( Exception e )
{
    Console.WriteLine( e.Message );
}
```

Test

To test the various outputs produced by the `ConvertToUInt16( char )` and `InvertHexString( string )` functions, a short test driver was used which called the functions using various characters and input strings. This test driver can be seen in Listing 6.2.

Results

The results generated by the test driver are as follows:

```
Char = '0'  Num = 0
Char = '9'  Num = 9
Char = 'A'  Num = 10
Char = 'F'  Num = 15
```

```
Char = 'f'  Num = 15
Char = 'G'  Num = 0
Char = '#'  Num = 0

Input: '1234567890'  Output: 'EFCDAB9078563412'
Input: 'qwerty'  Output: 'tyerqw'
Invalid HEX string: Odd number of characters.
```

This shows that the class is robust under a variety of correct and incorrect inputs. The `InvertHexString()` method does raise an exception on invalid data which must be handled by the calling program. It can also be seen that the `InvertHexString()` method does not check that the characters are proper hex values, as this does not impact the algorithm used by the function.

6.3.2 `CANPacket` Unit Test

Requirements

The `CANPacket` class is handled by all layers in this system and is required to present the information contained within itself in a variety of ways depending on its current context. The class header is shown in Listing 6.3 and because the number of different possible input values is quite large, only the basic operations and tests have been conducted. The main purpose for this class is to be able to extract key values from the identifier of the message, and provide access to individual bytes of the data field.

Test

The test driver shown in Listing 6.4 was used to show the various inputs and outputs provided by the class. While only a limited number of tests have been performed here, the class has been extensively tested during the integration test of the system and the general debugging process.

Listing 6.3: CANPacket class declaration.

```
public class CANPacket
{
    // Private data fields
    private LAWICEL.CANMsg innerMsg = new LAWICEL.CANMsg();

    // Class constructors
    public CANPacket( );
    public CANPacket( CANPacket original );
    public CANPacket( LAWICEL.CANMsg original );

    // Property accessors
    public bool Broadcast{ get; set; }
    public bool DirectedToMaster{ get; set; }
    public int NodeTypeNumber{ get; set; }
    public int NodeIDNumber{ get; set; }
    public int MessageType{ get; set; }
    public bool RTR{ get; set; }
    public int DataLength{ get; set; }
    public int this[ int index ]{ get; set; }
    public ulong DataLong{ get; set; }

    // Public functions
    public string CreateFromString( string buffer );
    public string ToHexString( );
    public override string ToString( );

    // Internal (protected) functions
    internal LAWICEL.CANMsg LawicelMsg( );
}
```

Listing 6.4: CANPacket test driver.

```
CANPacket msg = new CANPacket( );
msg.Broadcast = false;
msg.DirectedToMaster = false;
msg.NodeTypeNumber = 1; // DC motor
msg.NodeIDNumber = 0;
msg.MessageType = 0;     // Status request
msg.DataLength = 2;
msg[ 0 ] = 0x00;
msg[ 1 ] = 0x11;
msg[ 2 ] = 0x22;         // Automatically increases data length

Console.WriteLine( "Message: " + msg.ToString( ) );
Console.WriteLine( "Long Data (hex): " + msg.DataLong.ToString("X6") );
Console.WriteLine( "Data[1] (hex): " + msg[ 1 ].ToString( "X2" ) );
```

Results

The output of the `CANPacket` test driver is shown below. This is the hexadecimal representation of the message, which has been adopted as the standard representation of a `CANPacket` throughout the system. The second output also shows the difference between the internal representation of the data to the standard. Whereas the standard packet display presents the data bytes in numerical order of index from 0 to 7; the `DataLong` property accessor returns a single integer value containing the higher index data bytes in the most significant digits of the number.

```
Message: 0403001122
Long Data (hex): 221100
Data[1] (hex): 11
```

6.3.3 `BitFilter` Unit Test

Requirements

The `BitFilter` class acts as a configurable tool to extract information from the data section of a `CANPacket`. As such, it must be able to extract or insert a value of any size from any position in a byte without corrupting other information in the packet. The `BitFilter` class header is shown in Listing 6.5

Test

The test driver shown in Listing 6.6 shows a DC Motor message and how a variety of `BitFilters` may be used to extract information from various bit combinations.

Listing 6.5: BitFilter class declaration.

```
public class BitFilter
{
    // Private data fields
    private int byteNumber;      // Byte number ( 0 - 7 )
    private int bitStart;        // Start bit number ( 0 - 7 )
    private int numBits;         // Number of bits to include ( 8-bitStart )

    // Constructors
    public BitFilter( int byteNumber, int bitStart, int numBits );
    public BitFilter( string parseString );

    // Property accessors
    public int ByteNumber{ get; }
    public int StartBitNumber{ get; }
    public int FieldLength{ get; }

    // Public methods
    // Extracts a value from a CANPacket
    public int ParseValue( CANPacket msg );
    // Adds a value to a CANPacket
    public void AddValue( CANPacket msg, int value );
}
```

Listing 6.6: BitFilter class test driver.

```
CANPacket msg = new CANPacket( );
msg.Broadcast = false;
msg.DirectedToMaster = false;
msg.NodeTypeNumber = Node.DC_MOTOR;
msg.NodeIDNumber = 0;
msg.MessageType = CANPacket.MESSAGE_TYPE_1;
msg.DataLength = 8;

msg.DataLong = 0x7766554433221100;

Console.WriteLine( );
Console.WriteLine( "Original Message: " + msg.ToString( ) );


BitFilter filter1 = new BitFilter( "B1:S0:L4" );
BitFilter filter2 = new BitFilter( "B2:S1:L1" );
BitFilter filter3 = new BitFilter( "B3:S0:L8" );
BitFilter filter4 = new BitFilter( "B5:S2:L3" );

Console.WriteLine(
    "Filter 1 - B1:S0:L4 -> 0x" +
    filter1.ParseValue( msg ).ToString( "X2" ) );
Console.WriteLine(
    "Filter 2 - B2:S1:L1 -> 0x" +
    filter2.ParseValue( msg ).ToString( "X2" ) );
Console.WriteLine(
    "Filter 3 - B3:S0:L8 -> 0x" +
    filter3.ParseValue( msg ).ToString( "X2" ) );
Console.WriteLine(
    "Filter 4 - B5:S2:L3 -> 0x" +
    filter4.ParseValue( msg ).ToString( "X2" ) );

Console.WriteLine( );
```

Results

The output of the test driver is as follows. Note that the outputs have been converted to Hex for comparison with the original message data.

```
Original Message: 041 8 0011223344556677
Filter 1 - B1:S0:L4 -> 0x01     % Byte 1: ---- 0001
Filter 2 - B2:S1:L1 -> 0x01     % Byte 2: ---- --1-
Filter 3 - B3:S0:L8 -> 0x33     % Byte 3: 0011 0011
Filter 4 - B5:S2:L3 -> 0x05     % Byte 4: ---1 01--
```

This shows that the `BitFilter` class can handle a variety of valid inputs. The `BitFilter` constructor will throw an exception if the `B:S:L` string is invalid.

6.4 Integration Testing

6.4.1 `MRModel` Integration Test

Appendix D shows the configuration files, sample program and output listings used as a test case for the `MRModel` class. The output in Appendix D.5 shows the log file generated by the `CANUSB` layer when the test program is run.

Figure 5.4 shows a scatter plot of the time taken to transfer 30,000 individual CAN packets. The fact that every single message was transfered in under $50ms$ shows the outstanding speed and latency times capable of the developed system. The test code used to produce the 30,000 messages is shown in Listing 6.7. Essentially, the code uses the two pneumatic modules to count the test LEDs from 0 to 256^2. The test was terminated after about 15 minutes however due to the proportional pneumatic module turning itself off for an unknown reason. The 30,000 messages transfered during that time is used as the data for this test.

Listing 6.7: `MRModel` class test driver.

```
for ( int j = 0; j <= 0xFF; j++ )
{
    for ( int i = 0; i <= 0xFF; i++ )
    {
        robot.Set( "PP0", "valveall", i );
        robot.Set( "P2W0", "valveall", j );

        if ( robot.Get( "PP0", "valveall" ) != i )
        {
            Console.WriteLine( "Error PP0 Value: " + i );
            Thread.Sleep( 1000 );
        }
        if ( robot.Get( "P2W0", "valveall" ) != j )
        {
            Console.WriteLine( "Error P2W0 Value: " + i );
            Thread.Sleep( 1000 );
        }
    }
}
```

6.5 Chapter Overview

The preceding tests have shown that the individual and cumulative classes are valid and working under a variety of input conditions. Unfortunately not all tests performed could be shown here due to space and time restrictions. Due to the lack of documentation on the stepper motor, sensor and master nodes, they could not be effectively used during the testing process.

Chapter 7

Implementing the Program in Future Projects

7.1 Chapter Overview

This chapter brings together the important information from other chapters of this document and explains what is required to integrate this project in a higher level control programs.

7.2 Hardware Configuration

7.2.1 Lawicel CANUSB Driver Installation

In order to be able to talk to the Lawicel CANUSB device, the USB drivers provided by Lawicel must be installed onto the target computer. The executables are included in `Appendix E` on the CD or can be downloaded from the Lawicel CANUSB web site[1]. The filenames are as follows:

`canusb_d2xx.zip` This is the driver to talk to the USB port of the Lawicel CANUSB

[1] http://www.canusb.com/downloads.htm

device.

`canusbdrv014.EXE` This is the driver required to transfer data to and from the Lawicel CANUSB device and provides a C# DLL to talk directly to the device.

The `canusb_d2xx.zip` archive contains the driver files required by Windows XP when the device is initially connected to the computer. This archive must be extracted to a directory that is searched by the Windows new hardware installation wizard.

Once the Lawicel CANUSB device is recognised by the computer, the second driver can be installed by running the `canusbdrv014.EXE` file. This is installed with all default settings and the result is a new `LAWICEL` subdirectory in the `C:/Program Files` folder. The `LAWICEL` folder contains the `activeX` controls, sample code and diagnostic software for the device.

7.3 Software Configuration

7.3.1 .NET Framework

The code developed for this project was written in C# using the Microsoft .NET framework $V2.0$. To be able to run the code, version 2.0 or greater of the framework must be downloaded and installed from Microsoft.com. The redistributable installer (`dotnetfx.exe`) has been included in the `Appendix E` folder of the CD.

7.3.2 Visual Studio 2005

The Microsoft Visual Studio 2005 Integrated Development Environment (IDE) is highly recommended if any amount of code is to be developed. While it is possible to code, debug and run .NET source code using freely available text editors and the .NET framework SDK, it is well worth the expense to purchase Visual Studio to help manage the source code files, configuration and documentation associated with a large software project.

Listing 7.1: Simple XML document.

```
1 <?xml version="1.0"?>
2 <!-- This is a comment -->
3   <CAN
4     type =          "LawicelCANUSB"
5     firmware =      "0.0.14"
6     baudrate =      "125"
7     readinterval =  "5"
8     writeinterval = "5"
9     readtimeout =   "500"
10  >
11    <!-- Empty robot configuration -->
12  </CAN>
```

7.4 XML Robot Configuration

The XML configuration of the robot model is the key component to the simplicity of the communication interface. Although an advanced knowledge of XML is not required for this project, a basic knowledge of an XML document is helpful for the debugging process if errors occur.

The simplest XML document accepted by the program is shown in Listing 7.1 with the complete XML document listed in Appendix D.2. The key elements of the document are as follows:

Version declaration: Although the version declaration on line 1 is technically optional, it is good practice to include it so that future XML parsers will interpret the document correctly (Deitel et al. 2003, p. 658).

Root node: There can only be one root node in an XML document which in this example is the `<CAN>` element. All nodes in a XML document must be nested correctly with the appropriate closing tag. The name of root node has no effect on the running of this program, however it cannot contain spaces or any special characters or symbols.

Root node attributes: These attributes (`type, firmware, baudrate...`) are specific to the entire configuration and interface to the Modular Robot. Attributes are always interpreted as text and so must be enclosed in either single or double quotes.

`type:` The type of CAN bus adapter attached to the computer. Although it is not used in the program, it does help documentation purposes.

`firmware:` The firmware of the CAN device. This value is checked to ensure that the CAN device has the correct features.

`baudrate:` Baud rate of the CAN bus that the device will connect to.

`readInterval:` Time (ms) to wait between read polls on the CAN device. To improve response times, this value can be set to near zero. The minimum allowable value is $1ms$.

`writeInterval:` Time (ms) to wait after write intervals on the CAN device. To improve response times, this value can be set to near zero. The minimum operational value will depend on the amount of traffic sent to the CAN bus. If buffer errors occur on the CAN device, this value should be increased to ensure that messages are being transferred properly.

`readTimeout:` Time (ms) to wait for a response to a status request from the CAN bus. The minimum operational value will depend on the amount of CAN bus activity but should be set low enough to ensure real-time operation.

7.4.1 Defining Nodes

Once the basic XML document has been created and the global CAN settings have been added, the root `<CAN>` node can be extended by adding XML nodes in a nested pattern. The only first level node type accepted by the interface program at this stage is the `<node>` which although slightly confusing, is reference to a CAN distributed module or "node". The `<node>` node has the following attributes that must be defined in order for the program to configure correctly:

`name:` Unique module name. This name is case sensitive and must be unique within the list of defined CAN nodes.

`type:` CAN node type. Must be one of the following types: `DCMotor, Master, StepperMotor, ProportionalPneumatic, 2WayPneumatic` or `Sensor`.

`number:` Node ID number.

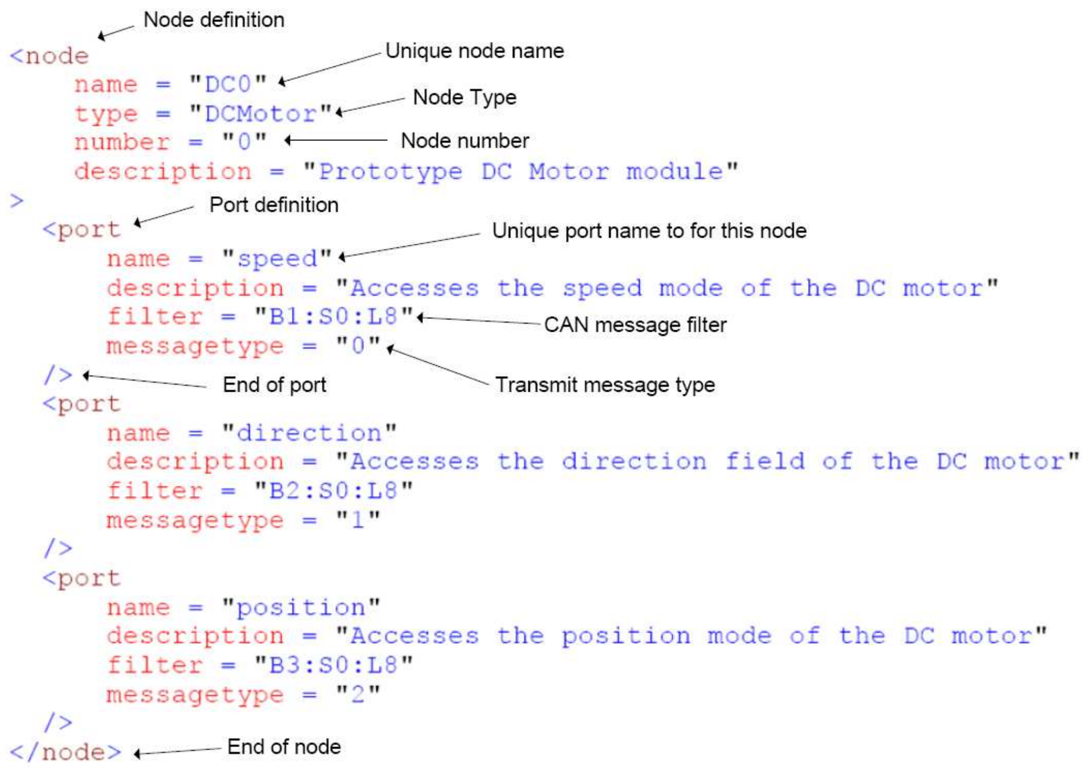


Figure 7.1: Sample XML node and port declaration showing important elements.

`description`: A brief description of the node. This is not currently used however it may be useful if a GUI interface were to be added to the system which could display the descriptions to the user.

An example XML node structure is shown in Figure 7.1.

7.4.2 Defining Ports

Within each of the CAN nodes, any number of ports can be defined which are essentially specific bits that are filtered from the data field of a CAN message. A port is defined in XML with the following attributes:

`name`: Unique port name within the context of the current node. The name is case sensitive and should represent the use of the port.

`description`: A brief description of the port that can be used as documentation or as

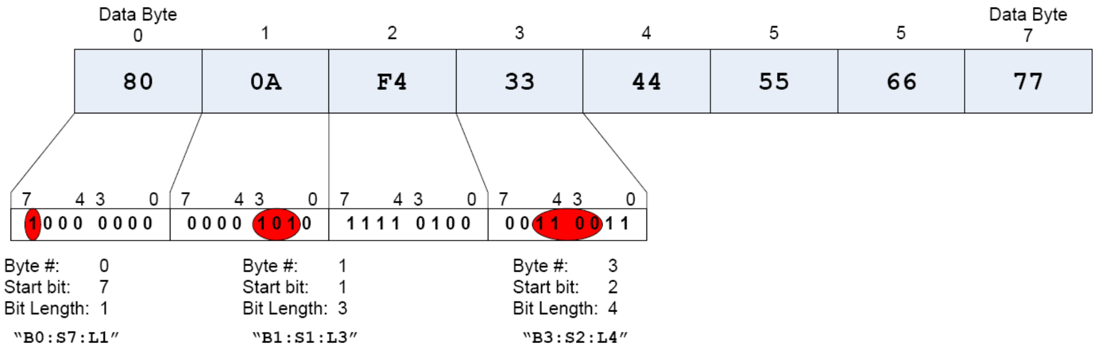


Figure 7.2: Data field showing example `filter` strings.

an aid on a graphical user interface.

`filter`: The CAN packet filter to be applied when sending or receiving data through this port. For an example on how to derive the `filter` string, see Figure 7.2.

`messagetype`: The message type to be used when sending a CAN packet. When a message of specific type is sent from a node, all of the ports that have a matching message type are included in the data field. If the message type is set to zero, the port value is included in all messages sent from the parent node.

An example XML port structure is shown in Figure 7.1.

7.5 Sample Program

The sample program listing shown in Appendix D.3 demonstrates the various ways that the functions provided by the `MRModel` class can be used for robot control. The important methods used in the sample program listing and their functionality are as follows:

`MRModel robot = new MRModel( );` - This method creates a new reference to a `MRModel` class and calls the default constructor. The `robot` variable can be any unique variable name and must be used consistently throughout the program.

`robot.ConfigFileName = "./robotCAN.xml";` - This property accessor tells the program where to find the XML configuration file. The filename given is a relative

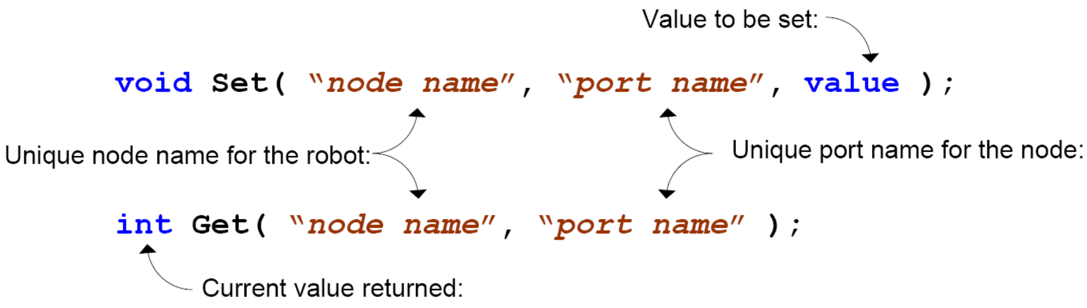


Figure 7.3: `Set( )` and `Get( )` commands.

path to the executable and will generate an error if the specified file cannot be found.

`if ( robot.Open( ) < 0 )` - This statement is used to initialise the `robot` using the XML from the configuration file defined above. It also opens the link to the CANUSB device and will return a number less than zero if the action fails. Once this method has been successfully called, the program is ready for operation.

`robot.Set(` *nodename*, *portname1*, *value1* `);`
...
`robot.Set(` *nodename*, *portname1*, *value1*, ... , *portname8*, *value8* `);` - The `Set` family of methods are used to set one or more values in a particular node and will generate at least one message on the CAN bus each time the function is called. If a particular *nodename* or *portname* cannot be found, it will display an error message and the function will return immediately.

`robot.Get(` *nodename*, *portname* `);` - The `Get` command is used to return the value of a specified port on a specified node. This will generate a status request message on the CAN bus and the value will be returned when the physical node replies. The format of the `Get()` and `Set()` commands are shown in Figure 7.3.

`robot.Close( );` - This method is used to clear the TX and RX buffers and to close the link to the CANUSB device. This method should be the last command given to the `MRModel` class in a program.

7.6 Summary

This chapter has given a overview on every element required to use the developed code in a higher level control program. The complete code listings used in this chapter can be found in Appendix D of this document and Appendix D of the CD.

Chapter 8

Conclusions and Further Work

8.1 Achievement of Project Objectives

The following objectives have been addressed:

Research & documentation of distributed control modules. Chapter 3 performed a dissection of each of the control modules and detailed the current hardware and software capabilities that each module offers. The research found that while the hardware capabilities of the modules are very promising, the assembly code software was originally designed only for testing and demonstration purposes and so cannot provide sufficient testing capabilities for this project to properly develop a positional control system. As a result, the project objectives were shifted to focus on providing a communication interface to the CAN bus for a higher level programming environment.

Design of data packet standard. Chapter 4 proposed a common data format for each of the distributed control modules to simplify communication between nodes. While the design of the data format for the modules is very application specific, a base format for the general order of bytes has been specified as a starting point for future implementations.

Development of communication interface. Chapter 5 outlines the implementation decisions and features of the developed system. The communication interface consists of two distinct layers with minimal dependency on each other. This independence proved invaluable when the message transfer layer was rewritten to change the CANUSB device interface from a serial `COM` port to a USB DLL without affecting any other part of the system.

Performance analysis and testing. Chapter 6 shows some of the testing stages and drivers that were used to gauge the reliability of the system. The code has been designed and written to handle most user inputs however complete stress and integration testing can only be achieved when all of the distributed modules are fully functional on the CAN bus.

Documentation of code and designs. Chapter 7 explains from a user's point of view how to install, use and configure the system to suit the required application. This chapter covers all of the technical information required to integrate the communication interface into a higher level program.

8.2 Further Work

Whilst the revised objectives for this project have been met and the developed software will simplify the interface to the modular robot, the initial objectives were to develop a much more advanced system that would be able to provide positional control and automation in a user-friendly scripting interface. Although the work developed by this project has greatly helped toward the realisation of that goal, the software in each of the distributed modules must now be updated to match the capabilities of the hardware.

This embedded software development would be in the form of a set of generic subroutines and functions that can be linked together and combined with some application specific logic. This would simplify the programming of the PICs and improve the development speed of the entire robot. This set of code modules could include Analogue-To-Digital (ADT) functions, motor control sub-routines, positional pneumatic and motor controls, generic interrupt routines and other basic I/O functions. The most effective development environment for this would be the C language which already offers strong real-time performance as well as the ability to develop very modular and generic sub-routines. This would enable novice users with little assembly language background to develop advanced embedded control programs for the PICs by simply adding calls to sub-routines and thus hiding the complicated device structure from the user.

Once the code in each of the distributed modules can easily provide positional control or feedback over its local environment, the next layer of control software on the PC side can be developed. This would utilise the processing power in each of the modules to be able to provide positional control over the entire robot structure with very little communication required on the CAN bus.

As for further work on this communication interface, some of the finer functionality and error handling could be refined to ensure robustness given all types of input data. A GUI application to simplify the creation and editing of the XML configuration file would further reduce the chance of human error and a run-time GUI showing the current values for important nodes and ports would provide advanced debugging capabilities. The latency of the `Get()` method could be improved by immediately returning the saved value of the port value if it has been recently updated, rather than always waiting on a new status message. A further improvement would be to to raise events whenever to any of the port values change. A higher level program could then attach callback functions to the events

8.3 Conclusion

The USQ Modular Robot development is a continually evolving project which can bring together a wide variety of disciplines such as mechanical, mechatronic, electronic, computer systems, instrumentation and control, and software engineering. This wide range of technical expertise working on a single project requires the use of simple designs and complete documentation of design rational and outcomes. Unfortunately for this project the previous documentation was unclear on the specific functionality of the distributed modules which resulted in a change of scope and objectives for this project.

Given the new objectives for this project, the developed program is highly successful at meeting the simplicity and performance requirements for such a system. The communication layer has outstanding latency times of $15 - 50ms$ and has a simple interface to higher level programs. This means it can be replaced with a simulation layer or wireless transfer layer if required at some time in the future. The virtual robot model layer simplifies the programmer interface by providing a set of simple commands that can be used to set and access data from any module attached to the CAN bus.

The greatest drawback to the system is the dependency on specifications of the CAN packet identifier by Francois Hoffman. Any change to the ordering or range of bits in the identifier will require a change in the `CANPacket` class and also all of the classes in the virtual robot model layer which generates the CAN messages. Any changes to the data field can be configured by the XML document which handles the `BitFilter` for each port. The use of XML has dramatically reduced the code required for the system, and also allows the one configuration file to be expanded later to add kinematic information without affecting the data for this program.

Overall, this project has developed a very useful tool for anyone wanting to demonstrate a practical application of the Modular Robot, and also for future software engineers to follow through on the initial project objectives of developing an advanced kinematic control system.

References

Billerwell, M. (2001), 'Modular robot controller', USQ BEng Thesis.

Bosch GmbH, R. (1991), *CAN Specifications, Version 2.0*, Adobe Acrobat Document Format.
http://www.semiconductors.bosch.de/pdf/can2spec.pdf
current May 2006.

Craig, J. J. (2005), *Introduction to Robotics: Mechanics and Control*, 3rd edn, Pearson Education, Inc., USA.

Deitel, H., Deitel, P., Listerfeild, J., Nieto, T., Yaeger, C. & Zlatkina, M. (2003), *C# for Experienced Programmers*, Deitel Developer Series, Prentice Hall, Upper Saddle River, New Jersey.

Fredriksson, L.-B. (1999), Bluetooth in automotive applications,
www.kvaser.se
current September 2006.

Hoffman, F. (2005), Distributed control system for different applications, Master's thesis, USQ.

Kimmel, P. (2002), *Advanced C# Programming*, McGraw-Hill/Osborne.

KVASER (2005), 'Controller Area Network Overview'.
http://www.kvaser.com
current June 2006.

Lawicel (2003), *CAN232 Manual Version 2.0A*.

http://www.can232.com
accessed 20-April-2006.

Lawicel (2006), *CANUSB Manual Version 1.0B*.
http://www.canUSB.com
accessed 20-April-2006.

Microchip (2002), *PIC16F87/88 Datasheet (DS30487A)*, Adobe Acrobat Document Format.
http://www.microchip.com.

MicroController.com (1999), 'CAN (Controller Area Network): Introduction & fundamentals'.
http://microcontroller.com/EmbeddedSystems.asp?c=27
current September 2006.

Microsoft Developer Network (2006), '.NET framework fundamentals'.
http://msdn.microsoft.com/netframework/programming/fundamentals/default.aspx
accessed 26-September-2006.

National Semiconductor (2003).
http://info.hobbyengineering.com/specs/LM340.pdf
current June 2006.

Netbeans (2006), 'Netbeans IDE'.
http://www.netbeans.org
current September 2006.

Scouller, C. (2002), 'Graphical user interface for modular robot', USQ BEng Thesis.

Teoh, L. P. (2001), 'Robot design kit', USQ BEng Thesis.

Wikipedia (2006*a*), '.net framework — wikipedia, the free encyclopedia'.
http://en.wikipedia.org/w/index.php?title=.NET_Framework&oldid=77275753
accessed 26-September-2006.

Wikipedia (2006*b*), 'Visual basic — wikipedia, the free encyclopedia'.
http://en.wikipedia.org/w/index.php?title=Visual_Basic&oldid=77612738
accessed 26-September-2006.

Appendix A

Project Specification

A.1 Issue A - 27 March 2006

See page 65.

A.2 Issue B - 15 August 2006

See page 65.

University of Southern Queensland
Faculty of Engineering and Surveying

ENG 4111 / 4112 Research Project
PROJECT SPECIFICATION

FOR:	**Kevin STARK** **0050009783**
TOPIC:	Script–Based Control Language for the Modular Robot Development
SUPERVISOR:	Mark Phythian
PROJECT AIM:	This project aims to investigate the kinematics involved with the control of various mechanical robot configurations and use the common elements to develop an abstract scripting language and user interface for the distributed CAN bus controller of the Modular Robot Development.

PROGRAMME: Issue A: 27 March 2006

1. Research basic robot arrangements capable of the Modular Robot Development.
2. Develop mathematical equations to model the kinematics of basic robotic movements and mechanical arrangements.
3. Develop & program a set of mathematical functions which can decompile a complex kinematic equation into a set of related equations for each motor, actuator or servo attached to the robot.
4. Design a minimal set of control, looping, logic, timing and function operators which can be used as a fundamental control language for the robot.
5. Implement the above language definition into a command parser and a CAN-bus controller.
6. Develop a basic development environment for the command parser to allow real-time feedback and debugging info.

As time permits:

7. Improve parser algorithms to improve speed and efficiency of commands by using the main CAN-bus controller and on-board chips to make calculations and broadcast data.
8. Improve the user interface of the script editor to allow for keyword highlighting and auto command completion.

AGREED:

____________________	____________________	_____ /_____ /_______
(Student)	*(Supervisor)*	*(Dated)*

University of Southern Queensland
Faculty of Engineering and Surveying

ENG 4111 / 4112 Research Project
PROJECT SPECIFICATION

FOR:	**Kevin STARK** **0050009783**
TOPIC:	Modular robot communication interface
SUPERVISOR:	Mr. Mark Phythian
PROJECT AIM:	This project aims to document guidelines for the CAN message protocol used by the modular robot and to develop a PC based message handling program to simplify future control of the modular robot.

PROGRAMME: Issue B: 15 August 2006

1. Research the hardware capabilities of the generic CAN modules.
2. Design a data packet standard that can be used as a guideline for future CAN module design.
3. Develop a configurable and expandable communication interface that can link a high level control program to the CAN modules.
4. Analyse and evaluate the performance of #3 using test drivers and various scenarios.
5. Fully document the API provided by #3 and also the data packet standards defined in #2.

As time permits:

6. Develop a graphical user interface tool to configure the robot model.

AGREED:

__________________ __________________ ____ / ____ / ______

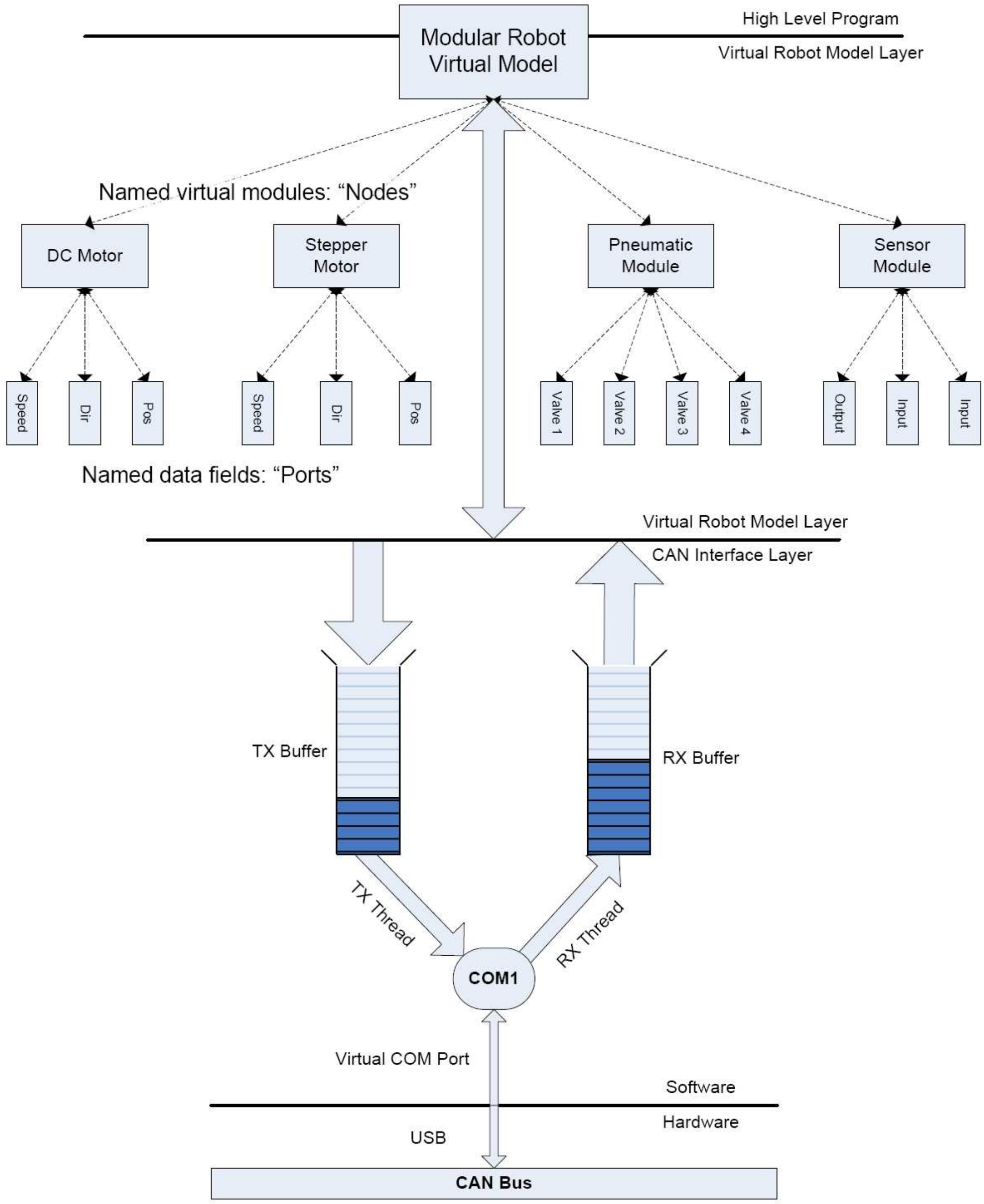
(Student) *(Supervisor)* *(Dated)*

Appendix B

System Diagrams

B.1 System Model

The following diagram is a simplified system structure showing the individual layers.



Appendix C

Source Code Listing

This appendix contains all of the source code files developed for this project. These files can also be found in the `Appendix C` folder on the CD.

C.1 Contents

The files are listed in alphabetical order and are as follows:

- **C.2**: `BitFilter.cs` - This class filters out specific data bits from a CANPacket data frame.
- **C.3**: `CANPacket.cs` - This class models a CAN data packet and includes a number of packet manipulation functions to aid in its use.
- **C.4**: `CANUSB.cs` - This class controls the access to the Lawicel CANUSB device and provides buffering and error checking.
- **C.5**: `HexConverter.cs` - This class converts HEX characters into integers values.
- **C.6**: `Log.cs` - This class provides logging capabilities for the CANUSB class.
- **C.7**: `MRModel.cs` - This class acts as the interface between the CANUSB and a higher level control program.
- **C.8**: `Node.cs` - This class models a CAN node.
- **C.9**: `Port.cs` - This class models a I/O data port on the nodes.

C.2 BitFilter.cs

Listing C.1: BitFilter class source code.

```
using System;
using System.Text;
using System.Collections.Generic;
using System.Windows.Forms;


namespace ModularRobot
{
    public class BitFilter
    {
        // ------------------------------------------------------------------
        // Private data fields

        private int byteNumber;     // Byte number ( 0 - 7 )
        private int bitStart;       // Start bit number ( 0 - 7 )
        private int numBits;        // Number of bits to include ( 8-bitStart )

        // ------------------------------------------------------------------
        // Public data fields / constants

        public const int DEFAULT_BYTENUM = 1;
        public const int DEFAULT_BITSTART = 0;
        public const int DEFAULT_NUMBITS = 8;

        // ##################################################################
        // Constructors

        /// <summary>
        /// BitFilter constructor.
        /// </summary>
        /// <param name="byteNumber">Byte number to select. ( 0 - 7 )</param>
        /// <param name="bitStart">Start bit number. ( 0 - 7 )</param>
        /// <param name="numBits">Number of bits to include.</param>
        public BitFilter( int byteNumber, int bitStart, int numBits )
        {
            Initialise( byteNumber, bitStart, numBits );
        }

        // ------------------------------------------------------------------

        /// <summary>
        /// BitFilter constructor.
        /// </summary>
        /// <param name="parseString">
        /// A valid string to parse from.  'Bx:Sx:Lx'
        /// </param>
        public BitFilter( string parseString )
        {
            ParseString( parseString );
        }

        // ##################################################################
        // Property accessors

        /// <summary>
        /// Gets the Byte index for this filter.
        /// </summary>
        public int ByteNumber
        {
            get { return byteNumber; }
        }

        // ------------------------------------------------------------------

        /// <summary>
        /// Gets the starting bit number.
        /// </summary>
        public int StartBitNumber
        {
            get { return bitStart; }
        }

        // ------------------------------------------------------------------

        /// <summary>
        /// Gets the bit field length.
        /// </summary>
        public int FieldLength
        {
            get { return numBits; }
        }
```

```
        // ####################################################################
        // Private methods

        /// <summary>
        /// Manages the initialisation for the BitFilter.
        /// </summary>
        /// <param name="byteNumber">Byte number to select. ( 0 - 7 )</param>
        /// <param name="bitStart">Start bit number. ( 0 - 7 )</param>
        /// <param name="numBits">Number of bits to include.</param>
        private void Initialise( int byteNumber, int bitStart, int numBits )
        {
            if ( byteNumber >= 0 && byteNumber <= 7 )
                this.byteNumber = byteNumber;
            else
            {
                this.byteNumber = DEFAULT_BYTENUM;
                throw new InvalidBitFilterException(
                    "Invalid Byte Number: " + byteNumber.ToString( ) );
            }
            if ( bitStart >= 0 && bitStart <= 7 )
                this.bitStart = bitStart;
            else
            {
                this.bitStart = DEFAULT_BITSTART;
                throw new InvalidBitFilterException(
                    "Invalid Start Bit: " + bitStart.ToString( ) );
            }
            if ( numBits > 0 && numBits <= ( 8 - StartBitNumber ) )
                this.numBits = numBits;
            else
            {
                this.numBits = ( 8 - StartBitNumber );
                throw new InvalidBitFilterException(
                    "Invalid Field Length: " + numBits );
            }
        }

        // --------------------------------------------------------------------

        /// <summary>
        /// Creates a new BitFilter from a formatted string.
        /// </summary>
        /// <param name="parseString">The BSL string to parse.</param>
        private void ParseString( string parseString )
        {
            string[] subs = parseString.Split( ':' );
            if ( !subs[ 0 ].StartsWith( "B" ) || subs[ 0 ].Length != 2 )
                throw new InvalidBitFilterException(
                    "Invalid sequence: \n\"" + parseString + "\"" );
            if ( !subs[ 1 ].StartsWith( "S" ) || subs[ 0 ].Length != 2 )
                throw new InvalidBitFilterException(
                    "Invalid sequence: \n\"" + parseString + "\"" );
            if ( !subs[ 2 ].StartsWith( "L" ) || subs[ 0 ].Length != 2 )
                throw new InvalidBitFilterException(
                    "Invalid sequence: \n\"" + parseString + "\"" );

            try
            {
                int b = int.Parse( subs[ 0 ].Remove( 0, 1 ) );
                int s = int.Parse( subs[ 1 ].Remove( 0, 1 ) );
                int l = int.Parse( subs[ 2 ].Remove( 0, 1 ) );
                Initialise( b, s, l );
            }
            catch ( Exception e )
            {
                throw new InvalidBitFilterException(
                    "Invalid sequence: \n\"" +
                    parseString + "\"\n" +
                    e.Message );
            }
        }

        // ####################################################################
        // Public methods

        /// <summary>
        /// Gets the integer value from the CANPacket at the BitFilter location.
        /// </summary>
        /// <param name="msg">CANPacket message to parse.</param>
        /// <returns>Integer value at the current BitFilter location.</returns>
        public int ParseValue( CANPacket msg )
        {
```

```
            int value = (int)msg[ ByteNumber ];
            value = value / ( ( int )Math.Pow( 2, StartBitNumber ) );
            value = value % ( ( int )Math.Pow( 2, FieldLength ) );
            return value;
        }

        // ------------------------------------------------------------------

        /// <summary>
        /// Adds the given value to the location specified by the BitFilter.
        /// Overflow may occur.
        /// </summary>
        /// <param name="msg">CANPacket to add to.</param>
        /// <param name="value">
        /// Value to be added.  Value is trimmed to the current field size.
        /// </param>
        public void AddValue( CANPacket msg, int value )
        {
            if ( value != value % ( ( int )Math.Pow( 2, FieldLength ) ) )
                MessageBox.Show(
                    "Field Overflow: \n" +
                    "Value: " + value.ToString( ) + "\n" +
                    "Max Value: " +
                    ( Math.Pow( 2, FieldLength ) - 1 ).ToString( ) );
            // Trim value to appropriate bit length
            value = value % ((int)Math.Pow(2, FieldLength) + 1);
            value = value * (int)Math.Pow(2, StartBitNumber);
            msg[ ByteNumber ] += ( byte )( value % 256 );
        }

        // ------------------------------------------------------------------

        /// <summary>
        /// Compares two BitFilters.  Returns true if all internal fields are
        /// identical.
        /// </summary>
        /// <param name="obj">BitFilter to compare.</param>
        /// <returns>True if equal.</returns>
        public override bool Equals( Object obj )
        {
            BitFilter bobj = ( BitFilter )obj;
            if (
                ( this.ByteNumber == bobj.ByteNumber ) &&
                ( this.StartBitNumber == bobj.StartBitNumber ) &&
                ( this.FieldLength == bobj.FieldLength ) )
            {
                return true;
            }
            else
            {
                return false;
            }
        }

        // ------------------------------------------------------------------

        /// <summary>
        /// Gets a unique hash code for the BitFilter.  Equal BitFilters will
        /// return the same hash code.
        /// </summary>
        /// <returns></returns>
        public override int GetHashCode( )
        {
            return ToString( ).GetHashCode( );
        }

        // ------------------------------------------------------------------

        /// <summary>
        /// A string representation of the BitFilter.
        /// </summary>
        /// <returns></returns>
        public override string ToString( )
        {
            return
                "B" + ByteNumber.ToString( "D1" ) +
                ":S" + StartBitNumber.ToString( "D1" ) +
                ":L" + FieldLength.ToString( "D1" );
        }

    } // public class BitFilter

    // ########################################################################
    //
```

```
    // ######################################################################

    /// <summary>
    /// Invalid BitFilter Exception.
    /// </summary>
    public class InvalidBitFilterException : ApplicationException
    {
        public InvalidBitFilterException( )
            : base("Illegal bit filter value.")
        {
        }

        public InvalidBitFilterException( string message )
            : base( message )
        {
        }

        public InvalidBitFilterException( string message, Exception inner )
            : base( message, inner )
        {
        }

    } // public class InvalidBitFilterException

} // namespace ModularRobot
```

C.3 CANPacket.cs

Listing C.2: CANPacket class source code.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;
using Lawicel;

namespace ModularRobot
{
    /// <summary>
    /// Contains a header and data structure for a CANUSB/CAN232 message.
    /// </summary>
    public class CANPacket
    {
        // ---------------------------------------------------------------------
        // Private data fields
        private LAWICEL.CANMsg innerMsg = new LAWICEL.CANMsg();

        // ---------------------------------------------------------------------
        // Public data fields / constants

        /// <summary>
        /// Integer representation of a Status Request message type.
        /// </summary>
        public const int MESSAGE_TYPE_STATUS_REQUEST = 0x0;   // -- --- --- 000
        /// <summary>
        /// Integer representation of a Type 1 message type.
        /// </summary>
        public const int MESSAGE_TYPE_1 = 0x1;                // -- --- --- 001
        /// <summary>
        /// Integer representation of a Type 2 message type.
        /// </summary>
        public const int MESSAGE_TYPE_2 = 0x2;                // -- --- --- 010
        /// <summary>
        /// Integer representation of a Type 3 message type.
        /// </summary>
        public const int MESSAGE_TYPE_3 = 0x3;                // -- --- --- 011

        // #####################################################################
        // Constructors

        /// <summary>
        /// Default CANPacket constructor.
        /// </summary>
        public CANPacket( ) { }

        // ---------------------------------------------------------------------

        /// <summary>
        /// Creates a new CANPacket using a deep copy method.
        /// </summary>
        /// <param name="original">CANPacket to copy from.  Must not be null.
        /// </param>
        public CANPacket( CANPacket original )
        {
            if ( original != null )
                this.CreateFromString( original.ToString( ) );
        }

        /// <summary>
        /// Creates a new CANPacket from a LAWICEL.CANMsg structure.
        /// </summary>
        /// <param name="original">LAWICEL.CANMsg to copy from.  Must not be
        /// null.</param>
        public CANPacket( LAWICEL.CANMsg original )
        {
            this.innerMsg.id = original.id;
            this.innerMsg.flags = original.flags;
            this.innerMsg.len = original.len;
            this.innerMsg.data = original.data;
        }

        // #####################################################################
        // Property accessors

        /// <summary>
        /// Gets or sets the Message Type (MT) bit of the identifier which
        /// selects either a broadcast or directed message.  If broadcast is
        /// set to true, the DirectedToMaster bit is set to false
        /// automatically.
```

```
        /// </summary>
        public bool Broadcast
        {
            // x - --- --- ---
            get { return ( int )( innerMsg.id / ( uint )1024 ) == ( uint )1; }
            set
            {
                if ( value != Broadcast )  // To be changed
                {
                    if ( Broadcast )         // Swap current status.
                    {
                        innerMsg.id -= ( uint )1024;
                        DirectedToMaster = false;
                    }
                    else
                        innerMsg.id += ( uint )1024;
                }
            }
        }

        // -----------------------------------------------------------------

        /// <summary>
        /// Gets or sets the DirectedToMaster (DTM) bit of the identifier.  If
        /// this bit is set to true, the Broadcast bit is set to false
        /// automatically.
        /// </summary>
        public bool DirectedToMaster
        {
            get
            {
                return ( int )
                    ( ( innerMsg.id / ( uint )512 ) % ( uint )2 ) == 1;
            }     // - x --- --- ---
            set
            {
                if ( value != DirectedToMaster )
                {
                    if ( DirectedToMaster )
                    {
                        innerMsg.id -= ( uint )512;
                        Broadcast = false;
                    }
                    else
                        innerMsg.id += ( uint )512;
                }
            }
        }

        // -----------------------------------------------------------------

        /// <summary>
        /// Gets or sets the Node Type Number (NTN) associated with this packet
        /// </summary>
        public int NodeTypeNumber    // - - xxx --- ---
        {
            get { return (int)( ( innerMsg.id / ( uint )64 ) % ( uint )8 ); }
            set
            {
                if ( value >= 0 && value <= 7 )
                {
                    innerMsg.id -= ( uint )( NodeTypeNumber * 64 );
                    innerMsg.id += ( uint )( value * 64 );
                }
            }
        }

        // -----------------------------------------------------------------

        /// <summary>
        /// Gets or sets the Node ID Number (NIN) assigned to the node.
        /// </summary>
        public int NodeIDNumber      // - - --- xxx ---
        {
            get { return ( int )( ( innerMsg.id / ( uint )8 ) % ( uint )8 ); }
            set
            {
                if ( value >= 0 && value <= 7 )
                {
                    innerMsg.id -= ( uint )(NodeIDNumber * 8);
                    innerMsg.id += ( uint )( value * 8 );
                }
            }
        }
```

```

        // --------------------------------------------------------------------

        /// <summary>
        /// Gets or sets the Message Sub-Type (MST) associated with this
        /// message.
        /// </summary>
        public int MessageType      // - - --- --- xxx
        {
            get { return ( int )( ( innerMsg.id / ( uint )1 ) % ( uint )8 ); }
            set
            {
                if ( value >= 0 && value <= 7 )
                {
                    innerMsg.id -= ( uint )MessageType;
                    innerMsg.id += ( uint )( value * 1 );
                }
            }
        }

        // --------------------------------------------------------------------

        /// <summary>
        /// Gets or set the RTR bit in the control field.  This bit signals a
        /// Remote Frame Request.
        /// </summary>
        public bool RTR
        {
            get { return innerMsg.flags == LAWICEL.CANMSG_RTR; }
            set
            {
                if ( value )
                    innerMsg.flags = LAWICEL.CANMSG_RTR;
                else
                    innerMsg.flags = 0x0;
            }
        }

        // --------------------------------------------------------------------

        /// <summary>
        /// Gets or sets the number of data bytes (DLC) to follow the message
        /// header.  Must be
        /// between 0 and 8 inclusive.
        /// </summary>
        public int DataLength
        {
            get { return (int)innerMsg.len; }
            set
            {
                // Ensures that the value set is between 0 - 8.
                innerMsg.len = ( byte )( Math.Max( Math.Min( value, 8 ), 0 ) );
            }
        }

        /// <summary>
        /// Gets or sets the data value the indexed location.  Index has a
        /// valid range from 0 - 7 inclusive, and will return (0) if out of
        /// bounds.
        /// </summary>
        /// <param name="index">A index into the .</param>
        /// <returns>Integer value of the byte at the indexed location.
        /// </returns>
        public int this[ int index ]
        {
            get
            {
                if ( index >= 0 && index <= 7 && index < DataLength )
                {
                    ulong val = DataLong / ( (ulong)Math.Pow( 256, index ) );
                    return (int)(val % 256);
                }
                return 0;
            }
            set
            {
                if ( index >= 0 && index <= 7 )
                {
                    value = value % 256;
                    int val = value - this[ index ];
                    DataLong += ((ulong)val) * (ulong)Math.Pow( 256, index );
                    if ( index >= DataLength )
                        DataLength = index + 1;
                }
```

```
            }
        }

        /// <summary>
        /// Returns a single unsigned long integer value of the entire Data field.
        /// </summary>
        public ulong DataLong
        {
            get { return innerMsg.data; }
            set { innerMsg.data = value; }
        }

        // ####################################################################
        // Private methods



        // ####################################################################
        // Public methods

        /// <summary>
        /// This method is redundant when using the USB driver interface.
        ///
        /// Fills packet header and data values from a given string of HEX
        /// values.  An invalid string sequence will result in default packet
        /// values, and the original string returned.  If the parse action is
        /// successful, the HEX values used from the string are removed and
        /// the remainder of the string is returned.
        /// </summary>
        /// <param name="buffer">A string of HEX values to parse from.</param>
        /// <returns>
        /// The original string on failure, unused character string
        /// on success.
        /// </returns>
        public string CreateFromString( string buffer )
        {
            if ( buffer.Length < 4 )    // Minimum header length.
                return buffer;

            UInt16[ ] bytes = new UInt16[ buffer.Length ];
            // Convert the HEX string into integer values.
            for ( int i = 0; i < buffer.Length; i++ )
                bytes[ i ] = HexConverter.ConvertToUInt16( buffer[ i ] );

            int identifier = 0;     // Initial header value.

            // Create the message identifier.
            identifier += bytes[ 0 ] * 256;
            identifier += bytes[ 1 ] * 16;
            identifier += bytes[ 2 ];

            innerMsg.id = ( uint )identifier;

            DataLength = bytes[ 3 ];    // Get the data length

            // Ensure that enough data bytes are supplied in the HEX string.
            if ( buffer.Length < 4 + DataLength * 2 )
                return buffer;

            // Copy the data bytes into the packet.
            for ( int i = 0; i < 8 && i < DataLength; i++ )
            {
                // High nibble
                this[ i ] = ( byte )( bytes[ i * 2 + 4 ] * 16 );
                // Low nibble
                this[ i ] += ( byte )bytes[ i * 2 + 5 ];
            }

            StringBuilder retString = new StringBuilder( buffer );

            // Remove the used bytes from the string.
            retString.Remove( 0, 4 + DataLength );

            return retString.ToString( );
        }

        // --------------------------------------------------------------------

        /// <summary>
        /// Converts the packet to a string of HEX values.
        ///
        /// This method is redundant when using the USB driver interface.
        /// </summary>
        /// <returns>A HEX string representation of the packet.</returns>
        public string ToHexString( )
```

```
        {
            // Construct the identifier
            int identifier = 0x000;
            if ( DirectedToMaster )
                identifier += 0x200;
            else if ( Broadcast )
                identifier += 0x400;
            identifier += NodeTypeNumber * 64;
            identifier += NodeIDNumber * 8;
            identifier += MessageType;

            //innerMsg.id = (uint)identifier;

            // Convert to ascii
            StringBuilder result = new StringBuilder(
                identifier.ToString( "X3" ) );

            // Append the data length code (Low nibble)
            result.Append( DataLength.ToString( "X2" )[ 1 ] );

            // Add each data byte
            for ( int i = 0; i < DataLength; i++ )
                result.Append( this[ i ].ToString( "X2" ) );

            return result.ToString( );
        }

        // --------------------------------------------------------------------

        /// <summary>
        /// Returns a string representation of the CANPacket
        /// </summary>
        /// <returns>String representation of the CANPacket.</returns>
        public override string ToString( )
        {
            return ToHexString( );
        }

        internal LAWICEL.CANMsg LawicelMsg( )
        {
            return new CANPacket( this ).innerMsg;
        }
    } // class CANPacket


} // namespace ModularRobot
```

C.4 CANUSB.cs

Listing C.3: CANUSB class source code.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;
using System.Threading;
using Lawicel;
using System.IO;

namespace ModularRobot
{
    public class CANUSB
    {
        // -------------------------------------------------------------------
        // Private data fields
        private uint handle;                                // CANUSB adapter handle.
        private System.Threading.Timer writeTimer;
        private System.Threading.Timer readTimer;
        private System.Threading.Timer statusTimer;
        private String baudRate = "";
        private int writeTimeout = 0;
        private int readTimeout = 0;

        private Queue<CANPacket> txQueue = null;
        private Queue<CANPacket> rxQueue = null;

        private Log canlog;                               // Manages a log file.
        private Object lockObject = new Object();

        // -------------------------------------------------------------------
        // Public data fields / constants

        public event EventHandler MessageReceived;

        public const String CAN_LOG_FILE = "./CanLog.txt";
        // Interval between CAN Status checks.
        public const int STATUS_CHECK_INTERVAL = 1000;

        public const string CAN_BAUD_1M = "1000";                  //   1 MBit / s
        public const string CAN_BAUD_800K = "800";                 // 800 kBit / s
        public const string CAN_BAUD_500K = "500";                 // 500 kBit / s
        public const string CAN_BAUD_250K = "250";                 // 250 kBit / s
        public const string CAN_BAUD_125K = "125";                 // 125 kBit / s
        public const string CAN_BAUD_100K = "100";                 // 100 kBit / s
        public const string CAN_BAUD_50K = "50";                   //  50 kBit / s
        public const string CAN_BAUD_20K = "20";                   //  20 kBit / s
        public const string CAN_BAUD_10K = "10";                   //  10 kBit / s

        // ###################################################################
        // Constructors

        /// <summary>
        /// Creates a new CANUSB interface.
        /// </summary>
        /// <param name="CAN_BAUD">
        /// A string representation of the baud rate.
        /// </param>
        /// <param name="readTimeout">
        /// Interval time between read attempts.  Dependent on baud rate.
        /// </param>
        /// <param name="writeTimeout">
        /// Delay time between write operations.  Dependent on baud rate.
        /// </param>
        public CANUSB( String CAN_BAUD, int readTimeout, int writeTimeout )
        {
            Initialise( CAN_BAUD, readTimeout, writeTimeout );
        }

        // -------------------------------------------------------------------

        /// <summary>
        /// Manages the CANUSB initialisation process.
        /// </summary>
        /// <param name="baudRate"></param>
        /// <param name="readTimeout"></param>
        /// <param name="writeTimeout"></param>
        private void Initialise(
            String baudRate, int readTimeout, int writeTimeout )
        {
            // Checks to ensure that the given baud rate is valid.
            if ( CheckBaudRate( baudRate ) )
```

```
                this.baudRate = baudRate;
            else
                throw new InvalidBaudRateException(
                    "Invalid Baud Rate: " + baudRate.ToString( ) );

            if ( readTimeout > 0 )
                this.readTimeout = readTimeout;

            if ( writeTimeout > 0 )
                this.writeTimeout = writeTimeout;

            txQueue = new Queue<CANPacket>();
            rxQueue = new Queue<CANPacket>();

            canlog = new Log( CAN_LOG_FILE );
        }

        // ####################################################################
        // Property accessors

        /// <summary>
        /// Adds CANPackets to the TX queue, and removes CANPackets from the
        /// RX queue.  Returns null if no messages available.
        /// </summary>
        public CANPacket Buffer
        {
            get
            {
                CANPacket retVal = null;
                lock ( rxQueue )   // Protects the buffer from multiple access.
                {
                    if ( rxQueue.Count > 0 )
                        // Attempts to remove a CANPacket.
                        retVal = rxQueue.Dequeue( );
                }

                if ( retVal != null )
                {
                    canlog.Value = "Dequeue: " + retVal.ToString( );
                    return new CANPacket( retVal );
                }
                else
                    return retVal;
            }
            set
            {
                lock ( txQueue )         // Protects buffer from multiple access
                {
                    if ( value != null )
                    {
                        txQueue.Enqueue( new CANPacket( value ) );
                        // Adds a log entry
                        canlog.Value = "Enqueue: " + value.ToString( );
                    }
                }
            }
        }

        // --------------------------------------------------------------------

        /// <summary>
        /// Gets or sets the baud rate of the bus.  Change will not take effect
        /// until port has been  re-opened.
        /// </summary>
        public String BaudRate
        {
            get { return baudRate; }
            set
            {
                if ( CheckBaudRate( value ) )
                    baudRate = value;
            }
        }

        // --------------------------------------------------------------------

        /// <summary>
        /// Gets or sets the read time interval.  Changes take effect
        /// immediately
        /// </summary>
        public int ReadTimeout
        {
            get { return readTimeout; }
            set
            {
                if ( value >= 0 )
```

```
                readTimeout = value;
            readTimer.Change( readTimeout, readTimeout );
        }
    }

    // ------------------------------------------------------------------

    /// <summary>
    /// Gets or set the write time delay.  Changes take effect immediately
    /// </summary>
    public int WriteTimeout
    {
        get { return writeTimeout; }
        set
        {
            if ( value >= 0 )
                writeTimeout = value;
            writeTimer.Change( writeTimeout, writeTimeout );
        }
    }

    // ------------------------------------------------------------------

    /// <summary>
    /// Thread safe TX queue count
    /// </summary>
    private int SafeTXCount
    {
        get
        {
            int count;
            lock ( txQueue )
            {
                count = txQueue.Count;
            }
            return count;
        }
    }

    // ------------------------------------------------------------------

    /// <summary>
    /// Thread safe RX queue count
    /// </summary>
    private int SafeRXCount
    {
        get
        {
            int count;
            lock ( rxQueue )
            {
                count = rxQueue.Count;
            }
            return count;
        }
    }

    // ###################################################################
    // Public methods

    /// <summary>
    /// Event to be triggered when a message is received.
    /// </summary>
    /// <param name="e"></param>
    protected virtual void OnMessageReceived( EventArgs e )
    {
        if ( MessageReceived != null )
            MessageReceived( this, e );
    }

    // -------------------------------------------------------------------

    /// <summary>
    /// Configures and opens the CAN bus ready for transfer.
    /// </summary>
    /// <returns>True on success, false on failure.</returns>
    public bool Open( )
    {
        handle = LAWICEL.canusb_Open( IntPtr.Zero,
            baudRate,
            LAWICEL.CANUSB_ACCEPTANCE_CODE_ALL,
            LAWICEL.CANUSB_ACCEPTANCE_MASK_ALL,
            LAWICEL.CANUSB_FLAG_TIMESTAMP );
        if ( handle <= 0 )
        {
            MessageBox.Show( "Failed to Open CANUSB" );
```

```
                return false;
            }

            readTimer = new System.Threading.Timer(
                new TimerCallback( this.Read ),
                null,
                readTimeout,
                readTimeout );

            writeTimer = new System.Threading.Timer(
                new TimerCallback( this.Write ),
                null,
                writeTimeout,
                writeTimeout );

            canlog.Value = "CANUSB Initialised";

            statusTimer = new System.Threading.Timer(
                new TimerCallback( this.StatusCheck ),
                null,
                STATUS_CHECK_INTERVAL,
                STATUS_CHECK_INTERVAL );

            return true;

        }

        // -------------------------------------------------------------------

        /// <summary>
        /// Clears the TX queue and appends the given message.
        /// </summary>
        /// <param name="msg">Emergency message to transmit.</param>
        public void EmergencyMsg( CANPacket msg )
        {
            // Adds a log entry
            canlog.Value = "EMERGENCY MESSAGE: " + msg.ToString( );
            lock ( txQueue )
            {
                canlog.Value = "Clearing " + txQueue.Count + " messages.";
                txQueue.Clear( );
                txQueue.Enqueue( new CANPacket( msg ) );
            }
            writeTimer.Change( 0, writeTimeout );
        }

        // -------------------------------------------------------------------

        /// <summary>
        /// Closes and releases the LAWICEL CANUSB adapter.
        /// </summary>
        public void Close( )
        {
            canlog.Value = "Closing CANUSB.";
            while ( txQueue.Count > 0 )
            {
                Thread.Sleep( 500 );

                if ( txQueue.Count > 0 )
                    canlog.Value = txQueue.Count.ToString( ) +
                        " items still remaining in TX queue.";
            }
            readTimer.Change(
                System.Threading.Timeout.Infinite,
                System.Threading.Timeout.Infinite );

            int res = LAWICEL.canusb_Close( handle );
            if ( LAWICEL.ERROR_CANUSB_OK == res )
            {
                canlog.Value = "Closed OK";
            }
            else
            {
                MessageBox.Show( "Failed to Close CANUSB" );
                canlog.Value = "Failed to Close CANUSB";
            }
        }

        // ###################################################################
        // Private methods

        /// <summary>
        /// Callback method to check the status of the canbus.
        /// </summary>
        /// <param name="state">State information passed in.</param>
        private void StatusCheck( Object state )
```

```
        {

            int rv;
            lock ( lockObject )
            {
                rv = LAWICEL.canusb_Status( handle );
            }

            if ( rv == 0 )
            {
                //canlog.Value = "STATUS: OK";
            }
            else
            {
                String errstr = "ERROR: ";
                if ( rv % 2 == 1 )
                    errstr += "'RX FIFO Full'  ";
                rv = rv / 2;

                if ( rv % 2 == 1 )
                    errstr += "'TX FIFO Full'  ";
                rv = rv / 2;

                if ( rv % 2 == 1 )
                    errstr += "'Error Warning'  ";
                rv = rv / 2;

                if ( rv % 2 == 1 )
                    errstr += "'Data Overrun'  ";
                rv = rv / 2;

                rv = rv / 2;

                if ( rv % 2 == 1 )
                    errstr += "'Error Passive'  ";
                rv = rv / 2;

                if ( rv % 2 == 1 )
                    errstr += "'Arbitration Lost'  ";
                rv = rv / 2;

                if ( rv % 2 == 1 )
                    errstr += "'Bus Error'  ";

                canlog.Value = errstr;
            }

        }

        // ---------------------------------------------------------------------

        /// <summary>
        /// Callback function used to periodically read messages from the
        /// CAN bus.
        /// </summary>
        /// <param name="state"></param>
        private void Read( Object state )
        {
            int iCont = 1;
            // Read one CAN message
            LAWICEL.CANMsg msg = new LAWICEL.CANMsg( );

            //canlog.Value = "Reading";
            while ( iCont == 1 )
            {

                int rv;
                lock ( lockObject )
                {
                    rv = LAWICEL.canusb_Read( handle, out msg );
                }
                if ( LAWICEL.ERROR_CANUSB_OK == rv )
                {

                    CANPacket canmsg = new CANPacket( msg );

                    canlog.Value = "RX: " +
                        msg.id.ToString( "X3" ) + " " +
                        msg.len.ToString( "X1" ) + " " +
                        HexConverter.InvertHexString(
                            msg.data.ToString( "X16" ) );

                    lock ( rxQueue )
                    {
                        rxQueue.Enqueue( canmsg );
                        OnMessageReceived( new EventArgs( ) );
                    }
```

```
                }
                else if ( LAWICEL.ERROR_CANUSB_NO_MESSAGE == rv )
                {
                    iCont = 0;
                }
                else
                {
                    iCont = 0;
                    canlog.Value = "Failed to read message.";
                    // Trigger the status check.
                    statusTimer.Change( 0, STATUS_CHECK_INTERVAL );
                }
            }
        }

        // ------------------------------------------------------------------

        /// <summary>
        /// Sends messages from the TX queue.  Active while there are messages
        /// to be sent.
        /// </summary>
        private void Write( Object state )
        {
            if(SafeTXCount > 0)
            {

                CANPacket msg = null;
                lock ( txQueue )
                {
                    if ( SafeTXCount == 0 )
                        return;
                    msg = txQueue.Dequeue( );
                }

                LAWICEL.CANMsg lawicelMsg = msg.LawicelMsg( );

                int rv;
                lock ( lockObject )
                {
                    rv = LAWICEL.canusb_Write( handle, ref lawicelMsg );
                }
                if ( LAWICEL.ERROR_CANUSB_OK == rv )
                {
                    canlog.Value = "TX: " +
                        lawicelMsg.id.ToString( "X3" ) + " " +
                        lawicelMsg.len.ToString( "X1" ) + " " +
                        HexConverter.InvertHexString(
                            lawicelMsg.data.ToString(
                            "X" + ( ( int )lawicelMsg.len * 2 ).ToString( ) ) );
                }
                else if ( LAWICEL.ERROR_CANUSB_TX_FIFO_FULL == rv )
                {
                    canlog.Value = "ERROR: FIFO full. Can't send message.  " +
                        msg.ToString( );
                    // Trigger the status check.
                    statusTimer.Change( 0, STATUS_CHECK_INTERVAL );
                }
                else
                {
                    canlog.Value = "ERROR: Failed to send message.  " +
                        msg.ToString( );
                    // Trigger the status check.
                    statusTimer.Change( 0, STATUS_CHECK_INTERVAL );
                }
            }
        }

        // ------------------------------------------------------------------

        /// <summary>
        /// Checks to see if the given string is a valid LAWICEL baud rate.
        /// </summary>
        /// <param name="br">Baud rate.</param>
        /// <returns>True if given string is a valid CAN baud rate.</returns>
        private bool CheckBaudRate( String br )
        {
            return ( br.CompareTo( CAN_BAUD_10K ) == 0 ||
                br.CompareTo( CAN_BAUD_20K ) == 0 ||
                br.CompareTo( CAN_BAUD_50K ) == 0 ||
                br.CompareTo( CAN_BAUD_100K ) == 0 ||
                br.CompareTo( CAN_BAUD_125K ) == 0 ||
                br.CompareTo( CAN_BAUD_250K ) == 0 ||
                br.CompareTo( CAN_BAUD_500K ) == 0 ||
                br.CompareTo( CAN_BAUD_800K ) == 0 ||
```

```
                    br.CompareTo( CAN_BAUD_1M ) == 0 );
        }
    } // public class CANUSB


    // ######################################################################
    //                                                                    //
    // ######################################################################

    public class InvalidBaudRateException : ApplicationException
    {
        public InvalidBaudRateException( )
            : base("Invalid Baud Rate.")
        {
        }

        // ------------------------------------------------------------------

        public InvalidBaudRateException( string message )
            : base( message )
        {
        }

        // ------------------------------------------------------------------

        public InvalidBaudRateException( string message, Exception inner )
            : base( message, inner )
        {
        }

    } // public class InvalidBaudRateException

} // namespace ModularRobot
```

C.5 HexConverter.cs

Listing C.4: HexConverter class source code.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ModularRobot
{
    /// <summary>
    /// A simple HEX converting class
    /// </summary>
    public class HexConverter
    {
        /// <summary>
        /// Converts a HEX based character (0:F) to its integer value
        /// (0x00:0x0F).
        /// </summary>
        /// <param name="hexVal">A HEX character in the range of 0 to F.
        /// </param>
        /// <returns>Integer value n the range 0x0 to 0xF.  Returns 0x00 if
        /// character cannot be converted.</returns>
        public static UInt16 ConvertToUInt16( char hexVal )
        {
            switch ( hexVal )
            {
                case '0':
                case '1':
                case '2':
                case '3':
                case '4':
                case '5':
                case '6':
                case '7':
                case '8':
                case '9':
                    return ( UInt16 )( hexVal - 0x30 );
                case 'A':
                case 'a':
                    return 0x0A;
                case 'B':
                case 'b':
                    return 0x0B;
                case 'C':
                case 'c':
                    return 0x0C;
                case 'D':
                case 'd':
                    return 0x0D;
                case 'E':
                case 'e':
                    return 0x0E;
                case 'F':
                case 'f':
                    return 0x0F;
            }
            return 0x00;
        }

        /// <summary>
        /// Inverts a given string of HEX values.  Inverts full byte values
        /// only.
        /// </summary>
        /// <param name="str">HEX input string.  Length of string must be even.
        /// </param>
        /// <returns>Inverted string.  ie: 0F4F8FCF will return CF8F4F0F.
        /// </returns>
        public static String InvertHexString( String str )
        {
            if ( str.Length > 1 )
            {
                if ( str.Length % 2 != 0 )
                    throw new Exception(
                        "Invalid HEX string: Odd number of characters." );
                StringBuilder s = new StringBuilder( );
                for ( int i = 0; i < str.Length; i += 2 )
                {
                    s.Insert( 0, str[ i + 1 ] );
                    s.Insert( 0, str[ i ] );
                }

                return s.ToString( );
            }
            else
```

```
                return str;
            }

    } // class HexConverter
}
```

C.6 Log.cs

Listing C.5: Log class source code.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;

namespace ModularRobot
{
    public class Log
    {
        // ------------------------------------------------------------------
        // Private data fields

        private StreamWriter file;
        private StringBuilder log;
        private bool newestontop = false;

        // ------------------------------------------------------------------
        // Public data fields / constants

        // ##################################################################
        // Constructors

        public Log( String name )
        {
            file = new StreamWriter( name );
            file.WriteLine(
                "Log Created:  " + DateTime.Now.ToString( ) + "\n" );
            log = new StringBuilder( );
        }

        // ##################################################################
        // Property accessors

        /// <summary>
        /// Determines whether new messages are placed at the top or bottom of
        /// log file.
        /// </summary>
        public bool NewestOnTop
        {
            get { return newestontop; }
            set { newestontop = value; }
        }

        // ------------------------------------------------------------------

        /// <summary>
        /// Writes a timestamped log entry to the file or gets the full log
        /// file in a string.
        /// </summary>
        public String Value
        {
            get
            {
                string templog;
                lock ( log )
                {
                    templog = log.ToString( );
                }
                return templog;
            }
            set
            {
                lock ( log )
                {
                    DateTime dt = DateTime.Now;     // Timestamp
                    String logString =
                        dt.Hour.ToString( "D2" ) + ":" +
                        dt.Minute.ToString( "D2" ) + ":" +
                        dt.Second.ToString( "D2" ) + "." +
                        dt.Millisecond.ToString( "D3" ) + " " +
                        value + "\n";
                    if ( newestontop )
                    {
                        log.Insert( 0, logString );
                    }
                    else
                    {
                        log.Append( logString );
                    }
                    file.Write( logString );
```

```
                        //MessageBox.Show( "Log" + logString );
                        file.Flush( );
                    }
                }
            }


            // ####################################################################
            // Private destructor

            ~Log( )
            {
                try
                {
                    //file.Close( );
                }
                catch ( Exception ) { }
            }

        } // public class Log

} // namespace ModularRobot
```

C.7 MRModel.cs

Listing C.6: MRModel class source code.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Xml;
using System.Windows.Forms;
using System.Threading;


namespace ModularRobot
{
    public class MRModel
    {
        // -------------------------------------------------------------------
        // Private data fields
        private String configFileName;
        private Dictionary<String, Node> nodes;
        private CANUSB comm;
        private int readtimeout;

        // -------------------------------------------------------------------
        // Public data fields / constants

        // ###################################################################
        // Constructors

        public MRModel( )
        {
            Initialise( );
        }

        private void Initialise( )
        {
            configFileName = "./robotCAN.xml";
            nodes = new Dictionary<string, Node>( );
            //comm = new CANText( );
        }

        // ###################################################################
        // Property accessors

        /// <summary>
        /// Sets the XML config file name for this robot model
        /// </summary>
        public String ConfigFileName
        {
            get { return configFileName; }
            set { configFileName = value; }
        }

        // ###################################################################
        // Private methods

        /// <summary>
        /// Function that is called when a CAN message is received.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ReceiveMessage( Object sender, EventArgs e )
        {
            CANPacket msg = comm.Buffer;
            foreach ( Node n in nodes.Values )
            {
                if ( n.NodeTypeNumber == msg.NodeTypeNumber &&
                    n.NodeIDNumber == msg.NodeIDNumber )
                {
                    n.ParseMessage( msg );
                }
            }
        }

        // -------------------------------------------------------------------

        /// <summary>
        /// Returns a named node or null
        /// </summary>
        /// <param name="name"></param>
        /// <returns></returns>
        private Node GetNode( String name )
        {
            Node temp;
            if ( !nodes.TryGetValue( name, out temp ) )
```

```
                return null;
            return temp;
        }

        // #####################################################################
        // Public methods

        /// <summary>
        /// Causes an emergency stop message to be sent to the CAN bus.
        /// </summary>
        public void EmergencyStop( )
        {
            CANPacket msg = new CANPacket( );
            msg.Broadcast = true;
            msg.DirectedToMaster = false;
            msg.NodeTypeNumber = Node.MASTER_NODE;
            msg.NodeIDNumber = 0;
            msg.MessageType = 0;
            msg[ 0 ] = 0;

            comm.EmergencyMsg( msg );
        }

        // ---------------------------------------------------------------------

        /// <summary>
        /// Initialises the model and opens the comms channel
        /// </summary>
        /// <returns>
        /// 0 on success, negative error message on failure
        /// </returns>
        public int Open( )
        {
            XmlDocument doc = new XmlDocument( );
            try
            {
                doc.Load( configFileName );
            }
            catch ( XmlException exp )
            {
                MessageBox.Show(
                    "Invalid XML File: " + configFileName + "\n" + exp.Message );
                return -1;
            }
            catch ( Exception exp )
            {
                // Just in case
                MessageBox.Show(
                    exp.Message, exp.Source,
                    MessageBoxButtons.OK, MessageBoxIcon.Error );
            }
            XmlElement root = doc.DocumentElement;
            XmlAttributeCollection settings = root.Attributes;

            try
            {
                String br = settings[ "baudrate" ].Value;
                int ri = int.Parse( settings[ "readinterval" ].Value );
                int wi = int.Parse( settings[ "writeinterval" ].Value );
                readtimeout = int.Parse( settings[ "readtimeout" ].Value );

                comm = new CANUSB( br, ri, wi );
            }
            catch ( Exception e )
            {
                MessageBox.Show( e.Message );
                return -2;
            }

            //              MessageBox.Show( "About to parse" );
            foreach ( XmlNode node in root.SelectNodes( "node" ) )
            {
                try
                {
                    Node newNode = new Node( ( XmlElement )( node ) );
                    nodes.Add( newNode.Name, newNode );
                }
                catch ( Exception e )
                {
                    //MessageBox.Show( "Exception Thrown" );
                    MessageBox.Show( e.Message );
                    return -3;
                }
            }

```

```
            comm.MessageReceived += new EventHandler( this.ReceiveMessage );

            if ( comm.Open( ) )
                return 0;
            return -1;
        }

        // ------------------------------------------------------------------

        /// <summary>
        /// Closes the communication channel
        /// </summary>
        /// <returns></returns>
        public int Close( )
        {
            comm.Close( );
            return 0;
        }

        // ------------------------------------------------------------------

        /// <summary>
        /// Returns a value from a named port on a node
        /// </summary>
        /// <param name="node">Node to access</param>
        /// <param name="port">Port name to get value</param>
        /// <returns></returns>
        public int Get( String node, String port )
        {
            Node n = GetNode( node );
            if ( n != null )
            {
                comm.Buffer = n.StatusMessage( );
                int timeout = readtimeout;
                while ( !n.UpToDate && timeout > 0 )
                {
                    Thread.Sleep( readtimeout / 50 );
                    timeout -= readtimeout / 50;
                }
                if(timeout < 0)
                {
                    MessageBox.Show(
                        "Status request '" + node + "." +
                        port + "' timed out." );
                }
                try
                {
                    return n.GetValue( port );
                }
                catch ( Exception e )
                {
                    MessageBox.Show(
                        "Port '" + port + "' on node '" +
                        node + "' not found.\n" + e.Message);
                    return 0;
                }
            }
            return 0;
        }

        // ------------------------------------------------------------------

        /// <summary>
        /// Returns a string representation of the class.
        /// </summary>
        /// <returns></returns>
        public override string ToString( )
        {
            StringBuilder str = new StringBuilder( );
            str.AppendLine( "File: " + ConfigFileName );
            foreach ( Node n in nodes.Values )
            {
                str.AppendLine( n.ToString( ) );
            }
            return str.ToString( );
        }

        // ------------------------------------------------------------------

        /// <summary>
        /// Sets a given number of ports with corresponding values
        /// </summary>
        /// <param name="node">Node name to access</param>
        /// <param name="port">String array of port names to set</param>
```

```
        /// <param name="value">Corresponding port values to be set</param>
        public void Set( String node, String[ ] port, int[ ] value )
        {

            if ( port.Length != value.Length || port.Length == 0 )
                throw new Exception(
                    "Port and value arrays have different lengths." );

            Node n = GetNode( node );
            List<int> mt = new List<int>( );
            if ( n != null )
            {
                for ( int i = 0; i < port.Length; i++ )
                {
                    try
                    {
                        int t = n.SetValue( port[ i ], value[ i ] );
                        if ( t != 0 && !mt.Contains( t ) )
                            mt.Add( t );
                    }
                    catch ( Exception e )
                    {
                        MessageBox.Show(
                            "Port '" + port[ i ] + "' on node '" + node +
                            "' not found.\n" + e.Message );
                        return;
                    }
                }
                mt.Sort( );
                foreach ( int i in mt )
                {
                    comm.Buffer = n.GetMessage( i );
                }
            }
        }

        // ---------------------------------------------------------------

        /// <summary>
        /// Sets "value1" to "port1" on "node"
        /// </summary>
        /// <param name="node">Node to be accessed</param>
        /// <param name="port1">Port name</param>
        /// <param name="value1">Value</param>
        public void Set( String node, String port1, int value1 )
        {
            String[ ] sa = { port1 };
            int[ ] ia = { value1 };
            Set( node, sa, ia );
        }

        // ---------------------------------------------------------------

        /// <summary>
        /// Sets "valueN" to "portN" on "node"
        /// </summary>
        /// <param name="node">Node to be accessed</param>
        /// <param name="portN">Port name</param>
        /// <param name="valueN">Value</param>
        public void Set( String node,
            String port1, int value1,
            String port2, int value2 )
        {
            String[ ] sa = { port1, port2 };
            int[ ] ia = { value1, value2 };
            Set( node, sa, ia );
        }

        // ---------------------------------------------------------------

        /// <summary>
        /// Sets "valueN" to "portN" on "node"
        /// </summary>
        /// <param name="node">Node to be accessed</param>
        /// <param name="portN">Port name</param>
        /// <param name="valueN">Value</param>
        public void Set( String node,
            String port1, int value1,
            String port2, int value2,
            String port3, int value3 )
        {
            String[ ] sa = { port1, port2, port3 };
            int[ ] ia = { value1, value2, value3 };
```

```
            Set( node, sa, ia );
        }

        // ----------------------------------------------------------------

        /// <summary>
        /// Sets "valueN" to "portN" on "node"
        /// </summary>
        /// <param name="node">Node to be accessed</param>
        /// <param name="portN">Port name</param>
        /// <param name="valueN">Value</param>
        public void Set( String node,
            String port1, int value1,
            String port2, int value2,
            String port3, int value3,
            String port4, int value4 )
        {
            String[ ] sa = { port1, port2, port3, port4 };
            int[ ] ia = { value1, value2, value3, value4 };
            Set( node, sa, ia );
        }

        // ----------------------------------------------------------------

        /// <summary>
        /// Sets "valueN" to "portN" on "node"
        /// </summary>
        /// <param name="node">Node to be accessed</param>
        /// <param name="portN">Port name</param>
        /// <param name="valueN">Value</param>
        public void Set( String node,
            String port1, int value1,
            String port2, int value2,
            String port3, int value3,
            String port4, int value4,
            String port5, int value5 )
        {
            String[ ] sa = { port1, port2, port3, port4, port5 };
            int[ ] ia = { value1, value2, value3, value4, value5 };
            Set( node, sa, ia );
        }

        // ----------------------------------------------------------------

        /// <summary>
        /// Sets "valueN" to "portN" on "node"
        /// </summary>
        /// <param name="node">Node to be accessed</param>
        /// <param name="portN">Port name</param>
        /// <param name="valueN">Value</param>
        public void Set( String node,
            String port1, int value1,
            String port2, int value2,
            String port3, int value3,
            String port4, int value4,
            String port5, int value5,
            String port6, int value6 )
        {
            String[ ] sa = { port1, port2, port3, port4, port5, port6 };
            int[ ] ia = { value1, value2, value3, value4, value5, value6 };
            Set( node, sa, ia );
        }

        // ----------------------------------------------------------------

        /// <summary>
        /// Sets "valueN" to "portN" on "node"
        /// </summary>
        /// <param name="node">Node to be accessed</param>
        /// <param name="portN">Port name</param>
        /// <param name="valueN">Value</param>
        public void Set( String node,
            String port1, int value1,
            String port2, int value2,
            String port3, int value3,
            String port4, int value4,
            String port5, int value5,
            String port6, int value6,
            String port7, int value7 )
        {
            String[ ] sa = {
                port1, port2, port3, port4, port5, port6, port7
            };
            int[ ] ia = {
```

```
                value1, value2, value3, value4, value5, value6, value7
            };
            Set( node, sa, ia );
        }

        // --------------------------------------------------------------------

        /// <summary>
        /// Sets "valueN" to "portN" on "node"
        /// </summary>
        /// <param name="node">Node to be accessed</param>
        /// <param name="portN">Port name</param>
        /// <param name="valueN">Value</param>
        public void Set( String node,
            String port1, int value1,
            String port2, int value2,
            String port3, int value3,
            String port4, int value4,
            String port5, int value5,
            String port6, int value6,
            String port7, int value7,
            String port8, int value8 )
        {
            String[ ] sa = {
                port1, port2, port3, port4, port5, port6, port7, port8
            };
            int[ ] ia = {
                value1, value2, value3, value4, value5, value6, value7, value8
            };
            Set( node, sa, ia );
        }

    } // public class MRModel

} // namespace ModularRobot
```

C.8 Node.cs

Listing C.7: Node class source code.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Xml;
using System.Windows.Forms;
namespace ModularRobot
{
    public class Node
    {
        // ---------------------------------------------------------------------
        // Private data fields

        private String name;
        private String description;
        private int nodetypenumber;
        private int nodeIDnumber;
        private Dictionary<String, Port> ports;
        private bool online = false;
        private bool uptodate = false;

        // ---------------------------------------------------------------------
        // Public data fields / constants

        /// <summary>
        /// Integer representation of Master node type number.
        /// </summary>
        public const int MASTER_NODE = 0x7;                 // -- 111 --- ---
        /// <summary>
        /// Integer representation of a DC Motor node type number.
        /// </summary>
        public const int DC_MOTOR = 0x1;                    // -- 001 --- ---
        /// <summary>
        /// Integer representation of a Stepper Motor node type number.
        /// </summary>
        public const int STEPPER_MOTOR = 0x2;               // -- 010 --- ---
        /// <summary>
        /// Integer representation of a Proportional Pneumatic node type number.
        /// </summary>
        public const int PNEUMATIC_PROPORTIONAL = 0x4;      // -- 100 --- ---
        /// <summary>
        /// Integer representation of a 2 Way Pneumatic node type number.
        /// </summary>
        public const int PNEUMATIC_TWO_WAY = 0x3;           // -- 011 --- ---
        /// <summary>
        /// Integer representation of a Sensor node type number.
        /// </summary>
        public const int SENSOR = 0x5;                      // -- 101 --- ---

        // #####################################################################
        // Constructors

        public Node( int NodeType, int NodeID, String Name )
        {
            Initialise( NodeType, NodeID, Name, "" );
        }

        // ---------------------------------------------------------------------

        public Node( int NodeType, int NodeID, String Name, String Description )
        {
            Initialise( NodeType, NodeID, Name, Description );
        }

        // ---------------------------------------------------------------------

        public Node( XmlElement node )
        {
            String nodeName = node.GetAttribute( "name" );
            String nodeType = node.GetAttribute( "type" );

            int nodeNumber;
            try
            {
                nodeNumber = Int16.Parse( node.GetAttribute( "number" ) );
            }
            catch ( Exception )
            {
                throw new Exception(
                    "Could not parse node 'number' from XML: \n" + node.OuterXml );
            }
```

```
            String nodeDesc = null; //node.GetAttribute( "description" );

            int ntn;
            switch ( nodeType )
            {
                case "Master":
                    ntn = Node.MASTER_NODE;
                    break;
                case "DCMotor":
                    ntn = Node.DC_MOTOR;
                    break;
                case "StepperMotor":
                    ntn = Node.STEPPER_MOTOR;
                    break;
                case "ProportionalPneumatic":
                    ntn = Node.PNEUMATIC_PROPORTIONAL;
                    break;
                case "2WayPneumatic":
                    ntn = Node.PNEUMATIC_TWO_WAY;
                    break;
                case "Sensor":
                    ntn = Node.SENSOR;
                    break;
                default:
                    throw new Exception( "Unknown NodeType: " + nodeType );
            }

            if ( nodeDesc == null )
                Initialise( ntn, nodeNumber, nodeName, "" );
            else
                Initialise( ntn, nodeNumber, nodeName, nodeDesc );

            //MessageBox.Show( "Node Init" );

            foreach ( XmlNode port in node.SelectNodes( "port" ) )
            {
                Port newPort = new Port( ( XmlElement )( port ) );
                ports.Add( newPort.Name, newPort );
            }
        }

        // ####################################################################
        // Property accessors

        /// <summary>
        /// Gets the node name.
        /// </summary>
        public String Name
        {
            get { return name; }
        }

        // --------------------------------------------------------------------

        /// <summary>
        /// Gets the node description
        /// </summary>
        public String Description
        {
            get { return description; }
            set { description = value; }
        }

        // --------------------------------------------------------------------

        /// <summary>
        /// Gets the node type number.
        /// </summary>
        public int NodeTypeNumber
        {
            get { return nodetypenumber; }
        }

        // --------------------------------------------------------------------

        /// <summary>
        /// Gets the node ID number
        /// </summary>
        public int NodeIDNumber
        {
            get { return nodeIDnumber; }
        }

        /// <summary>
        /// Determines if this node it up to date (any ports set changed)
        /// </summary>
        public bool UpToDate
```

```
            {
                get { return uptodate; }
            }

            // ####################################################################
            // Private methods

            private void Initialise(
                int nodeType, int nodeID, String name, String description )
            {
                if ( nodeType >= 0 && nodeType <= 7 )
                    nodetypenumber = nodeType;
                else
                    throw new Exception(
                        "Invalid NodeTypeNumber: " + nodeType.ToString( ) );

                if ( nodeID >= 0 && nodeID <= 7 )
                    nodeIDnumber = nodeID;
                else
                    throw new Exception(
                        "Invalid NodeIDNumber: " + nodeID.ToString( ) );

                if ( name == null || name.Contains( " " ) || name.Contains( "." ) )
                    throw new Exception(
                        "Node NAME cannot be null or contain spaces or periods." );

                this.name = name;
                this.description = description;
                ports = new Dictionary<string, Port>( );
            }

            // ####################################################################
            // Public methods

            /// <summary>
            /// Returns a status request message to be sent to the node.
            /// </summary>
            /// <returns></returns>
            public CANPacket StatusMessage( )
            {
                uptodate = false;
                CANPacket msg = new CANPacket( );
                msg.Broadcast = false;
                msg.DirectedToMaster = ( NodeTypeNumber == MASTER_NODE );
                msg.NodeTypeNumber = NodeTypeNumber;
                msg.NodeIDNumber = NodeIDNumber;
                msg.MessageType = CANPacket.MESSAGE_TYPE_STATUS_REQUEST;
                msg.DataLength = 0;
                return msg;
            }

            // --------------------------------------------------------------------

            /// <summary>
            /// Gets a specific port value.
            /// </summary>
            /// <param name="pName">Port name</param>
            /// <returns></returns>
            public int GetValue( String pName )
            {
                Port p = GetPort( pName );
                if ( p == null )
                    throw new Exception( "Port '" + pName + "' not found." );
                else
                    return p.Value;
            }

            // --------------------------------------------------------------------

            /// <summary>
            /// Sets a specific port value
            /// </summary>
            /// <param name="pName">Port to be set</param>
            /// <param name="val">value to be set</param>
            /// <returns>Message type required for this port</returns>
            public int SetValue( String pName, int val )
            {
                Port p = GetPort( pName );
                if ( p == null )
                    throw new Exception( "Port '" + pName + "' not found." );
                else
                {
                    p.Value = val;
                    return p.MessageType;
                }
```

```
        }

        // ---------------------------------------------------------------------

        /// <summary>
        /// Extracts individual port information from the message.
        /// </summary>
        /// <param name="msg">CAN status message</param>
        public void ParseMessage( CANPacket msg )
        {
            if ( msg.Broadcast == true ||
                ( msg.NodeTypeNumber == NodeTypeNumber &&
                msg.NodeIDNumber == NodeIDNumber ) )
            {
                foreach ( Port p in ports.Values )
                {
                    if ( msg.Broadcast == false )
                    {
                        // Only update ports with matching MT
                        if ( msg.MessageType == p.MessageType )
                            p.ExtractInfo( msg );
                    }
                    else
                    {
                        // Update all fields on for a broadcast message.
                        p.ExtractInfo( msg );
                    }
                }
                uptodate = true;
            }
        }

        // ---------------------------------------------------------------------

        /// <summary>
        /// Gets a CAN message of specific type.  Will return a CAN message
        /// with port information from all ports with matching 'type'.
        /// </summary>
        /// <param name="type">Message type to be sent</param>
        /// <returns></returns>
        public CANPacket GetMessage( int type )
        {
            CANPacket msg = new CANPacket( );
            foreach ( Port p in ports.Values )
            {
                if ( p.MessageType == 0 || p.MessageType == type )
                {
                    p.AddInfo( msg );
                }
            }
            msg.Broadcast = false;
            msg.DirectedToMaster = ( NodeTypeNumber == MASTER_NODE );
            msg.NodeTypeNumber = NodeTypeNumber;
            msg.NodeIDNumber = NodeIDNumber;
            msg.MessageType = type;
            return msg;
        }

        // ---------------------------------------------------------------------

        /// <summary>
        /// Returns true if the port has received a status message from the bus.
        /// </summary>
        /// <returns></returns>
        public bool IsOnline( )
        {
            return online;
        }

        // ---------------------------------------------------------------------

        /// <summary>
        /// Returns a list of port names attached to this node.
        /// </summary>
        /// <returns></returns>
        public String[ ] PortNames( )
        {
            Dictionary<String, Port>.KeyCollection keys = ports.Keys;

            if ( keys.Count == 0 )
                return null;

            String[ ] arr = new String[ keys.Count ];
            int i = 0;
```

```
            foreach ( String s in keys )
            {
                arr[ i ] = s;
                i++;
            }
            return arr;
        }

        // -------------------------------------------------------------------

        /// <summary>
        /// Returns the number of ports attached to this node.
        /// </summary>
        /// <returns></returns>
        public int NumPorts( )
        {
            return ports.Count;
        }

        // -------------------------------------------------------------------

        /// <summary>
        /// Returns a specific port
        /// </summary>
        /// <param name="name"></param>
        /// <returns></returns>
        public Port GetPort( String name )
        {
            Port temp;
            if ( !ports.TryGetValue( name, out temp ) )
                return null;
            return temp;
        }

        // -------------------------------------------------------------------

        /// <summary>
        /// Returns the entire port list.
        /// </summary>
        /// <returns></returns>
        public Dictionary<String, Port> GetAllPorts( )
        {
            return ports;
        }

        // -------------------------------------------------------------------

        /// <summary>
        /// Adds a port to the node.
        /// </summary>
        /// <param name="p"></param>
        public void AddPort( Port p )
        {
            ports.Add( p.Name, p );
        }

        // -------------------------------------------------------------------

        /// <summary>
        /// String representation of the node.
        /// </summary>
        /// <returns></returns>
        public override string ToString( )
        {
            StringBuilder str = new StringBuilder( );
            str.AppendLine(
                "Node: " + Name +
                ", NTN: " + NodeTypeNumber +
                ", NID: " + NodeIDNumber );

            //MessageBox.Show( str.ToString( ) );
            foreach ( Port p in ports.Values )
            {
                str.AppendLine( "  " + p.ToString( ) );
            }
            str.AppendLine( );
            //MessageBox.Show( str.ToString( ) );
            return str.ToString( );
        }

    } // class Node

} // namespace ModularRobot
```

C.9 Port.cs

Listing C.8: Port class source code.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;
using System.Xml;

namespace ModularRobot
{
    /// <summary>
    /// Holds current information about a physical port on a node.
    /// </summary>
    public class Port
    {
        // -------------------------------------------------------------------
        // Private data fields

        private BitFilter parser;
        private String name;
        private String description;
        private int messageType;
        private int currentValue;
        private int setValue;
        private bool changed = false;

        // -------------------------------------------------------------------
        // Public data fields / constants

        // ###################################################################
        // Constructors

        public Port( String name, BitFilter bf, int MessageType )
        {
            Initialise( name, bf, MessageType, "" );
        }

        // -------------------------------------------------------------------

        public Port(
            String name, BitFilter bf, int MessageType, String description )
        {
            Initialise( name, bf, MessageType, description );
        }

        // -------------------------------------------------------------------

        /// <summary>
        /// Creates a new port from an XML string.
        /// </summary>
        /// <param name="port"></param>
        public Port( XmlElement port )
        {
            String portName = port.GetAttribute( "name" );
            String portDesc = null;// = port.GetAttribute( "description" );
            String portFilter = port.GetAttribute( "filter" );
            //MessageBox.Show( "Port Init" + portName);
            int portMT;
            try
            {
                portMT = Int16.Parse( port.GetAttribute( "messagetype" ) );
            }
            catch ( Exception )
            {
                throw new Exception(
                    "Could not parse messagetype from XML: \n" + port.OuterXml );
            }

            BitFilter bf = new BitFilter( portFilter );

            if ( portDesc == null )
                Initialise( portName, bf, portMT, "" );
            else
                Initialise( portName, bf, portMT, portDesc );

            //MessageBox.Show( "Port created: " + portName );

        }

        // ###################################################################
        // Property accessors

        /// <summary>
```

```
        /// Returns the message type used by this port.
        /// </summary>
        public int MessageType
        {
            get { return messageType; }
        }

        // -------------------------------------------------------------------

        /// <summary>
        /// Returns the port name.
        /// </summary>
        public String Name
        {
            get { return name; }
        }

        // -------------------------------------------------------------------

        /// <summary>
        /// Returns the port description
        /// </summary>
        public String Description
        {
            get { return description; }
        }

        // -------------------------------------------------------------------

        /// <summary>
        /// Returns the BitFilter used by this port
        /// </summary>
        public BitFilter Filter
        {
            get { return parser; }
        }

        // -------------------------------------------------------------------

        /// <summary>
        /// Gets the current value of the port
        /// </summary>
        public int CurrentValue
        {
            get
            {
                changed = false;
                return currentValue;
            }
        }

        // -------------------------------------------------------------------

        /// <summary>
        /// Sets the port value and accesses the current port value.
        /// </summary>
        public int SetValue
        {
            get { return setValue; }
            set
            {
                setValue = value;
                changed = true;
            }
        }

        // -------------------------------------------------------------------

        /// <summary>
        /// Sets a new port value and returns the current port value
        /// </summary>
        public int Value
        {
            get { return CurrentValue; }
            set { SetValue = value; }
        }

        // -------------------------------------------------------------------

        /// <summary>
        /// Determines whether the port has been modified by a status message
        /// </summary>
        public bool Changed
        {
            get { return changed; }
        }
```

```
        // ####################################################################
        // Private methods

        /// <summary>
        /// Initialises the port and checks all input values
        /// </summary>
        /// <param name="name">Port Name</param>
        /// <param name="bf">BitFilter structure to use</param>
        /// <param name="mt">Message type used by this port</param>
        /// <param name="desc">Short description</param>
        private void Initialise(
            String name, BitFilter bf, int mt, String desc )
        {
            if ( name == null || name.Contains( " " ) || name.Contains( "." ) )
                throw new Exception(
                    "Port NAME cannot be null or contain spaces or periods." );
            if ( bf == null )
                throw new Exception(
                    "BitFilter cannot be NULL in port initialisation: " + name );
            if ( mt < 0 && mt > 7 )
                throw new Exception( "Message type must be between 0 and 7." );

            this.name = name.ToLower( );
            this.parser = bf;
            this.messageType = mt;
            this.description = desc;
            currentValue = 0;
            setValue = currentValue;
        }

        // ####################################################################
        // Public methods

        /// <summary>
        /// Passes the BitFilter over the given status message and updates the
        /// current information.
        /// </summary>
        /// <param name="msg"></param>
        public void ExtractInfo( CANPacket msg )
        {
            currentValue = parser.ParseValue( msg );
            if ( currentValue == setValue )
                changed = false;
        }

        // --------------------------------------------------------------------

        /// <summary>
        /// Adds the port's SetValue to the CAN packet to be sent
        /// </summary>
        /// <param name="msg">CAN message to be sent.</param>
        public void AddInfo( CANPacket msg )
        {
            parser.AddValue( msg, setValue );
        }

        // --------------------------------------------------------------------

        /// <summary>
        /// Gets a string representation of the port.
        /// </summary>
        /// <returns></returns>
        public override string ToString( )
        {
            String str =
                "Port: " + Name +
                ", MT: " + MessageType +
                ", Val: " + CurrentValue +
                ", BF: " + Filter.ToString( );
            //MessageBox.Show( str );
            return str;
        }

    } // public class Port

} // namespace ModularRobot
```

Appendix D

Sample Program

D.1 Overview

This appendix contains the configuration file, test program and output log file for the Modular Robot. The program shows the current functionality of the distributed controllers and how it is harnessed by the communication interface.

D.2 Sample XML Configuration File

Listing D.1: A clever MATLAB function.

```
<?xml version = "1.0"?>

<!-- A prototype robot description -->
<!-- robotCAN.xml -->

<!-- Root node is the type of interface used to communicate with the robot. -->
  <CAN
    type="LawicelCANUSB"
    firmware="0.0.14"
    baudrate = "125"
    readinterval = "5"
    writeinterval = "5"
    readtimeout = "500"
  >
    <!-- CAN Node configuration -->
<!--
    <node
        name = "nod1"
        type = "Master"
        number = "0"
        description = "This type of node is not supported."
    ></node>
-->
    <node
        name = "DC0"
        type = "DCMotor"
        number = "0"
        description = "Prototype DC Motor module"
    >
      <port
          name = "speed"
          description = "Accesses the speed mode of the DC motor"
          filter = "B1:S0:L8"
          messagetype = "0"
      />
      <port
          name = "direction"
          description = "Accesses the direction field of the DC motor"
          filter = "B2:S0:L8"
          messagetype = "1"
      />
      <port
          name = "position"
          description = "Accesses the position mode of the DC motor"
          filter = "B3:S0:L8"
          messagetype = "2"
      />
      <port
          name = "auto"
          description = ""
          filter = "B0:S7:L1"
          messagetype = "3"
      />

    </node>

    <node
        name = "STEP0"
        type = "StepperMotor"
        number = "0"
        description = ""
    >
      <port
          name = "position"
          description = "Accesses the position mode of the DC motor"
          filter = "B3:S0:L8"
          messagetype = "2"
      />
      <port
          name = "speed"
          description = "Accesses the speed mode of the DC motor"
          filter = "B1:S0:L8"
          messagetype = "0"
      />
      <port
```

```
            name = "direction"
            description = "Accesses the direction field of the DC motor"
            filter = "B2:S0:L8"
            messagetype = "1"
        />
        <port
            name = "auto"
            description = ""
            filter = "B0:S7:L1"
            messagetype = "3"
        />
      </node>

      <node
          name = "PP0"
          type = "ProportionalPneumatic"
          number = "0"
          description = "Prototype proportional pnuematic module"
      >
        <port
            name = "valve1"
            description = "Valve control 1"
            filter = "B1:S0:L1"
            messagetype = "1"
        />
        <port
            name = "valve2"
            description = "Valve control 2"
            filter = "B1:S1:L1"
            messagetype = "1"
        />
        <port
            name = "valve3"
            description = "Valve control 3"
            filter = "B1:S2:L1"
            messagetype = "1"
        />
        <port
            name = "valve4"
            description = "Valve control 4"
            filter = "B1:S3:L1"
            messagetype = "1"
        />
        <port
            name = "valveAll"
            description = "All valves"
            filter = "B1:S0:L4"
            messagetype = "1"
        />
      </node>

      <node
          name = "P2W0"
          type = "2WayPneumatic"
          number = "0"
          description = "Prototype 2 way pneumatic module"
      >
        <port
            name = "valve1"
            description = "Valve control 1"
            filter = "B1:S0:L1"
            messagetype = "1"
        />
        <port
            name = "valve2"
            description = "Valve control 2"
            filter = "B1:S1:L1"
            messagetype = "1"
        />
        <port
            name = "valve3"
            description = "Valve control 3"
            filter = "B1:S2:L1"
            messagetype = "1"
        />
        <port
            name = "valve4"
            description = "Valve control 4"
            filter = "B1:S3:L1"
```

```
            messagetype = "1"
        />
        <port
            name = "valveAll"
            description = "All valves"
            filter = "B1:S0:L4"
            messagetype = "1"
        />
    </node>

    <node
        name = "SENS0"
        type = "Sensor"
        number = "0"
        description = "Prototype sensor module"
    >
        <port
        name = "analoge1"
        description = ""
        filter = "B1:S0:L8"
        messagetype = "1"
        />
    </node>

  </CAN>
```

D.3 Sample Program Listing

Listing D.2: A clever MATLAB function.

```
using System;
using System.Threading;

namespace ModularRobot
{
    public static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main( )
        {

            MRModel robot = new MRModel( );
            robot.ConfigFileName = "./robotCAN.xml";
            if ( robot.Open( ) < 0 )     // Ensure device is operational
                return;

            Console.WriteLine(
                "MRModel Initialised.\nConfig: " + robot.ConfigFileName );

            // ------------------------------------------------------------

            robot.Set( "PP0", "valveall", 0 );
            robot.Set( "P2W0", "valveall", 0 );

            for ( int i = 1; i <= 8; i = i * 2 )
            {
                robot.Set( "PP0", "valveall", i );
                robot.Set( "P2W0", "valveall", i );

                if ( robot.Get( "PP0", "valveall" ) != i )
                    Console.WriteLine( "Error PP0 Value: " + i );
                if ( robot.Get( "P2W0", "valveall" ) != i )
                    Console.WriteLine( "Error P2W0 Value: " + i );
                Thread.Sleep( 1000 );
            }

            Console.WriteLine(
                "PP0 valve1: " + robot.Get( "PP0", "valve1" ).ToString( ) );
            Console.WriteLine(
                "PP0 valve2: " + robot.Get( "PP0", "valve2" ).ToString( ) );
            Console.WriteLine(
                "PP0 valve3: " + robot.Get( "PP0", "valve3" ).ToString( ) );
            Console.WriteLine(
                "PP0 valve4: " + robot.Get( "PP0", "valve4" ).ToString( ) );

            // ------------------------------------------------------------

            robot.Set( "DC0", "speed", 1 );
            Random rand = new Random( );
            for ( int i = 0; i < 3; i++ )
            {
                int setpos = rand.Next( 0xFF );
                Console.WriteLine(
                    "Setting DC0 position: " + setpos.ToString( ) );
                robot.Set( "DC0", "position", setpos );
                Thread.Sleep( 2000 );
                Console.WriteLine(
                    "Final DC0 position: " + robot.Get( "DC0", "position" ) );
            }

            // ------------------------------------------------------------

            Console.WriteLine(
                "Sensor analoge1 value: " + robot.Get( "SENS0", "analoge1" ) );

            // ------------------------------------------------------------

            robot.Close( );

```

```
            }
        }
}
```

D.4 Console Output

```
MRModel Initialised.

Config: ./robotCAN.xml

PP0 valve1: 0

PP0 valve2: 0

PP0 valve3: 0

PP0 valve4: 1

Setting DC0 position: 117

Final DC0 position: 117

Setting DC0 position: 66

Final DC0 position: 66

Setting DC0 position: 211

Final DC0 position: 151

Sensor analoge1 value: 9
```

D.5 Run-Time Log

The following is a enhanced version of the log file produced by the program. The standard log file output generates the Timestamp and Log Message columns, however the Delay and Message purpose columns were generated by importing the log file into an Excel spreadsheet and formatting it so that it was more intuitive to the reader.

Log Created: 26/10/2006 10:26:57 PM

Time (hh:mm:ss) (ms)	Delay (s)	Log Message	*Message purpose*
09:10:12 . 609	0.000	CANUSB Initialised	*TX & RX queues ready*
09:10:12 . 625	0.016	Enqueue: 10120000	*Command to Proportional Pneumatic*
09:10:12 . 625	0.000	Enqueue: 0C120000	*Command to 2 Way Pneumatic*
09:10:12 . 625	0.000	Enqueue: 10120001	*Command to Proportional Pneumatic*
09:10:12 . 625	0.000	Enqueue: 0C120001	*Command to 2 Way Pneumatic*
09:10:12 . 625	0.000	Enqueue: 1000	*Status request - Proportional Pneumatic*
09:10:12 . 625	0.000	TX: 101 2 0000	*Transmit*
09:10:12 . 640	0.015	TX: 0C1 2 0000	
09:10:12 . 656	0.016	TX: 101 2 0001	
09:10:12 . 671	0.015	TX: 0C1 2 0001	
09:10:12 . 687	0.016	TX: 100 0 0	*Status request transmit - Proportional Pneumatic*
09:10:12 . 703	0.016	RX: 501 8 2001223344556620	*Status request reply - Proportional Pneumatic*
09:10:12 . 703	0.000	Dequeue: 50182001223344556620	*Queue read*
09:10:12 . 703	0.000	Enqueue: 0C00	*Status request - 2 Way Pneumatic*
09:10:12 . 718	0.015	TX: 0C0 0 0	*Status request transmit - 2 Way Pneumatic*
09:10:12 . 734	0.016	RX: 4C1 8 2001223344556618	*Status request reply - 2 Way Pneumatic*
09:10:12 . 734	0.000	Dequeue: 4C182001223344556618	*Queue read*
09:10:13 . 734	1.000	Enqueue: 10120002	*Next value to Proportional Pneumatic*
09:10:13 . 734	0.000	Enqueue: 0C120002	*Next value to 2 Way pneumatic*
09:10:13 . 734	0.000	Enqueue: 1000	*Status request - Proportional Pneumatic*
09:10:13 . 750	0.016	TX: 101 2 0002	
09:10:13 . 765	0.015	TX: 0C1 2 0002	
09:10:13 . 781	0.016	TX: 100 0 0	
09:10:13 . 796	0.015	RX: 501 8 2002223344556620	
09:10:13 . 796	0.000	Dequeue: 50182002223344556620	
09:10:13 . 796	0.000	Enqueue: 0C00	
09:10:13 . 812	0.016	TX: 0C0 0 0	
09:10:13 . 828	0.016	RX: 4C1 8 2002223344556618	
09:10:13 . 828	0.000	Dequeue: 4C182002223344556618	
09:10:14 . 828	1.000	Enqueue: 10120004	
09:10:14 . 828	0.000	Enqueue: 0C120004	
09:10:14 . 828	0.000	Enqueue: 1000	
09:10:14 . 843	0.015	TX: 101 2 0004	
09:10:14 . 859	0.016	TX: 0C1 2 0004	
09:10:14 . 875	0.016	TX: 100 0 0	
09:10:14 . 890	0.015	RX: 501 8 2004223344556620	
09:10:14 . 890	0.000	Dequeue: 50182004223344556620	
09:10:14 . 890	0.000	Enqueue: 0C00	
09:10:14 . 906	0.016	TX: 0C0 0 0	
09:10:14 . 921	0.015	RX: 4C1 8 2004223344556618	
09:10:14 . 921	0.000	Dequeue: 4C182004223344556618	
09:10:15 . 921	1.000	Enqueue: 10120008	
09:10:15 . 921	0.000	Enqueue: 0C120008	
09:10:15 . 921	0.000	Enqueue: 1000	
09:10:15 . 937	0.016	TX: 101 2 0008	
09:10:15 . 953	0.016	TX: 0C1 2 0008	
09:10:15 . 968	0.015	TX: 100 0 0	
09:10:15 . 984	0.016	RX: 501 8 2008223344556620	
09:10:15 . 984	0.000	Dequeue: 50182008223344556620	
09:10:15 . 984	0.000	Enqueue: 0C00	
09:10:16 . 000	0.016	TX: 0C0 0 0	
09:10:16 . 015	0.015	RX: 4C1 8 2008223344556618	
09:10:16 . 015	0.000	Dequeue: 4C182008223344556618	
09:10:17 . 015	1.000	Enqueue: 1000	
09:10:17 . 031	0.016	TX: 100 0 0	

09:10:17 . 046	0.015	RX: 501 8 2008223344556620	
09:10:17 . 046	0.000	Dequeue: 50182008223344556620	
09:10:17 . 046	0.000	Enqueue: 1000	
09:10:17 . 062	0.016	TX: 100 0 0	
09:10:17 . 078	0.016	RX: 501 8 2008223344556620	
09:10:17 . 078	0.000	Dequeue: 50182008223344556620	
09:10:17 . 078	0.000	Enqueue: 1000	
09:10:17 . 093	0.015	TX: 100 0 0	
09:10:17 . 109	0.016	RX: 501 8 2008223344556620	
09:10:17 . 109	0.000	Dequeue: 50182008223344556620	
09:10:17 . 109	0.000	Enqueue: 1000	
09:10:17 . 125	0.016	TX: 100 0 0	
09:10:17 . 140	0.015	RX: 501 8 2008223344556620	
09:10:17 . 140	0.000	Dequeue: 50182008223344556620	
09:10:17 . 156	0.016	Enqueue: 042400010075	*Command to DCMotor*
09:10:17 . 156	0.000	TX: 042 4 00010075	
09:10:19 . 156	2.000	Enqueue: 0400	
09:10:19 . 156	0.000	TX: 040 0 0	
09:10:19 . 171	0.015	RX: 441 8 20FF017580556608	
09:10:19 . 171	0.000	Dequeue: 441820FF017580556608	
09:10:19 . 171	0.000	Enqueue: 042400010042	
09:10:19 . 187	0.016	TX: 042 4 00010042	
09:10:20 . 609	1.422	ERROR: 'Bus Error'	*Occasional bus error acceptable*
09:10:21 . 171	0.562	Enqueue: 0400	
09:10:21 . 187	0.016	TX: 040 0 0	
09:10:21 . 203	0.016	RX: 441 8 20FF024280556608	
09:10:21 . 203	0.000	Dequeue: 441820FF024280556608	
09:10:21 . 203	0.000	Enqueue: 0424000100D3	
09:10:21 . 218	0.015	TX: 042 4 000100D3	
09:10:23 . 203	1.985	Enqueue: 0400	
09:10:23 . 218	0.015	TX: 040 0 0	
09:10:23 . 234	0.016	RX: 441 8 2001029780556608	
09:10:23 . 234	0.000	Dequeue: 44182001029780556608	
09:10:23 . 234	0.000	Enqueue: 1400	*Status request to Sensor*
09:10:23 . 250	0.016	TX: 140 0 0	
09:10:23 . 265	0.015	RX: 541 8 2009223344556628	
09:10:23 . 265	0.000	Dequeue: 54182009223344556628	
09:10:23 . 265	0.000	Closing CANUSB.	*Preparing to close the CANUSB device*
09:10:24 . 265	1.000	Closed OK	*All buffers empty, device closed OK*