IMPLEMENTATION OF A HEALTH MONITORING SYSTEM WITHIN A FAULT TOLERANT STEER-BY-WIRE SYSTEM

by

MO LI

B.Sc., Beijing University of Aeronautics and Astronautics, 2004

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Mechanical Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

July, 2007

Abstract

The objective of this thesis is to examine the improvement of the design and implementation of a fault tolerant Steer-by-Wire system. This application integrates the methods of multi-level redundancy developed in previous work with a tool set for monitoring the health of the computer controlled system. It allows the system to tolerate faults at all levels of its organization.

The health monitoring system adopted is based on a simplified representation of the dynamic model of the Steer-by-Wire application. By comparing the output of a system model to the actual output achieved by the application, it provides three measures of component health, all of which indicate errors based on the performance of a system state relative to a base model. The analytical redundancy provided by the health monitoring system may be used to either reduce overall system cost through replacement of a physical sensor, or as an additional sensor to allow continued voting after the first failure of a physical sensor thus avoiding significant system degradation.

The health monitoring system has been implemented on a prototype of a three-wheeled vehicle. The vehicle has two "independent" steering systems: the first resembles a conventional power assist, the second a Steer-by-Wire configuration.

Table of Contents

List of Tables

List of Figures

List of Symbols and Acronyms

$u(t)$	current system output state with respect to time, t
$y(t)$	estimate of the system output state at time, t
Δ	discrete time interval
h_d	system dynamics health indicator
h_i	instantaneous error health indicator
h_a	average state change health indicator
ω_n	closed position loop natural frequency
ζ	damping ratio
$K_{lead,lag}$	lead/lag filter gain.
a	lead parameter in continuous time
b	lag parameter in continuous time
A	lead parameter in discrete time
B	lag parameter in discrete time
k_{ref}	reference gain
k_{loop}	loop gain of adjustable drive amplifier velocity control
k_{tach}	tachometer gain of adjustable drive amplifier velocity control
τ_{vel}	first order transfer function time constant
T_s	sampling time
T_1	torque output from the first drive
T_2	torque output from the second drive
K_T	motor torque constant
K_A	amplifier gain
K_{torque}	torque loop gain
F_{load}	steering rack load
F_y	lateral tire force
F_z	vertical tire force

α	tire slip angle
$C_{F\alpha}$	cornering stiffness
F_{zT}	rated tire load
$C_{T\alpha}$	alignment stiffness
l_t	tire contact length
M_z	self-aligning momentum
M_w	wheel turning momentum
J_w	wheel inertia
a_w	steering rack acceleration
l_c	distance between wheel centre and pivot

ARI	Average Response Indicator
CAN	Controller Area Network
CPU	Central Processing Unit
DRI	Dynamics Response Indicator
ECU	Electronic Control Unit
FSU	Fail-Silent Unit
FTU	Fault-Tolerant Unit
IRI	Instantaneous Response Indicator
PWM	Pulse Width Modulated
SCM	Smart Control Moduler
TMDO	Digital Output Module
TMDI	Digital Input Module
TMAO	Analog Output Module
TMAI	Analog Iutput Module
TMAC	FPGA Based Motion Control Module

Acknowledgements

First, I would like to thank Dr. Ian Yellowley, my supervisor, for his continuous support and invaluable advice during the completion of this work. Many of the ideas in this thesis were developed at his suggestion. His expertise and constant guidance have led me through the M.A.Sc program and made it a rewarding academic experience of my life. And without his financial assistance, my dream of studying overseas would not have been possible.

I would also like to thank my second reader, Paul Winkelman, for taking the time to proofread and give helpful comments on a very short notice.

A special thanks to Green College, and its founder Dr. Cecil H. Green, for the amazing accommodation and fabulous community. The friendships with people having diverse backgrounds and interests are my precious asset forever.

I am grateful to my parents for always standing behind me with their love whenever I was depressed or happy. Their encouragement was indispensable to my dream pursuing. They have been like a beacon of light letting me see the happiness on the road to the future.

Chapter 1: Introduction

1.2 Safety Critical Function

1.2.1 Introduction

A safety critical function is a function whose failure necessarily causes unsafe operating conditions; any unhandled error will lead to elevated risk to the system, its users, or environs. For example, unanticipated catastrophic failure of a car's steering system will cause a loss of vehicle control leading to the possibility of accidents, injuries and damage. A safety–critical system must ensure that system functions deemed safety critical have no possibility of failure.

A number of techniques are used to achieve service dependability. During the design phase, the engineer can design to fault avoidance. This involves using various design methodologies, modelling, and validation techniques to prevent fault from being incorporated in the initial design. Then the designer looks to fault removal techniques. This involves system testing or prototyping to discover faults, which are then removed from the design.

Once a system prototype is produced, the engineer can implement fault tolerance techniques. This means the inclusion of fail-operational behaviour within the device by having control that can detect, diagnose, treat and recover from errors that occur. This also

includes fault detection and fault diagnosis mechanisms as well as protection, supervision and appropriate safety actions.

1.2.2 Passive / Active Safety System

While a number of potential strategies for improving application safety exist, generally strategies can be classified as either active or passive safety systems.

Passive safety systems operate by attempting to minimize harm caused by a failure. These systems attempt to make the system react to failure in a particular way, or constrain errors in a particular way. Examples of passive safety systems include seatbelts, compound glass, airbags, and surge protectors.

Active safety systems change system operating-conditions to reduce the probability of failure. Examples of active safety systems include ABS, Electronic Traction Control, and Electronic Stability Protection.

In brief then, a passive safety system seeks to manage the results of failure, whereas an active system seeks to reduce the incidence of failure.

1.2.3 Dependability

Safety critical functions have to be dependable regardless of implementation. Dependability is that property of a computing system which allows reliance to be justifiably placed on the service it delivers. It is usually expressed in terms of other measures such as safety, reliability and availability. Laprie [1] defines these terms as follows:

- **Safety** is a measure of the continuous delivery of service free from occurrences of catastrophic failures.

- **Reliability** is a measure of the continuous delivery of proper service.
- **Availability** is a measure of the readiness for correct service.

1.3 Recent Developments of Steering Systems

The proliferation of electronic control systems is nowhere more apparent than in the modern automobile. Automotive systems are increasingly being designed with integrated electronic sensors, actuators, microcomputers, information processing for single component, and engine, drivetrain, suspension, and brake systems. During the last two decades, advances in electronics have revolutionized many aspects of automotive engineering, especially in the areas of engine combustion management and vehicle safety systems such as anti-lock brakes (ABS) and electronic stability control (ESC). However, only recently has the electronic revolution begun to find its way into automotive steering systems in the form of electronically controlled variable assist and, within the past two years, fully electric power assisted steering systems.

1.3.1 Conventional Steering System

The basic design of automotive steering systems has changed little since the invention of the steering wheel: a conventional steering system as shown in Figure 1.1 [2] typically consists of the handwheel (steering wheel), the steering column, intermediate shaft, rotary spool valve (an integral part of the hydraulic power assist system), the rack and pinion, and the steering linkages. Since the steering column and pinion are almost never collinear, they are joined to the intermediate shaft via two universal joints matched to minimize torque and speed variations between steering column and pinion.

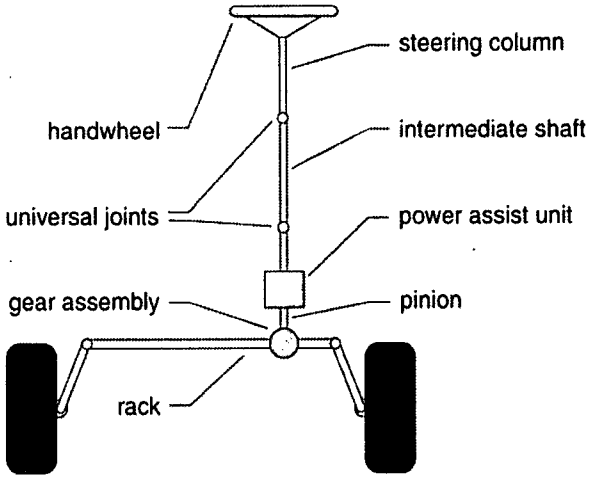


Figure 1.1: Conventional Steering System

1.3.2 Power Steering

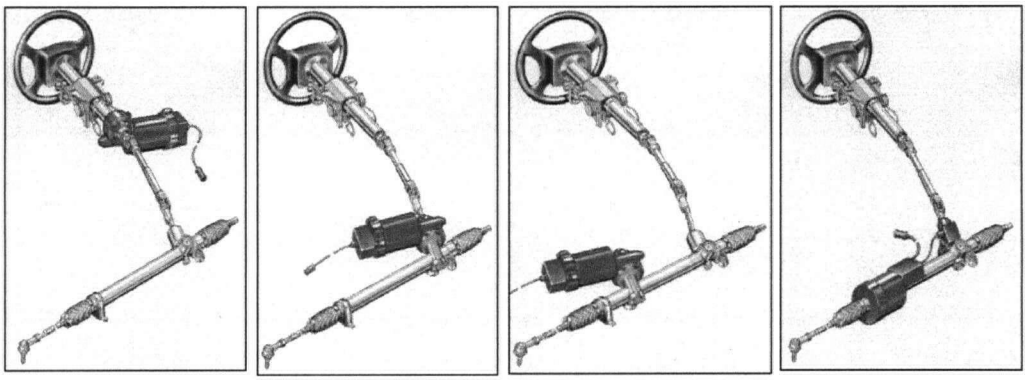
Power steering is a system for reducing the steering effort by using an external power source to assist in turning the wheels. Power steering technology was first introduced to the mass market in the 1950's and has continued to use hydraulic systems. These systems have achieved a remarkable performance in regard to ride and handling, cost and comfort and power assist has become a standard component in modern automotive steering systems. Using hydraulic pressure supplied by an engine-driven pump, power steering amplifies and supplements the driver-applied torque at the steering wheel so that steering effort is reduced. In addiction to improved comfort, reducing steering effort has important safety implications as well, such as allowing a driver to more easily swerve to avoid an accident.

Electric Power Steering arrived first on small cars in Europe in the mid-1990s, and is now found on cars such as the Chevrolet Cobalt, Acura NSX, Saturn VUE V6, and on most FIAT and Lancia cars. The basic system uses sensors to detect the motion and torque applied to the steering column and a computer module to generate torque commands to an amplifier motor system based on position error and torque input [3].

Compared to hydraulic systems, electric systems are significantly more efficient, (because the hydraulic pump used in conventional systems is usually running constantly), and this is the main reason for their introduction. The assist level is also easily tunable to the vehicle type, road speed and even driver preference [4]. An added benefit is the elimination of the environmental hazard posed by leakage and disposal of hydraulic power steering fluid. Due to all these advantages, electric power steering likely to gain a considerable proportion of the power steering system market.

Electric power steering is currently limited to smaller vehicles. This is because of the 12 volt electrical system and typical current limits of around 80 amps. Larger vehicles such as trucks and SUVs require a larger power output than the 1 Kw currently available. A new 42 volt electrical system standard should enable use of electric power steering on larger vehicles.

Four basic variants of electrical steering system are being developed and can be separated into categories based on the location of the electric motor that provides steering assistance [5]. Each of the four solutions focuses on one type of operation, drives of steering column, pinion, double pinion (gear, double gear) and rack Figure 1.2.



Motor at Column Motor at Pinion Motor with Double-Pinion Motor at Rack

Figure 1.2: Four Types of Electric Power Steering Configurations

1.3.3 Steer By Wire

The Next step in steering system evolution is termed "Steer-by-Wire" and involves the complete removal of the steering column and shaft. This represents a dramatic departure from traditional automotive design practice. The term "By-wire" refers to the lack of physical connection between the steering wheel and the steering mechanism, brake pedal and the brake actuators or accelerator pedal and engine controller. This project focuses on the health monitoring of a Steer-by-Wire system. This idea is not new as many modern aircraft, both commercial and military, rely completely on fly-by-wire fight control systems [6].

The main difference between the conventional steering systems and the "Steer-by-Wire" systems is the connection between the steering wheel and the steering actuator. One may view the steering system as having two main subsystems:

1). Command input subsystem (steering wheel),

2). Steering power circuit (e.g., the pump, cylinder, and valves of electro-hydraulic power steering).

The conventional systems include different forms of mechanical and hydraulic connections between the steering wheel and the steering actuator. In simplest terms, the steering wheel rotation is proportionally amplified by the steering actuator to obtain a proportional articulation angle. Since the two systems are mechanically coupled, there is a built in inherent force feedback to the operator at the steering wheel proportional to the steering conditions. A "Steer-by-Wire" system has only electrical signal connection between the steering wheel sub-system and the steering power sub-systems and cannot provide "natural" force feedback.

A number of current production vehicles already employ by-wire technology for the throttle and brakes Figure 1.3 [7]. One of the main advantages of By-Wire systems is the opportunity for performance improvement. Using software, it is considerably easier to implement more flexible control strategies, which also allow for better tunability and performance of the control systems. As an example, in the conventional car steering combined with mechanical linkage and hydraulic power steering, the effort required to manoeuvre a car is inversely proportional to the vehicle's speed, and therefore turning the steering wheel for parking always requires more torque than that required in high-speed situation. With a Steer-by-Wire system, it is easy to incorporate the vehicle's speed into the control loop running in the embedded application, and thereby adjust the tactile force feedback to guarantee effortless use to the user. The absence of a steering column also greatly simplifies the design of car interiors. The absence of a steering column allows

better space utilization in the engine compartment. Without a direct mechanical connection between the steering wheel and the road wheels, noise, vibration, and harshness from the road no longer have a path to the driver's hands and arms through the steering wheel.

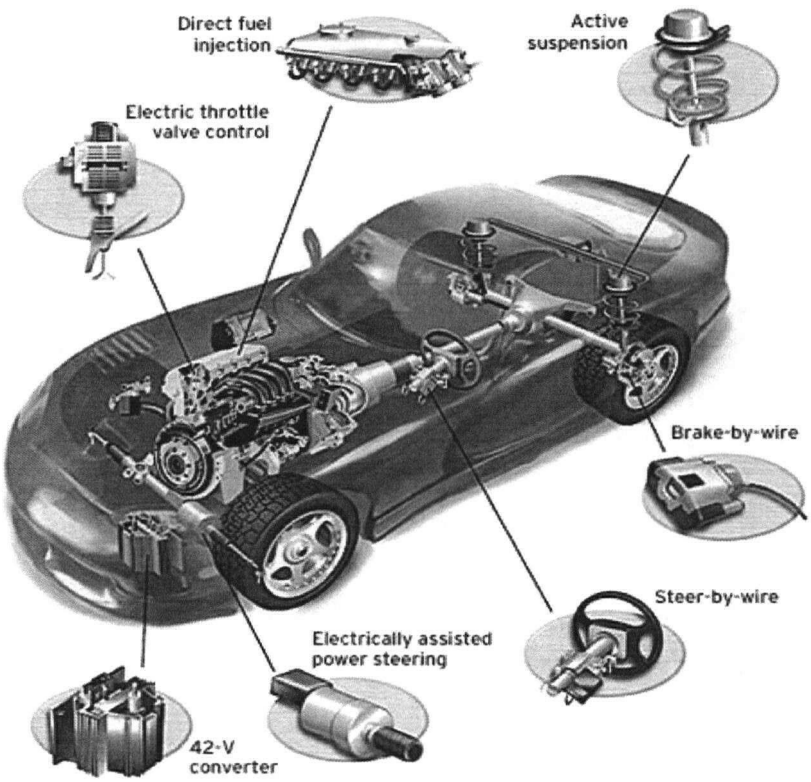


Figure 1.3: Automotive Applications for By-wire Technology. Source: Motorola

1.3.4 Challenge

Despite the potential benefits outlined in section above, the Steer-by-Wire concept also brings many challenges to the control engineer. Safety considerations and fault

tolerance become major issues. One of the risks of embedded computing systems is the binary failure mode of an electronic component: either it works, or it does not. The design of an integrated embedded system must consider the system's ability to cope with such faults. The greatest challenge in design is to guarantee that the system will continue to perform safely even if one or more components fail.

Existing technological solutions to the problem are dependent on high levels of redundancy and high costs correspondingly. This type of solution is not appropriate for the development of products in cost sensitive markets such as consumer devices, automobiles, and pleasure boats, nor is it ideal for industrial or military applications. Thus the special constraints of the automobile industry (mass production, certification rules etc.) have also to be taken into account.

1.4 Objective

The objective of this thesis is to examine the implementation of a health monitoring system and to develop this within a fault–tolerant Steer-by-Wire environment. The complete framework will then integrate multi–level redundancy, perform both local and distributed error detection, and allow the system to tolerate faults at all levels of its organization. With certain levels of redundancy of both hardware and software, each ECU could perform both local and distributed error detection, (in conjunction with its peers), to detect sensor, ECU, and communication errors.

The thesis introduces a Steer-by-Wire/Power Assist prototype steering system installed on an existing three-wheeled chassis. The dynamic model of the complete system is analyzed and used to track physical vehicle performance. The overall control of

the system is performed using a highly object-oriented software framework that incorporates data acquisition, control and communication functions.

The thesis examines the creation of the health monitoring system and use of this as an "analytical sensor". The idea behind the use of the analytical sensor is to provide an additional level of security to augment the normal 3 sensor voting routines. (Alternatively in very cost sensitive and non threatening applications one may wish to use this to reduce cost). The development and testing of an appropriate set of software measures to recognize different forms of error is also presented.

1.5 Outline

This thesis is organized into five chapters. Chapter 2 provides a review of previous research and development efforts in the design of fault–tolerant, safety–critical systems. The chapter discusses the existing technologies that are used in implementing fault-tolerant features among sensors, actuators and communications from both hardware and software point of view.

Chapter 3 addresses the architecture of the proposed system. It specifies the approach adopted to build a dynamic model for the specific application, provides the background to the health monitoring system proposed in [8], and describes the approach taken to create the health indicators.

Chapter 4 describes assembly and implementation of a laboratory automotive Steer-by-Wire system, which was designed and built to illustrate the concepts developed within this thesis. This is followed by the experimental results. An analysis of the health

monitoring system compared with physical sensors state values shows that system failure modes can be detected and diagnosed through a combination of selected health indicators.

Chapter 5 presents the overall conclusions of the work and makes recommendations for further work.

Chapter 2: Achieving Fault-tolerance

2.1 Introduction

Fault-tolerance is the property that enables a system to continue operating properly in the event of the failure of one or more of its components. Complex safety-related functions in future automotive systems will be increasingly based on electronic components, which are susceptible to a variety of failure modes caused by either interior or exterior factors. Consequently fault-tolerance is emerging as a key technology which needs to be applied to the design of By-wire systems. The goal for a By–Wire system designer should be to create as robust and reliable a system as possible within reasonable cost constraints; Ideally, the range of faults tolerated by the system should be as wide as possible, so that the probability of the catastrophic failure can be minimized.

2.2 Terminology

Hiller [9] provides a good summary of terms used in fault–tolerant system design. Although his work concentrates on software fault–tolerance, an identical terminology is applicable to the whole electro–mechanical system. The three most important terms are **fault**, **error** and **failure**.

- A **fault** exists when there exists a state of operation in a system that leads it to non-conformance of its specifications. Faults are classified by their duration, as either transient or permanent, and their realization, as dormant or active.
- An **error** is the manifestation of an active fault; it is an occurrence of the system entering a state of non-conformance to its specification. Undetected errors are termed latent [10]. The key difference between a fault and an error is that whereas errors are the results of faults, only an error is measurable.
- A **failure** is the result of an unresolved error. Note that a failure of a low-level system might be considered a fault by a higher level system, which could in turn trigger a different error. In a safety–critical system, a failure will be called catastrophic if it happens at a high enough level to put the user's safety at risk.

Since typical By-Wire system requires the correct action to be taken at the correct time, one needs only consider terminology commonly used in the discussion of real time systems. Kopetz defines a **real–time system** as follows [11]:

- A **real-time system** is a computer system in which the correctness of the system behaviour depends not only on the logical results of the computations, but also on the physical time when the results are produced.

In addition to this, a hard real-time system - as opposed to soft real-time system - is one for which a missed deadline implies a failure. In general, real-time computer systems interact with a physical environment, such as sensors and actuators. Safety-critical systems such as Steer-by-Wire are typical examples of hard real-time systems: a missed execution

deadline is not tolerable since it may lead to global catastrophic failure. So in a hard real-time application, the system must always produce the correct value at the correct time. The manifestation of an active time fault will be called a time error and defined as a computation result which is: either never produced or produced outside its allowed predefined time zone.

2.3 Fault Sources

A typical Steer-by-Wire system normally includes physical components such as sensors and actuators. Usually, these are monitored and controlled by a distributed computing platform, hence one must also consider the ECU's and communication channels. Before one may examine the architecture of a fault-tolerance system, it is necessary to have a good understanding of the nature of the potential faults for each of these components.

2.3.1 Sensors

In general, faults that can be expected in sensors are either due to mechanical failure of some components (e.g., fatigue or accidental impact) or to environmental disturbances (e.g., vibrations, electromagnetic interferences, etc). Mechanical failures usually lead to permanent faults, while electrical disturbances or vibrations may cause transient faults. "Intelligent" sensing units, which are combined with a dedicated microprocessor and memory at the sensor level, should obviously be considered separately, since they feature logic components, which are themselves subject to electromagnetic interferences and other environment–caused disturbances. Although the use of intelligent sensing units

simplifies the system from a global perspective, it adds a degree of complexity at the local level and therefore introduces additional fault sources in the system.

2.3.2 Actuators

DC motors are extensively applied into the area of Electric Power Steering and Steer-by-Wire. Common DC motor failures are due to wear of the brushes and integrity of armature insulation. The brushes tend to wear quickly because of their friction on the commutator ring. For applications where system reliability and longevity are important requirements, it may be preferable to use brushless DC motors. The amplifier now becomes more complex and sensors are needed to synchronize current with rotation. Other typical DC motor failure sources include bearings or seal failure, insulation breakdown, demagnetization and damaged connections.

2.3.3 Electronic Controller Units

When examining faults on ECU's, it is necessary to distinguish physical hardware failures from software faults. Indeed, although an ECU can be considered "just another physical component" in the system, it also runs a soft application, which is itself subject to faults.

Apart from accidental physical damage to the ECU hardware, which is obviously outside of the scope of normal operation, ECU hardware faults are typically caused by electromagnetic interferences, temperature changes or vibrations [12] and are therefore transient by nature. Such faults may corrupt the values present in memory or inside the processor's registers, and may lead to failure of the ECU if no mechanisms are

implemented to detect and tolerate them. They can lead to software errors in both the time and value domains.

Software design faults ("bugs") should also be considered. Design faults exist "when the design of the system does not match the specifications" [12]. Therefore, they are not locally detectable (since the software performs according to its own erroneous specifications). However, especially in complex software systems, guaranteeing the absence of design faults implies that exponential numbers of combinations need to be considered and tested; this may often be impossible. Ways to detect and treat software design errors are therefore required. This is usually achieved through the use of diverse redundancy.

In the context of distributed computing systems, malicious processor behaviour, i.e., a fault which results in a processor outputting inconsistent and incorrect data at correct times, is called a "Byzantine" error. This refers to the classic Byzantine generals problem [13].

2.3.4 Communication Channels

A communication channel is faulty if the data it carries is corrupted during transmission or not transmitted. The most likely cause for corruption of the signal is electro-magnetic noise. Many communication protocols have built-in checks (typically Cyclic Redundancy Check), which detect if a message has been corrupted during its transmission. These faults are by definition transient, but they can cause higher level time faults, since the recovery mechanism may introduce non–deterministic transmission times.

Other types of faults can be caused by failure of the communication controller chips themselves, e.g., from the effects of electromagnetic interferences or ageing. Finally damage to the physical medium itself (e.g., a broken wire) will induce a permanent communication fault.

In a computing network, a faulty node, which attempts to gain access to the bus repetitively and thereby prevents the remaining fault-free nodes to communicate, is said to exhibit a babbling idiot behaviour. Such behaviour must absolutely be avoided in hard real–time, safety–critical applications, for obvious reasons.

Sensor measurements acquired by ECU's can be erroneous because the analog or digital link from the considered sensor to the considered ECU is physically damaged. Possible damages can include broken wires or damaged contacts. In a similar manner, it is possible that a correct analog command is sent in an incorrect manner to a non-faulty actuator because of a damaged link, thereby triggering an actuation error.

2.4 Existing Approaches to Achieve Fault-tolerance

In this section different approaches taken to design fault–tolerant safety-critical systems are examined and contrasted. Although the projects reviewed rely on different methodologies, one concept is universal and compulsory in fault-tolerant system design: redundancy.

2.4.1 Redundancy

In the event of a failure of one or more components of an embedded system is detected, alternate hardware must be available to allow the system to tolerate the fault(s).

Redundancy is necessary then for error treatment as well as for error diagnosis, (which uses a voting arrangement between alternate components).

Redundancy is often described as being either "hot" or "cold". A cold redundant component does not run in normal operation, but can be activated and take over if the primary component fails. A hot redundant unit or component, on the other hand, runs in parallel with its primary counterpart. Hot redundancy is advisable in hard real-time safety-critical environments, where the switching time must meet stringent requirements. "Redundancy" will therefore refer to "hot redundancy" in this chapter.

Redundancy can also be categorized as exact or diverse. Exactly redundant units are identical in all points, perform the same tasks at the same times, and are therefore expected to produce identical results at a given time. On the other hand, diversely redundant units use separate methods to perform a given task. Examples of diversely redundant units for position sensing are a potentiometer and an optical encoder both used for position sensing (but with different characteristics and different methods required for their use). Unlike the tolerances of identically redundant units, acceptable tolerances of diversely redundant units must be judged individually to account for each unit's unique physical characteristics, such as speed, accuracy and environmental effects.

The use of diversity allows a system to tolerate software design faults, which are not detectable using exact redundancy. Diversity also decreases the probability of environmentally caused simultaneous failures of the redundant units, by allowing the selection of physically different units. Finally, it aids in cost reduction, since high–quality, expensive units can be used in conjunction with lower performance redundant units.

2.4.2 Fly-by-Wire

Fly-By-Wire systems are common in commercial aerospace applications, and have been in use since the late 1980s. The first commercial airplane to be equipped with a total Fly-By-Wire system, with no mechanical back-up, was the Airbus A320, in 1988. Since then, Boeing has developed its own Fly-By-Wire aircraft, the Boeing 777, and Airbus has expanded its technology to newer models, such as the A330 and A340. Although the aeronautics example provides a good starting point for personal vehicle By-Wire systems design, the technological solutions adopted in Fly-By-Wire applications are not necessarily directly transferable to large scale, cost-sensitive markets.

Fault-tolerance of the Boeing 777 Primary Flight Computers (PFCs) is accomplished using triple-triple redundancy [14]. This means that TMR, which consists of three redundant units, coupled with a voter [12], is used at two levels in the PFCs: each of the three redundant computing units which form the PFC is itself comprised of 3 diversely redundant computers. Diversity is ensured at the hardware level by selecting hardware processors from different manufacturers (namely AMD, Intel and Motorola), and at the software level by compiling the Ada source code using three different Ada compilers.

2.4.3 X-by-Wire

The "X–By–Wire: Safety Related Fault Tolerant Systems in Vehicles" project was conducted in Europe from 1996 to 1998 and involved members of both the automobile industry and academia: Daimler-Benz, Centro Ricerche Fiat (CRF), Ford Europe, Volvo, Magneti Marelli, Bosch, Mecel, Technical University of Vienna, and Chalmers University of Technology [15].

The objective of this project was to achieve a framework for the introduction of safety related fault tolerant electronic systems without mechanical backup in vehicles. The "X" in "X-by-Wire" (XBW) represents the basis of any safety related application,such as steering, braking, power train or suspension control or multi-airbag systems. These applications will greatly increase overall vehicle safety by liberating the driver from routine tasks and assisting the driver to find solutions in critical situations.

The approach taken by the XBW team is based on exact redundancy and fail-silence, at all levels in the system's organization. In a Fail-Silent Unit (FSU), the component must be able to self check and output either the correct value or nothing at all. Furthermore, each atomic subsystem is a fault-tolerant unit (FTU), which is composed of two exactly redundant FSUs Figure 2.1. This is valid for both sensor level and ECU level.

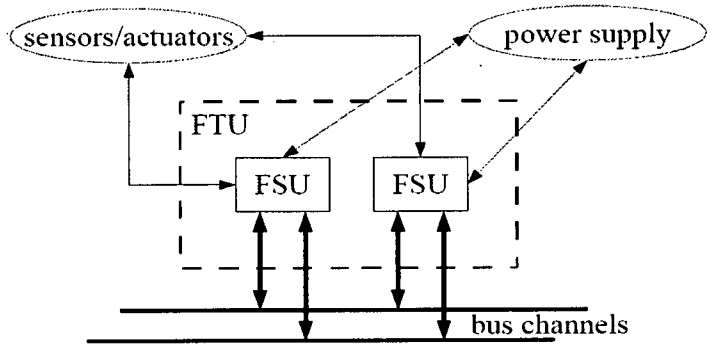


Figure 2.1: Structure of a Fault-tolerant Unit
X-by-Wire Project

The fail-silent property is likewise enforced for each of the two bus channels, based on error detection coding and checking for compliance with expected communication behaviour. Thus, the architecture is capable of tolerating a failure of any bus channel.

In XBW's implementation prototype, actuation was achieved by having three redundant DC motors connected to a specialized gear box. Each actuator provides one third of the required torque in normal operation, and half of the required torque in an error state. The report concluded that it was more cost effective to have three motors capable of delivering 50% of the required torque than two that deliver output 100% of required torque.

The XBW project was conducted in parallel and in close partnership with the Time Triggered Architecture project, which also aimed at demonstrating an architecture of fault-tolerant distributed real-time systems in safety-critical transportation application. As

Time Triggered Protocol supports the use of a duplicate physical communication channel, it employs exact redundancy.

Achieving the desired fail-silence property appears, however, particularly challenging and costly. For example, at the ECU level, fail-silence implies that each FSU is able to perform self error detection, and is able to shut itself down. In practice, only the combination of two ECU's can make this possible. Hence, quadruple redundancy is required to form a single Electronic Controller FTU. The increased cost makes it unsuitable for large scale production in cost-sensitive markets. Furthermore, the architecture is extremely rigid and does not make optimal use of the available resources. However, some of the techniques used for software error detection and for fault-tolerant actuating are quite ingenious.

2.4.4 Error Detection

While some redundancy is required, many projects avoid the use of total hardware redundancy in an effort to reduce costs. For example, at the sensor or actuator levels, model-based methods have been developed which exploit the "inherent redundancy contained in the dynamic system equations that relate the different sensors outputs" [16]. This is called analytical redundancy. It essentially takes two forms:

- Direct redundancy (the relationship among instantaneous outputs of sensors)
- Temporal redundancy (the relationship among the histories of sensor outputs and actuator inputs).

Analytical redundancy allows the outputs of dissimilar sensors to be compared. The residuals resulting from these comparisons are then measures of the discrepancy

between the behaviour of observed sensor outputs and the behaviour that should result under normal conditions. [17] gives an example of a triple redundant unit replaced by a double redundant hardware system used jointly with analytical redundancy, resulting in significant cost reduction.

Error detection can take on different forms, but these are generally classified as either data replication or executable assertions. Data can be replicated through analytical redundancy, double execution, and hardware replication. Executable assertions include limit checking, certification, signature checking, self-tests, and watchdog timers [18]. Furthermore, detection can be performed within a component or via a network by peer-to-peer checking or supervisory elements. Distributed environments require a mechanism to synchronize their timing, and when error detection is performed in a distributed environment, errors may be detected in the time domain: each computer can check the communication timing of the other computers.

The ECU level of the system has been the subject of many research efforts on error detection. Some workers have attempted to design processor architectures, which inherently provide on-line self-testing capabilities [19]. In the latter example, a primary and back-up microprocessors are integrated in a single element. Other approaches use codes to detect hardware ECU failures [20]. In these applications, computation results are coded using a code generator, and later checked by a code checker, thereby providing self-test capabilities. This concept is similar to methods used for communication error detection, such as CRC. Unfortunately, it requires development of additional logic circuits, and can therefore not be directly used with off-the-shelf components. The latter

works are mainly targeted at the fault tolerance of large computer networks, and are relatively poorly adapted to smaller embedded applications.

In an effort to minimize the overall system cost Bouvier [21] reduced the number of components and optimized the use of the components. Bouvier developed a distributed fault-tolerant architecture and demonstrated it in a Steer-by-Wire application. In his system triple hardware diverse redundancy was used at both the sensor and ECU levels while the object-oriented model of the system was combined with on-line error detection and both software and hardware dynamic reconfiguration[1] utilities. Each ECU could perform local error detection on the data it acquired, and also a distributed error detection with its peers to detect sensor, ECU, and communication errors. The system response strategy uses a dynamic reconfiguration approach, in which the system can reconfigure resources at run-time. This architecture uses an abstraction of system hardware and resources in combination with an execution task flow chart to select tasks. The fail-silence property described by the X-By-Wire project is also implemented on each ECU. Each ECU is connected to a second set of channels called state-lines, which operates as a voting mechanism, where two peers must validate the local ECU as being functional to allow it to broadcast data over the communications bus. This research uses and enhances the fault-tolerant architecture developed by Bouvier.

2.4.5 Reconfiguration

Dynamic reconfiguration is the process of making changes to an executing system without requiring that the system be temporarily shut down. This feature should be built into systems that have requirements for adaptability and/or high availability.

1 See section **2.4.5**

A distributed reconfigurable network of ECU's to control hyper-redundant space robot manipulators was developed in [22]. Their system is able to adapt itself in real-time to partial failures and to the changing operating conditions through the use of a distributed consensus algorithm. The inverse kinematics problem is solved on all ECU's, and the optimal solution is selected and applied after exchanges over the communication channel have led to an agreement.

Oldknow has developed a dynamically reconfigurable architecture for machining applications [23]. The goal of his work is to optimize the machining process from an economic point of view. Depending on the operating conditions, different constraints are active. Relying on reconfigurable architecture, the system is able to adapt itself to the current active constraint, in real-time.

A further application of reconfigurable systems lies in possibilities for off-line modifications. If the system's architecture is flexible enough, modifications such as sensor upgrades are possible in a way that is transparent to higher-level software. [23] in particular details a way to use self-instantiating intelligent components, from which the code required to use them is uploaded to the high-level controller, hence providing "plug and play" functionality.

2.4.6 Graceful Degradation

To cope with failures, a system may reduce performance capabilities, cancel less important tasks, or switch to different control algorithms. This method of tolerating errors is called Graceful Degradation.

The most active research group currently tackling graceful degradation issues in distributed embedded real-time systems is RoSES (Robust Self-configuring Embedded Systems). The RoSES project was started at Carnegie Mellon University in 2000, under the supervision of Dr. Phil Koopman [24]. The RoSES approach to graceful degradation is based on Product Family Architecture (PFA). A PFA can be thought of as the whole range of offerings of a given manufacturer for one type of products. The idea behind the RoSES project is that upon failure of some of the components of a system, it is possible to shift to another configuration, i.e., to another "product" in the PFA.

However, the initial effort of the RoSES team concentrates on off-line reconfiguration of non safety-critical systems, and is therefore not directly applicable to the problem described here.

2.5 Health Monitoring

The majority of work in fault-tolerant systems discussed so far relates to the detection and handling of errors, where errors are classified as existing in a binary state: either the error exists (the system is in a state of non-conformance with specification), or the error does not exist. A weakness with the binary error approach of these fault-tolerant systems is the lack of error predictability. The ability to predict impending failure allows the operator or system to take preemptive action to prevent failure by altering system functionality or scheduling maintenance. Indicators of system health give insight into the likelihood of failure.

Health indication seeks to represent the system's health state by selecting indicators which describe system performance or correlate to known failure modes. There are a

number of approaches that have been used to develop health indicators, which can be classified as either Condition Monitoring, Parameter Identification, or State Estimation.

2.5.1 Condition Monitoring

Condition monitoring can be defined as a technique or a process of monitoring the operating characteristics of machine in such a way that changes and trends of the monitored characteristics can be used to predict the need for maintenance before serious deterioration or breakdown occurs, and/or to estimate system's health.

In general, the condition monitoring approach involves adding physical sensors to measure a direct indicator of health. Condition monitoring will often require the creation or addition of new sensors to measure signals which directly indicate system performance. These sensors are specific to each application, and each recorded signal is usually indicative of a specific fault or operating condition.

A variety of internal system parameters may indicate errors; temperature can reveal lubrication problems, misalignment, or overload; noise may indicate valve, gear, or bearing wear. The use of these internal states to predict failure is discussed in, Goode [25], who proposes a prediction model theory based on statistical process control, and a failure model using a stable zone and failure zone to predict time to failure.

Condition monitoring systems are programmed with awareness of a number of fault conditions, and a diagnosis mechanism compares the system state with the condition signals to provide a diagnosis of system health.

2.5.2 Parameter Identification

Parameter identification techniques seek to determine and monitor the model parameters for the system over time; a common way to do this is to use observers. Usually, a system model of a given order is defined, and then a least squares regression is performed to determine the parameters for the given relationship between system input states and measured system output states. The system parameters can then have limit values applied to them that describe acceptable operation.

Isermann [26], describes the parameter estimation method for linear systems and how significant parameter changes can be detected by reference to the normal values using statistical methods like the Two-Probe T-Test.

One weakness in this method is that the parameters of a given order model may not represent system characteristics in a predictable way. Typically, these systems would then need to be developed for each application experimentally, and the limits must also be set experimentally. This reduces the potential design benefits of using a system model.

2.5.3 State Estimation

The process of state estimation involves using a known system model, and applying a set of inputs to known system states to predict the output state at a given interval. The health indicators that are built from state estimation are generally composed of the residual error between the predicted state and the measured state.

State estimation techniques such as Kalman filters or various other tracking filters can also be implemented as a prognostic technique. The Kalman filter is a dynamical systems tool for estimating unknown states by combining current measurements with the

most recent state estimate. It can be considered as a virtual sensor in that it takes current available sensor measurements and provides optimal estimates (or predictions) of quantities of interest that may in themselves not be directly be measurable. It is typically implemented with the use of a linear system model, but can also be extended to non-linear systems through the use of the extended Kalman filter algorithm that linearizes the system about an operating point. The linearization allows the application of other linear algebra techniques.

Other parameter estimation techniques rely on models to predict future state values. If the observer output states reconstruct measurements of the process, then the observer creates analytical redundancy of those states. Typically, a diagnostic observer generates output states that are indicative of faults, whereas the state observers generate data needed for control. Control observers also tend to operate within a closed-loop environment, whereas a diagnostic observer tends to operate in an open-loop configuration. This requires the diagnostic observer to be more complete, or more robust when considering model uncertainties.

2.6 Summary

The main challenge of developing "By-wire" systems is the achievement of suitable levels of fault-tolerance at a cost which is acceptable to the market. One must be able to guarantee that continued functionality is provided even if one or more of the system components is faulty.

This chapter first described the terminology used to define faults together with the common fault sources for each of the typical components of a Steer-by-Wire system. It then reviewed previous research projects and approaches of fault-tolerance systems.

Finally health monitoring was introduced in the main areas of condition monitoring, parameter estimation and state estimation.

Chapter 3: System Architecture and Dynamic Modelling

3.1 Introduction

The steering system was chosen to be used as an example in this thesis is the steering system on a prototype three-wheeled urban vehicle designed in the laboratory. The steering systems transmits the input from the steering wheel to the steering actuators, which in turn move the steering rack and tie rods to steer the wheels.

The steering action is usually achieved using a mechanical connection often assisted with hydraulic or electrical components ("Power Assist"). In this particular case it was deemed advantageous to build a modular steering system that can accommodate manual steering, a power assisted mode and a steer by wire mode, each either alone or in parallel. This complicates the overall design, but provides considerable flexibility in the provision of performance and robustness. In order to simplify the building of the initial prototype and to expedite the experimental part of this project, the experiments were conducted using a torque command rather than manual torque supplied by the steering wheel. The torque command to the power steering system was generated by the error signal in the steer by wire position loop. Additional discussion of this arrangement is given later in the Chapter.

Next section provides a detailed description of the parallel power steering and steer by wire actuators together with a discussion of suitable control strategies. This is followed by an introduction to the idea of Health Monitoring developed in previous work by Nicholas Cullingham [8] in the laboratory (concerned with the monitoring of hydro-mechanical marine steering systems) and the provision of improved fault tolerance through the use of soft sensors. The basic premise here is that one may utilize a virtual model of the system dynamics, together with the inputs to the real system, to track the expected values of major physical states within the system and to make comparison with actual values.

Finally in this chapter, the dynamics of the steering system are modelled by analyzing the control loops of the parallel steering actuators and an initial simulation of the behaviour of the proposed health monitor is developed. This then allows the development of suitable approaches to the design and implementation of the real time software system which will be embedded within the experimental steering system.

3.2 Steering Actuation Design and Control Strategy

Steer-by-wire and power steering are both safety-critical functions. While power steering does have mechanical backup, the loss of the power assistance can cause significant problems. A true steer by wire system poses even greater risks and it is likely that this will require the use of redundant actuation components as well as redundant sensors and computational elements. It is not necessary however that these redundant systems be duplicates and in many cases one may achieve both a lowering of cost and improvement in reliability by having different types of actuator and drive. Some systems

performance may suffer as a result of not simply duplicating the best alternative. However, in the case of steering, the required performance parameters are quite easily satisfied and exceeded by a wide variety of approaches.

Within the steering system the following major groups of elements need to be considered:

a) Actuators and associated elements, (drive trains, amplifiers, etc)
b) Sensors
c) Communication networks
d) Computation and logic elements
e) Power supplies and associated wiring

Ideally, to maintain system functionality in the case of single component's failure and prevent catastrophic failures, each component should be fault-tolerant, reliable, and to achieve this will require some degree of redundancy. The major interest in this research is to find ways to minimize the increased cost of redundancy by incorporating a parallel software system which can offer "analytical redundancy".

Among the electronic components needed in the application, actuators are almost always the most expensive to duplicate, hence the consideration of alternate approaches to actuator redundancy is one of the major issues to be considered during the prototype design phase. A fault-tolerant system with a single motor drive may overcome most of the problems in critical applications by having redundant and reliable controllers, the result of a complete motor failure is of great concern.

As with any design problem there are many ways to achieve actuator redundancy, X-By-Wire project's prototype had three DC motors connected to a specialized gear box. Each actuator would provide one third of the required torque in normal operation, and half of the required torque in an error state. The report concluded that it was more cost effective to have three motors capable of delivering 50% of the required torque than two each delivering 100% of the required torque. However, to support a high number of redundant actuators, more controllers, feedback systems and communication systems have to be considered for reliable operation. At the same time one may wish to anticipate potential transmission failure and provide alternate drive possibilities. This research suggests that a triplicate system including the necessary sensors and actuators/transmission elements is unlikely to be cost effective.

For research purposes, the prototype vehicle in the laboratory uses two actuators with different means of transmission. The provision will be made for the possibility of manual override in the limit case, which in practice would require a flexible transmission element to take full advantages of Steer-by-Wire system (This is likely practical on a temporary "limp home" basis). The motors will also utilize different control modes, one in position mode (steer by wire), the second in torque mode (power steering). Realistically, the power steering would be driven by either torque applied to the steering wheel or a combination of torque and position. The prototype vehicle does not have the rigid connection between steering wheel and steering mechanism so output from the first motor amplifier (torque signal) is used to drive the second loop, as shown in Figure 3.1:

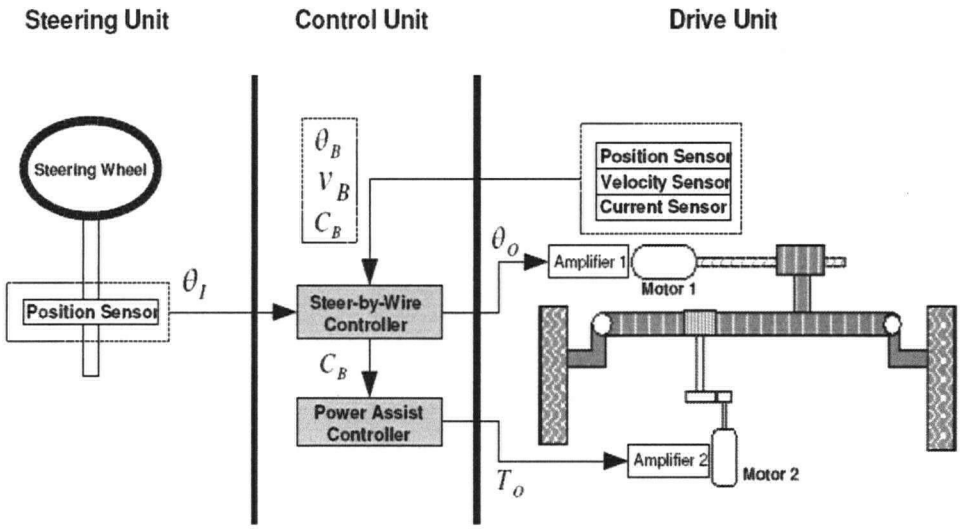


Figure 3.1: Two-actuator System Schematic Illustration

3.2.1 Steer by Wire Actuation

The first control loop in the system (Steer-by-Wire) receives commands from the user by measuring the position sensors mounted at the steering wheel and controls an actuator driving the pinion and the steering rack. This controller implements closed loop position control and has position and velocity sensors to measure the rack's movements. To facilitate the operation of the second control loop (power steering), it also measures the current output supplied to the actuator.

3.2.2 Electric Power Steering Actuation

Most modern conventional power steering systems consist of an engine driven hydraulic pump and a hydraulic actuator. However, electric power steering systems are more compatible with Steer-by-Wire applications. In addition, electric power steering

systems offer better fuel economy, are more compact and more environmentally friendly than traditional hydraulic systems.

To simplify the building and controlling of the experimental prototype, the input to the power assist mode in this case comes from the first motor amplifier's output. As shown in Figure 3.1, the current output from the first motor amplifier is detected and transferred to the power assist controller; this in turn determines the appropriate output magnitude and drives the motor (i.e., the power steering actuation is operated in an open loop torque mode). With the power assist, the amount of steering torque required by Steer-by-Wire actuator can be significantly reduced.

In the normal situation, these double actuators will be operated in their different modes. Should the position mode motor fail, the other motor would still be able to acquire the current output from the amplifier in the position control loop, steer the wheels itself and back-drive the other actuator. This guarantees the fail-safe capability of the overall system at the actuation level.

3.3 Fault-tolerant Framework

Before presenting the Health Monitoring concept, it is useful to provide an introduction to the fault-tolerant framework which is to be used as a platform for the Health Monitoring system. This fault-tolerant system architecture is based on Bouvier's multiple embedded hardware [21].

To achieve a particular system function, a fault-tolerant system relies on having redundant methods, which consist of software or analytical replication, and hardware or physical replication. Each system function can be thought of as a causal system that has a

set of input states, parameters, output states, and relations between each of those states. Redundancy is needed to maintain any high level function when that function depends on generating a defined output from a given set of inputs even if a sub-component fails. Input states are replicated through analytical or physical redundancy; the relationship between input and output, the actuators or plant, are replicated only through physical redundancy.

To facilitate fault-tolerant management and system reconfiguration, object oriented programming techniques are used in the development environment. Thus each component can be described by a software class whose attributes and methods provide an abstraction of that component. Where components are replicated, each class can be instantiated as a software object, whose attributes describe that particular component. The inheritance property, which allows a class to by defined as a child of a previously designed class, causes the child class to include all of the attributes and methods of the parent. This property is particularly useful when designing for diverse redundancy, where each replication has similar attributes, or may require more attributes, and has similar methods, which need to be slightly altered for each case. As in terms of the whole system, this allows one to isolate the high-level software from the low-level redundant sensors and actuators. By doing this, one can allow the high-level software to be "unaware" of the low-level redundancy and to be unaware of where the information originated from. Thus, programming and many kinds of error detection can be done in a relatively simple manner.

A representation of the fault-tolerant framework in Bouvier's work is shown in Figure 3.2, which displays the hierarchical nature of software configuration including atomic methods, mid-level methods or high-level methods. The architecture shown here is representative of one Electronic Control Unit (ECU).

Atomic methods describe the actual instantiation of components available to the system. Each atomic method has a one-to-one relationship with an input component (a sensor) or an output component (an actuator). Atomic methods make up the system description and the hardware abstraction layer.

The mid-level methods make up the redundancy manager, where the treatment of sensor and actuator errors is done locally. These methods include the error detection, masking, and error response functionality. Redundancy manager is used to determine the state of all physical devices in the system, and provide a correct, single reading from redundant sensors , even if one of them is faulty.

The high level methods sit at the highest level in the system, and are independent of the low-level hardware. Their inputs and outputs rely solely on fault-tolerant methods and data that have been checked against errors in the mid-level methods. A typical example of a high-level method is one that performs a closed-loop control action: it takes a reference value and a feedback value, compares them, applies a control strategy and outputs a command.

The architecture utilize triple hardware diverse redundancy at both the sensor and ECU levels while the object-oriented model of the hierarchical system is implemented to allow the dynamic reconfiguration of both software and hardware utilities.

3.4 Health Monitoring System Development

As seen from Bouvier's work, at least three components are needed for every level of fault tolerance, as 3 is the smallest number which allows majority voting. Should one of these components fail or become isolated, voting can no longer be used.

Ideally, the system should provide some protection and allow for modifications following initial failure, without increasing the level of redundancy. Wherever possible, protection should be provided by the software since it has lower value. This is achieved by building models of the system and using the relationships between input and output as an auxiliary check on system performance. In this way, one can develop a fault-tolerant as well as cost-effective system by further reducing the amount of hardware while maintaining system's fail-safe functions. This model based detection as well as the treatment procedures is termed a Health Monitoring system.

The following section introduces the particular Health Monitoring concept developed in Cullingham's thesis project. Cullingham implemented this approach to provide additional software redundancy within an electro-hydraulic marine steering system [8]. The system is adapted here for the dual actuator electro-mechanical automobile steering system.

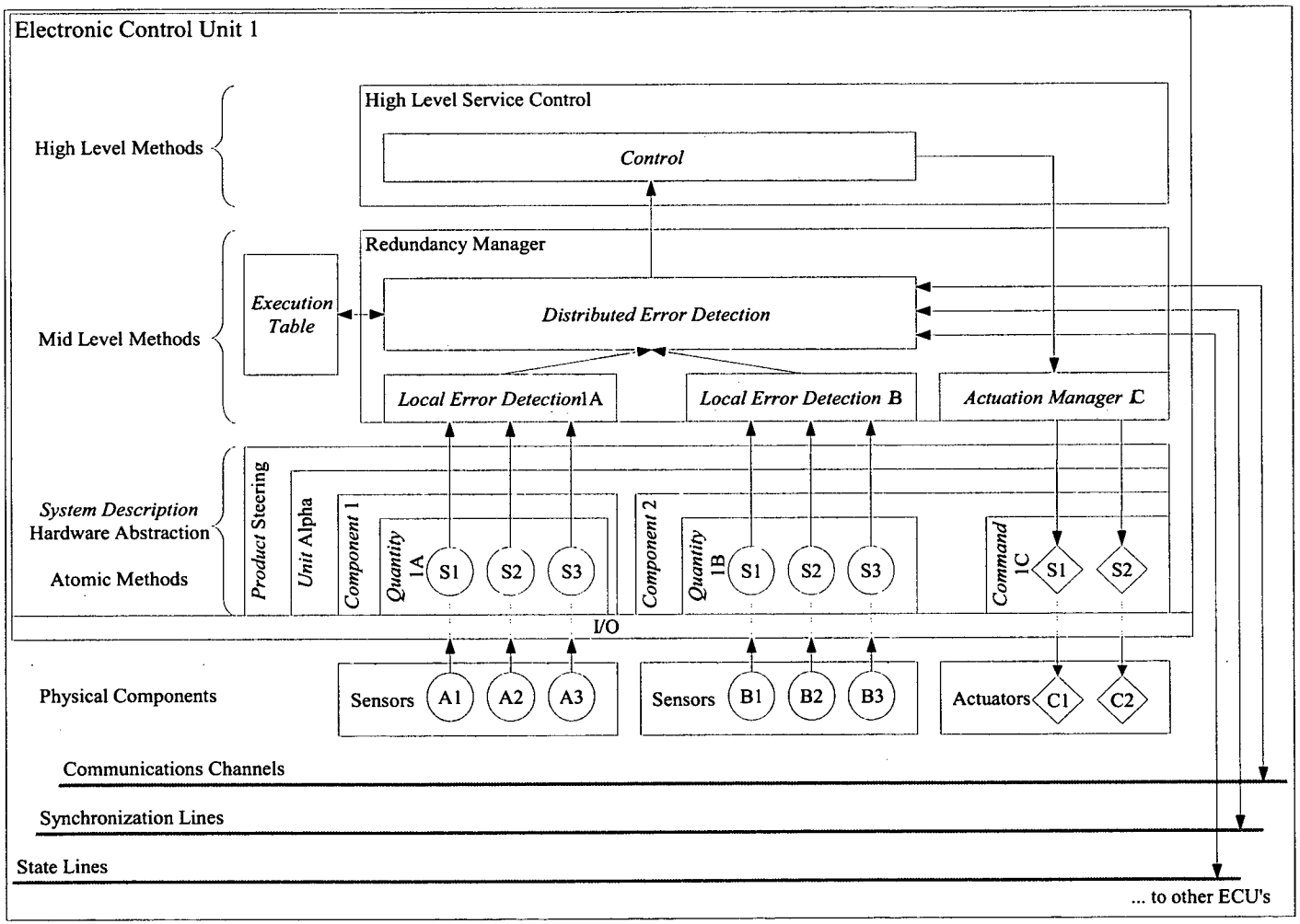


Figure 3.2: Fault-Tolerant System Architecture

3.4.1 Model-based Approaches and Model Selection

Section 2.5 discussed a number of approaches to achieving health awareness: condition monitoring, parameter identification, and state estimation. In general, the condition monitoring approach involves adding physical sensors to measure a direct indicator of health. Alternatively, parameter identification and state estimation use model-based approaches, and sensors that may already exist for control purposes, to create a measure of health. Model-based approaches are preferred for this project because they allow the use of existing hardware.

The greatest difficulty with parameter estimation occurs when the system model is non-linear. Typically, parameter identification approaches rely on the least squares algorithm to determine the parameters, and this algorithm requires the model to be not only constant, but also linear.

Most real mechanical systems cannot be accurately represented by a single linear model. Mechanical systems typically have a number of non-linearities (e.g. saturations, relays, friction). Furthermore, a number of mechanical systems are not time-invariant; they change not only with their internal parameters, but also with time. To accurately model such system, non-linear model elements and multiple model states are required.

Although there are few non-linearities (saturations, dead-bands) in the prototype steering system, state estimation approach is still more compatible with this application. First, it is straightforward to compare the analytical sensor with physical sensors to assess system health. Second, state estimation is also the best approach for systems controlled in real time since it minimizes memory requirements and is computationally efficient.

For this study, a complete simulation model is constructed from a number of system parameters and second-order transfer-functions, including gains, filters as well as integrals. In this manner, a number of small processes or system components can be individually modelled, and then cascaded in series to produce an overall model of the system. This facilitates the process integrating both control loops, (closed position loop and open torque loop), and to simplify the modelling. The use of a second-order system model to estimate system dynamics has a number of benefits. In particular, reducing the system to its dominant second order model allows its performance to be specified simply in terms of gain, damping and natural frequency.

To allow the dynamic model to continue to accurately replicate application's behaviour, the closed loop state estimation is updated in the real time. The model is adjusted after each comparison with the physical system to ensure that the starting values of the parameters for the next period correspond to the actual values observed. The details of this procedure are to be detailed in section 3.4.3.

3.4.2 Indicator Selection

The model-based approach used by Cullingham [8] re-used the existing control, sensors and ran the system model in parallel with the physical plant. Differences in performance then needed to be identified that would provide meaningful information about failures or impending failures of the system. The study of potential failures of the marine steering system lead Cullingham to propose three different indicators of failure: the Instantaneous Response Indicator, the Dynamics Response Indicator and the Average Response Indicator.

The **Instantaneous Response Indicator** is the current error between input and output to the system. Obviously a lower absolute value of the indicator indicates a healthy measure for the system, whereas a high absolute value indicates that the system health may have deteriorated. In this research, the IRI represents the error between command and actual positions of the steering rack. The instantaneous error is a good initial indicator of system health, for one can only tolerate a set amount of error. However the instantaneous error does not always provide enough information to localize component problems. Typically one needs to examine the change in these parameters over time and to incorporate reasonable dynamic models so that one can compensate for transient periods of high error which are normally the result of the input transients.

The **Dynamics Response Indicator** is determined from errors between the actual system states and those estimated from a system model. This process requires that the model know the inputs to the system and that the model be restarted with actual physical values at regular intervals to prevent errors in the output from the system model being confused with deterioration of the system. The procedure is described in detail in the next section. The model developed for the prototype estimates the expected amplifier output to the drive motor which is then compared to the actual output of the amplifier within the steer by wire loop. Since the Dynamics Response Indicator is run over a short period of time and its states are reset with measured states on a regular basis, health errors arising from slow changes in the system may be missed. The third type of response indicator is used to verify system health.

The **Average Response Indicator** is the difference between expected change and the measured change over a period of time. The response to a slow moving input is checked over a period of time that is significantly longer than the settling time of the dominant system dynamics. For the purpose of this thesis, this indicator is the difference between motor's expected average velocity and actual average velocity over the time window. A large error signal here indicates that the system is not responding as expected. A change in performance can be attributed to either the input (actuator fault) being incorrect, or that the model structure or model parameters no longer adequately approximates the actual system (mechanical fault).

3.4.3 State Estimation Process

The Health Monitoring System has three available inputs: the system state vector, system reference vector, and control signal vector.

The model is not design to be run continuously, instead it is only operated over short intervals at which comparisons are made between the model and physical system states. Once the comparison is made, the model is reset with current system states and allowed again to run for a short time. This principle described in the previous paragraph is shown in Figure 3.3. In this figure, a single estimation process is shown. The model is updated at each loop closing to include all of the inputs, $u(t)$, $u(t+\Delta)$ ... $u(t+n\Delta)$ and the starting states of the system, $y(t)$; it then produces the system end point, $y(t+n\Delta)$. After each estimation, the actual system states are stored by the model and used to generates the state value at the end of the next period.

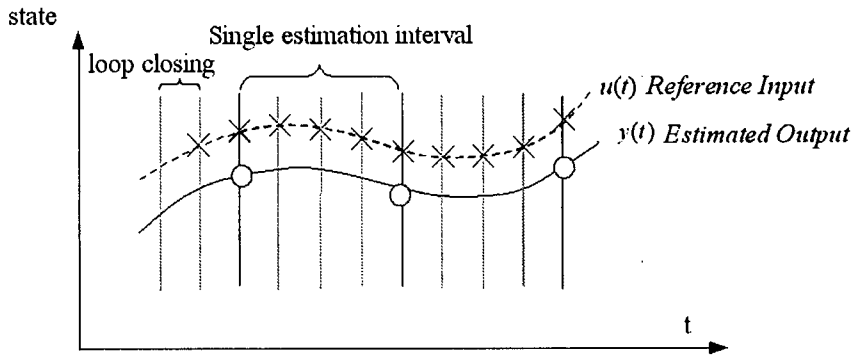


Figure 3.3: Single State Estimation Interval

In Figure 3.3, the period of estimation is significantly longer than the loop-closing interval, but the system health indicators are required to be updated at each loop closing. Then when an estimation occurs at each loop closing, it shares the same inputs with other estimations in the same interval and acquires the starting state generated by the dynamic model from previous estimations, as shown in Figure 3.4. During this multiple estimation process, each set of estimation then represents results for a window of system data that moves with time. When an estimation occurs at each loop closing, the moving windows overlap with previous estimations.

To accurately simulate the actual system's behaviour and prevent the tolerated errors in the model estimation being confused with deterioration of the system, at the end of each estimation interval, (e.g. 5 times of loop closing), the dynamic model is restarted to the actual system states and performs another interval's estimation after being synchronized with the actual system inputs and states.

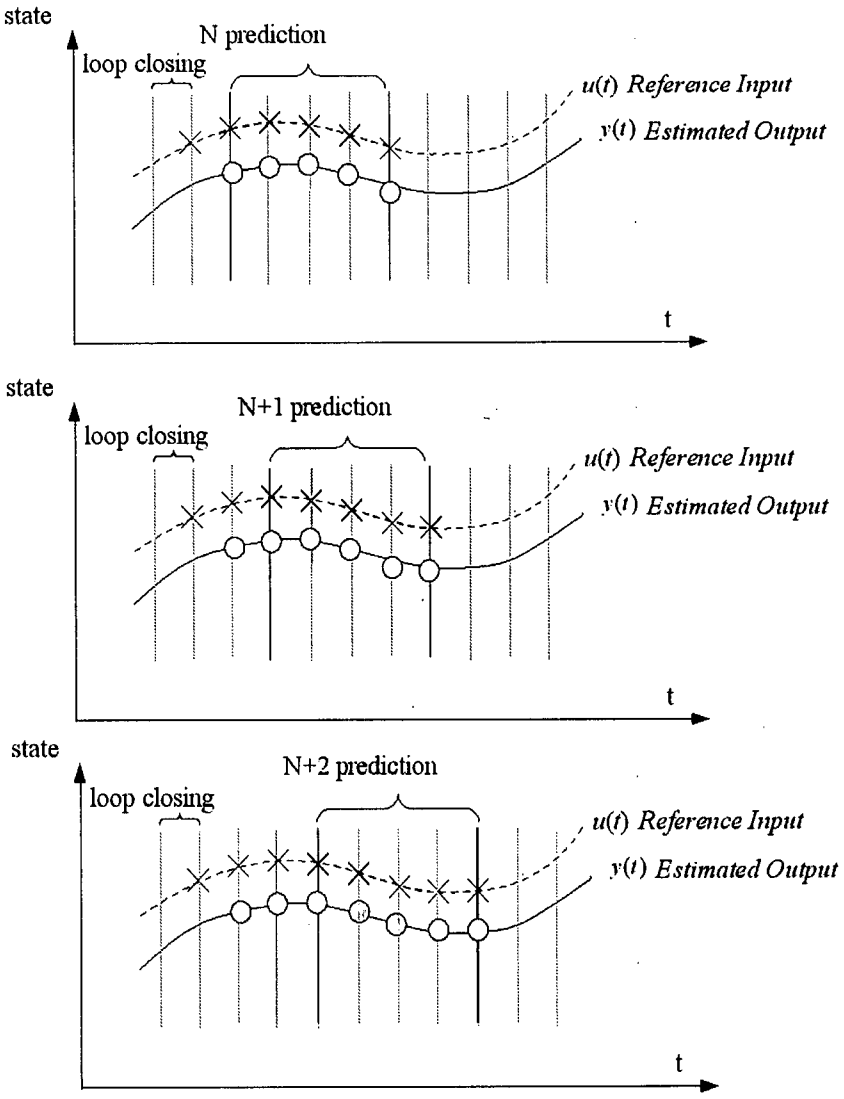


Figure 3.4: Multiple State Estimation Intervals

A parabolic approximation is applied to estimate a history of inputs, as shown in Equation 3.1. In this case, three of the input points are used: the start point, end point, and mid point. The parabolic input was tested in an algebraic solution to the equation of a

second order system by Cullingham [8] and it is proved that the parabolic approximation is fairly accurate across the entire range of the sine wave input signal and adequate when the system input is a continuous function.

$$u(t)=\left\{ a_1 t^2+a_2 t+a_3 \;\middle|\; a_1=\frac{u_0-2u_1+u_2}{2\Delta^2}, a_2=\frac{-3u_0+4u_1-1u_2}{2\Delta}, a_3=u_0 \right\}$$

Equation 3.1

Where:

Δ is a discrete time interval representing half the estimation period.

$u_0=u(t)$, $u_1=u(t+\Delta)$, and $u_2=u(t+2\Delta)$

3.5 System Dynamics Modelling

To implement the Health Monitoring System one must first build the models of the various steering elements and extract the equations relating the states of interest. Both the Steer-by-Wire and power assist loops are implemented using Microprocessor without Interlocked Pipeline Stages (MIPS) processors with high speed serial communication to the control and data acquisition modules. A detailed description of the actual components is included in Chapter 4.

3.5.1 Steer-by-wire Motion Control

Motion control of the Steer-by-Wire actuation is achieved using the FPGA Based Motion Control Module (TMAC), an FPGA based motion control module. The module performs closed loop servo position control of up to 2-axes and it executes both the interpolation task and the measurement of quadrature encoder input, and outputs the

required analog voltage to the amplifiers. A layout of this control process is shown in Figure 3.5.

A supervisory PC is used to communicate with the embedded system. This system operates using a highly object oriented framework of methods running within the MIPS master which allows simple motion, monitoring and communication tasks to be easily achieved. The motion commands result in coarse increments being transmitted to the FPGA module which in turn interpolates these and performs the functions already described. The current amplifier within the control loop utilizes tachometer feedback to allow easier and more robust tuning of the system.

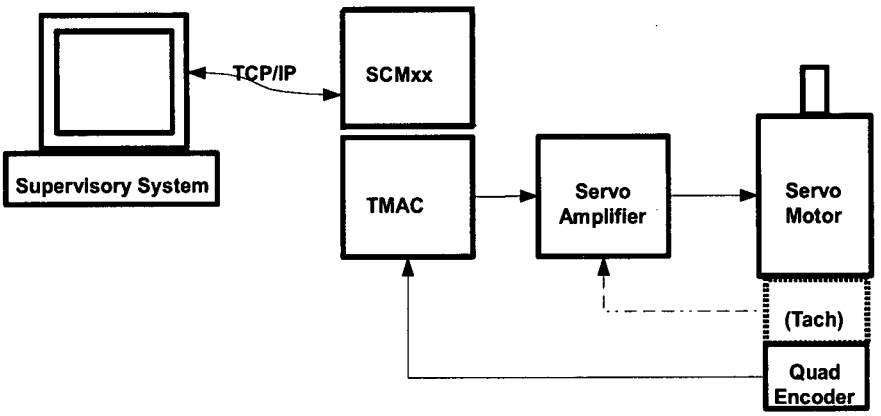


Figure 3.5: Closed-loop Position Control for Steer-by-wire Actuation

Velocity Loop

The block diagram of the internal velocity loop is shown below in Figure 3.6 and has 3 variable gains, (K_{ref} , K_{loop} and K_{tach}).

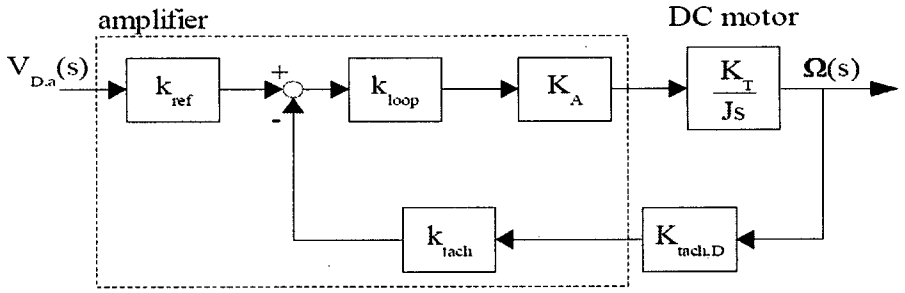


Figure 3.6: DC Motor and Servo Amplifier Control Algorithm

The velocity response of the servo drive can then be modelled as a first-order lag as shown in Equation 3.2.

$$\frac{\Omega(s)}{V_{D,a}(s)} = \frac{K_{vel}}{1+\tau_{vel}(s)} \qquad \text{Equation 3.2}$$

where

$$\begin{cases} K_{vel} = \dfrac{k_{ref}}{k_{tach} k_{tach,D}} \\ \tau_{vel} = \dfrac{J}{K_T K_A k_{loop} k_{tach} k_{tach,D}} \end{cases} \qquad \text{Equation 3.3}$$

The parameters of this firs-order function can be verified by applying a step input to the actuator and measuring the steady–state gain and rise time.

Position Loop

The controller used by TMAC is a "lead/lag" filter. After having identified the parameters for the velocity loop, the lead/lag filter parameters are chosen so that the

closed position loop response is a second-order lag with a natural frequency of ω_n and a damping ratio of ζ. The lead/lag filter can be represented by the following transfer function in the Laplace domain:

$$H_{lead,lag}(s) = K_{lead,lag}\frac{(s+a)}{(s+b)}$$

Equation 3.4

Proper tuning of the lead/lag filter can greatly simplify the analysis of the system's behaviour. Including the first-order transfer function of servo amplifier and DC motor as shown above results in the continuous-time block diagram of position control in the Laplace domain, as shown in Figure 3.7.

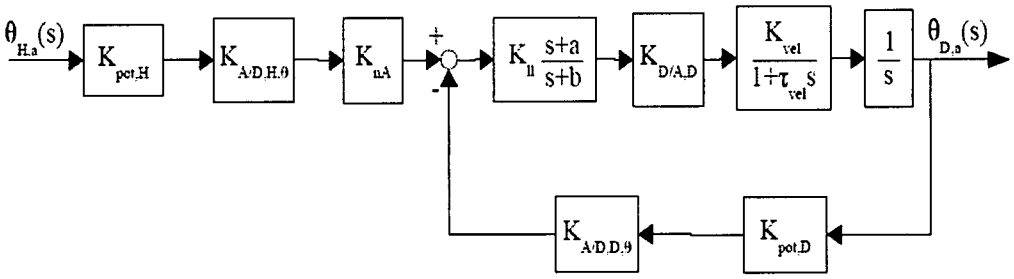


Figure 3.7: Lead/lag Filter in Closed-loop Position Control

The global transfer function of this control loop can be expressed as:

$$H(s)=\frac{\dfrac{K_{vel}K_{DIA,D}K_{lead,lag}(s+a)K_{nA}K_{potH}K_{AID,H,0}}{s(1-\tau_{vel}s)(s+b)}}{1+\dfrac{K_{vel}K_{DIA,D}K_{lead,lag}(s+a)K_{pot,D}K_{AID,D,\theta}}{s(1-\tau_{vel}s)(s+b)}}$$

Equation 3.5

By choosing $a=\frac{1}{\tau_{vel}}$, the effect of τ_{vel} is cancelled. Furthermore, it is possible to obtain a second order global transfer function characterized by its natural frequency ω_n and the damping ratio of ζ , by selecting b and $K_{lead,lag}$ in the following manner :

$$\begin{cases} b=2\zeta\omega_n \\ K_{lead,lag}=\dfrac{\omega_n^2\tau_{vel}}{K_{pot,D}K_{AID,D,0}K_{DIA,D}K_{vel}} \end{cases}$$

Equation 3.6

In order to adapt this analysis to discrete time (Z-transform) form, the following simple transformation is used, where T_s is the loop-closing period :

$$S=\frac{z-1}{T_s}$$

Equation 3.7

This gives the following discrete expression for the controller :

$$K_{lead,lag}\frac{s+a}{s+b}=K_{lead,lag}\frac{z-A}{z-B},\ where\ \begin{cases} A=1-aT_s \\ B=1-bT_s \end{cases}$$

Equation 3.8

The filter can now be implemented as a digital controller for numerical integration within the object-oriented software environment.

3.5.2 Power Assist Motion Control

A separate power assisted steering system is used to supplement the steering torque of the Steer-by-Wire system and to act as a back up system in case the Steer-by-Wire loop fails. Ideally, the steering system would be modular and the power steering system would input from a torque sensor mounted on the steering wheel and knowledge of current position error in the system; this enables easy motion of the system from rest and enhanced dynamics during transients. However the prototype used for this research has only two loops and the manual steering component is not physically available. In order to approximate the ideal system, the required steering torque from the power steering system is assumed to be proportional to the current position error.

$$T=k_{torsion}\cdot\Delta\theta$$

Equation 3.9

Where

$\Delta\theta$: current position error,

T : the steering torque and

$K_{torsion}$: the steering column stiffness or the ratio between them.

This approximation allows the input of what is essentially a torque reference input without requiring an additional torque sensor. At the same time, one can adjust its magnitude by simply modifying the gain in this open torque loop, as shown below:

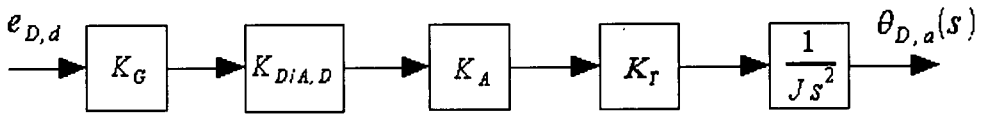


Figure 3.8: Open-loop Torque Control

The input is position error and output is position. K_G , $K_{D/A,D}$, K_A, K_T, J represent torque loop gain, drive unit DAC gain, amplifier gain, torque constant and inertia of servo system respectively. Since the position loop and torque loop are mechanically connected, the first step in the design of the system is to model the dynamics and examine the required torques in the various modes of operation.

3.5.3 Dynamic Performance Simulation

An off-line simulation of system dynamics offers an opportunity to examine the dynamic performance, to verify the parameters and balance the control loops. This modelling was performed by using MatLab/Simulink before it was implemented within the embedded system.

The Steer-by-Wire loop includes a DC brush motor and a ball screw connected to the steering rack to move the wheels according to position reference. The power assist loop includes a second DC brush motor connected to the same steering rack and pinion

mechanism through a pair of pulleys. Thus, the steering rack is moved by the combined outputs of both motors. The complete system diagram is shown in Figure 3.9, T_1 and T_2 represent the torque outputs from the two driving motors. The reaction force from the tires is included in F_{load}, which is estimated using approaches adopted in [27] , see Appendix A.

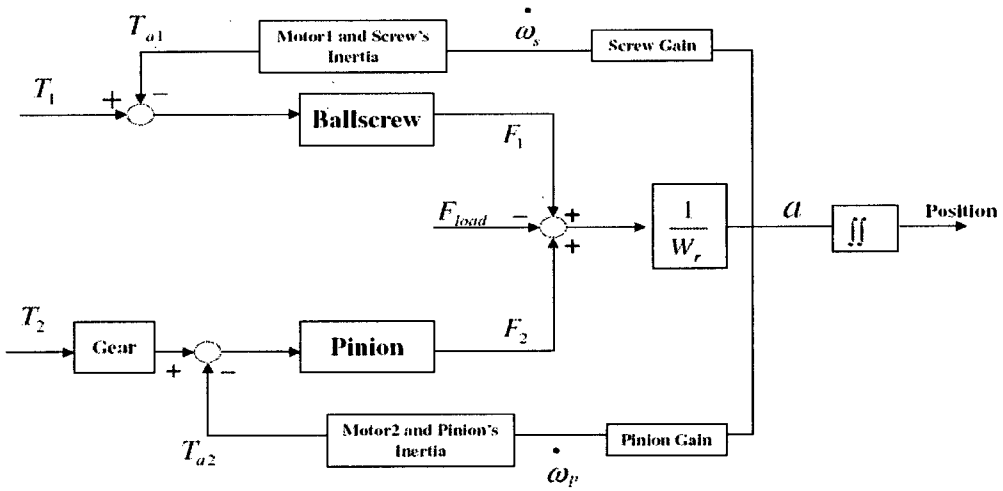


Figure 3.9: Ballscrew and Pinion, Rack Coupling Mechanism

The integration of both of the control loops and the actuation mechanisms results in a continuous-time block diagram of the overall system model as shown in Figure 3.10. Primarily reference input is sinusoidal wave, which represents the common input from steering wheel. Closed-loop position control and open-loop torque control are coupled by the actuation mechanism developed in the application design. To protect against motor overload, saturation is used to limit the current outputs from amplifiers.

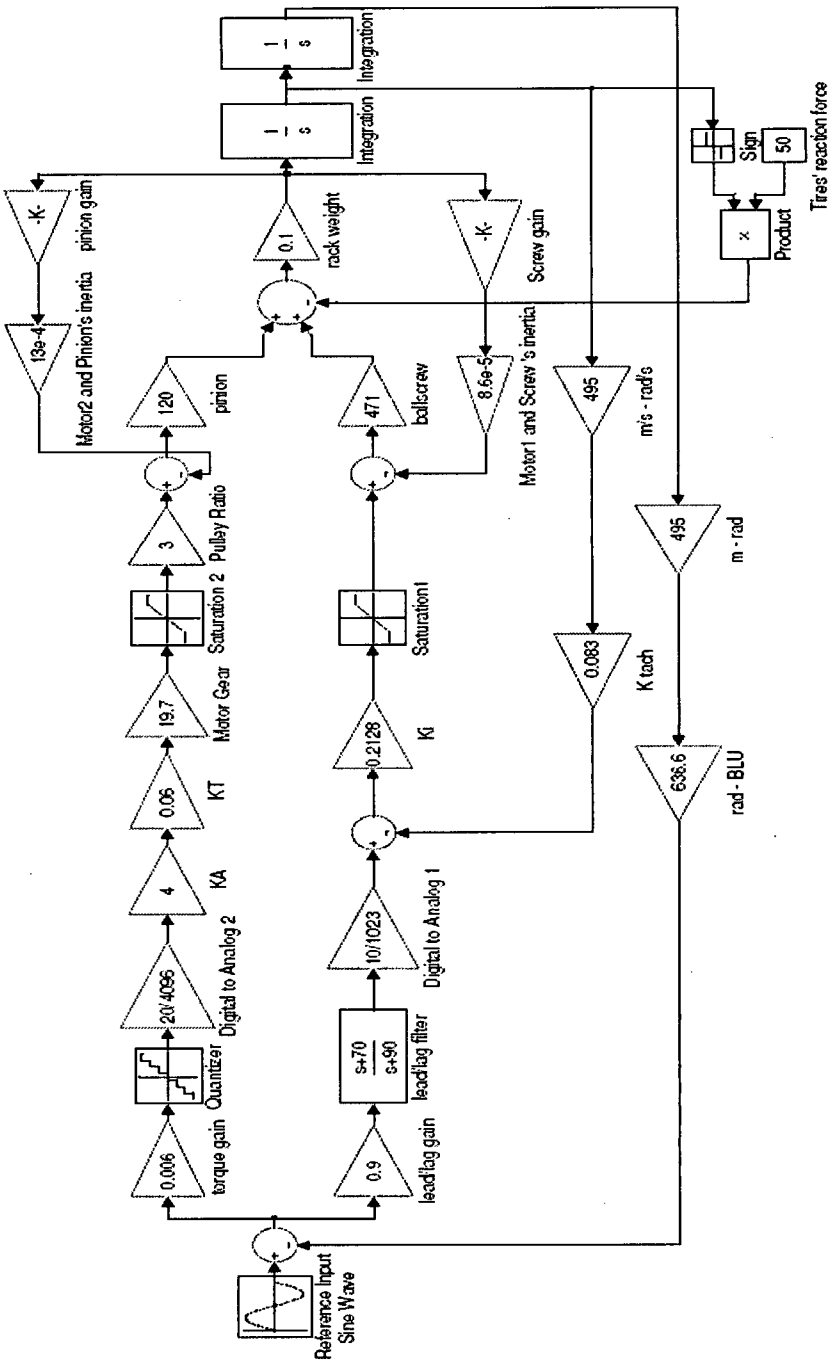


Figure 3.10: Block Diagram of Steer-by-wire and Power Assist Application

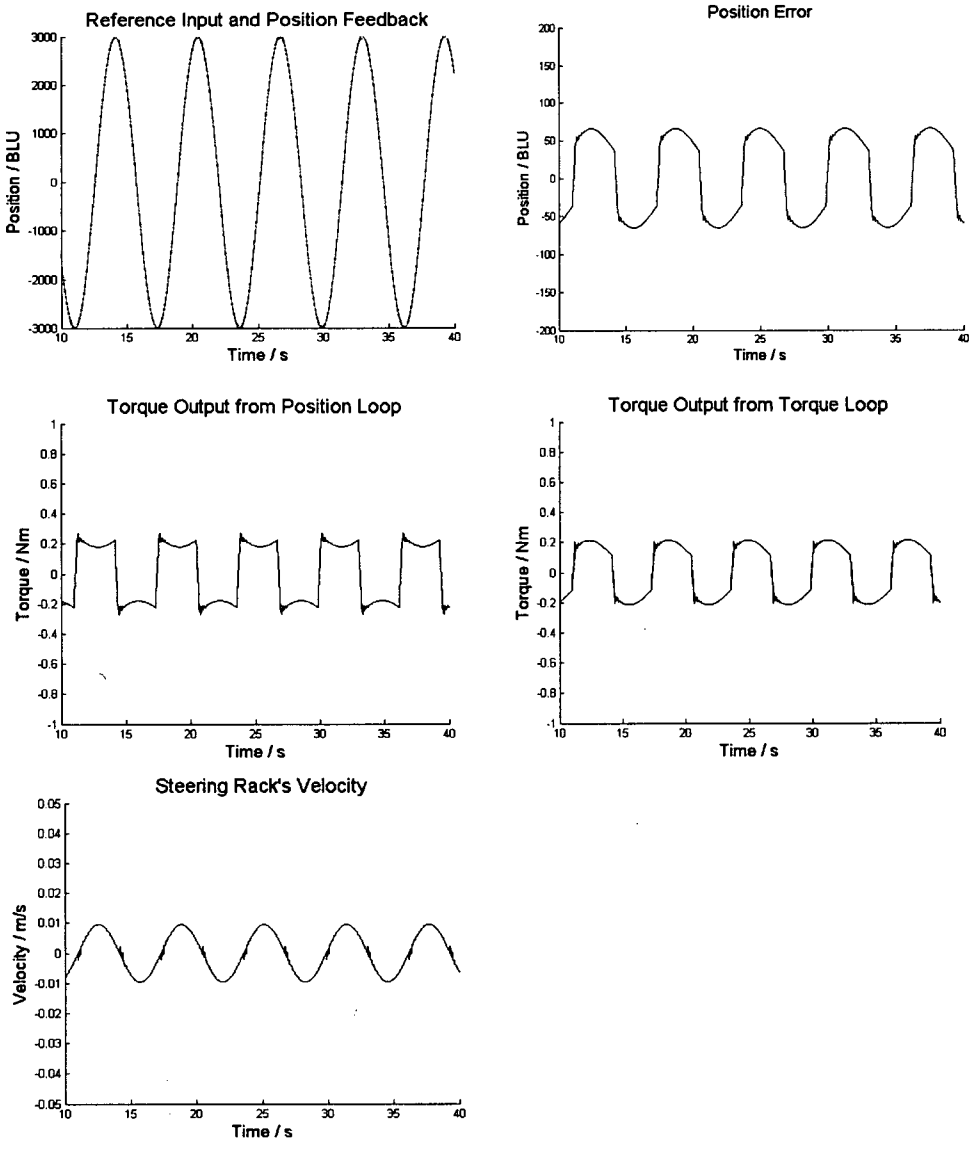


Figure 3.11: Simulation of Output States in Matlab

After initially identifying all the parameters, and running the simulation in Matlab, the system's dynamic performance can be assessed with the reference to the position

feedback, position error, torque outputs from both loops as well as the steering rack's travelling status. A simulation of these output states is shown in Figure 3.11 with a sinusoidal input at 1Hz.

Off-line simulation allows one to check system performance and stability, to tune the system parameters, and at the same time create a benchmark for implementing the Health Monitoring System in the embedded application

3.6 Health Monitoring Framework

The desired output from a health monitoring layer is a set of values that are indicative of the current status for a given component. Each unique health monitor should aggregate a set of indicators, as well as all of the methods that describe those indicators. The Health Monitoring indicators are model-based, as described in Section 3.4, which makes each indicator dependent on the current system state and the input applied over a period. The Health Monitoring layer necessarily relies on the systems' sensors, observable states, and controller inputs. Thus an abstraction of the hardware system as well as the current error status of all the state variables are needed if the Health Monitoring layer is to function properly.

The Health Monitoring needs to be integrated with the error detection functions so that the health indicators can be generated and, if disagreements are found, faulty sensor can be detected and disabled.

Figure 3.12 shows a flow chart of the data used by the health monitoring system during normal operation. During the first step, the RunModel method is used, which updates the model elements, and generates an estimation for system states. In this

application, the estimated states include steering rack's position and velocity as well as current outputs from amplifiers. At the same time, by acquiring the physical sensors, (such as the tachometer, encoder and potentiometer), one can also obtain the actual physical states of the system. Data from the health monitoring system and the sensors must then be compared.

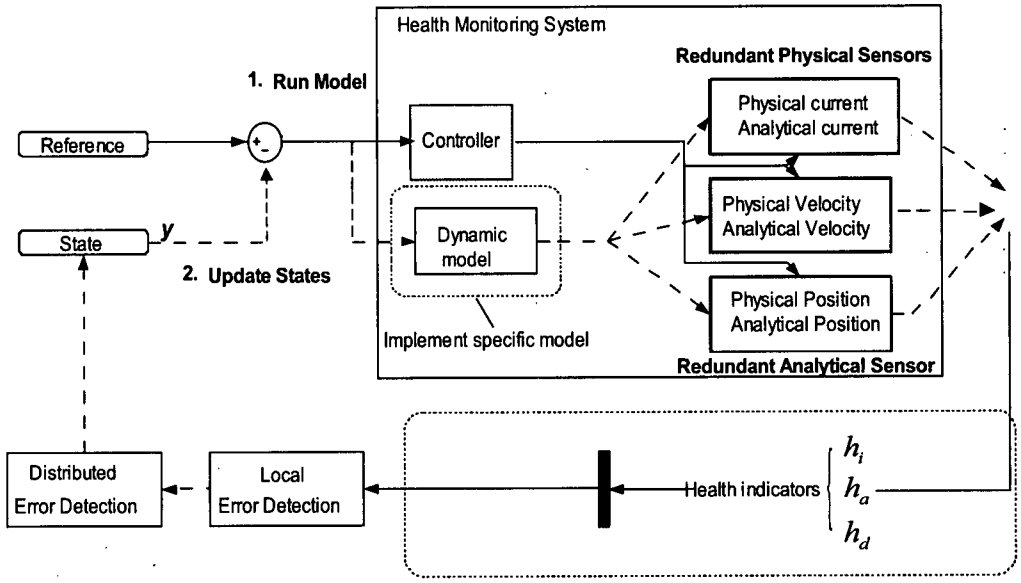


Figure 3.12: Health Monitoring System Data Flow

The second step in the data flow for the Health Monitoring system is updating the system states with the values produced by the Distributed Error Detection routine. This then allows the calculation of the output values for each of the Health Indicators.

3.7 Summary

A combination of Steer-by-Wire mode and power assisted mode is proposed as the steering strategy for the purpose of experiment. Then it introduces the fault-tolerant

platform that the health monitoring functions are designed to interface with. This platform runs on multiple embedded hardware.

The development and potential benefits of the Health Monitoring platform are discussed and the three health indicators introduced by Cullingham are defined and integrated into the system.

This chapter also explains the process of building simulation models to examine the behaviour of the combined steering system and shows how these may be utilized to determine the effectiveness of the proposed system models. System response to a typical sinusoidal input are presented.

Finally the Chapter describes how health monitoring is integrated with the physical fault-tolerant system and how the health indicators are obtained from the comparisons of physical sensors and analytical sensors within the error detection process.

Chapter 4: Design and Test of a Laboratory Steer-by-wire System

4.1 Introduction

Model-based Health Monitoring is performed by comparing the outputs of a real-time model with actual sensor values. Demonstration of the Health Monitoring scheme then requires a realistic prototype steering system with suitable sensors and monitoring equipment.

This chapter first provides a detailed description of hardware components of the Steer-by-Wire application with the power assist, and the assembly and setup for the steering and actuating function. This experimental system has been implemented in the Product Development Laboratory at UBC. The second part describes the experiments used to demonstrate performance of the Health Monitoring, which is designed to detect failure modes in different scenarios. At last, a fault diagnosis strategy is proposed to verify the fault symptoms and causes and supervise the overall behaviour of the specified application.

4.2 Description of Steer-by-wire Setup

4.2.1 Embedded System Configuration

The Health Monitoring system described in this thesis sits atop a prototype embedded system that was developed for the fault-tolerant system architecture in Mathew Bouvier's previous research [21]. It is a distributed computing platform based on the exact redundancy strategy, which is applied to all the levels of configuration, including sensors, actuators and ECUs. However, in this thesis, the major objective is to provide a model based software system with the ability to monitor the health conditions of the physical hardware in real time. The idea is to allow additional redundancy to be achieved at a lower cost and provide a flexible tool for a variety of applications.

Under normal operating conditions, it is expected that the Steer-by-Wire system must be controlled by multiple computers. The distributed computing environment introduced by Bouvier included error checking and data sharing via CAN bus network communications. To isolate and test the health monitoring functions, the experiments done described in this Chapter involve only one ECU's however no network based exchange of fault data is transmitted between them. Figure 4.1 shows the ECU's and other electronic components such as amplifier, power supply and communication interface used in the system. The majority of the embedded computer configuration and basic programming was completed by Bouvier, who set up a distributed fault tolerant architecture with the ability to tolerate faults at both sensor and ECU's levels.

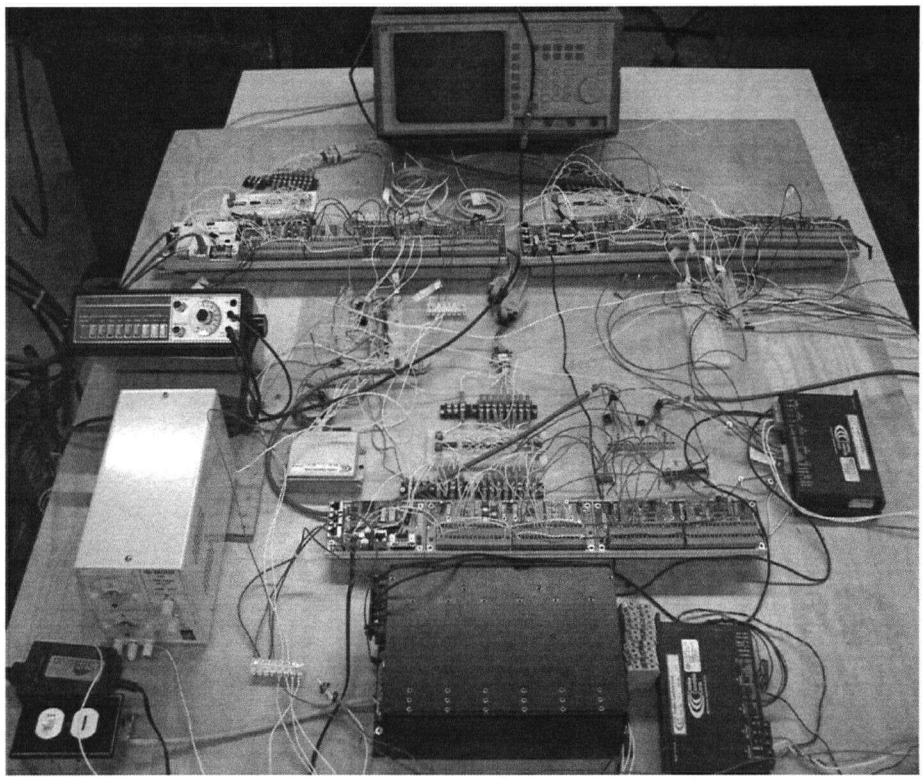


Figure 4.1: Embedded Computer Configuration

Embedded Computing Platform

The embedded computing platform is based on MIPS processors of varying performance. The characteristics of the SCM20 and SCM40 modules are shown in Table 4.1.

Parameter	SCM20	SCM40
CPU speed	33MHz	200MHz
Bit size	32	64
SDRAM	8MB	16MB
Flash disk	16MB	16MB
Power supply	24VDC	24VDC

Table 4.1: MIPS Processors Parameters

Each module is mounted on a carrier board which provides Ethernet 10 Base-T, and CAN2.0 network support. The modules communicate with I/O modules from the same manufacturer over a proprietary high speed serial link. The I/O modules used here are:

- Digital output module, 16 channels, each capable of 12 bit PWM output. The board is configured to provide 4 channels of PWM output, and 12 channels of controlled output.
- Digital input module,16 channels of digital input.
- Analog input module, 12 bit, 4/8 channel, programmable input range to +/-10V.
- Analog output module, 12 bit, 8 output channels of -10V to 10V.
- 2 axis position controller, FPGA based, loop closing 4KHz, incorporates encoder feedback and analog out to amplifier.

Software Environment

The Steer-by-wire and the health monitor applications are developed in Forth and run under a real time operating system on th MIPS modules. In order to ensure portability

of the application over multiple hardware platforms, the ANS Forth standard was selected. The object oriented extension used was developed by McKewan, and is ANS Forth compliant; the resulting language supports the major object-oriented concepts such as inheritance, polymorphism, and aggregation.

Within this object-oriented extension, classes are defined between the delimiting words :Class and ;Class. Inside each class definition, instance variables may be declared as any ANS Forth variable, or as an aggregated class. The scope of these variables is limited to the each object declared, and cannot be accessed directly from other objects of the same type. Also inside each class definition, methods are defined between the delimiting words :M and ;M, and the last character of every method name must be a colon (":"). After defining a class (e.g. Class1), an object (e.g. Object1) can be instanced by calling the command Class1 Object1. It is then possible to access a method within that object (e.g. Method1:) by calling the command Method1: Object1. The method ClassInit: is automatically called when the class is instantiated. The method's definition may be changed, which allows the developer to initialize the object's instance variables to a set of initial values.

4.2.2 Electrical Actuation Circuit

As described in section 3.2 the steering actuation strategy adopted for the initial prototype provides both Steer-by-Wire and electrical power assist loops but no manual steering facility. The two actuators are connected through different means to the steering mechanism operated in parallel so that the steering functionality is improved and actuator fault tolerance is achieved. To meet this requirement, each actuator includes its own

ECU, digital and analog input/output, power amplifier and necessary sensors. The embedded computer platform interfaces with the steering system via the actuation circuit shown in Figure 4.2.

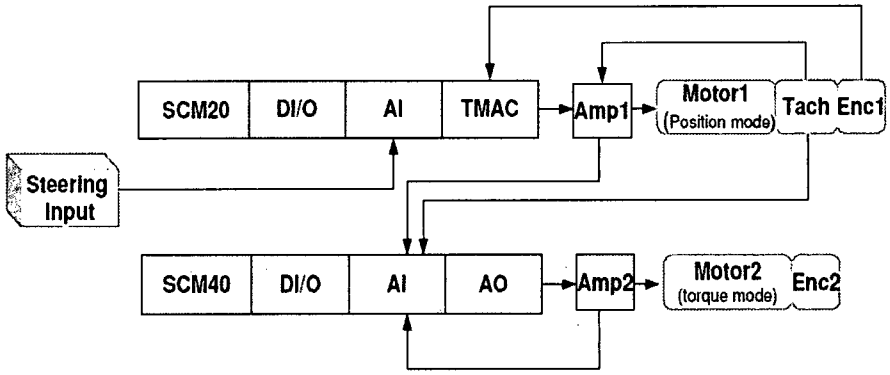


Figure 4.2: Circuit Diagram of Actuators

The SCM20 runs the first motor in the closed loop position control; this motor drives a ballscrew which in turn powers the steering rack, (Steer-by-Wire loop). The command position of the steering wheel is measured by the analog input module and passed by the ECU to the motion control module. The motion control module performs smoothing of the position input and closes the position loop, (outputting an analog voltage to the power amplifier). The brush type DC motor actuator provides both position feedback to the motion control module through an encoder and velocity feedback direct to the power amplifier through a tachometer.

The SCM40 operates the second motor in the open loop torque control mode, (power assist loop). In most modes of operation the torque command would be provided

by a torque sensor attached to the steering wheel, (whether or not the steering wheel is physically connected to the rack) and the complete steering system would then have parallel torque and position inputs. In this case there is no input torque signal from the steering wheel. For the purpose of testing the health monitoring system, the torque reference signal was taken from the applied torque exerted by the Steer-by-Wire loop actuator. The SCM40 module determines the magnitude of the torque being commanded from the analog input module, which is used to detect the current output from the first motor's amplifier.

Two amplifiers used in the application are identical and the motors driven by them are different. The description of these components are shown in the tables of Appendix D.

4.2.3 Laboratory Steering Assembly

The experimental steering system is built on a three-wheeled chassis as shown in Figure 4.3, where it gives side view and front view. This testbed includes all required electrical steering components such as motors, ballscew or pulley, rack and pinion as well as steering rack and tie rods.

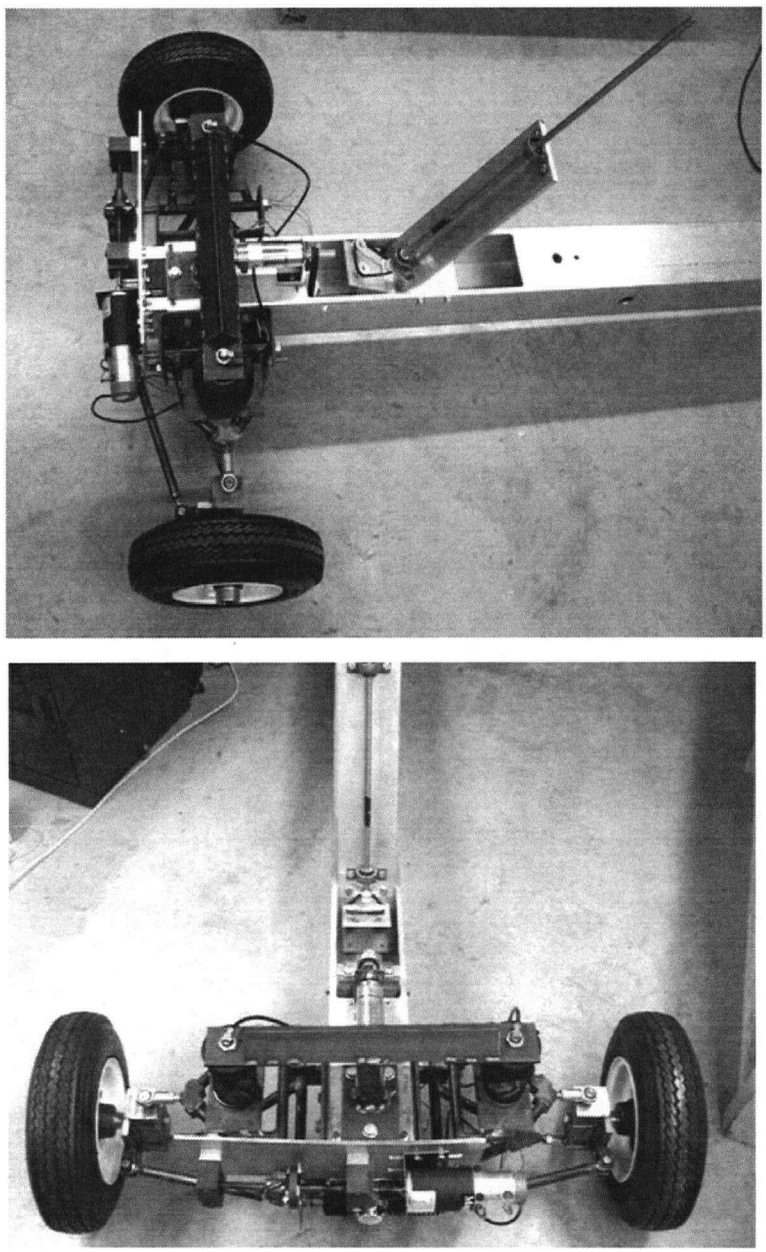


Figure 4.3: Steer-by-Wire with Power Assist Testbed

In this assembly, the leftmost motor in the picture is operated in position mode, (Steer-by-Wire loop). This motor is coupled to a ballscew and the rotational displacement of this ball screw is converted into linear motion of a nut, which moves the steering rack and therefore steers the two front wheels. The ballscew used is 0.5 inches diameter with a lead of 0.5 inches, the large lead angle allows easy back driving of the ballscrew, (this is a required feature of the design and forces the selection of a drive motor with high torque capacity).

The second motor sitting on the chassis is the one running as power assist. The motor is much smaller but has both an integral epicyclic gearbox (ratio 1:19.7) and is driving a steering pinion via a pair of pulleys with ratio 1:3 so that enough driving torque is available.

The steering column has no direct connection with steering rack, its displacement is measured by double potentiometers, which are read by the analog input module.

4.3 Instantiation of Health Monitoring System

Health Monitoring is a model based approach that detects the system's faults and diagnoses the failure modes by comparing the output of a system model to the actual output from system, and generating the differences between them. To monitor the health conditions of the physical hardware in real time, a real time "virtual system model" must be programmed and implemented in the embedded computing system. The control loops and vehicle dynamics of the system are approximated in this case as a simple second order system with the undamped natural frequency and damping ratio being determined by experiment.

Each element of the model previously simulated in MatLab/Simulink is implemented as a software object within an objected-oriented Forth language framework, which resides in the ECU's and is capable of real-time operation. The overall behaviour of the power assisted Steer-by-Wire system is thus provided by a collection of software objects. These objects do not need to be coded, they are instantiated from class definitions. The class definition of an object defines it completely, and specify its instance variables and methods. Multiple objects (with different names) can be instantiated from the same class definition within a given object–oriented model.

Figure 4.4 shows the detailed description of the classes from which the on-line Health Monitoring utilities are instantiated and executed.

Objects instantiated from the classes, variables as well as methods are directly used for data processing by passing the data via the stack according to the control flow. Instance variables and methods of the classes are specifically introduced in Appendix B.

Counterparts of actual outputs of physical sensors can be obtained from the instance variables defined in the corresponding objects. This is known as the analytical redundant sensor, which joins in the detection of sensor faults and protect against continuous system degradation. As discussed in section 3.6, the output from the analytical sensor can be used by the Local Error Detection object and the redundancy management architecture. In a normal operation, each state used by the model is dependent on a previous model estimate. The model adapts to small deviations between these expected and observed values, and it treats large differences as faults.

Health Monitoring Classes	Description
InputandFeedback	*Acquire sensors information from the input and feedback source (every 10 ms)and make a record of them*
Parabolic	*Perform the parabolic approximation based on Equation 3.1 and prepare for the numerical integration*
LeadLag_Health	*Represent the position loop's Lead Lag filter, gains and saturation in discrete expression based on Equation 3.8*
PID_Health	*Represent the torque loop's PID filter (only proportion normally) and gains*
Physicalsystem	*Aggregate results from both loops. apply vehicle dynamics and numerical integration of acceleration and velocity*
Execinst	*Instantiate the classes above, execute the methods and close the position loop*

Figure 4.4: Health Monitoring Classes Description

4.4 Test Setup and Results Achieved

4.4.1 Health Monitoring Configuration

The embedded computing system was run with a loop closing and sampling interval of 10 ms. The health monitoring model was configured with an estimation scale of 5 samples, for an estimation interval of 50 ms. The parameters used by the dynamic model for the tests are shown in Table 4.2. A complete list of model parameters is given in Table C.1.

Parameter	Value	Unit
Position Loop		
Ω_n	18	rad/s
Z	0.75	
$K_{lead,lag}$	0.06	
A	22	Sec^{-1}
B	27	Sec^{-1}
Torque loop		
K_{torque}	0.06	

Table 4.2: Health Monitoring Parameters

So as to have a well controlled position reference input for these tests then this has been generated in software rather than trying to provide a regulated input from the steering column via the analog input module.

4.4.2 Test Results of Injected Failures

The health monitoring system is used to identify when system performance deviates from that expected. From a safety standpoint one can identify a number of potential failure modes and focus the experimental investigation. It might be noted that one is not concerned here with a problem of lack of performance (e.g. not tracking fast transients etc), one is simply concerned with the health of the system as designed.

The steering system is tested in two different configurations; the first is the pure Steer-by-Wire system without any power assist, the second is the complete system including Steer-by-Wire and power assist loops. Clearly the model used in each case simply reflects the components selected.

Pure Steer-by-wire

Figure 4.5 shows the behaviour of the Health Monitoring system in the pure Steer-by-Wire operating condition in the absence of faults. The figure shows the comparisons between states predicted by the Health Monitoring and those measured in the physical system. The selected observers are the steering rack position, the position error (control error signal), the current output from the amplifier (converted to torque), and the velocity of the steering rack. Actual values of these observers are obtained from the motor encoder, tachometer, and the amplifier current monitor.

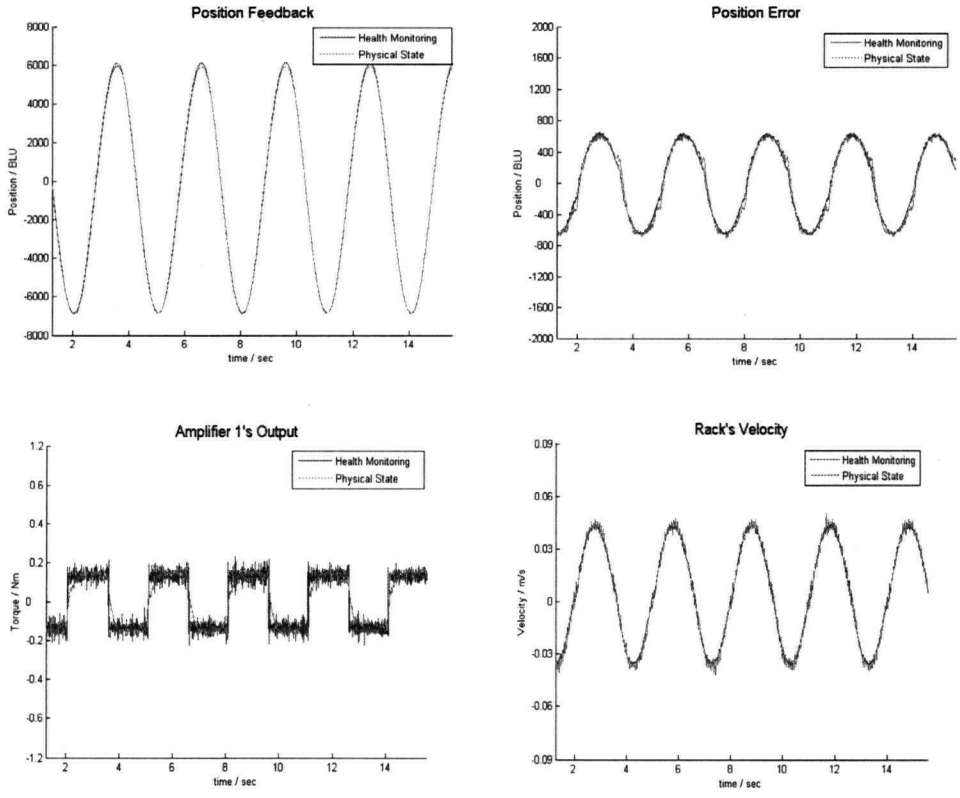


Figure 4.5: Steer-by-Wire in Normal Status

In Figure 4.5, input is a sinusoid wave with 6000 BLU at 0.33 Hz, which represents a steering rate approximately 2.5 rad/s at the steering wheel.

One can see that when there is no failed component or external disturbance, the system states estimated by the Health Monitoring follow their physical counterparts with small error. One can assume then that the system models are fairly accurate and reliable.

Figure 4.6 is also in normal operation but its input has an increasing frequency from roughly 0.5 to 2 Hz. With the varying frequency, we can further prove that Heath Monitoring resembles the system dynamic well.

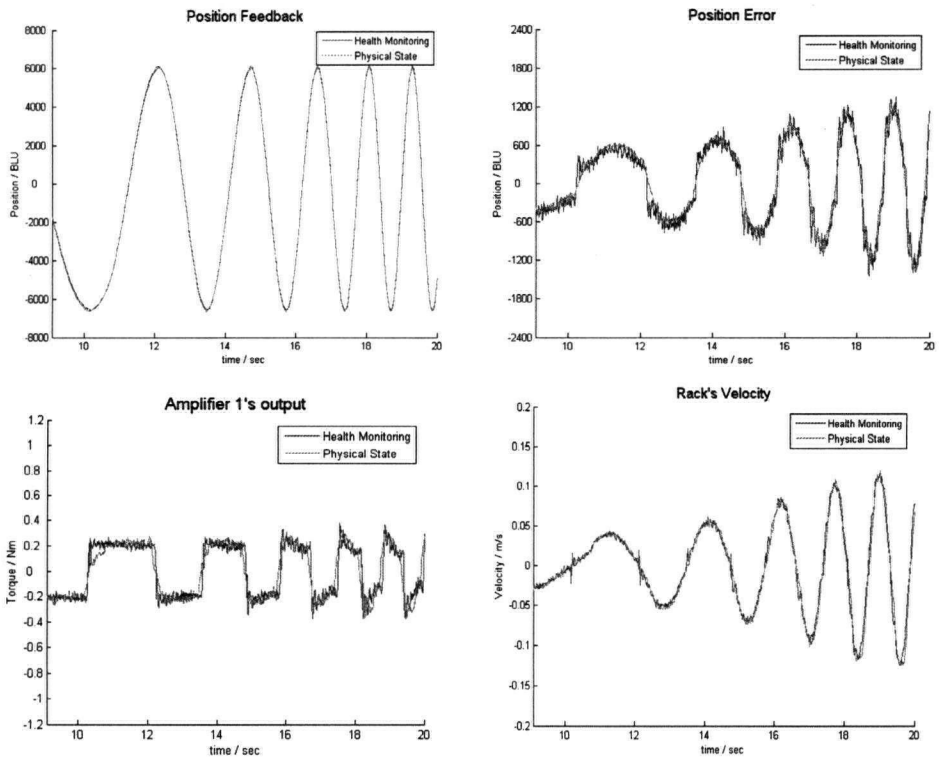


Figure 4.6: Steer-by-Wire With Varying Input Frequency

One failure fault was conducted on one of the transmission components, the flexible coupling, which is used as the connection of motor and ballscrew's shafts Figure 4.7. It is able to compensate for misalignment between shafts and prevent the transmission of overload power. The failure mode is injected by loosing the coupling. A loose coupling results in insufficient driving power transferred from the motor to the steering rack. Consequently the steering rack is not able to travel at the commanded speed and the power output from the motor is apparently reduced. Figure 4.8 shows the comparison results with the injection of the loose coupling failure mode.

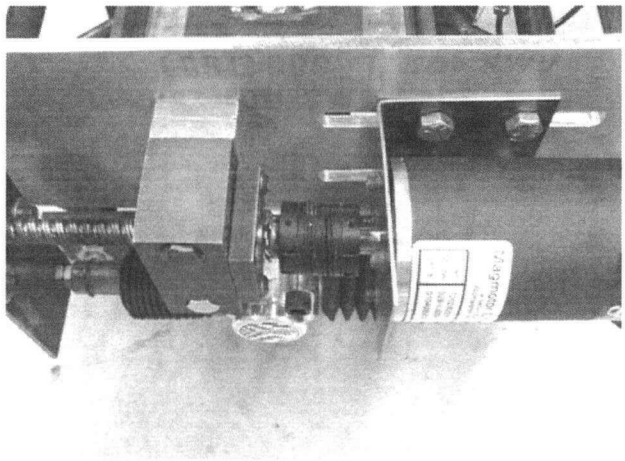


Figure 4.7: Flexible Coupling Between Motor and Ballscrew

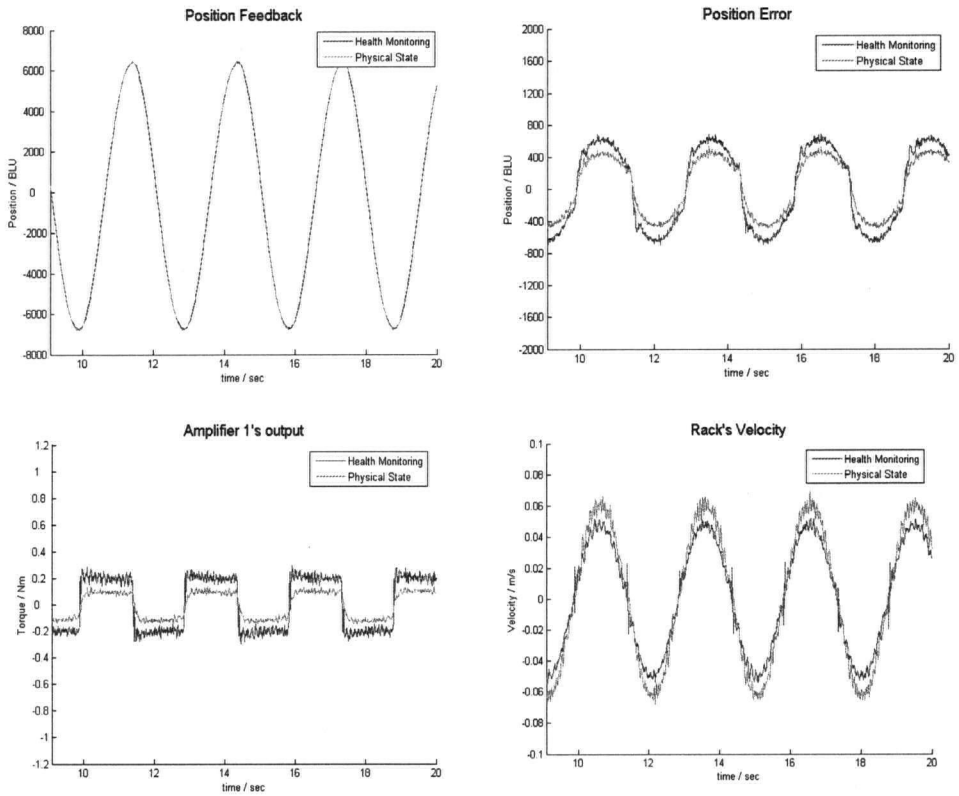


Figure 4.8: Steer-by-Wire With Loose Coupling

One can see that this failure reflects on the position error, current output and rack's velocity as big differences between Health Monitoring estimations and actual system observations, since Health Monitoring is a representation of system in healthy status. Both position error and amplifier's output have bigger Health Monitoring estimations than actual sensors' values. Since the tachometer is coupled to the motor, so it gives a bigger velocity output.

Steer-by-wire with Power Assist

With power assist, we have the second motor to supply the torque and the load on the first motor is relieved. Figure 4.9 gives the normal operation of power assisted Steer-by-Wire condition. To compare it with the normal status of pure Steer-by-Wire, we can see that with the same input, the current output from the first amplifier (the one for the motor in position mode) is reduced for half.

Another failure mode is tested under the power assisted Steer-by-Wire condition and it is shown in Figure 4.10. A resistance force is applied to one side of the wheels, which represents possible scenarios such as mechanical failure of steering rack, tie rods or wheels. So in this case, there are always bigger actual outputs from both of amplifiers and position error. Since the wheels are not able to travel to the required position, both of position feedback and velocity from the sensors show smaller values on this side. Once such faults detected, we may diagnose it as a failure mode similar to a resistance force.

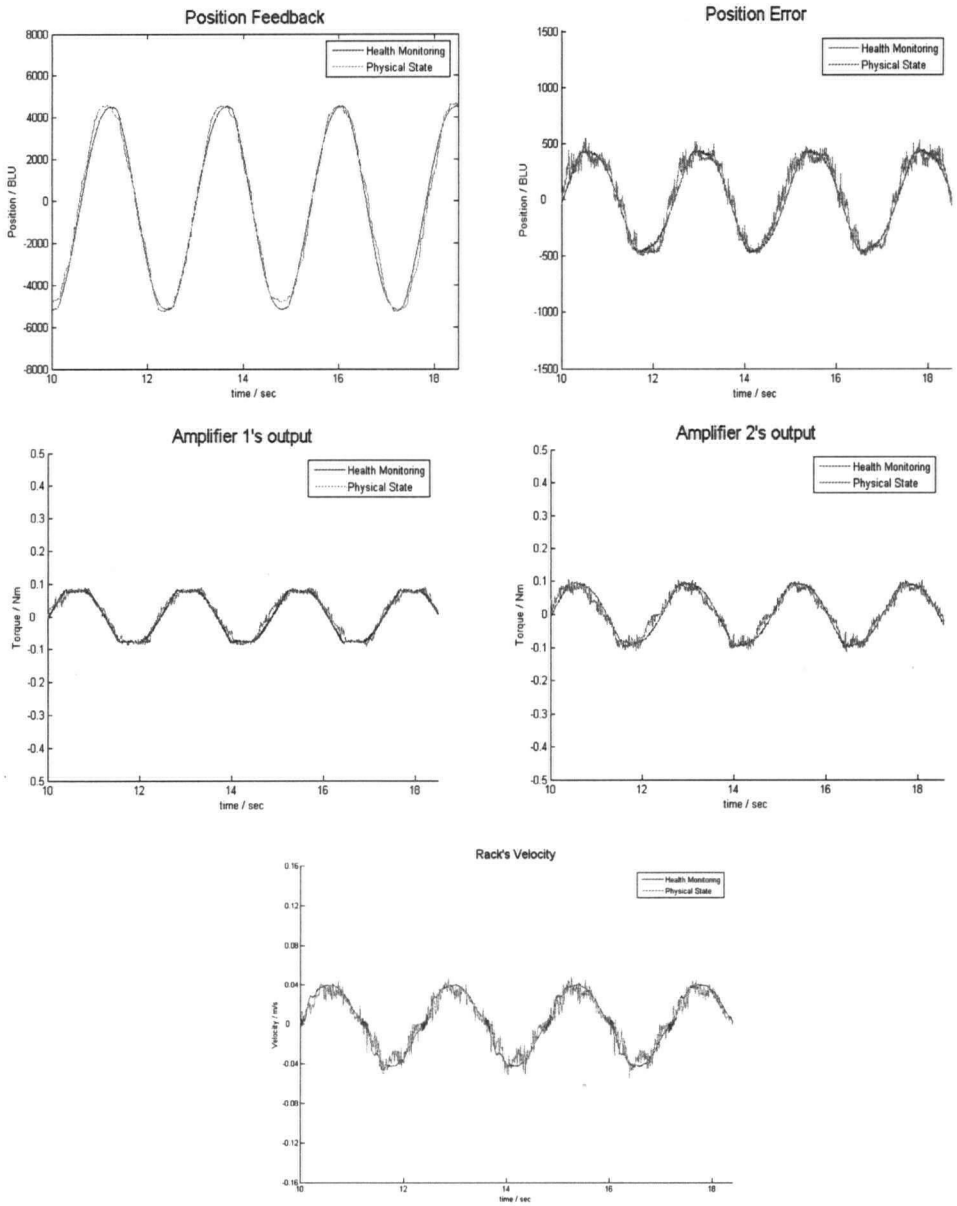


Figure 4.9: Power-Assisted Steer-by-Wire In Normal Status

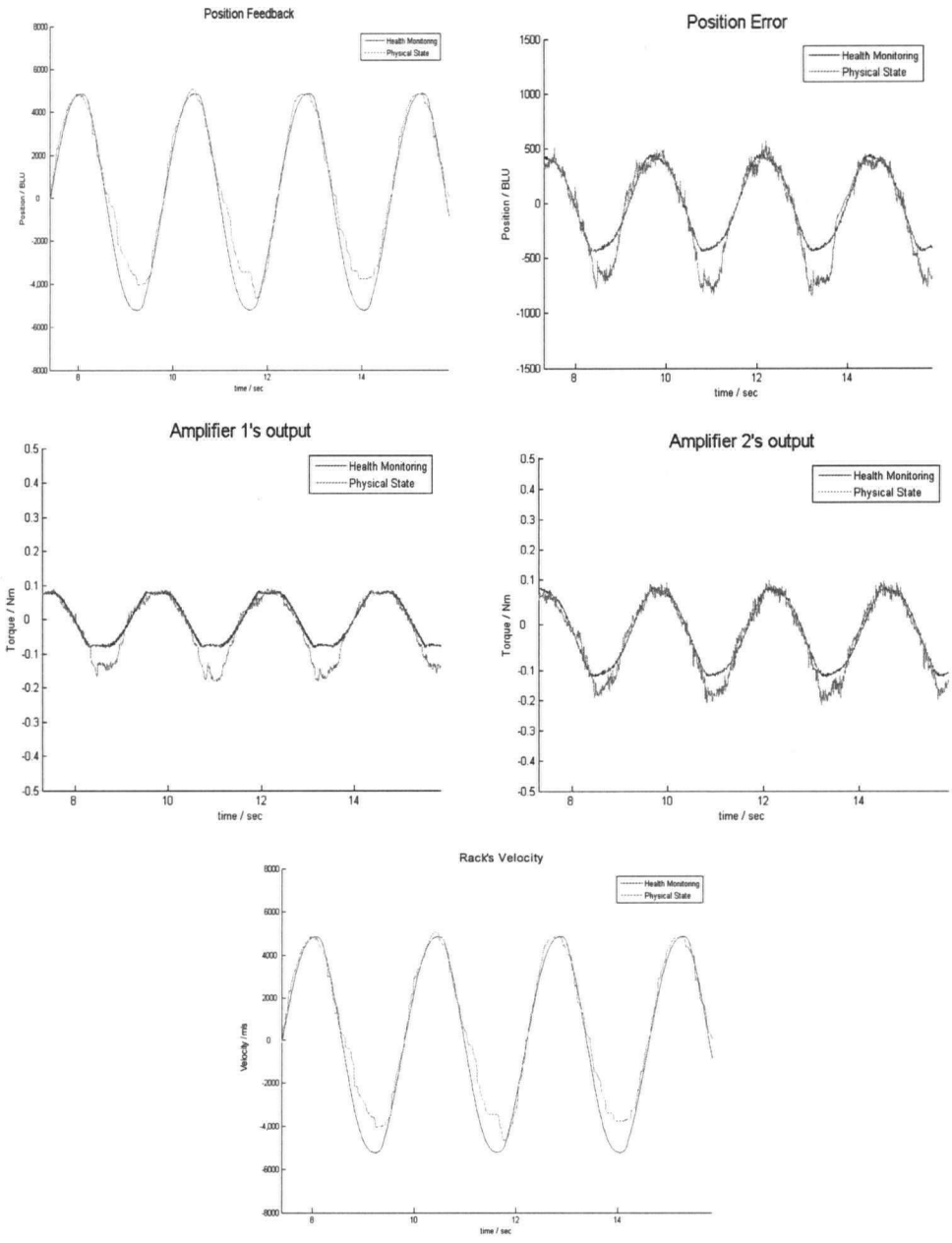


Figure 4.10: Power-Assisted Steer-by-Wire With Resistance On One Side

As shown above, by injecting faults such as modifying system's components, shutting down the power or disturbing data communication, one may be able to detect different fault symptoms by simultaneously supervising different system states such as position, velocity and force.

A subsequent fault diagnosis strategy can be developed by finding the best match between fault symptom and cause. This requires one to develop an adequate system model which characterizes the dynamics of vehicle steering, which means sufficient system dynamical characteristics should be known during the process of steering. Then a complete system model can allow one to distinguish different fault symptoms and locate the fault causes among the failure modes which have been known. For example, in order to have the ability to diagnose faults from external environment, one needs additional force or torque sensor mounted on the wheels to have knowledge of external disturbances acting on the vehicle. At the same time, a more sophisticated filter (e.g. Kalman filter) is also required to improve the performance on estimating the external disturbance. However, such developments are outside the scope of this thesis and no diagnostic module has been introduced yet.

4.5 Summary

In this chapter, the concepts detailed in Chapter 2 and Chapter 3 are illustrated through experiments conducted on a prototype steering system that was designed and built as part of this thesis. The system has been operated in both Steer-by-Wire and "power assisted steer-by-wire" modes.

The Health Monitoring system is implemented in as a virtual system environment which runs in parallel with the control activities on MIPS based processors. This chapter has given a brief account of the software development, the hardware design and the mode of operation of the steering system.

Experiments have been performed to verify the system models used and to demonstrate the ability to recognize component failure or unusual operating conditions. These experiments have examined both a simple Steer-by-Wire arrangement as well as the combination of power assist and Steer-by-Wire loops. The failure modes chosen are simply and fairly gross in nature but were easily recognized by the system. Clearly a precise method of examining outputs and assigning tolerances to the allowable errors between model and measured states is still required.

Chapter 5: Summary

5.1 Conclusions

By-Wire systems and associated embedded computing applications are expected to have wide-spread use in personal vehicles. This results from the improvements, that may be attained in performance, flexibility and efficiency as well as the chance of providing additional feature. However, the potential risk involved in these electrical systems, means that one must guarantee vehicle and passengers safety. At the same time one must provide the required level of safety efficiently, i.e., cost is an issue in these applications.

Previous approaches developed by other research groups to attain these goals have relied on high redundancy levels. The Boeing 777 controller is comprised of three triple redundant ECU's, while the X-by-Wire automobile project is controlled by three fault-tolerant nodes, each of which is locally comprised of four ECU's. Most of the approaches resort to exact redundancy, specific physical sensors to detect faults and computationally expensive techniques.

The architecture presented in this thesis contains a health monitoring layer within the overall framework; this framework functions as a fault detection and diagnosis mechanism to protect the monitored system against degradation when one or more sensor failure occurs. Since it is based on the system dynamic model, the health monitoring

system can provide the awareness of the health of the system's components by estimating the performance of sensors and actuators and detecting their faults, without the burden of adding extra physical sensors and without adding high computational loads. Consequently its analytical redundancy feature provides the safe-critical system with the fault tolerant capability at minimal cost.

The concepts and architecture described in this thesis are illustrated through application to a prototype automobile Steer-by-Wire/Power Assisted system. The experimental testbed was designed and assembled as part of this thesis. The health monitoring architecture described was implemented within the object oriented environment of the embedded computing platform. Testing has shown that the health monitoring system is capable of representing normal behaviour, and also of detecting component failure or unusual operating conditions. A fault diagnosis strategy was proposed to match the fault symptoms and causes, and distinguish different failure modes.

5.2 Recommendations for Future Work

Three kinds of health indicators and a framework used to organize them were introduced in this work. However, the existing fault-tolerant architecture does not include this analytical redundancy within the local and distributed error detection procedures. These fault detection functions should be established in order to implement a completed health monitoring layer,

The sensitivity of the technique needs to be further tested. The accuracy of the health indicators affects the thresholds observed when the outputs of the analytical sensor represents the healthy status and are compared with the actual system states. An analytical

sensor with low accuracy can not be trusted alone and it would be only involved in the voting scheme to arbitrate between two sensors that do not agree after the third has failed. Meanwhile, the communication of the health monitoring states needs to be implemented so that the health monitoring system is able to collect data from multiple ECUs, where each of the system states are measured and shared across a network. This requires a distributed functionality of the system to be developed.

Non-linearities should also be taken into consideration in the health monitoring system so that it can accurately represent the real system. These non-linearities include saturation, friction, relay and backlash and have been incorporated in the health monitoring system by Cullingham.

Bibliography

1 Laprie, J.C., Dependability: a unifying concept for reliable computing and fault tolerance. ANDERSON, T. (Ed.): "Dependability of Resilient Computers", chapter 1. Blackwell Scientific Publications, 1989

2 Paul Yih, Thesis:, "Steer by wire: implications for vehicle handling and safety". Stanford University, 2004

3 Kurishige, M., T. Kifugu, N. Inoue, S. Zeniya, and S. Otagaki, A Control Strategy to Reduce Steering Torque for Stationary Vehicles Equipped with EPS, SAE Technical Papers, 1999

4 R. McCann, Variable effort steering for vehicle stability enhancement using an electric power steering sytem, SAE Technical Paper Series 2000-01-0817, Mar. 6-9, 2000, p. 1-5.

5 D.Peter and R.Gerhard, Electric power steering - the fist step on the way to steer by wire, SAE Technical Paper Series 1999-01-0401

6 C.L.Seacord and D.K.Vaughn, Preliminary design for a digital fly-by-wire flight control system for an F-8C aircraft, NASA Centre for Aerospace Information NASA-CR-2609, 1976

7 H.Inagaki, K.Akuzawa and M.Sato, Yaw rate feedback braking force distribution control with control-by-wire braking system., In the Proceedings of the International Symposium on Advanced Vehicle Control, Nagoya, Japan, 213-217, 1998

8 Nicholas Eoghan Cullingham, Thesis: "Definition of a Distributed Health Monitoring System: Application to the Design of a Hydraulic Marine Steer-by-Wire System", The University of British Columbia, 2005

9 Hiller, M, Using Software to Handle Data Errors in Embedded Control Systems, Chalmers University of Technology, 1999

10 Laprie, J. C., "Dependability - Its Attributes, Impairments and Means", Predictably Dependable Computing Systems, 3-24, Springer-Verlag, Berlin; Heidelberg; New York, 1995

11 Kopetz, H, Design Principles for Distributed Embedded Applications, Kluwer Academic Publisher, 1997

12 PALBUS team, PALBUS-Validation of Dependable Distributed Computing Systems, Published on the internet, http://www.sp.se/electronics/rnd/palbus/ [accessed on 06/14/2005], 2001

13 Lamport, L.,R.Shostak and M.Pease, The Byzantine Generals Problem, ACM Transactions on Programming Languages and Systems, 382-401, July, 1982

14 Yeh, Y. C, Design Considerations in Boeing 777 Fly-By-Wire Computers, In Proceedings of the Third IEEE International High-Assurance SystemsEngineering Symposium, 64-72, 1998

15 X-By-Wire team, X-By-Wire, Safety Related Fault Tolerant Systems inVehicles, Final Report, http://www.vmars.tuwien.ac.at/projects/xbywire/docs/final.doc [accessed on 10/25/2005], 1998

16 Chow,E.Y. and A.S.Willsky, Analytical Redundancy and the Design of Robust Failure Detection Systems, 1984

17 Chow,E.Y. and A.S.Willsky, Analytical Redundancy and the Design of Robust Failure Detection Systems, IEEE Transactions onAutomatic Control, 603-514, 1984

18 Askerdal, O., Gafvert, M., Hiller, M., Suri, N., Analyzing the Impact of Data Errors in Safety-Critical Control Systems, IEICE Transactions on Information and Systems, E86-D, 12, 2623-2633, 2003

19 Pflanz, M., F. Pompsch, and H. T. Vierhaus, An Efficient On-line-Test and Back-up Scheme for Embedded Processors, In Proceedings of the International Test Conference, 964–972, 1999

20 Sogomonyan, E. S. and M. Gssel, Concurrently Self-Testing Embedded Checkers for Ultra-Reliable Fault-Tolerant Systems, In Proceedings of the 14th VLSI Test Symposium, 138–144, 1996

21 Bouvier, M., Thesis: "Definition of a Cost-Effecive, Fault-Tolerant Control Architecture: Application to the Design of a Steer-by-Wire System", University of British Columbia, 2002

22 Kimura, S. and T. Okuyama, Consensus Making of Multi-CPU Control of Hyper-redundant Manipulator, In IEEE International Conference on Systems, Man, and Cybernetics, volume 4, 3623–3628, 1998

23 Oldknow, K. D., Thesis: "A Dynamically Reconfigurable System Architecture and FPGA Based Servo Controller for Distributed Machine Tool Control", University of British Columbia, 2000

24 Nace, W. and P. Koopman, Graceful Degradation in Distributed Embedded Systems, http://www.ddjembedded.com/ressources/articles/2001/0106em001/0106em001a.htm [accessed on 03/25/2006], 2001

25 Goode, K. B., Moore, J., Roylance, B. J., Plant machinery working life prediction method utilizing reliability and condition-monitoring data, Proc Instn Mech Engrs, Volum 214.E, pp.109-122, 2000

26 Isermann, R., Fault Diagnosis of Machines via Parameter Estimation and Knowledge Processing, Automatica, Vol. 29.4, pp. 815-835, 1993

27 N. L. Azad, J. McPhee and A. Khajepour, Tire Forces and Moments and On-road Lateral Stability of Articulated Steer Vehicles., SAE TECHNICAL PAPER SERIES, 2005-01-3597

Appendix A: Tire Lateral Force and Steering Rack Load Estimation

The external forces that can cause longitudinal or lateral motion of the vehicle are mainly generated at the tires. Research on the force and moments generated by tires on roads has been conducted using different analysis and measurements. This section introduces the approach to estimate the lateral force developed in [27]. As shown in Figure A.1, there are three forces and three moments which can be generated at the tire/road contact area. Among them, The lateral tire force F_y is very important for the lateral stability and steering behaviour of road vehicles. This force depends on the slip angle α, the vertical tire load F_z, and also the friction coefficient μ.

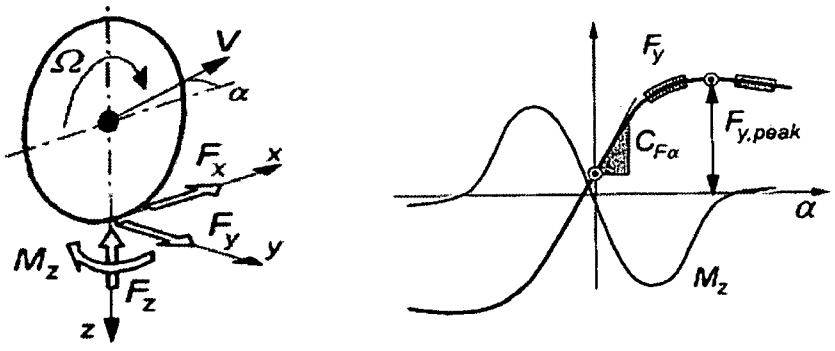


Figure A.1: Lateral tire force and aligning moment characteristic curves on hard surfaces [27]

Because the effective line of action of the lateral force F_y does not intersect the centre of the wheel axis system, a self-aligning moment is produced. The moment arm is known as pneumatic trail, which decreases once sliding beings and approaches zero at higher slip angles. Typical on-road tire force and moment characteristics in the form of F_y and M_z versus curves α are represented inFigure A.1. And the lateral slip stiffness or cornering stiffness $C_{F\alpha}$ can be computed as:

$$C_{F\alpha} = A B F_z$$

Equation A.1

where B is determined for a specific conditions of the tire and surface by the following equation:

$$B = \frac{C}{A}\left(\frac{F_{zT}}{F_z}\right)^m + \frac{D}{A}$$

Equation A.2

F_{zT} is the rated tire load and m is an exponent equal to 0.14. For different contact surface the coefficients of A, C and D are given in Table A.1.

Surface	*A*	*C*	*D*
High way	*0.67*	*0.677*	*-0.563*
Plowed Field	*0.65*	*0.267*	*-0.222*
Gravel	*0.52*	*0.588*	*-0.49*
Corn Field	*0.53*	*0.440*	*-0.365*
Meadow	*0.88*	*0.784*	*-0.652*

Table A.1: Metz's Coefficients For Different Surfaces

Then aligning stiffness $C_{T\alpha}$, which is also the negative slope of M_z versus curves α at zero slip angle, can be predicted by

$$C_{T\alpha} = \frac{C_{F\alpha} l_t}{6}$$

Equation A.3

where l_t is the tire contact length.

Therefore for small slip angle , self-aligning moment is

$$M_z = -C_{T\alpha}\alpha$$

Equation A.4

And since the wheel is not turning around its centre but a pivot aside, there is wheel turning inertia torque:

$$M_w = J_w \alpha_w$$

Equation A.5

where J_w is the wheel's moment of inertia and α_w is the wheel's angular acceleration.

Then the force applied by the wheels on the steering rack can be obtained by

$$F_{load} = 2 \cdot \frac{M_z + M_w}{l_c}$$

Equation A.6

where l_c is the distance between the wheel centre and tie rod's end.

Appendix B: Health Monitoring Software Documentation

This appendix provides a detailed description of the classes from which the Health Monitoring utilities are instantiated. It is meant to supplement the description given in 4.2.

Table B.1: Input and Feedback Class Description

InputandFeedback		
Inherits from: none		
Instance Variables		
Class: InputandFeedback		
Name	**Type**	**Description**
r1	var (integer)	Acquired position reference input at 0 ms.
r2	var (integer)	Acquired position reference input at 10 ms.
r3	var (integer)	Acquired position reference input at 20 ms.
r4	var (integer)	Acquired position reference input at 30 ms.
r5	var (integer)	Acquired position reference input at 40 ms.
Fy	var (integer)	Acquired position feedback at 100ms from sensors.
HPT	constant (integer)	Loop closing interval (10 ms).

Methods

Class: InputandFeedback		
Name	**Stack Diagram**	**Description**
Acquire:	(current_time —)	Obtain reference input at each loop closing and feedback
Get_r1:	(— r1)	Fetch the value from the beginning point.
Get_r3:	(— r3)	Fetch the value from the middle point.
Get_r5:	(— r5)	Fetch the value from the end point.

Methods

Class: InputandFeedback		
Name	**Stack Diagram**	**Description**
Get_fy:	(— Fy)	Fetch the value from the feedback.
ClassInit:	(—)	Execute initiation when the class is instantiated.

Table B.2: Parabolic Class Description

Parabolic		
Inherits from: InputandFeedback		
Instance Variables		
Class: Parabolic		
Name	**Type**	**Description**
a1	Fvar (float)	Parabolic curve parameter.
a2	Fvar (float)	Parabolic curve parameter.
a3	Fvar (float)	Parabolic curve parameter.
deta	Fvar (float)	Parabolic curve time step.
paraTs	Fvar (float)	Time sample.
num	var (integer)	Number of data in a segment of parabolic curve.

Methods

Class: Parabolic		
Name	**Stack Diagram**	**Description**
Acquire:	(current_time —)	Obtain reference input at each loop closing and feedback
Get_r1:	(— r1)	Fetch the value from the beginning point.
Get_r3:	(— r3)	Fetch the value from the middle point.
Get_r5:	(— r5)	Fetch the value from the end point.
Get_fy:	(— Fy)	Fetch the value from the feedback.
Compute_ parameters:	(—)	Compute parameters for parabolic form.
Compute_x	(— x4 x3 x2 x1 x0)	Compute estimated reference positions
ClassInit:	(—)	Execute initiation when the class is instantiated.

Table B.3: Physical System Class Description

Physicalsystem		
Inherits from: none		
Instance Variables		
Class: Physicalsystem		
Name	**Type**	**Description**
der2y	Fvar (float)	Steering rack's acceleration
dery	Fvar (float)	Steering rack's velocity
y	Fvar (float)	Steering rack's position
e	Fvar (float)	Velocity's sign
pi	Fconstant(float)	π
ls	Fconstant(float)	Screw lead
nse	Fconstant(float)	Ballscrew efficiency
Dp	Fconstant(float)	Pinion diameter
nre	Fconstant(float)	Pinion efficiency
G2	Fconstant(float)	Pulley ratio
Wr	Fconstant(float)	Rack weight
Ft	Fconstant(float)	External force
Jm1	Fconstant(float)	Motor 1's inertia
Js	Fconstant(float)	Screw's inertia
Jm2	Fconstant(float)	Motor 2's inertia
Jp	Fconstant(float)	Pinion's inertia
Kpos	Fconstant(float)	Position feedback gain

Methods

Class: Physicalsystem		
Name	**Stack Diagram**	**Description**
Compute_ac:	(T1 T2 — der2y)	Calculate acceleration from torques.
Compute_vel:	(der2y — dery)	Calculate velocity from acceleration.
Compute_pos::	(dery — y)	Calculate position from velocity.
Get_dery:	(— dery)	Fetch the velocity value.
Put_dery:	(dery —)	Store the velocity value.

Methods

Class: Physicalsystem		
Name	**Stack Diagram**	**Description**
Get_y:	(— y)	Fetch the position value.
Put_y:	(y —)	Store the position value.
Reset:	(—)	Reset the position and velocity.
ClassInit:	(—)	Execute initiation when the class is instantiated.

Table B.4: Lead/Lag_Health Class Description

LeadLag_Health		
Inherits from: Physicalsystem		
Instance Variables		
Class: LeadLag_Health		
Name	**Type**	**Description**
Kll	Fvar (float)	LeadLag gain.
Ac	Fvar (float)	LeadLag coefficient a (continuous time)
Bc	Fvar (float)	LeadLag coefficient b (continuous time)
Kp1	Fvar (float)	TMAC's voltage/PWM
Ktach	Fvar (float)	Velocity feedback gain
Ki	Fvar (float)	Amplifier gain
sat_high	Fvar (float)	Saturation high limit
sat_low	Fvar (float)	Saturation low limit
Ad	Fvar (float)	LeadLag coefficient a (discrete time)
Bd	Fvar (float)	LeadLag coefficient b (discrete time)
T1	Fvar (float)	Torque output from position loop
in	Fvar (float)	Current input to LeadLag filter
out	Fvar (float)	Current output from LeadLag filter
last_in	Fvar (float)	Last input to LeadLag filter
last_out	Fvar (float)	Last output from LeadLag filter

Methods		
Class: LeadLag_Health		
Name	**Stack Diagram**	**Description**
Compute_Ad:	(—)	Calculate the discrete time coefficient a.
Compute_Bd:	(—)	Calculate the discrete time coefficient b.
Compute_ LeadLag:	(in — out)	Calculate LeadLag filter's output.
Compute_out:	(out — T1)	Calculate torque output from position loop
Compute_ac:	(T1 T2 — der2y)	Calculate acceleration from torques.
Compute_vel:	(der2y — dery)	Calculate velocity from acceleration.
Compute_pos::	(dery — y)	Calculate position from velocity.
Get_dery:	(— dery)	Fetch the velocity value.
Put_dery:	(dery —)	Store the velocity value.
Get_y:	(— y)	Fetch the position value.
Put_y:	(y —)	Store the position value.
Get_last_in:	(— last_in)	Fetch the value of last input to LeadLag.
Put_last_in:	(last_in —)	Store the value of last input to LeadLag.
Get_last_out:	(— last_out)	Fetch the value of last output from LeadLag.
Put_last_out:	(last_out —)	Store the value of last output from LeadLag.
Get_T1:	(— T1)	Fetch the torque value.
ClassInit:	(—)	Execute initiation when the class is instantiated.

Table B.5: PID_Health Class Description

PID_Health		
Inherits from: None		
Instance Variables		
Class: PID_Health		
Name	**Type**	**Description**
K2	Fvar (float)	Torque loop gain
Kp	Fvar (float)	PID proportion gain

PID_Health		
Inherits from: None		
Instance Variables		
Kd	Fvar (float)	PID derivative gain
Ki	Fvar (float)	PID integral gain
Kp2	Fvar (float)	TMAO's voltage/PWM
KA2	Fvar (float)	Amplifier gain
KT2	Fvar (float)	Motor torque constant
G1	Fvar (float)	Gear ratio
e_prev	Fvar (float)	Last error value
i_prev	Fvar (float)	Last sum value
kds	Fvar (float)	Kd/time step
sat_high	Fvar (float)	Saturation high limit
sat_low	Fvar (float)	Saturation low limit
T2	Fvar (float)	Torque output from torque loop

Methods

Class: PID_Health		
Name	**Stack Diagram**	**Description**
Compute_out:	(out — T2)	Calculate torque output from torque loop
Get_e_prev:	(— e_prev)	Fetch the value of last error value.
Put_e_prev:	(e_prev —)	Store the value of last error value.
Get_T1:	(— T1)	Fetch the torque value.
Reset:	(—)	Reset the values of error and error sum.
ClassInit:	(—)	Execute initiation when the class is instantiated.

Appendix C: Software Configuration and Vehicle Parameters

Table C.1: Health Monitor Parameters

Parameter	Value	Unit
Software Configuration		
Position loop		
Gain of velocity/PWM	0.02	V
Velocity loop gain	0.0476	V/rad/s
Amplifier gain	0.62	A/V
Position loop feedback gain	315117	BLU/m
Natural frequency	18	rad/s
Damping ratio	0.75	
LeadLag gain	0.06	
Lead	22	
Lag	27	
Torque loop		
Torque loop gain	0.02	
Gain of velocity/PWM	0.005	V
Amplifier gain	4	A/V
Motor torque constant	0.0612	Nm/A
Motor gearhead	19.7	
Pulley ratio	3	
PID proportion gain	0.06	
PID derivative gain	0	
PID integral gain	0	

Parameter	Value	Unit
Vehicle Parameters		
Vehicle weight	800	Kg
Wheelbase	2	M
Front wheels distance	1	M
Steering ratio	11	
Tire max Load (sure trail 4.80)	2575	N
Tire weight	4	Kg
Tire contact length	0.08	M
Pivot and centre distance	0.15	M
Slip angle	6	Degree
Rack transverse Acceleration	0.03	m/s^2
Total weight of steering rack	10	Kg
Pinion diameter	0.015	M
Pinion length	0.2	M
Rack slide friction coefficient	0.1	
Rack and pinion efficiency	80	%
Friction torque of pinion	0.1	Nm
Friction force of bearings	30	N
Ballscrew diameter	0.0127	M
Screw length	0.5	M
Ballscrew friction coefficient	0.02	
Ballscrew efficiency	0.9	
Screw lead	0.0127	M
Pulley ratio	3	
Motor 1's inertia	1.4*10^-4	Kg*m^2
Screw's inertia	10^-5	Kg*m^2
Motor 2's inertia	10^-6	Kg*m^2
Pinion's inertia	7.85*10^-6	Kg*m^2

Appendix D: Amplifier and Motors Parameters

Copley Controls Inc. 513, DC brushless servo amplifier	Power supply	+24V – +90V
	Output power	Continuous: 13A, 2360W Peak: 20A, 3600W
	Bandwidth	25kHz
	Resolver option	Tachometer emulation Encoder emulation
	PWM transconductance stage	4A/V
	Monitor output	±6.5V @ ±26A (4A/volt)

Table D.1: DC Brushless Servo Amplifier

Magmotor S28-I-300ET1, with coupled tachometer and optical encoder.	Power Supply	24 – 120 VDC
	Rotor inertia	0.025oz-in-sec^2
	Continuous torque	200oz-in
	Peak torque	1500oz-in
	Power range	200W
	Back EMF	48.5V/krpm
	Max current	23A
	Torque constant	65.6oz-in/amp

Table D.2: Brushed Servo Motor Applied in Position Loop

Pittman Gearmotor GM14904S016, with gearbox	Power supply	24VDC
	Rotor inertia	3.7 E-03oz-in-sec^2
	Continuous torque	374oz-in
	Peak torque	2934oz-in
	Power range	100W
	Back EMF	6.41V/krpm
	Max current	23.8A
	Torque constant	8.67oz-in/amp
	Gearbox reduction ratio	19.7

Table D.3: DC Servo Gearmotor Applied in Torque Loop