Embedded Software Control Design for an Electronic Throttle Body

by

Paul G. Griffiths

B.S. Mechanical Engineering (Michigan Technological University) 2000

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Masters of Science

in

Mechanical Engineering

in the

GRADUATE DIVISION
of the
UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor J. Karl Hedrick, Chair
Professor Pravin Varaiya

2002

The dissertation of Paul G. Griffiths is approved:

Chair Date

Date

Date

University of California, Berkeley

2002

Embedded Software Control Design for an Electronic Throttle Body

Paul G. Griffiths

Abstract

Embedded Software Control Design for an Electronic Throttle Body

by

Paul G. Griffiths

Masters of Science in Mechanical Engineering

University of California, Berkeley

Professor J. Karl Hedrick, Chair

This paper examines the complete design of an electronic throttle control system. This system is a safety-critical, embedded control application, and so the design of the system extends beyond the analysis of the closed-loop behavior of the plant and a controller. The necessary reliability and ability to detect and handle hardware failures creates particular hardware requirements. Differential equations that describe the hardware are derived so that control strategies can be designed and simulated before they are tested on the hardware. The presence of Coulomb and static friction in the dynamics of the throttle plate motivates the use of a non-linear control law. Sliding mode and adaptive sliding mode control laws are derived for controlling the throttle plate dynamics. The adaptive control law provides an on-line estimate of the friction, which can identify a throttle body with excessive friction. Besides the control of the throttle position, the system must be integrated into the higher-level goals of a powertrain controller. Various operational modes and the conditions to

transition between modes are identified. Software needed to implement both the logic of choosing an appropriate controller and the algorithms of each controller is mapped into a structured software architecture. Models are developed to simulate the closed-loop behavior of the controller software and the code for the target embedded processor is written to match the structure. Experimental results are presented, which validate various system requirements.

Professor J. Karl Hedrick
Dissertation Committee Chair

To Kristina

Contents

List of Figures

List of Tables

Acknowledgements

This work was conducted under sponsorship of DARPA/ITO (Contract #F33615-00-C-1698, SmartVehicles: An Open Platform for the Design, Testing, and Implementation of Automotive Embedded Systems)

Part I

First Part

Chapter 1

Introduction

1.1 Background

The electronic throttle control (ETC) system is a drive-by-wire system in which the direct linkages between the accelerator and the throttle or the steering wheel and the steering gear are replaced with pairs of sensors and actuators. ETC systems have existed for more than a decade but have only entered mass production in the past few years. The systems are entering the passenger car market in high-end vehicles, in which manufacturers want to enhance the driver's experience by dynamically adjusting pedal to throttle position transfer function in response to changes in the temperature, altitude, or vehicle speed. [10]

Electronic throttle control systems are usually packaged as one assembly with a motor, springs, and throttle position sensors all fit into the throttle body. Figure 1.1 shows a simplified diagram of the internals of an electronic throttle body. In the center are the throttle bore and the plate. As a safety mechanism, the spring provides a torque to close the throttle when the motor is off. The equilibrium position of the spring in some electronic throttle bodies is set to a small positive angle and the throttle is used to control idle speed. The motor on the left end of the throttle shaft actuates the throttle and the potentiometer on the right end of the shaft is the throttle position sensor.

1.2 Motivation

The electronic throttle control (ETC) system presents two interesting problems in one application. From the automatic control perspective, the throttle plate has non-linear dynamics, so control of the throttle plate position is most appropriately addressed by the use of non-linear control theory. Of equal concern is the development of software, which

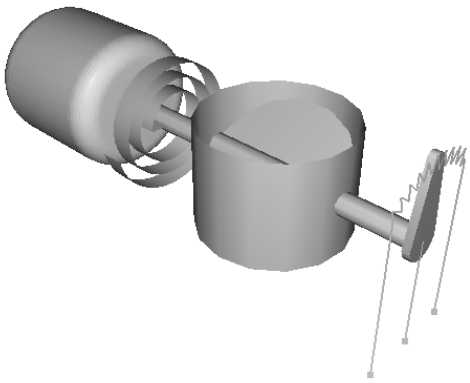


Figure 1.1: Simplified diagram of the electronic throttle body

implements the stabilizing control law. The controller software is an embedded application on a powertrain microprocessor and it must meet high reliability and safety requirements. In this thesis, a complete ETC system will be designed that considers the hardware, software, and control theory design problems.

The alternative to the drive-by-wire electronic throttle system is the standard pull-cable throttle with return spring. This is still the predominant solution in use in passenger cars. There are a number of ways in which the electronic throttle system performs better than the mechanical linkage. The only disadvantage of the drive-by-wire solution may be the cost. A natural concern about removing the mechanical linkage between the accelerator pedal and the throttle mechanism is that the non-mechanical system might be inherently less safe and less reliable. The drive-by-wire system can, in fact, be more reliable particularly

when considering problems with sticky and dirty throttle bodies. The electronic system can adapt to the friction in the system in order to maintain the accelerator pedal tracking performance. Since the system makes an estimate of the friction, it can also diagnose dirty, sticky, or otherwise worn out hardware. Hardware and software redundancy is also used to maintain a very high level of reliability. Integration of various engine and vehicle control systems can be accomplished with a single ETC system and can offset the additional cost of the ETC hardware. Cruise control, idle control, engine over-rev. protection and traction control features might all need to modify the throttle position. With the ETC system, the switching or blending of control algorithms occurs in software and there is no need for separate actuation components for each feature. There are also advanced features that can be accomplish only with the ETC system. If multiple throttles are used, sophisticated engine power management can shut down individual cylinders and, in doing so, increase the efficiency of the power cycles in the other engine cylinders. The replacement of the connection between the driver's foot and the throttle plate with software allows the designer to adjust the pedal-to-plate transfer function. For instance, initial pedal travel can correspond to smaller throttle plate motion compared with pedal travel closer to the wide-open-throttle (WOT) position. This transfer function can also be adjusted for vehicle speed or altitude to make the engine feel more responsive to the driver.

1.3 Overview

This thesis is organized roughly in the order of design activities for the ETC system. First, the hardware is described in detail. Important details of the design, which

are relevant to the ETC system, are discussed. An in-depth look is given to the power electronics for driving the throttle actuation motor. Given a particular electronic throttle body, a mathematical model of this plant is developed and then the system is fit to data taken from the throttle. Position control of the throttle plate dynamics is considered given the model of those dynamics. A sliding mode and adaptive sliding mode controller are developed theoretically and then tested on the mathematical model of the throttle. The system requirements and operational description prescribe the throttle behavior given inputs such as the accelerator pedal position, vehicle speed, and engine speed. The next design consideration is the software architecture, in which the algorithms and logic that implement the system requirements are fit. Some abstractions of the software are developed and these abstractions are used to model the software. A detailed model of the plant, the controller and interface between the two in the form of driver, actuator and sensor models are created in MathWorks Simulink®/Stateflow® [1]. Simulations of the models verify that the system requirements are met before the controller software is tested with the physical plant. The software implementation of the controller is tested on the hardware and the experimental results are compared with the simulation results.

[1]Simulink and Stateflow are registered trademarks of The MathWorks, Inc., Natick, MA

Chapter 2

Hardware Description

2.1 Introduction

The hardware for the ETC system is a combination of purchased and built components. A schematic of the complete system is shown in figure 2.1. A Motorola MPC555 micro-processor based board (ES200.2 sold by ETAS GmbH & Co.KG.) implements the embedded control. A slightly modified PC power-supply provides 5V and 12V lines to the micro-processor board and the driver electronics. The micro-processor board runs exclusively on 12V and can tolerate some variation in supply voltage. The driver electronics for the pedal position sensor use a 12V line to create a regulated 5V signal for the sensor's potentiometer. The other driver electronics, which drive the motor and the throttle position sensors in the electronic throttle body, use both 5V and 12V supplies. The connections between the micro-processor board and the driver electronics carry digital signals to the H-bridge IC and the analog signals from the pedal position sensor and the throttle position sensors.

2.2 Throttle Hardware

The throttle selected for the system is a BMW electronic throttle. See the picture of the throttle in figure 2.3. There are several important features that make it well suited for the electronic throttle application. It has a DC servo motor that drives the throttle plate through a set of reduction gears. There are only two types of motors appropriate to this type of application, a stepper motor and a servo motor. In terms of efficiency, the stepper motor is an optimal solution when one position must be held but it is not very efficient for quick movements. The ability to make quick precise movements of the throttle

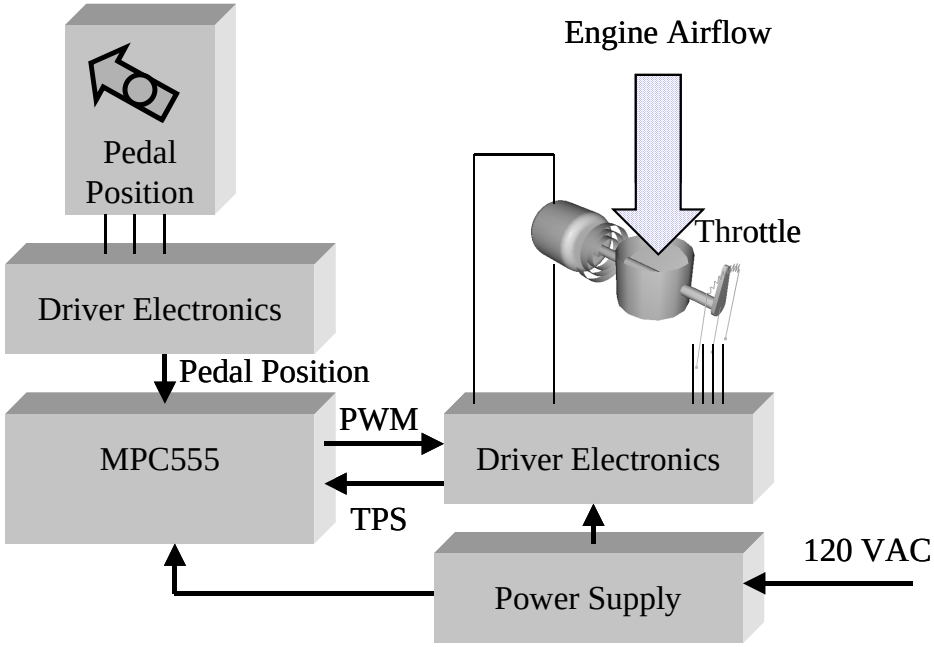


Figure 2.1: Schematic of the ETC hardware configuration

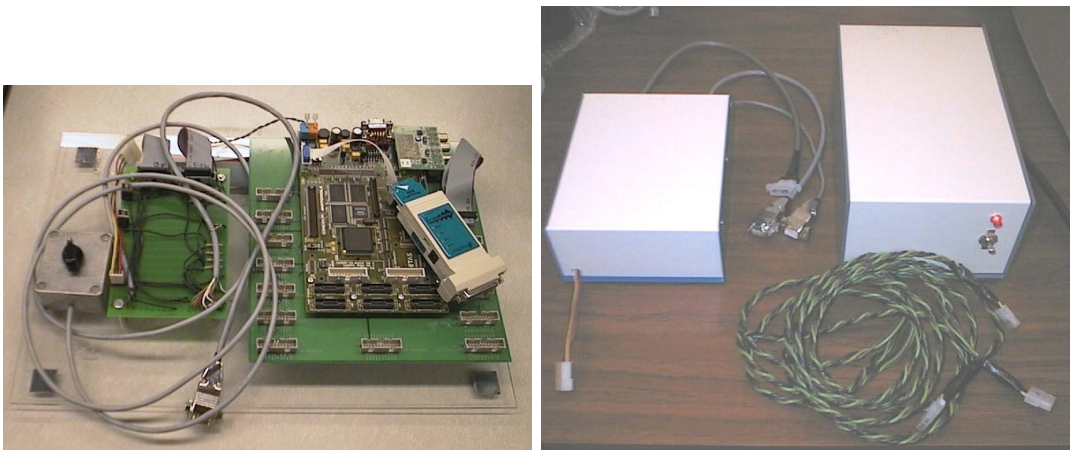


Figure 2.2: The MPC555-based micro-processor board and break-out connections are shown on the left, and on the right, are the power supply and driver electronics boxes

plate is more important than the potential efficiency gain of a stepper motor for holding one position for long periods of time. Also, the stepper motor is built to hold a finite number of positions and this affords very little flexibility in the design of the controller. Another important feature is the gear reduction between the servo motor and the throttle plate. The gear reduction cuts down the static friction from the throttle plate and allow a smaller motor to be used. Compared to a system without the gear set, the throttle can be controlled more precisely and requires less power. The servo motor provides the actuation and the throttle position sensor provide the position feedback necessary for closed-loop control. Because the throttle position control is a safety critical function, there must be hardware redundancy to make it likely that a hardware fault is detected. Since the throttle position sensor is the sole observer of the states of the throttle, a redundant sensor greatly increases the likelihood that a hardware failure will be identified and that the system will be correctly brought to a safe shutdown or limited ability mode. The throttle position sensors are two linear potentiometers that operate on the same power supply. A common 5V signal is provided to the throttle position sensors by a regulated by an IC in the power electronics box. In a clever design, the TPS potentiometers have been connected so that the outputs are complementary. The sum of the two signals equals the supply voltage and then a failure of the regulator due to a chip failure or an under-voltage condition of the battery can be distinguished from a single potentiometer failure. The last notable feature of the throttle is a positive equilibrium position of the return spring. If the servo motor is not energized, the throttle plate will remain open by a few degrees. This design ensures that the engine continues to run at a high idle speed in the case of a failure in the servo actuation system.

Maintaining this fast idle is both a safety and convenience concern. If the engine stalls, then there is no power to accessories such as power steering and power brakes. If the electronic throttle shuts down when the driver is on the road, a fast idle will help the driver get the vehicle to a safe location. And of course, it is important that the throttle defaults to a nearly closed condition so that the power of the engine is low and the vehicle does not run out of control.

Figure 2.3: Picture of the BMW electronic throttle body (donated by BMW)

The BMW electronic throttle body was donated to the MoBIES project and it came with a connector but no cabling of power electronics. I decided that the cabling and power electronics should be modular and well-shielded. There are six pins in the electronic throttle body connector: two for the motor and four for the TPS potentiometers. The motor cables and the TPS potentiometer wires are bundled separately so as to reduce the noise that the current going to the motor would cause in the TPS readings. Both cables have shielding and inside the TPS cable, the supply wires are independently shielded from the

signal wires. The motor and TPS cables are shielded large gauge three-conductor cable and small gauge four-conductor cable, respectively. In between the MPC555 and the throttle is a power electronics box. The cables from the throttle have connectors in between the throttle and the power electronics box. (This makes it much easier to quickly test the throttle, switch in a different throttle or a different power electronics box.) For the same reason, the cable that connects the power electronics with the MPC555 also has a connector. This nine-conductor shielded cable carries only signals and no power between the processor and the power electronics. The heart of the power electronics is the H-bridge, which uses digital inputs to control large currents (over 3A) to the motor. Four of the wires from the MPC555 are TTL signals, which can control the H-bridge. In addition, the signals going back to the H-bridge consist of a status signal from the H-bridge, two throttle position sensor signals, the motor current signal and a ground line.

2.3 Design of Power Electronics

2.3.1 Introduction

The design of the power electronics that drive the servo motor is a bit of an art. The purpose of the power electronics is to provide a digital logic interface to the actuation of the servo motor. In order to control the position of the throttle, it is necessary to control the torque out of the servo, which is simply proportional to the direct current in the servo windings. There are two methods types of power supplies that can be used to adjust the current going through the windings: linear and switching power supplies. The linear design is quite simple; a variable resistance is placed in series with the motor, with which the

current through the motor is given by Ohm's law. Unfortunately, at about half of the motor's rated current, only half of the power is going into the motor and the other half is going into the variable resistor. The heat dissipated in the resistance is wasteful and cooling the resistor can be a challenge. Of course, there is essentially no loss when the variable resistor is at a very high or very low resistance with respect to the resistance of the motor windings. This leads to the idea of the switching power supply. The idea is that by only using the power supply at the low or high resistance (on or off) operating points and switching between the two very rapidly, the load will only receive a fraction of the power and the losses will be much less than using a linear supply.

2.3.2 Switching Power Supplies

If only one polarity is needed, the switching power supply can be represented by a voltage source across a switch in series with the load. If the load is a resistance, then the instantaneous power to the resistance when the switch is closed is the product of the current and the supply voltage. The average power dissipated by the load is a portion of that power, where that portion is the fraction of the time that the switch is closed.

There are several of devices that will open and close the circuit using an input signal. A mechanical relay, which uses a small control current to open and close large contacts, can be used. However, the mechanical motion of the relay is too slow for a switching power supply and sparking can pit the contacts or weld them together. Transistors are the correct solution for a switching power supply because they can be turned on and off much faster than a relay and do not have mechanical wear problems. There are two common classes of transistors: the bipolar junction transistor (BJT) and the field effect transistor

(FET). The former is a current controlled device and the latter is a voltage controlled device. Although they both could be used for the switching power supply, this distinction, as well as the fact that BJTs are subject to thermal-runaway makes the FET a better choice. The BJT in its linear operating range uses a small current to meter a large current like a set of reduction gears. The ratio of the control current to the large current is generally in the range or 1:10 to 1:100. (Darlington transistors are simply two BJTs cascaded to achieve a higher ratio.) For use in the switching power supply, the transistor will be driven into its saturation region by supplying more control current than is needed to reach the maximum load current. For large load currents there will be a substantial control current, which will create heat dissipation problems and there will have to be multiple BJTs to reduce the control current level within that of transistor-transistor logic (TTL). Heating up a BJT is a problem for another reason; the resistance to the large current flow drops as the BJT heats up and allows more current flow. This positive feedback is know as thermal-runaway and it can destroy the transistor in an instant. FETs do not suffer from this problem. For FETs, a voltage applied to the gate terminal controls a large current. Only a tiny current ($< 10^{-}9A$) into the gate is possible and for all practical purposes the gate terminal can be considered an infinite resistance. This means that no power is required to hold the FET in the open or closed state. Power is only consumed in changing the state of the FET from on to off. The on and off states of the FET are characterized by very high and very low resistance, on the order of over 1MΩ and less than 0.1Ω respectively. During transitions there is a brief period of time that the resistance of the FET is of a comparable magnitude and during this time there is power dissipation. One other important feature that is commonly built into

FETs is a diode that allows for reverse current flow through the FET. The low on-state resistance, the fast switching time and the lack of power needed to hold the FET open or closed makes it an excellent choice for the switching power supply.

2.3.3 Power to Inductive Loads

In the previous discussion of the appropriate switching device, the load was considered to be a resistance. In contrast the servo motor is a load characterized by a large inductance in series with a small resistance. (The inductance of the servo motor in the BMW electronic throttle body is 1.7mH and the resistance is 3Ω.) When the switch is closed in the circuit with the motor as the load, there is no instantaneous current through the motor. The inductance guarantees that the current through the motor is a continuous function of time. This means that ideally it is not possible to open the switch again when there is current through the motor. If the switch is a set of contacts, then at the instant after the contacts open the induction of the motor applies whatever voltage is necessary to maintain its current flow. In the case of physical contacts, thousands of volts are applied, which ionizes the air and the current arcs across the gap. The large voltage quickly eliminates the current flow, so the spark only lasts a brief moment. The situation is not much different in the case of the FET except that it cannot spark, but the voltage applied from the drain to the source can easily exceed the break-down voltage and destroy the transistor. Clearly, it is important to provide a safe outlet where the current flow can decay in a continuous, controlled manner. A diode that connects the motor terminals and allows current to circulate out of the motor and back into the positive terminal solves this problem. This diode is referred to as a fly-back diode, because it allows the current to fly-back from the

negative to the positive terminal. See the schematic of the circuit in figure 2.4. Because it allows current flow only from the negative to the positive motor terminals, when the switch is closed the power supply cannot short circuit across the diode. There are three design considerations for the diode. First, similar to the FET, there is a break-down voltage, which the power supply to the motor must not exceed or the diode will be destroyed. Also, there are two kinds of diodes from which to choose. The standard diode begins to allows current flow in the forward direction with a forward voltage of about 0.6 volts and the Schottky diode begins to allow current flow at only 0.3 volts. Since the power dissipated by the device is the product of the current flow and the voltage across the device, the Schottky diode will draw only have the power of the standard diode. If the current does not need to be stopped when the switch is open, the reduced forward voltage reduces the power wasted by the diode. Finally, the power rating of the diode must be matched to the application.

The single FET with a fly-back Schottky diode is a good solution for the switching power supply, but it is limited to one polarity across the motor terminals. If the servo motor must be capable of applying torque in both directions, then an H-bridge design is needed. The H-bridge connects each motor terminal to both the positive and ground points of the voltage supply with a FET in each connection. The four FETs operate in two pairs, each of which connects one motor terminal to the positive supply voltage and one motor terminal to ground. Switching on one or the other of these pairs causes current to flow either one way or the other through the motor. When both pairs are off, no current can flow, except backwards through one or the other pair of FETs. This path is through the built-in diodes and is against the supply voltage. Again, the inductance of the motor must be considered

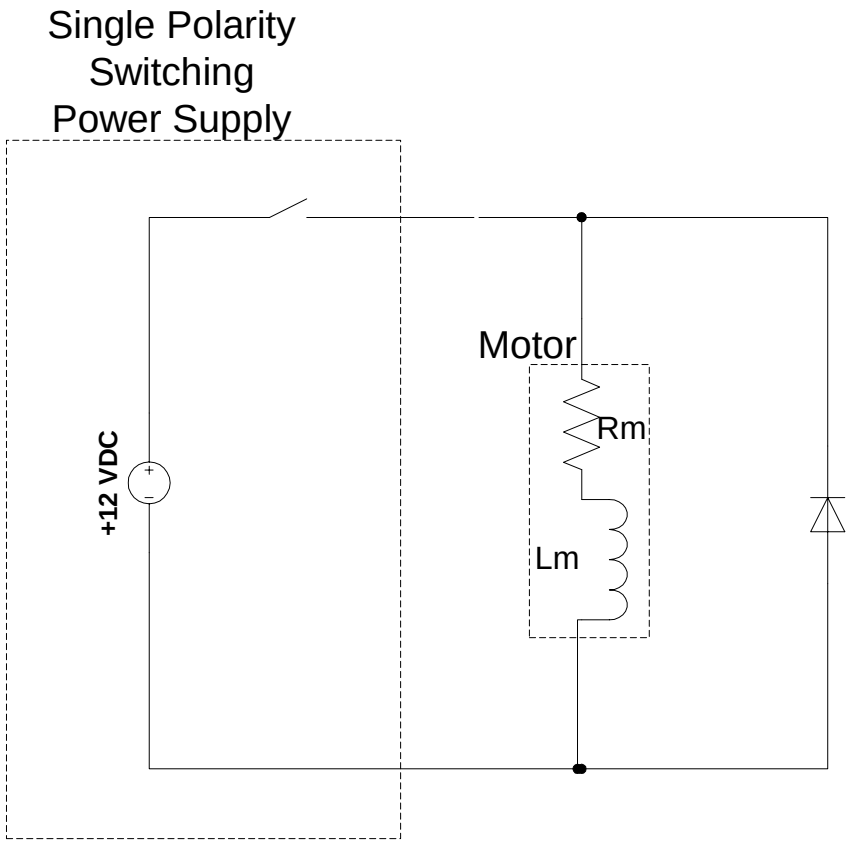


Figure 2.4: Schematic of the single fly-back diode design for relief of inductive currents

and a path for the current flow must be maintained at all times. Although the motor current can flow backwards through a pair of FETs after the FETs have been turned off, this current flow is against the supply voltage.

The switching power supply is to implement pulse-width modulation (PWM) and ideally, the power delivered would be linearly proportional to the duty-cycle and not be affected by the frequency of the PWM. That is to say, that a 50% duty cycle PWM would deliver only half the power of a 100% duty cycle PWM and the power at any duty cycle would be constant throughout all frequencies. In order for this to happen, once the power is delivered to the motor during the "on" part of one PWM wave, it should not return to the power supply during the "off" part. This is what will happen in the simple H-bridge configuration and there are two ways to fix the circuit so that this does not happen. Both are extensions of the fly-back diode design discussed in the context of the single FET, which use two fly-back diodes.

2.3.4 Double Fly-back Diode Designs

The first design requires that all four gates of the FET can be controlled independently. Two fly-back diodes are used, but instead of connecting the diodes across the motor terminals, one end of each diode connects to one motor terminal and the other ends connect to the power supply ground. The diodes are oriented to allow current flow from ground back into one or the other motor terminals. When one pair of FETs transitions from on to off, the FET that connects to ground can be left on and then current can continue to circulate through one of the diodes and the FET that is still on. If independent control of each gate is not possible, then a more sophisticated design is needed.

The H-bridge power electronics, which were designed for the BMW throttle, uses logic to insert or remove the fly-back diodes at the correct moment. (It is important to note that this design is specific to the load and should not be used with any generic load without recalculating a few key parameters.) The design has two fly-back diodes that together allow current to flow in either direction from the motor terminals. To prevent a short circuit when the supply voltage is applied to the motor terminals, the potentially short-circuiting diode is removed from the circuit by a FET acting as a switch. The schematic of this circuit is shown in figure 2.5. The resistance, R_s, and inductance, L_s, protect the circuit against a short during any brief moments that the H-bridge is on and the diodes have not yet been removed from the circuit by the FETs. The resistance, R_p, and the capacitance, C_p, provide a path for current from the motor for brief periods of time if the H-bridge has just turned off but the FETs have not inserted the diodes back into the circuit.

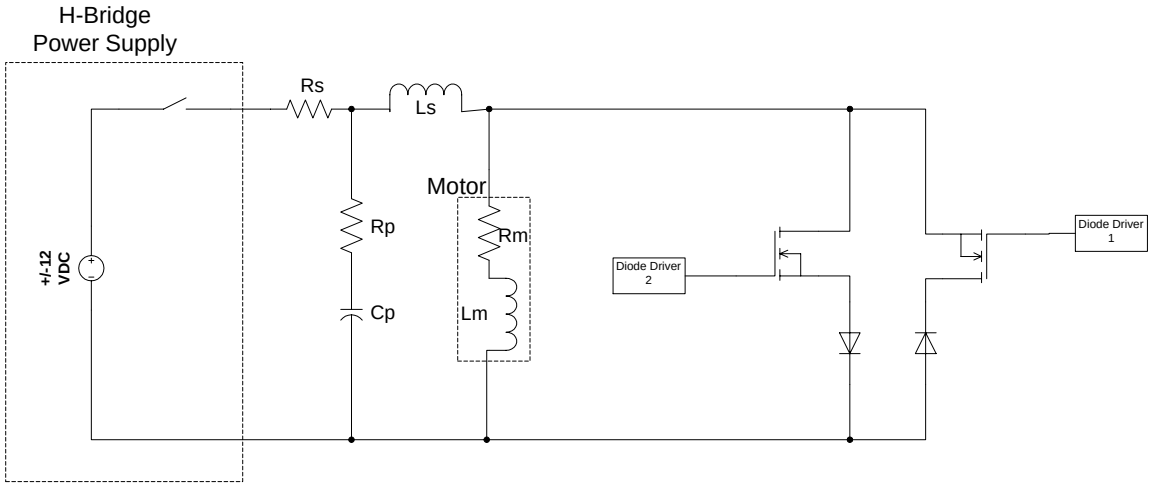


Figure 2.5: Schematic of the double fly-back diode design for relief of inductive currents with a dual polarity power source

Chapter 3

Mathematical Model

3.1 Introduction

The need to control the dynamics of the throttle plate motivates the derivation of the differential equations for all the hardware components and the experimental identification of hardware parameters. The hardware will be conceptually divided into four sections: the driver, the actuator, the plant and the sensors. The driver is the interface through which the software can affect the plant. In the ETC system, two pulse-width modulation PWM units drive the power electronics. The actuator section models the combined dynamics of the power electronics and the electrical characteristics of the motor. The actuator applies a torque on the plant, which is the mechanical system including the throttle plate, the reduction gear set and the motor. The throttle position sensors and the analog-to-digital converters are lumped together in the sensors section. The behavior of each part will be described in the following sections.

3.2 State Equations of the Driver

There are two models different models of the PWM, which are used in the ETC system. One is a static duty-cycle PWM, in which the duty-cycle input is sampled once per PWM period. The following defines the state-machine that describes this PWM. $p(t)$ is the PWM output, d is the input duty-cycle, d_L is the latched duty-cycle, and T is the PWM period.

$$\begin{aligned} p(t) &\in \{0,1\} \\ d(t), d_L &\in [0,1] \\ T &\in \Re_+ \end{aligned}$$

$p(0) = 0$

if ($t = nT | n = 0, 1, 2, \dots$) then $d_L(t) = d,$ and $p(t) = 1$

if ($p = 1$ and $t \bmod T > d_L$) then $p = 0$

An alternative PWM design does not have a static duty-cycle during one period. Instead $p(t) = 1$ until an input value falls to zero or below.

$$\begin{aligned} p(t) &\in \{0,1\} \\ e(t) &\in \Re \\ T &\in \Re_+ \end{aligned}$$

$p(0) = 0$

if ($t = nT | n = 0, 1, 2, \dots$) then $p(t) = 1$

if ($p = 1$ and $e < 0$) then $p = 0$

3.3 Differential Equations of the Actuator

The PWM drives the H-bridge of the power electronics. Two PWM units are used, where one is for forward polarity output and the other is for reverse polarity output. For the purposes of modeling, only one PWM is used to drive the H-bridge and the sign of the PWM determines the output polarity. In order to write down the circuit equations, the following states and parameters are defined.

$i(t)$:= Armature Current (A)

$p(t)$:= PWM

$T(t)$:= Torque on Throttle N-m

$V_b(t)$:= Back EMF (V)

R_a := Armature Resistance (Ω)

L := Armature Inductance (H)

R_{Bat} := Internal Resistance of the Source (Ω)

V_{Bat} := No-Load Voltage (V)

K_T := Motor Torque Constant $\left(\frac{N-m}{A}\right)$

The Kirchoff loop equation for the armature circuit is given by equation 3.1

$$\frac{di}{dt} = -\frac{R_a}{i} - \frac{V_b}{L} + \frac{V_{Bat}}{L}p(t) - \frac{R_{Bat}}{L}i, \text{where } p(t) \in \{-1, 0, 1\} \quad (3.1)$$

$$V_b(t) = K_T\ \omega \quad (3.2)$$

$$T(t) = K_T\ i \quad (3.3)$$

Equation 3.2 gives the back EMF, which is proportional to the motor speed, and equation 3.3 gives the torque applied to the throttle. This set of equations couples the actuator and the plant.

3.4 Differential Equations of the Plant

The throttle plate, the reduction gears and the motor rotor are lumped together as a single inertia, which is acted upon by the torque defined in equation 3.3. The states and parameters of the throttle plate dynamics are defined.

$\theta(t) :=$ Throttle Angle (rads)

$\omega(t) :=$ Throttle Angular Velocity $\left(\frac{\text{rads}}{\text{s}}\right)$

$J :=$ Lumped Inertia of throttle plate, reduction gears and motor rotor (Kg-m^2)

$K_s :=$ Spring Constant $\left(\frac{\text{N-m}}{\text{rad}}\right)$

$K_d :=$ Viscous Friction Constant $\left(\frac{\text{N-m-s}}{\text{rad}}\right)$

$K_f :=$ Coulomb Friction Constant (N-m)

$\theta_{eq} :=$ Zero Displacement of Throttle Return Spring wrt $\theta = 0$ (rads)

$C_s :=$ Torque Constant (spring torque at $(\theta = 0)$ (N-m)

The parameter, C_s is related to K_s and θ_{eq} by the following,

$$Ts = -K_s(\theta - \theta_{eq}) = -K_s\theta - C_s \tag{3.4}$$

or solving for C_s,

$$C_s = -K_s\theta_{eq} \tag{3.5}$$

The moments due to viscous and Coulomb friction are,

$$T_d = -K_d\omega \tag{3.6}$$

$$T_f = -K_f \text{sgn}(\omega) \tag{3.7}$$

The throttle plate dynamics can now be expressed by summing the moments about the throttle plate shaft. There are four moments acting upon the shaft: the moment from the return spring given by 3.4, the viscous and Coulomb friction moments given by 3.6 and 3.7, respectively, and the moment applied by the electric motor given by 3.3.

$$J\dot{\omega}(t) = -K_s(\theta(t) - \theta_{eq}) - K_d\omega(t) - K_f\text{sgn}(\omega) + T(t) \tag{3.8}$$

So the complete system is given by,

$$\begin{bmatrix} \dot{\theta}(t) \\ \\ \dot{\omega}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ \\ -K_s & -K_d \end{bmatrix} \begin{bmatrix} \theta(t) \\ \\ \omega(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \\ C_s - K_f\text{sgn}(\omega(t)) \end{bmatrix} + \begin{bmatrix} 0 \\ \\ T(t) \end{bmatrix} \tag{3.9}$$

3.5 Sensor Models

Two throttle position sensors and a motor current sensor are the three analog sensors in the ETC system. The throttle position sensors are potentiometers that, given a constant input voltage, provide an output voltage that varies linearly between about 0.5V and 4.5V with the throttle angle, $\theta(t)$. Also, the sum of the output voltage of the two sensors is equal to the input voltage. The motor current sensor also provides an output voltage that varies linearly with the motor current.

$TPS_1(t) :=$ Throttle Position Sensor 1 Output (V)

$TPS_2(t) :=$ Throttle Position Sensor 2 Output (V)

$n_1(t) :=$ Noise on $TPS_1(t)$ (V)

$n_2(t) :=$ Noise on $TPS_2(t)$ (V)

$$V_{ref} := \text{Constant Voltage Source} \quad \text{(V)}$$

$$\alpha := \text{Fraction of } V_{ref} \text{ for } TPS_1(t) \text{ at } \theta = 0$$

$$\beta := \text{Fraction of } V_{ref} \text{ for } TPS_2(t) \text{ at } \theta = \frac{\pi}{2}$$

$$TPS_1(t) = \alpha V_{ref} + \left(\frac{\beta V_{ref} - \alpha V_{ref}}{\frac{\pi}{2}} \right) \theta(t) + n_1(t) \tag{3.10}$$

$$TPS_2(t) = V_{ref} - TPS_1(t) + n_1(t) + n_2(t) \tag{3.11}$$

3.6 System Identification

3.6.1 Introduction

The constant terms in the equations for the actuator, plant, and sensors must be identified to realize a controller for the system. Some of these terms can be measured directly, some can be determined indirectly from dynamic performance and some cannot be measured at all. The sensor noise terms n_1 and n_2 must also be characterized for simulation and design of appropriate filters.

3.6.2 Determination of Plant Parameters

The easiest parameters to measure are those that do not involve the dynamic behavior of the system. The resistance and inductance of the electric motor can be measured directly as well as the source voltage and internal resistance of the source.

Parameter	Measured Value
R_a	3.5 Ω
L	1.7 H
V_{Bat}	12.0 V
R_{Bat}	0.5 Ω

Table 3.1: Directly Measured Parameters

Measuring other parameters such as Coulomb and viscous friction requires the throttle to be in motion, and even then these values cannot be measured directly. The signals that are available from the hardware are $p(t)$ and $\theta(t)$, from which $\omega(t)$, $\dot{\omega}(t)$ and $\mathrm{E}[p(t)]$ can be easily derived. The dynamics of the power electronics are simplified by replacing the input, $p(t)$, by a mean value approximation, and assuming that the inductance of the motor, L, is zero.

$$u(t) := \mathrm{E}[p(t)] \in [-1, 1] \tag{3.12}$$

$$0 = -R_a \bar{i} - V_b(t) + V_{Bat} u(t) - \bar{i} R_{Bat} \tag{3.13}$$

$$\bar{i} = \left(\frac{1}{R_a + R_{Bat}} \right) [u(t) V_{Bat} - V_b(t)] \tag{3.14}$$

The throttle plate dynamics can be rewritten in terms of $\bar{i}(t)$

$$\dot{\omega}(t) = -\frac{K_s}{J}\theta(t) - \frac{C_s}{J} - \frac{K_d}{J}\omega(t) - \frac{K_f}{J}\text{sgn}(\omega(t)) + \frac{K_T}{J}\bar{i}(t)$$

Substituting equation 3.2 in for V_b and defining $\frac{K'_d}{J}$

$$\begin{aligned}\frac{K'_d}{J} &:= \frac{K_d}{J} + \frac{K_t^2}{J(R_a + R_{Bat})} \\ \dot{\omega}(t) &= -\frac{K_s}{J}\theta(t) - \frac{C_s}{J} - \frac{K'_d}{J}\omega(t) \\ &\quad - \frac{K_f}{J}\text{sgn}(\omega(t)) + \frac{K_T}{J}\frac{V_{Bat}}{R_a + R_{Bat}}u(t)\end{aligned} \tag{3.15}$$

Given that $\theta(t)$, $\omega(t)$, $\dot{\omega(t)}$ and $u(t)$ are available signals, then sets of these values at different times, t, can be used to solve for the best values of $\frac{K_s}{J}$, $\frac{C_s}{J}$, $\frac{K'_d}{J}$, $\frac{K_f}{J}$ and $\frac{K_T}{J}$ in a least squares sense. Solving for $\frac{K_d}{J}$ from $\frac{K'_d}{J}$ requires knowledge of either K_T or J. Obtaining K_T would require either measuring the motor torque to current relationship or the motor angular velocity to induced armature voltage relationship. J is more easily estimated by knowing the material densities, approximate geometries of the throttle plate, motor and gears and the gear ratio. The only complication is the presence of the nonlinear term, $\text{sgn}(\omega(t))$. To eliminate this problem, the data is simply divided into data points where $\omega(t)$ is positive and data points where $\omega(t)$ is negative. Then the combined positive and negative velocity data sets are fit to the differential equation. The difference in the parameter, $\frac{C_s}{J}$ between the positive velocity data set and the negative velocity data set provides the estimate of $\frac{K_f}{J}$.

3.6.3 Noise Characteristics

The noise, $n_1(t)$ and $n_2(t)$, on the throttle position sensor signals is assumed to be a white Gaussian stationary random process. It can then be characterized by its mean value and variance.

$$m_1 := \mathrm{E}[n_1(t)] \tag{3.16}$$

$$m_2 := \mathrm{E}[n_2(t)] \tag{3.17}$$

m_1 and m_2 are assumed to equal 0

$$\mathrm{E}\left[\frac{TPS_1(t) + TPS_2(t)}{2}\right] = \theta(t) \tag{3.18}$$

$$\sigma_{TPS} := \mathrm{E}\left[\left(\theta(t) - \frac{TPS_1(t) + TPS_2(t)}{2}\right)^2\right] \tag{3.19}$$

A zero-phase filter is applied to the complete set of throttle position sensor readings in order to obtain a good approximation of $\theta(t)$. Then the data and the filtered data are used to estimate the variance, σ_{TPS}, of the noise signal.

3.6.4 System Identification Results

Parameter	Measured Value	
$\frac{K_d}{J}$	3.00e1	s^{-1}
$\frac{K_s}{J}$	1.35e1	s^{-2}
$\frac{K_f}{J}$	6.29e1	$\frac{\mathrm{rad}}{\mathrm{s}^2}$
$\frac{K_t}{J}$	2.50e1	$\frac{\mathrm{rad}}{\mathrm{A\text{-}s}^2}$
$\frac{C_s}{J}$	2.01e2	$\frac{\mathrm{rad}}{\mathrm{s}^2}$

Table 3.2: Parameters fit to model of throttle dynamics using experimental data

Signal Variance	Measured Value
σ_{TPS}	9.28e-6 rad^2
σ_{Pedal}	1.63e-5 rad^2

Table 3.3: Variance of the average TPS and pedal signals

Chapter 4

Control of Throttle Plate Dynamics

4.1 Introduction

The presence of non-linear friction in the throttle dynamics motivates the use of a sliding mode controller. This non-linear controller makes full use of the plant model to guarantee robust control even in the presence of plant uncertainty and external disturbances. An adaptive sliding mode controller is considered as a method for tuning the controller to the actual Coulomb friction. This can essentially minimize the controller gain with respect to the Coulomb friction as well as provide an estimate of the friction for fault diagnosis.

4.2 Sliding Surface Control

To begin the derivation of the sliding mode controller, a Lyapunov function V is defined in equation 4.1. V is positive definite and if $\dot{V}$ is negative definite, then s will globally, asymptotically approach zero. s is selected so that the surface in the state-space defined by $s = 0$ corresponds to desired plant dynamics (i.e. the plant states approach the desired values.) Equation 4.3 defines a sliding surface, s, such that for positive values of λ, the throttle angle, $\theta(t)$, and angular velocity, $\omega(t)$, will exponentially approach the desired throttle angle, $\theta_d(t)$, and the desired angular velocity, $\omega_d(t)$

$$V := \frac{1}{2}s^2 \tag{4.1}$$

$$\dot{V} = s\dot{s} \tag{4.2}$$

$$s := \omega(t) - \omega_d(t) + \lambda(\theta(t) - \theta_d(t)) \tag{4.3}$$

$$\dot{s} = \dot{\omega}(t) - \dot{\omega_d}(t) + \lambda(\omega(t) - \dot{\theta}_d(t)) \tag{4.4}$$

Replacing $\dot{\omega}(t)$ with the plant dynamics given by 3.15, and defining K_a,

$$K_a := \frac{K_t V_{Bat}}{R_a + R_{Bat}} \tag{4.5}$$

$$\begin{aligned} \dot{s} = &-\frac{K_s}{J}\theta(t) - \frac{C_s}{J} - \frac{K'_d}{J}\omega(t) - \frac{K_f}{J}\text{sgn}(\omega(t)) \\ &+ \frac{K_a}{J}u(t) - \dot{\omega_d}(t) + \lambda(\omega(t) - \omega_d(t)) \end{aligned} \tag{4.6}$$

An important property of the sliding surface, s, is that its derivative contain the control input so that the control input can force $\dot{V}$ to be negative definite. The presence of $u(t)$ in $\dot{s}$ validates the selected surface, given by 4.3.

Consider the definition 4.7 for $\dot{s}$, which is negative definite. It is possible to define $\dot{s}$ in this way because, given the plant parameters and states, $u(t)$ can cancel all the plant terms in $\dot{s}$ and insert any other expression.

$$\dot{s} := -\eta\text{sgn}(s) \tag{4.7}$$

$$s\dot{s} = s(-\eta\text{sgn}(s)) < 0, \forall s \neq 0 \tag{4.8}$$

In the expression for $\dot{V}$, only $\dot{s}$ depends on the control input, so only $\dot{s}$ can be used to force $\dot{V}$ to be negative definite. The terms in $\dot{s}$ due to plant dynamics can be canceled so that $\dot{s} = 0$ if $u(t)$ is defined as follows:

$$\begin{aligned} u(t) = \left(\frac{K_a}{J}\right)^{-1} \Bigg[&\frac{K_s}{J}\theta(t) + \frac{C_s}{J} + \frac{K'_d}{J}\omega(t) \\ &+ \frac{K_f}{J}\text{sgn}(\omega(t)) + \dot{\omega_d} - \lambda(\omega(t) - \omega_d(t)) \Bigg] \end{aligned} \tag{4.9}$$

By adding an additional term, $-\eta\ \mathrm{sgn}(s)$, a control law is formed which makes $s\dot{s}$ negative definite, and then s must be globally asymptotically stable about 0.

$$u(t) = \left(\frac{K_a}{J}\right)^{-1}\left[\frac{K_s}{J}\theta(t) + \frac{C_s}{J} + \frac{K'_d}{J}\omega(t) + \frac{K_f}{J}\mathrm{sgn}(\omega(t)) + \dot{\omega_d} - \lambda(\omega(t) - \omega_d(t)) - \eta\ \mathrm{sgn}(s)\right] \tag{4.10}$$

4.3 Adaptive Sliding-Mode Control

Online parameter estimation of the Coulomb friction, K_f, in the plant would be advantageous and seems should be possible because the parameter varies very slowly with time. With an estimate of this value, the control input could be reduced because it would not have to handle the worst case friction. If the estimate shows that the friction is abnormally high (compared with some design specification for the throttle body), a diagnostic message could be displayed.

The Lyapunov function defined in equation 4.1 is altered to include a parameter estimation error term,

$$\frac{\tilde{K_f}}{J} = \frac{\hat{K_f}}{J} - \frac{K_f}{J} \tag{4.11}$$

$$V = \frac{1}{2}s^2 + \frac{1}{2\gamma}\left(\frac{\tilde{K_f}}{J}\right)^2 \tag{4.12}$$

$$\dot{V} = s\dot{s} + \frac{1}{\gamma}\frac{\tilde{K_f}}{J}\frac{\dot{\tilde{K}}_f}{J} \tag{4.13}$$

Because $\frac{\hat{K_f}}{J}$ in equation 4.13 is not equal to the actual Coulomb friction in the plant, the control, $u(t)$, cannot exactly compensate for it. $u(t)$ becomes,

$$u(t) = \left(\frac{K_a}{J}\right)^{-1}\Bigg[\frac{K_s}{J}\theta(t) + \frac{C_s}{J} + \frac{K'_d}{J}\omega(t) + \frac{\hat{K_f}}{J}\text{sgn}(\omega(t)) + \dot{\omega_d} - \lambda(\omega(t) - \omega_d(t)) - \eta\ \text{sgn}(s)\Bigg] \tag{4.14}$$

Using the sliding-mode control law derived before, $\dot{V}$ becomes,

$$\dot{V} = -\eta s\text{sgn}(s) + \frac{\tilde{K_f}}{J}\text{sgn}(\omega)s + \frac{1}{\gamma}\frac{\tilde{K_f}}{J}\frac{\dot{\tilde{K}}_f}{J} \tag{4.15}$$

The first term, $-\eta s\text{sgn}(s)$, is already negative, so the adaption law for $\frac{\hat{K_f}}{J}$ must be chosen such that the remaining terms do not disturb the negative definite property of $\dot{V}$.

$$\frac{\tilde{K_f}}{J}\text{sgn}(\omega)s + \frac{1}{\gamma}\frac{\tilde{K_f}}{J}\frac{\dot{\tilde{K}}_f}{J} \leq 0 \tag{4.16}$$

$$\frac{\dot{K_f}}{J} = 0 \tag{4.17}$$

$$\frac{\tilde{K_f}}{J}\left(\text{sgn}(\omega)s + \frac{1}{\gamma}\frac{\dot{\tilde{K}}_f}{J}\right) \leq 0 \tag{4.18}$$

$$\frac{1}{J}\text{sgn}(\omega)s + \frac{1}{\gamma}\frac{\dot{\tilde{K}}_f}{J} = 0 \tag{4.19}$$

$$\frac{\dot{\tilde{K}}_f}{J} = -\gamma\text{sgn}(\omega)s \tag{4.20}$$

The adaption law for $\frac{\tilde{K_f}}{J}$ given by equation 4.20 causes the terms with $\frac{\hat{K_f}}{J}$ in $\dot{V}$ to cancel each other out of the equation exactly. The expression for $\dot{V}$ becomes the same as it was in the original formulation of the sliding mode controller by the addition of the adaption law. However, there is a substantial difference between the Lyapunov functions. In the new Lyapunov function there is no explicit guarantee on the convergence of $\hat{K_f}$ to K_f. Initially it would appear that once $s = 0$ then $\dot{V} = 0$ and so if $\frac{\tilde{K_f}}{J} \neq 0$, it will not get any better.

Persistency of excitation is the additional requirement for convergence of $\tilde{K_f} \rightarrow 0$. This means that reference signals that do not sufficiently exercise the controller will not cause parameter convergence. It will be shown the simulation results that the reference signal must contain switches in the sign of the desired angular velocity for parameter convergence.

Chapter 5

System Requirements & Operational Description

5.1 Dynamic Response Requirements

The purpose of the throttle is to meter air flow into the intake manifold. In the most common mode of operation, the driver provides a desired throttle position with the throttle pedal. Trying to emulate the behavior of the mechanical linkage with the electronic throttle, places performance requirements on the dynamics of the pedal to throttle position transfer function. It is not practical nor useful to match the driver's requested throttle position with as much accuracy as possible. The speed of response to a change in the requested position and the accuracy with which the throttle is positioned only needs to be on the same order as the speed of response of the engine to a change in throttle position and sensitivity to the throttle position. Using this as a guide, the following system performance requirements were developed.

Rise Time

Rise Time is defined as the time required for the throttle plate angle response to a step change in pedal position to rise from 10% of the steady state value to 90% of the steady state value. The rise time for step changes from closed to fully open is 100ms; and the rise time for step changes from WOT to closed is 60ms.

Settle Time

Settle Time is defined as the minimum time after which the throttle plate angle remains within +5% of steady-state value. ETC shall guarantee that the settle time is less than 40ms after the throttle plate angle reaches 90% of the steady-state value.

Percent Overshoot

ETC throttle plate shall never hit the stops (or zero overshoot).

Steady State Tracking Error

Steady State Tracking Error is defined as the difference between the desired throttle plate angle and final steady state angle. The ETC shall guarantee the Steady State Tracking Error is within the +2

Throttle Plate Angle Resolution

The ETC system shall be able to control the throttle plate angle with a resolution of 0.2 degrees for a range of 0 to 90 degrees.

5.2 Operational Description

Defined below are modes of operation for the ETC system. Each mode may have different inputs and a different objective.

Human Driver

Input:	Commanded throttle angle from accelerator pedal sensor
Function:	Track commanded throttle angle

Cruise Control

Input:	Vehicle speed at the instant that the cruise set button was pressed
Function:	Maintain the set vehicle speed

Rev Limiting

Input:	Engine speed
Function:	Reduce the engine speed below a maximum RPM

Traction Control

Input:	Desired engine torque to regain traction
Function:	Reduce the engine torque to the desired engine torque

Startup

Function:	Power-up hardware, initialize software and enter human driver mode of operation when all components are ready.

Shutdown

Function:	Power-down hardware

Condition	Definition
Cruise Condition	Vehicle speed above 30 mph AND gear selector in drive AND brake switch is inactive AND Cruise switch "On" AND Cruise coast not pressed
Over-rev Condition	Engine speed > maximum engine speed
Traction Ctrl. Condition	Engine torque > maximum allowable engine torque

4. Transitions

Mode 1 $\longrightarrow$	Mode 2	Condition
Startup	Driving	No sensor or actuator faults
Inactive	Cruise	Cruise condition
Cruise	Inactive	NOT cruise condition
Driving	Limiting	Over-rev condition OR traction control condition
Limiting	Driving	NOT (Over-rev condition OR traction control condition)
Inactive	Over-Rev	Over-rev condition
Over-Rev	Inactive	NOT over-rev condition with hysteresis
Inactive	Traction ctrl	Traction control condition
Traction ctrl	Inactive	NOT traction control condition with hysteresis
Driving	Shutdown	Ignition turned off
Limiting	Shutdown	Ignition turned off

5. Throttle Actuation Rules

Mode	Rule
Startup Mode	No actuation of throttle motor
Driving Mode	Throttle motor voltage is calculated with the human control law and the cruise control law, if active, and the greater of the two values is selected.
Limiting Mode	Throttle motor voltage is calculated with the rev limiting control law, if active, and with the traction control law, also if active, and the lesser of the two values is selected.
Shutdown Mode	No actuation of throttle motor

The modes of operation and the possible transitions are shown in figure 5.1. The label, "XOR" indicates only one of the states at the same level can be active, whereas the "AND" label indicates that all the states at the same level are active. When the vehicle ignition switch is turned on, the ETC system enters the startup mode. The throttle actuator should be off while the embedded processor boots. The startup software for ETC should check that the actuators and sensors are not exhibiting any faults and then transition to the driving state. In the driving state, both the human control state and the inactive cruise control state are active. If the cruise control state is activated, then the throttle actuation is calculated based on both the human control and the cruise control and the greater of the two values is used. This method replicates the current human driver interface for cruise control. In current production cruise control systems, the driver is able to increase the throttle angle to pass, but if the driver is not pushing the throttle down as much as the cruise control decides is needed, then the cruise control actuator takes over. If either or both of the limiting mode (rev limiting and traction control) conditions are true, then the limiting state is entered. To exit this state, both the rev limiting and traction control

conditions must be false. Because the objective of either the rev limiting control or the traction control is the reduction of the throttle angle, if both the rev limiting control and the traction control are active, the throttle actuation values are calculated based on both control laws and the smaller value is selected. The shutdown mode must turn off the throttle actuation and then halt the ETC software.

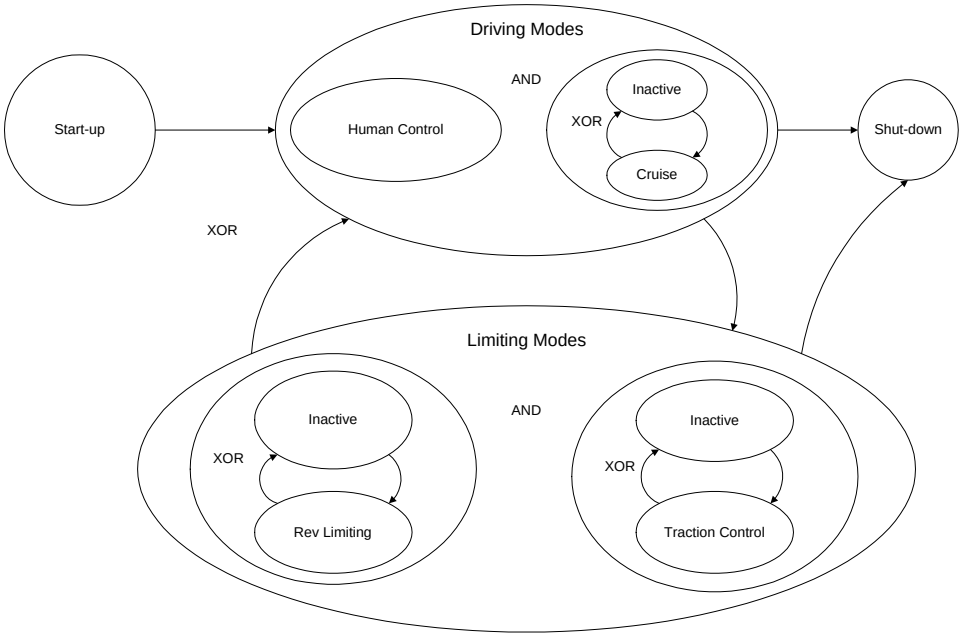


Figure 5.1: Controller Modes

Chapter 6

Embedded Software Design

6.1 Introduction

A model-based design approach for the software development improves on the current industry practice by using the model of the controller as the lowest level of implementation. Ideally, a compiler would use the model as source code and generates executable code for the target platform without any manual steps. There are several potential benefits to this process.

- Reduction in lines of hand-written code
- Reusability of components
- Improved accuracy of simulations
- Better control design

The goal is to eliminate hand-written source code, but if it is not possible to eliminate it completely, at least the portion associated with the controller can be compiled from the models. Reducing the amount of hand-written code translates into a reduction in the amount of time spent debugging and testing code. The reuse of components such as controllers or filters can reduce software development time. Currently, it is common for the source code of controllers to be rewritten or tuned for each new application, but this is an inefficient practice because it does not fully leverage previous control design work. If the plant, sensor, and actuator models contain sufficient detail, stock controllers can be modified at the model level, which minimizes redundant development work. [1] Besides

providing a more detailed model of the environment, the information about the software and hardware environment, which is needed by the model compiler, improves the accuracy of simulation. If software effects such as latency and jitter affect the controller performance significantly, it will be seen during simulation rather than during testing. Also, information from the software can be used to improve the performance of the controller or to reduce the controller's use of hardware resources, while maintaining the same performance. All of these benefits equate to reduction in development time and system cost.

6.2 Background

The current automotive industry practice for design of control systems is to separate the task of control design from the task of implementing the controller in software. [12] There are two problems with this division of labor. First, there is a duplication of effort when the controllers are implemented once as models and again as source code. The control engineers model the controllers in a high-level modeling language for simulation and the software engineers re-implement the controllers in a low-level language such as C. The second problem with the separation of the control design and the software design is that the control is unable to use information from the software environment to improve the performance or reduce the demand on the hardware. This means that the controller is not making optimal use of the available information. Moreover, there are hardware and software interactions that are not present in simulation because there is no model of the software part of the system. The primary effects of the software environment on the controller are jitter and additional end-to-end delay. The result of these software phenomena

on the system performance can be significant enough that the system design requirements are not met even though they were met during simulation. These are the problems that the model-based design approach targets.

6.3 Model-Based Design

The principle of the model-based design is that the high-level model of a controller is the source code. Instead of modeling the system merely for simulation, the controller portion of the model serves as the simulation and the implementation. There is no separate low-level implementation. There are potentially large benefits of this approach to embedded software design, measured in reduction of development time and reduction of system cost when compared with the current industry practice.

6.3.1 Motivation

The standard process for design and testing of embedded systems is shown in figure 6.1. [12] The process begins on the top left with the specification of requirements and flows downward through design until an implementation is realized. On the right side of the "V" are the testing phases that correspond to design phases at the same level. Iteration is a necessary part of the design process when a problem is discovered that requires some design change at a previous step. A change that requires repeating many parts of the process can be very costly. If, for example, a system level test fails and it is determined that the requirements specifications were incorrect, the entire design process may need to be repeated. The objective of the model-based approach is to do as much design work as

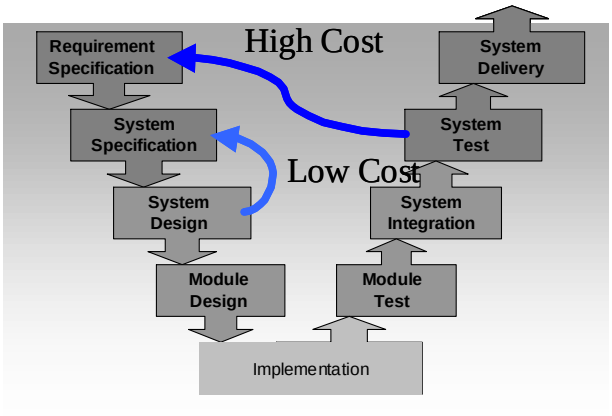


Figure 6.1: "V" Process for Design and Testing of Embedded Systems (Source: Man-Feng Cheng, General Motors Corp.)

possible by simulation, analysis and testing with models, so design changes are relatively cheap and quick to implement. In order to do this work, modeling tools are required, which facilitate the modeling of the controller and the software with sufficient accuracy so that problems can be discovered or even prevented during the design phases by simulation or analysis. This need to model the software of the embedded control system motivates the subsequent discussion of modeling constructs needed to model the software.

6.3.2 Embedded Software Modeling

There are some basic modeling constructs that are needed to model a real-time software system properly:

- Tasks

- Scheduler
- Control-flow and Data-flow

Tasks

A task is a piece of sequential code that can be scheduled to run by the operating system. In terms of a language such as C, a task could be some function. The scheduler in the operating system knows how to call the function and accepts requests for the task to run. [9] There are some properties of a task that are useful but not always necessary. Usually an integer priority is assigned to a task. This value is used to determine, which of task of several that are waiting to run should go first. Another property that can be used for analysis or by the scheduler is worst-case execution time. This is an upper bound on the time required for the task to execute once. In terms of the model, when a section of the controller model is associated with a task, it is updated only when the scheduler tells it to run, and there is a delay between the invocation of the task and the availability of the output.

Scheduler

One of the basic services of a real-time operating system is the scheduler. Although the exact functionality of the scheduler varies by operating system, the most basic functions of the scheduler are the ability to take requests for tasks to run, to decide which of a number of tasks waiting to run should run, and to invoke a task. The scheduler decides which task should run based on an algorithm such as earliest-deadline-first (EDF) or priorities

generated by rate-monotonic analysis (RMA). For EDF scheduling, a deadline must be associated with the request for a task to run, and for rate-monotonic analysis, each task has a static period and the shortest period task receives the highest priority. [9]

Control-flow and Data-flow

Any algorithm that can be written in a programming language such as C, can be thought of as a combination of control-flow and data-flow. Programming constructs such as if-then-else are part of the control-flow of a program and calculations and assignment operations are part of the data flow. [4] Because the objective is to replace the lower level source code with the model of the controller, there needs to be a way of representing control-flow and data-flow of the controller algorithm in the modeling language. There are various ways of representing control and data-flow. It is important that the representation is intuitive and maps to an efficient and unambiguous implementation. One such way to represent the logic of control-flow is with a finite state machine. Actions can be taken upon entering, leaving or remaining in a state. It is a particularly good choice in a control system, in which a change of state is rare, because knowledge of the current state can be used to achieve the minimum number of guards (i.e. conditions that cause a state transition) that must be checked frequently.

6.4 Electronic Throttle Control Software Design

6.4.1 Modeling

Simulink and Stateflow were selected as the modeling environment for the ETC system because this is an automotive industry standard. [12] A model in the Simulink environment consists of a hierarchical structure of blocks whose input and output are specified by wires drawn between blocks. Figure 6.2 shows the top-level of the ETC system as it appears in Simulink (except for the removal of wire labels for readability). The simulation contains the controller and the dynamics of the environment. The controller interacts with the physical environment through the sensors and the drivers. These two components are the interface of the controller software to the physical world. When code is generated from the controller model, it requires the same interface to a set of sensors to measure plant states and drivers to run the actuators. The focus of the ETC case study is on the modeling of the controller.

Sufficient detail in the model of the system is important when designing a control system that must make full use of the limited hardware capabilities. It is convenient to assume that sampling is sufficiently fast, that the code requires approximately zero time to run, or that the pulse-width modulation (PWM) of the actuator is of sufficiently high enough frequency to be ignored. These approximations were not made in the ETC models so that the performance could be examined when the system is running at the margin. The expected benefit is that a more intelligent controller can be developed.

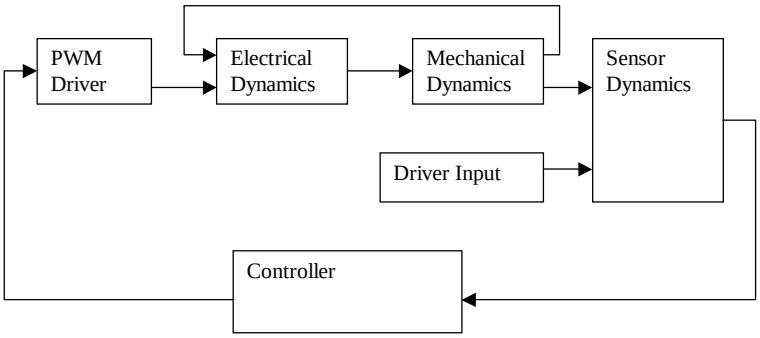


Figure 6.2: Top-level view of the ETC system

6.4.2 Controller Design

Control Flow

The ETC system has multiple modes of operation that require different control law calculations. Control flow is required to route the path of execution through one or two calculations and not through others. Figure 6.3 shows the states of the ETC system. In the driving mode, the human driver mode and one of the cruise control modes are active in parallel. This means that both the control law for following the accelerator pedal position and the control law for maintaining the vehicle speed are calculated, which gives two values for the desired motor current. The greater of the two values is used as the final output. In the limiting mode, there is a similar situation with parallel states in the rev. limiting and traction control modes. Again two calculations are made, but in the limiting mode, the lesser of the two values is used for the final output. A State-chart (part of Stateflow) is

used to represent the modes, invoke a Simulink calculation from each mode and express the conditions for switching between modes. The order of execution of parallel (i.e. AND) states in a State-chart is established by the relative position of the two states. Order of execution is an important part of the control-flow, so it is necessary that this property is specified, but using the graphical position is a poor method and a weakness of the Stateflow interface. Nevertheless, specifying order of execution makes it possible to create one state, whose calculations are used to affect the state transitions or calculations of a second parallel state, which are executed later but without a one controller period delay. In the ETC system, a fault-detection state is executed before the controller mode state is selected. This way, if a fault is detected, the subsequent evaluation of the controller mode state can immediately switch into the fault-handling mode instead of being delayed by one period of the controller.

Data-flow

Wires that connect blocks show the assignment of the output of one block to the input of the next. In the Simulink models of the ETC system, there is a distinction between continuous-time and discrete-time valued wires. For instance, all the state values of the plant are continuous time values and all values in the controller and in its set of input and output values are discrete. The sensor dynamics include sampling of continuous time values, which converts those values to a train of discrete values. On the output side of the controller, discrete time values for desired current are produced as controller output and at the beginning of each PWM cycle, the PWM receives the value, copies it locally and applies voltage to the armature until the desired current level is reached. Copying the desired current value locally is similar to a zero-order hold, although it is not precisely the

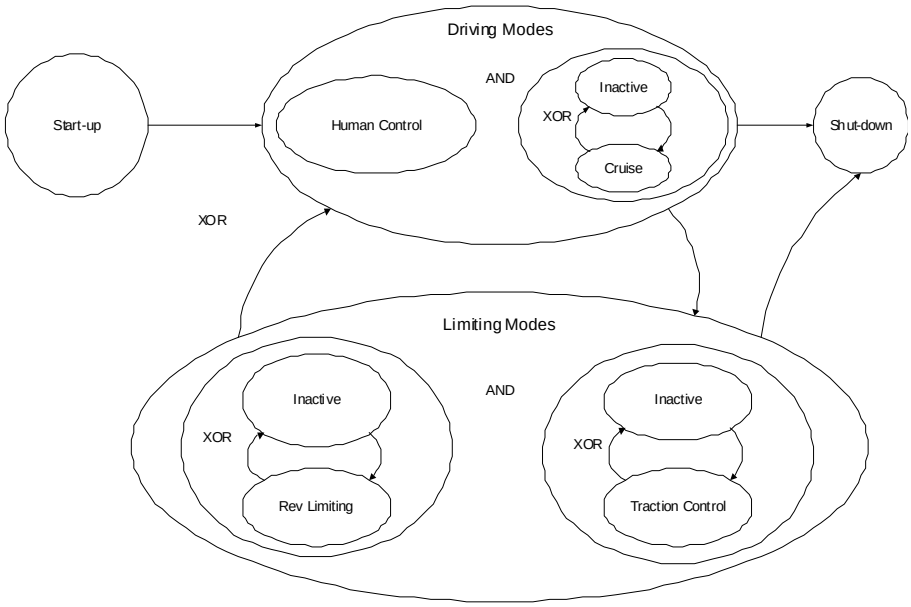


Figure 6.3: States of the ETC system

same, because the PWM cycle does not match in phase or frequency with the period of the controller.

Inside the controller there are three tasks, each of which runs periodically at a different rate. Because a task has some execution time, then the inputs must be buffered by copying to a local set of values. Then if an input is changed during execution of a task, that change in value cannot affect the calculations being performed within the task.

Tasks

A stylized usage of Simulink is used to represent tasks. Within the controller subsystem, there are subsystems that represent tasks and switching a particular trigger signal runs a task. Because one run of a subsystem requires zero simulation time, the output of a task must be delayed by some amount to account for the true delay from input to output. One simple way to handle this is to apply a one-step delay to the outputs of each task. The assumption is that the output from the previous invocation of a task must be ready by the time of the next invocation. If a more realistic delay is required to improve the system performance, then the scheduler can be used to retrieve the output of a task some time after the invocation of the task, which then corresponds to the execution time.

The ETC system is composed of three tasks, which are triggered by a scheduler. Figure 6.4 shows the structure. In the Simulink/Stateflow representation of the model, the tasks, manager, monitor and servo-control are subsystems triggered by the scheduler. The scheduler is driven by a 1 ms clock input. The tasks have different periods because some parts of the control can be run much slower than other parts. More specifically, the servo-control is responsible for the closed-loop control of the throttle position and needs to be

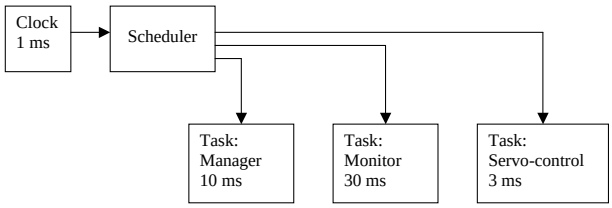
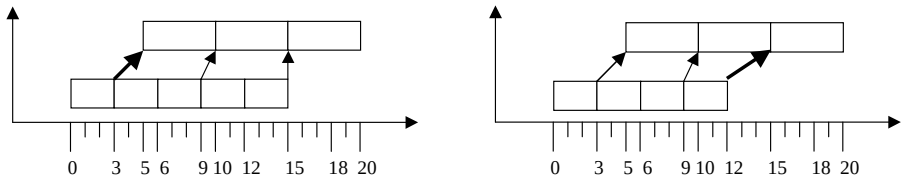


Figure 6.4: Triggering of tasks inside the ETC controller

run at over 300 Hz, but the manager is responsible for determining which of several types of control (pedal-following control, cruise control, traction control or engine RPM limiting control) should be active, and need only be performed about 100 times per second. The monitor implements some fault detection, one of which is checking that the manager and servo-control have run at least once in the previous period. By setting its period longer than either the manger task or the servo-control task, then the manager and servo-control should run one or more times for each run of the monitor.

The effect on controller performance of dividing the controller into tasks is the addition of delay. As an example, consider the delay between two tasks, A and B, that transfer data from one to the other. Suppose the following:

- A has a period of 3 ms
- B has a period of 5 ms
- The output of A is not available until the end of its period



(a) A runs before B : max 2 ms delay (b) A runs after B : max 3 ms delay

Figure 6.5: Task timing

- The output of A is the input of B
- If A and B both need to run at a given instant, then A runs before B

The worst-case delay from A to B is 2 ms. If the last condition is inverted so that B runs before A if at a given instant they both need to run, then the worst-case delay from A to B is 3 ms. Clearly, the latency of data-flow between tasks can be of the same magnitude as the latency from input to output of a task.

Scheduler

The scheduler is responsible for triggering the execution of tasks. In the ETC system, all the tasks have constant rates, so the scheduler has a static logic that triggers the three subsystems at different multiples of the base clock rate. This triggering is done by a Stateflow block, which has a fire and wait state associated with each task. Upon entering

a wait state, a counter is set to zero, and for each subsequent clock tick, the counter is incremented by one. When the counter reaches a threshold value, a transition to the fire state is taken and the task is invoked. After the fire state, the wait state is re-entered and the process repeats itself. The Stateflow diagram that implements this simple scheduler is shown in figure 6.6. Before entering the fire states that invoke the tasks, the output of previous invocations of the tasks is retrieved and the new inputs for the next invocations are buffered. The buffering of the inputs and outputs makes this diagram into a simple Giotto program. See [5] for a description of Giotto. One feature of this design is that the order of execution for the parallel fire states is irrelevant. More complicated scheduling logic is possible, but there is also a limitation to this model framework. One type of task that cannot be simulated in this framework is one with an internal synchronization point. When Stateflow invokes the task, it is an uninterruptible action. Moreover, it would be more useful for code generation to specify a type of scheduler and parameters rather than the logic for the scheduler.

6.5 Conclusions

The ETC system is suitable for the model-based design approach of software design because it is a practical real-time embedded software system with stringent cost limitations and high safety and reliability requirements. Simulink/Stateflow is used as the modeling environment because it is a current industry standard. With respect to software modeling, however, there are some deficiencies in the Simulink/Stateflow modeling environment. Modeling the scheduler and tasks can be done by using a pre-defined style or the libraries

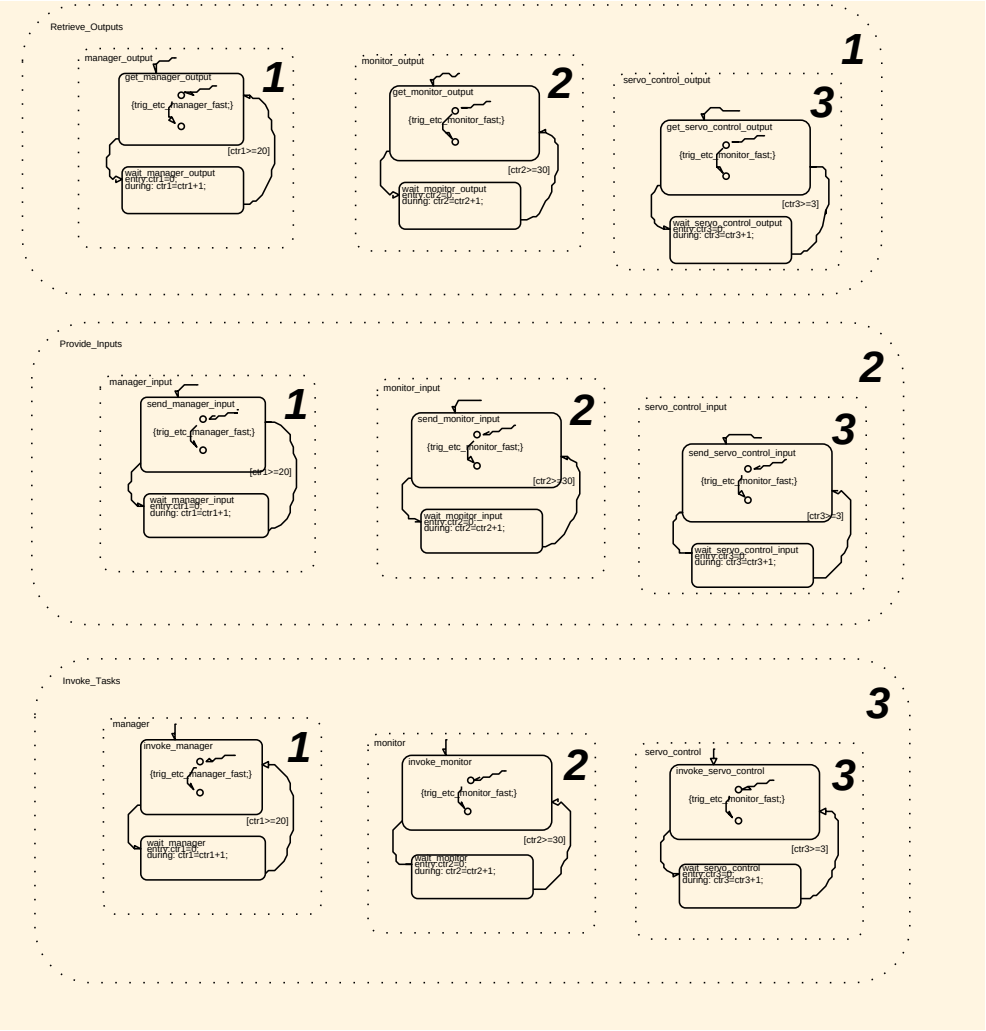


Figure 6.6: ETC static scheduler implemented in Stateflow

could be extended to include some generic scheduler and task blocks. It appears that efficiently modeling complicated tasks (a task with internal synchronization points) would require altering the Simulink/Stateflow simulation engine.

The model-based design requires extra modeling effort up front in return for benefits later in the design process. Creating a model that is used for simulation, analysis, testing as well as the source for the control software, eliminates the usual duplication of effort needed to convert the models manually to source code. Reducing the amount of hand-written code in the embedded software system will reduce the time spent in debugging and maintenance of the code. The additional modeling detail, such as software structure, increases the accuracy and information available from simulation, and as a result increases the likelihood that the actual system will behave like the simulation and that problems will be discovered during simulation. Once a generic model component has been fully developed and tested, it can be stored in a library for later use. In this manner, an organization can fully leverage previous design work. The replacement of source code with models and the reuse of model components requires more detailed and structured models, but promises to reduce the development time and total system cost of embedded control systems.

Chapter 7

Simulink/Stateflow Models

7.1 Modeling in Simulink/Stateflow

7.2 Complete System Model

The top most level of the ETC model contains the driver, actuator plant, sensor and controller models and specifies the interconnection of these models. This level of the model contains a mixture of continuous and discrete signals. For instance, the sensor values provided to the controller and the controller output are discrete signals, but the motor current and the throttle position are continuous signals. The driver inputs and powertrain inputs provide dummy data as a substitute for inputs from the human driver and the powertrain. The feedback around the actuator and plant model reveals the coupling between the electric dynamics of the motor and the mechanical dynamics of the throttle plate. The signals available to the sensors are strictly those that can be measured with the hardware and the controller receives these signals with noise.

7.3 Plant Model

The plant model in figure 7.2 represents graphically the second-order, nonlinear differential equation of the throttle plate mechanical dynamics. The presence of the integrator blocks in the diagram requires that the system be a continuous model. The nonlinear sign block models the Coulomb friction and the nonlinear saturation block models the physical constraint of the throttle plate within the range of 0 to 90 degrees. The back EMF output is feedback to the actuator model and the throttle angle is provided to the sensors.

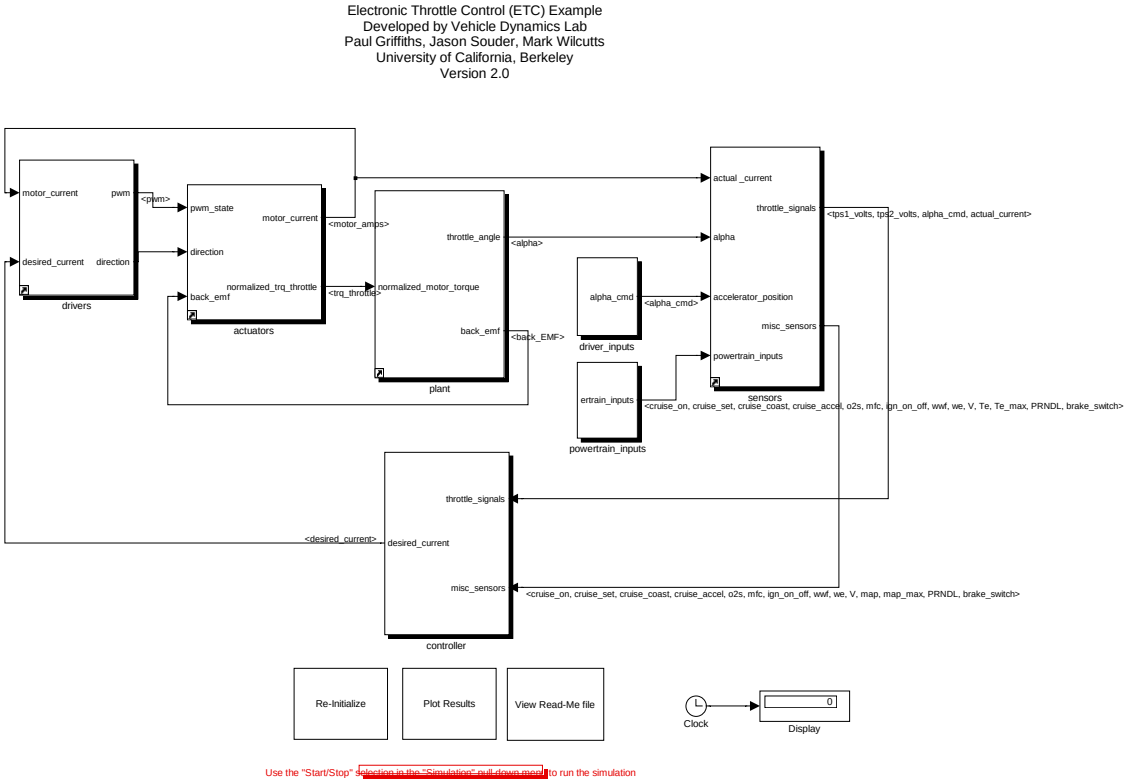
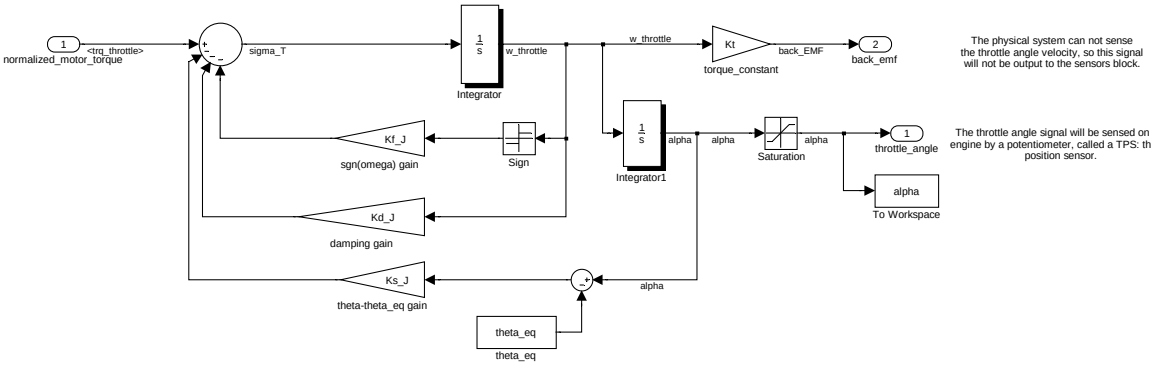


Figure 7.1: Top-level ETC model



This is a model of the mechanical portion of the throttle valve.
Note that the throttle valve and the electric motor used to actuate the valve
have been separated out here into a "Actuators" and "Plant" section.
In the vehicle model, they may be integrated into a single actuators block.

Figure 7.2: Model of the throttle plate dynamics

7.4 Sensor Models

The sensor model in figure 7.3 converts the ideal signals that are measured by the hardware into realistic signals with noise and then samples the signals to provide a train of discrete sensor values to the controller. The single throttle position is split into two throttle position signals with uncorrelated noise. They are also correctly converted to voltages that complement each other such that they add to approximately 5 volts under normal operation. A switch block is provided so that the signals can be grounded to simulate a fault. After the sample block, the signals are delayed by one sample period to model the acquisition delay of the analog-to-digital converter hardware. Figure 7.4 looks inside the acquisition delay block, which reveals a unit delay block. Figure 7.5 shows that the sampling behavior is accomplished by simply passing the signals straight through a sub-system, but since the sub-system is triggered, the continuous input shows up on the output only when triggered.

7.5 Controller Model

The top-most level of the controller model, shown in figure 7.6, reveals the scheduling of the controller tasks and the input and output that the controller has with the environment. The scheduler component is triggered by a 1 ms clock and generates triggers for the three controller tasks. Figure 7.7 shows how the Stateflow chart generates triggers at the three different rates for the three tasks. A software failure can be simulated by grounding the trigger signal for one or more of the tasks.

Whereas the top-most level of the controller model captures the controllers interaction with the environment, figure 7.8 reveals how data moves between tasks.

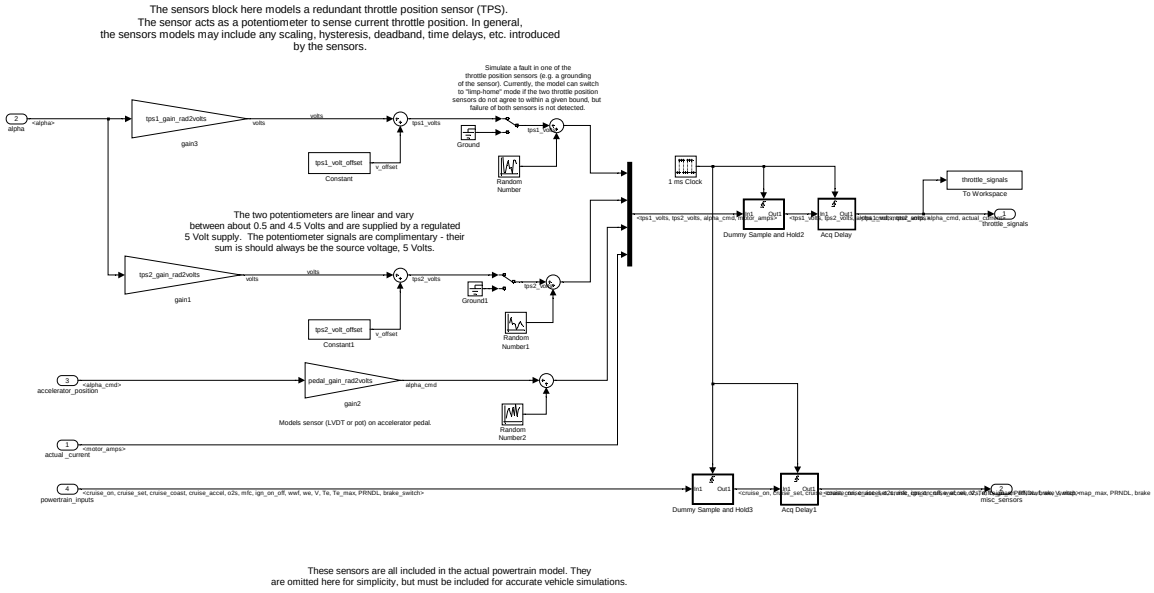


Figure 7.3: Model of the sensors with noise

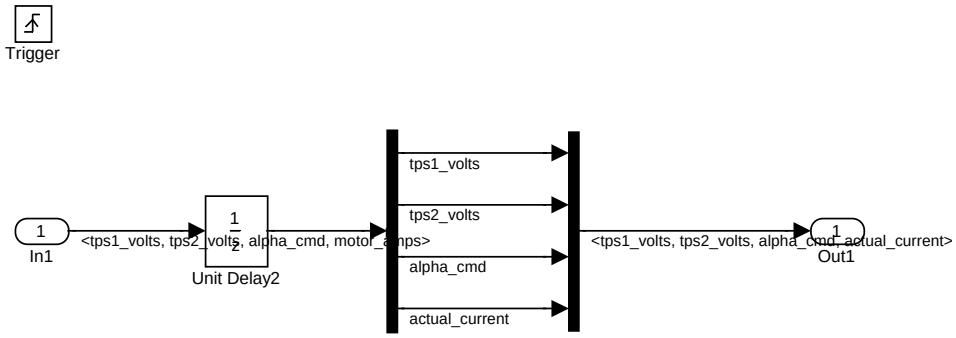


Figure 7.4: Model of the acquisition delay

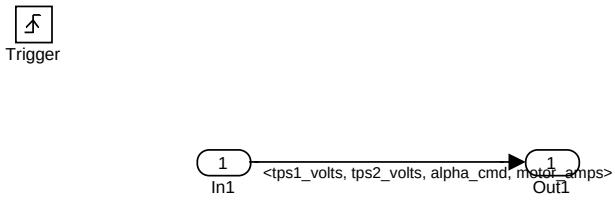


Figure 7.5: Model of the sensor sampling

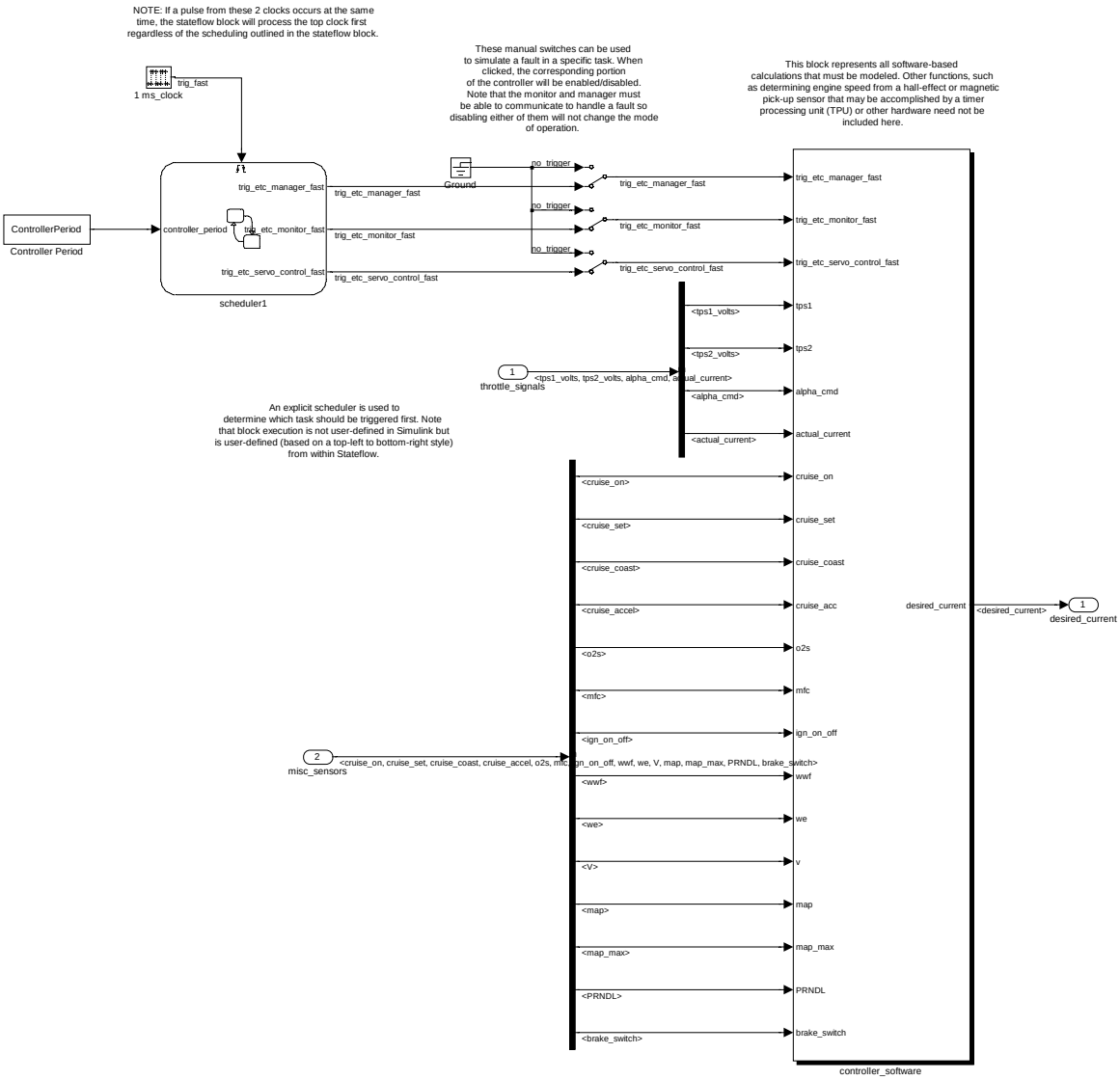


Figure 7.6: Top-most model of the control software

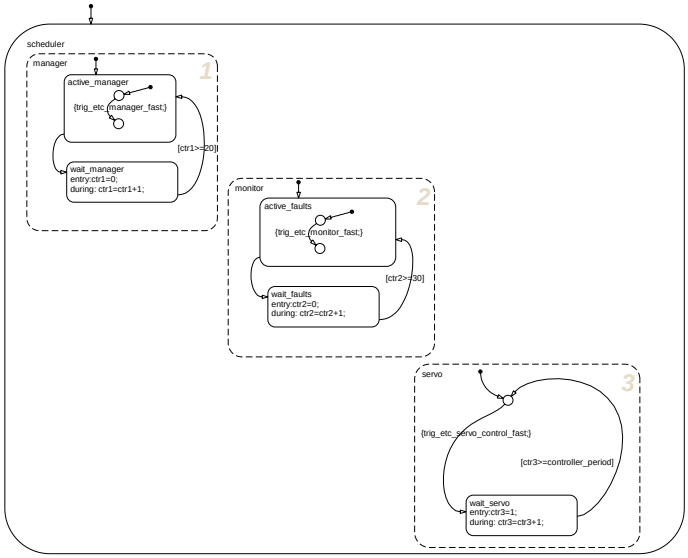


Figure 7.7: State-chart to trigger the execution of controller tasks

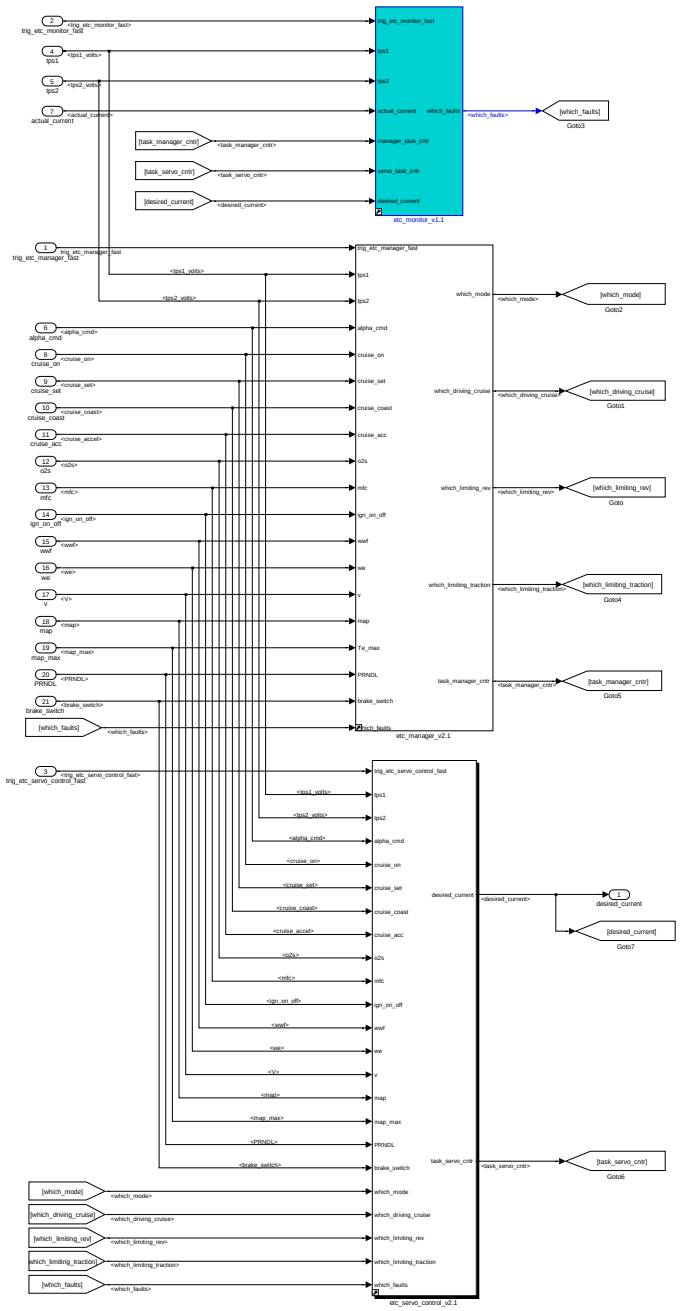


Figure 7.8: Model of the data-flow between tasks

7.5.1 Manager Task Model

Figure 7.9 and 7.10 define the complete model of the manager task. The numbers in the parallel states of the manager logic indicate the sequence in which the states are processed. First the task counter is incremented and the inputs are sampled. Then the real work of the manager is taken care of, which requires deciding which mode of control should be run based on the current inputs from the driver and the powertrain.

7.5.2 Monitor Task Model

The monitor task checks for software and hardware faults. The top-level model of the task is shown in figure 7.11. The Stateflow chart in figure 7.12 models the control-flow of the task. First the inputs are sampled. Then checks are made for software faults and hardware faults.

To check for a software fault, the monitor task reads the task counters from the manager and servo-control tasks each time it runs. Because the monitor task runs at the lowest frequency of the three tasks, the manager and servo-control task counters should have incremented several times. If a comparison against the last task counter value shows that either of the tasks has not run, then a software error is flagged.

There are two possible failures of the throttle position sensors that may occur. If the voltage source fails, then both signals will be off. If one of the potentiometers fails, then only one signal will be incorrect. In either case, a mismatch between the signals will indicate an error.

The motor current measurement is used to detect a failure of the motor. The

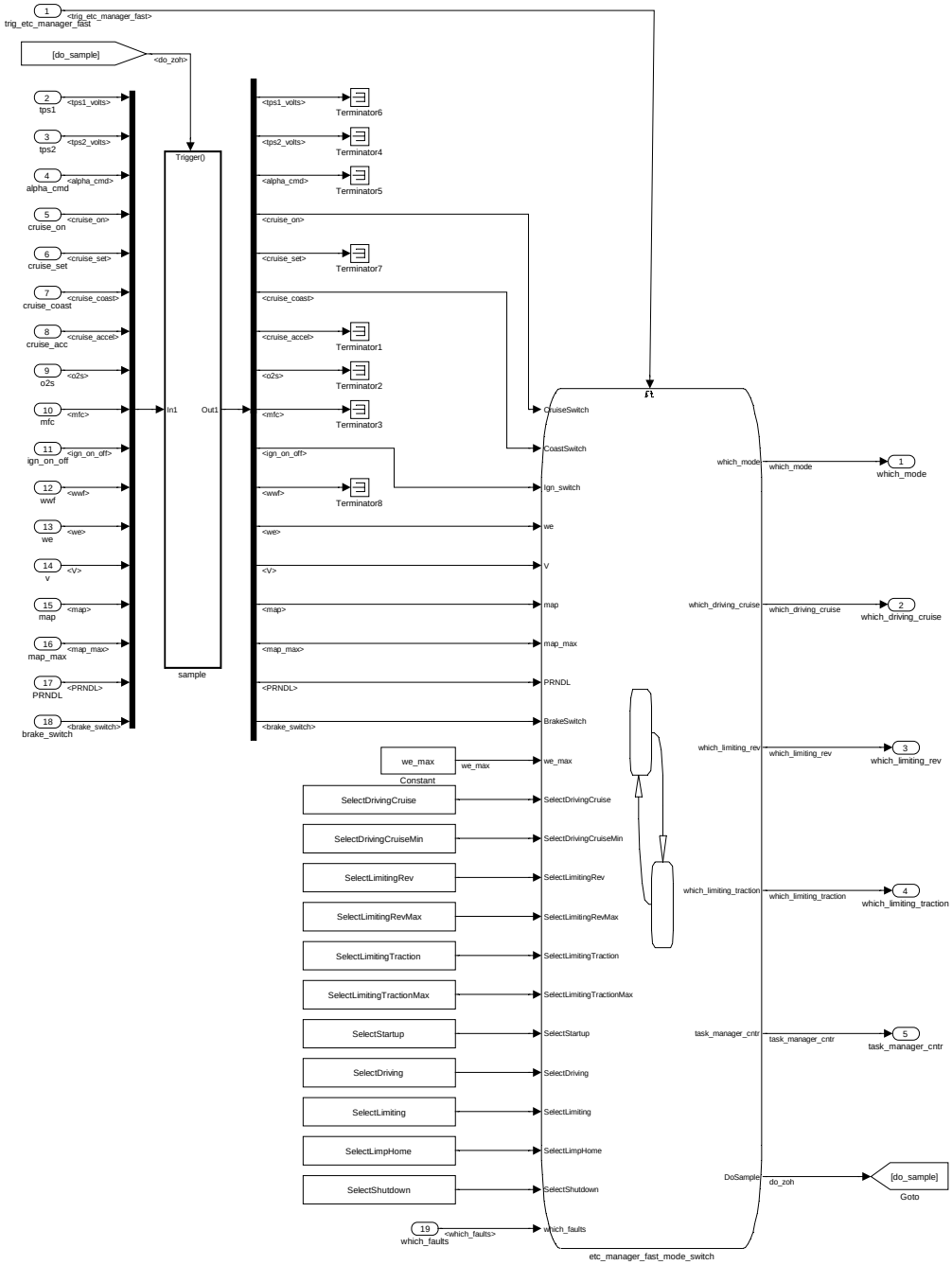


Figure 7.9: Model of the manager task

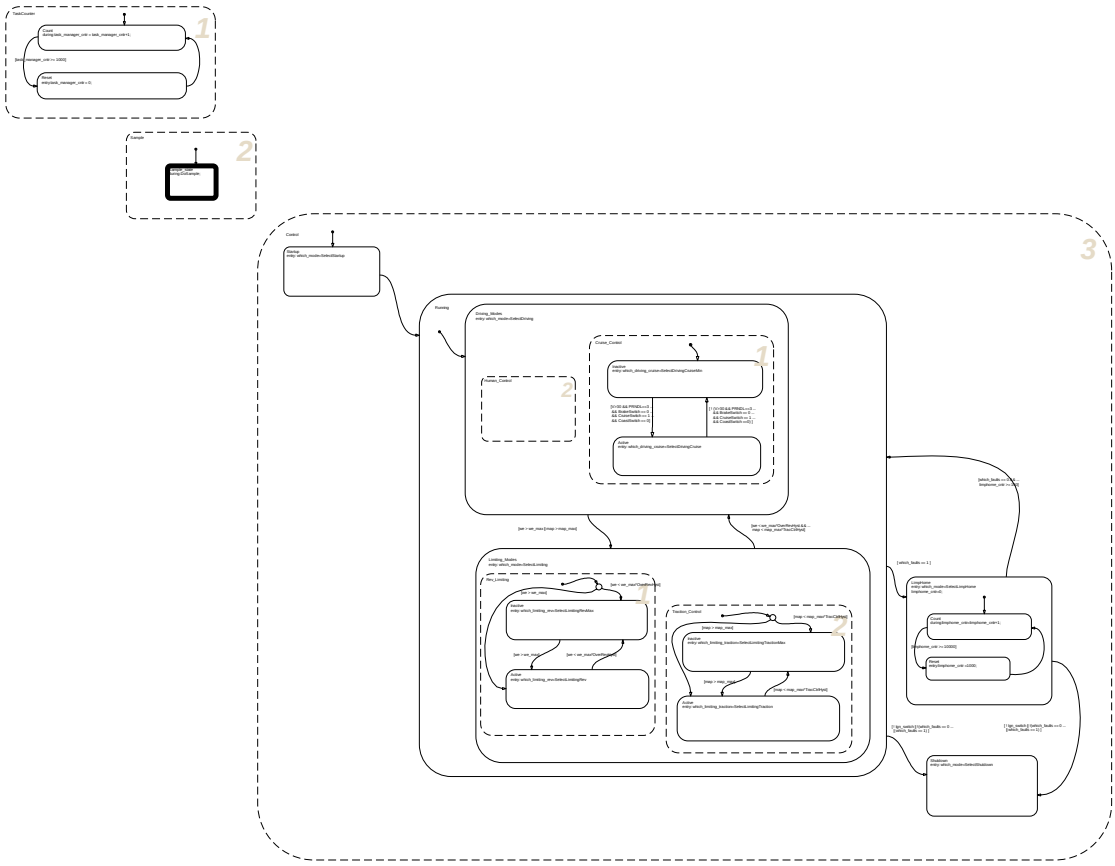


Figure 7.10: Model of the manager task logic

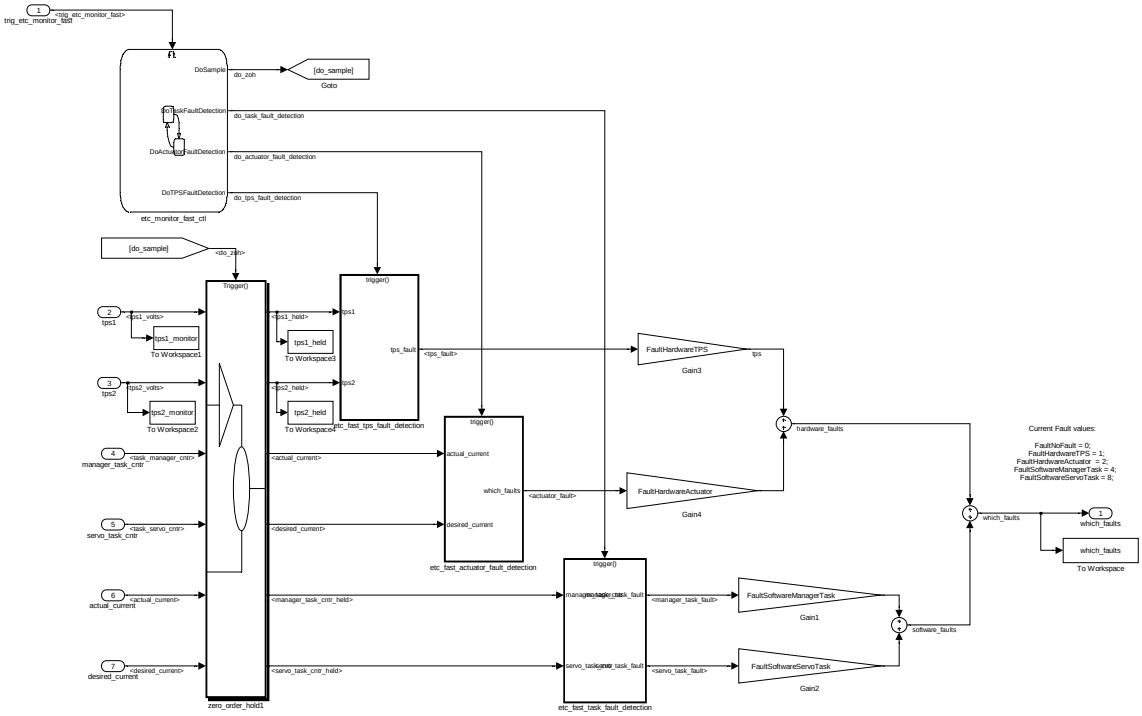


Figure 7.11: Model of the monitor task

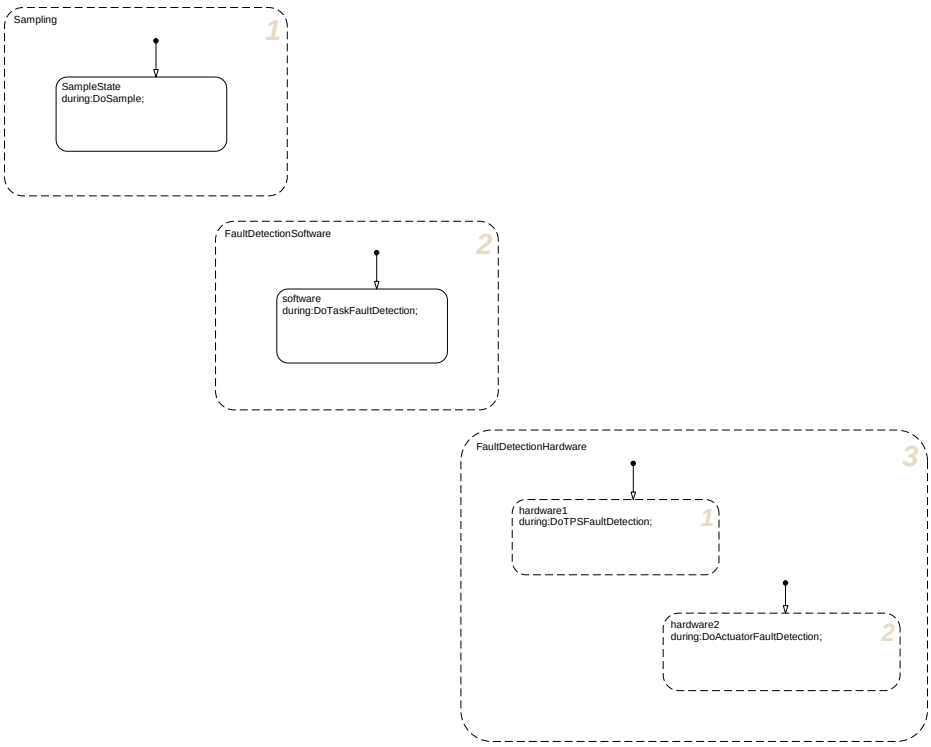


Figure 7.12: Model of the monitor task logic

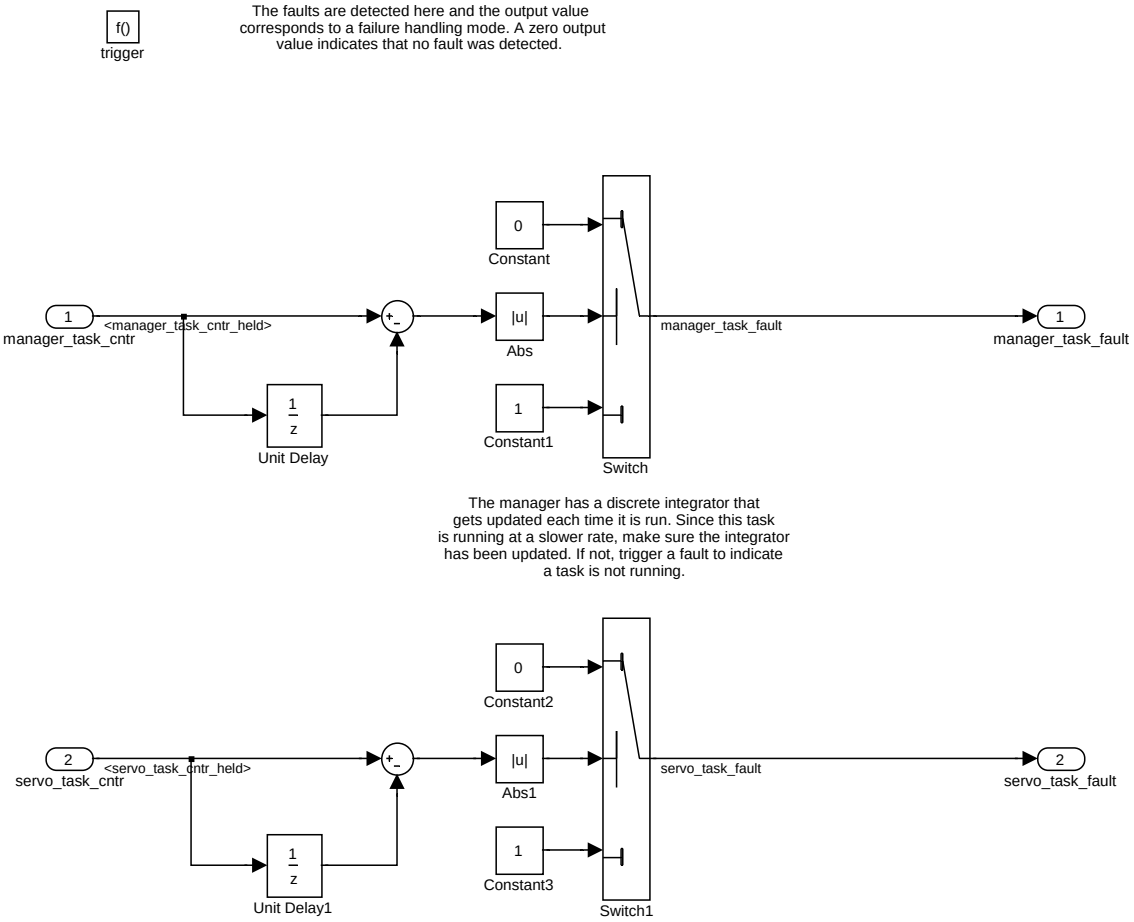


Figure 7.13: In this sub-system, the current task counters of the manager and servo-control task are compared against the last set of values.

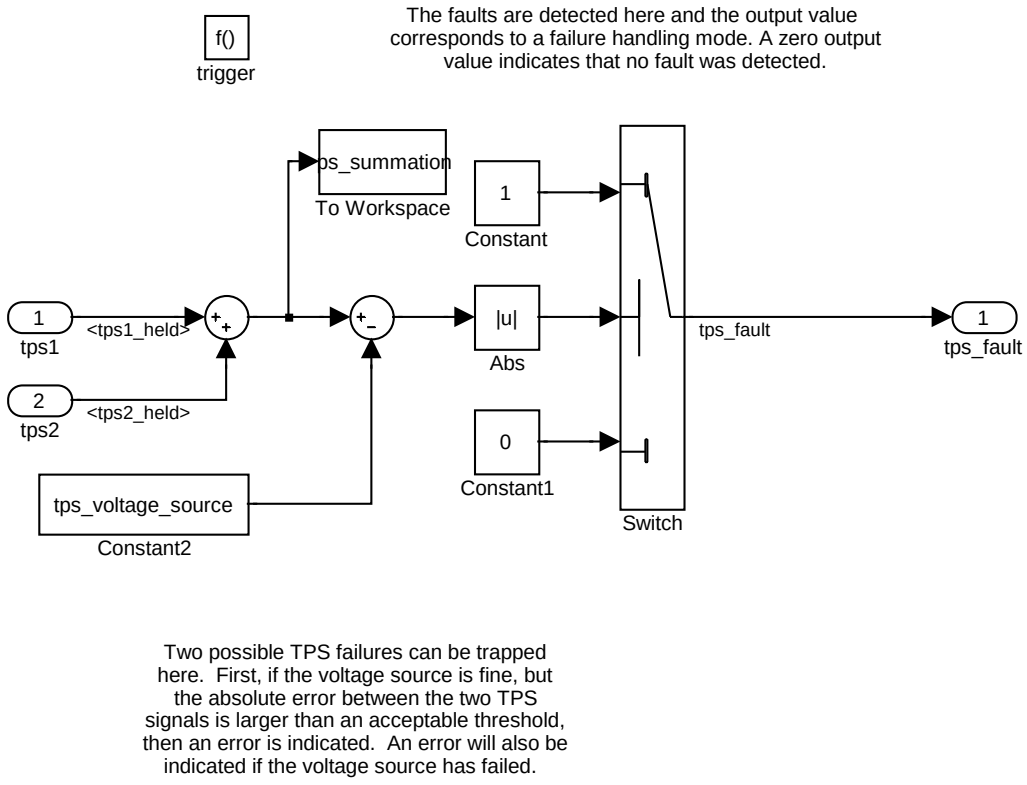


Figure 7.14: An attempt to detect a failure of the throttle position sensors is made in this model.

controller produces a desired motor current as an output, against which the measured motor current can be compared. Some allowance most be made for a mismatch between the two since the desired current may change more rapidly than the actual current can change.

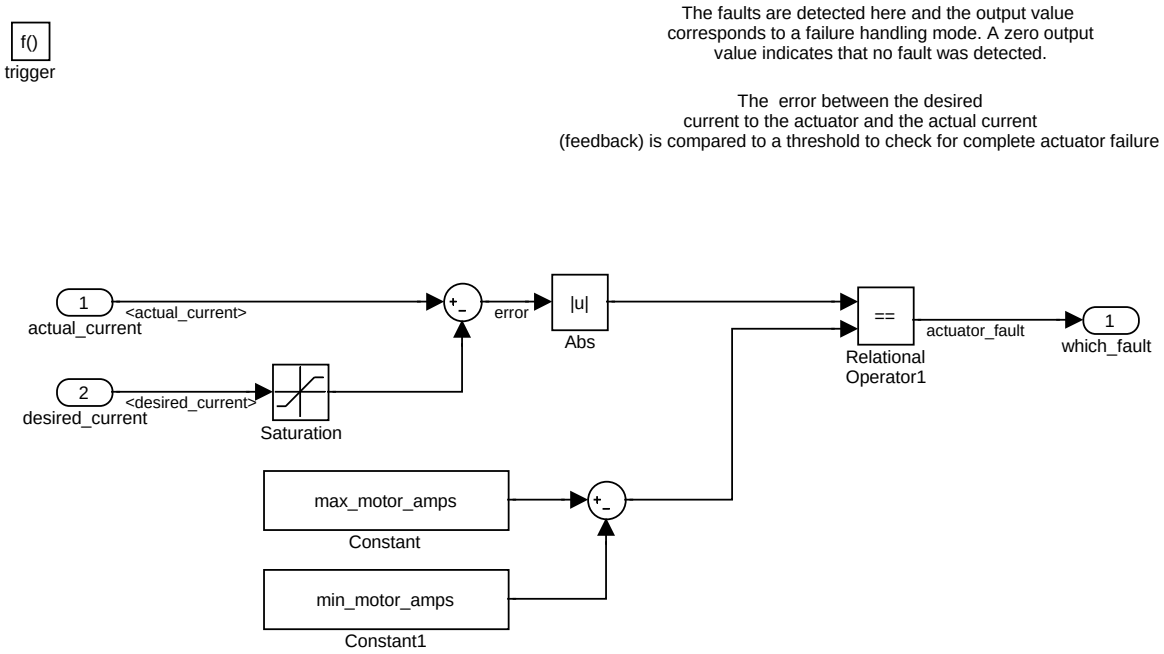


Figure 7.15: An attempt to detect a bad failure of the motor is made in this model.

7.5.3 Servo-control Task Model

The top-level model of the servo-control model in figure 7.16 shows the Stateflow chart that triggers the appropriate sub-systems. Just like the manager and monitor tasks, the inputs are first sampled. Then filtering of the input signals is triggered. Based on the inputs from the manager, the appropriate control algorithm is selected by triggering one of the controller sub-systems and a switch input is provided so that the matching output is captured in the end and returned as the final controller output.

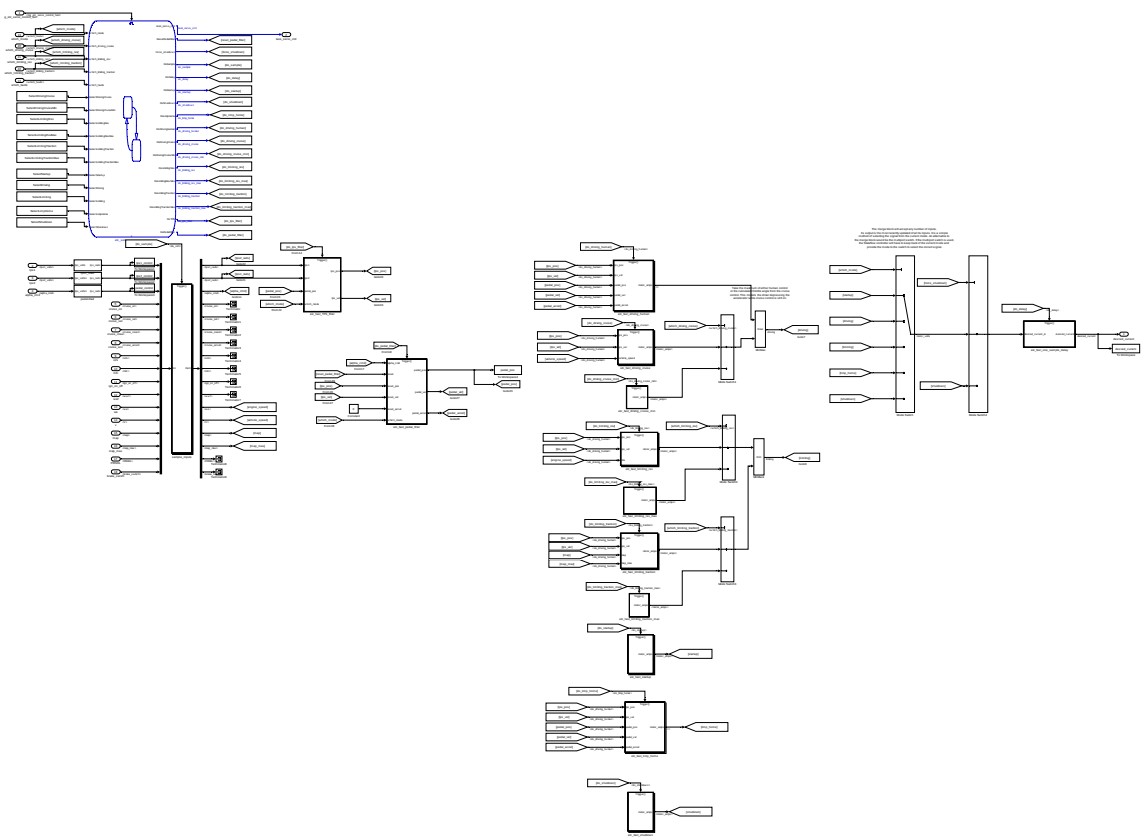


Figure 7.16: The top-level of the servo-control task model

There are two TPS signals provided to the controller. In the model of the TPS filter (figure 7.17) a choice is made between using the average of both signals or trying to determine if the one best signal. Figure 7.18 shows the averaging of the two signals and figure 7.19 shows how the signal which is closest to the desired value. (This method would be more reliable if the TPS signals were compared with the expected response based on the commanded motor current and the measured current.)

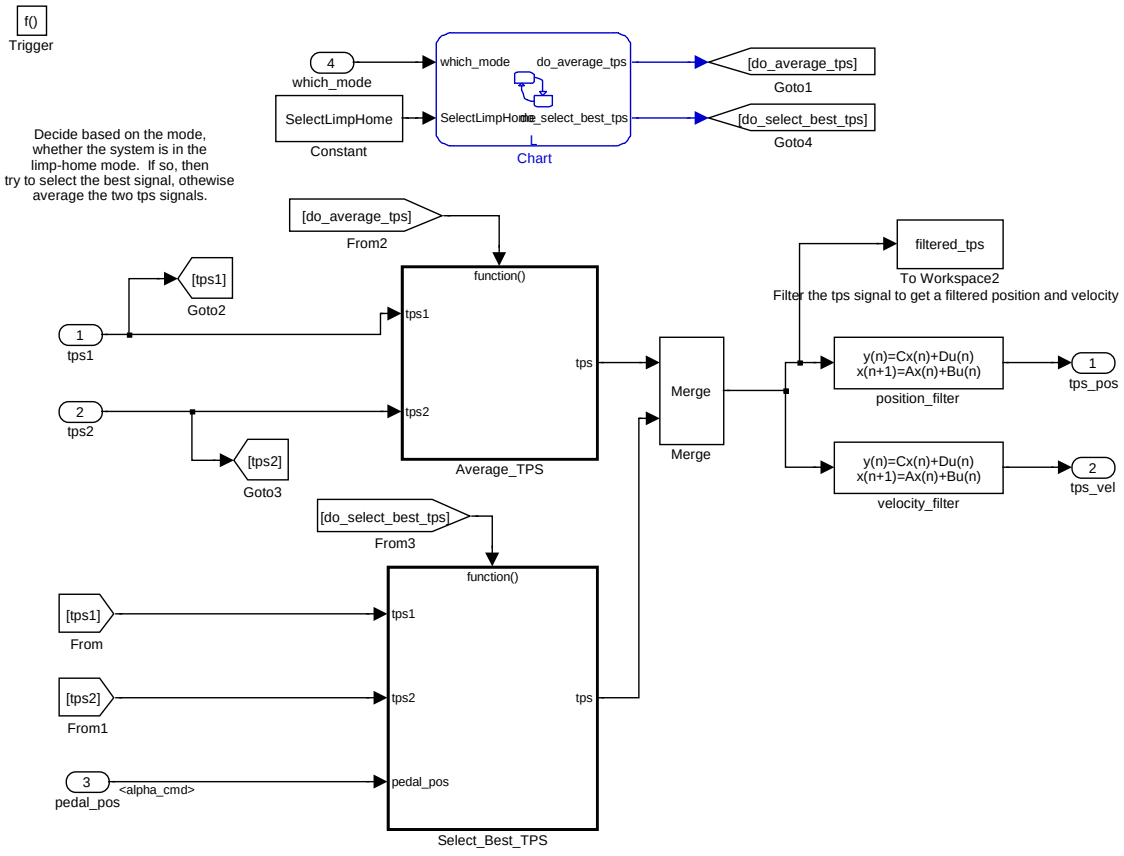


Figure 7.17: Filter for the TPS signal

A resettable non-linear filter is applied to the pedal position in order to provide

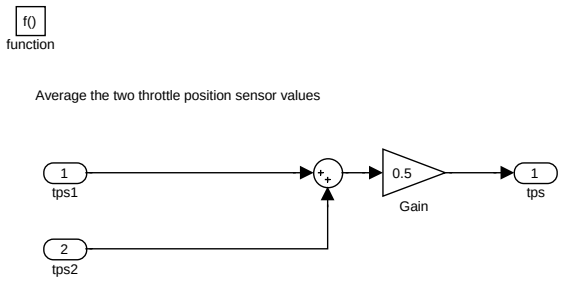


Figure 7.18: The two TPS signals are averaged to obtain a better estimate of the throttle position.

a reference position and velocity that are continuous and "slow" enough that the closed-loop system can track them. If the reference signals deviate too much from the actual plant states, the response to standard test signals, like a step input, is degraded. This filter design simulates the closed-loop system with a model of the plant in figure 7.21 and a simplified model of the controller in figure 7.22. There are two important features offered by this filter structure. The plant model includes the saturation of the motor so the dynamics of the plant states are limited by this important nonlinearity. Also, the fact that the filter states are also the plant states makes resetting the values to the plant values relatively easy. This reset feature is used whenever there is a switch between control laws.

The controller modeled in figure 7.23 is the control law defined in equation 4.14 with the adaption update given by equation 4.20. A manual switch towards the top of the

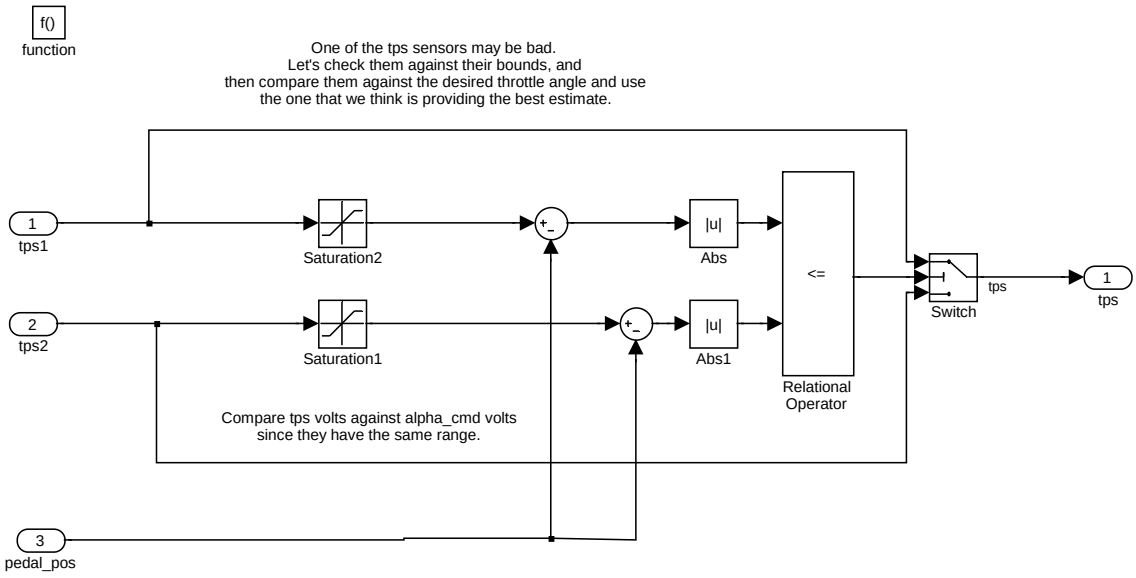


Figure 7.19: This model is used to select one TPS if the monitor has indicated that one has failed

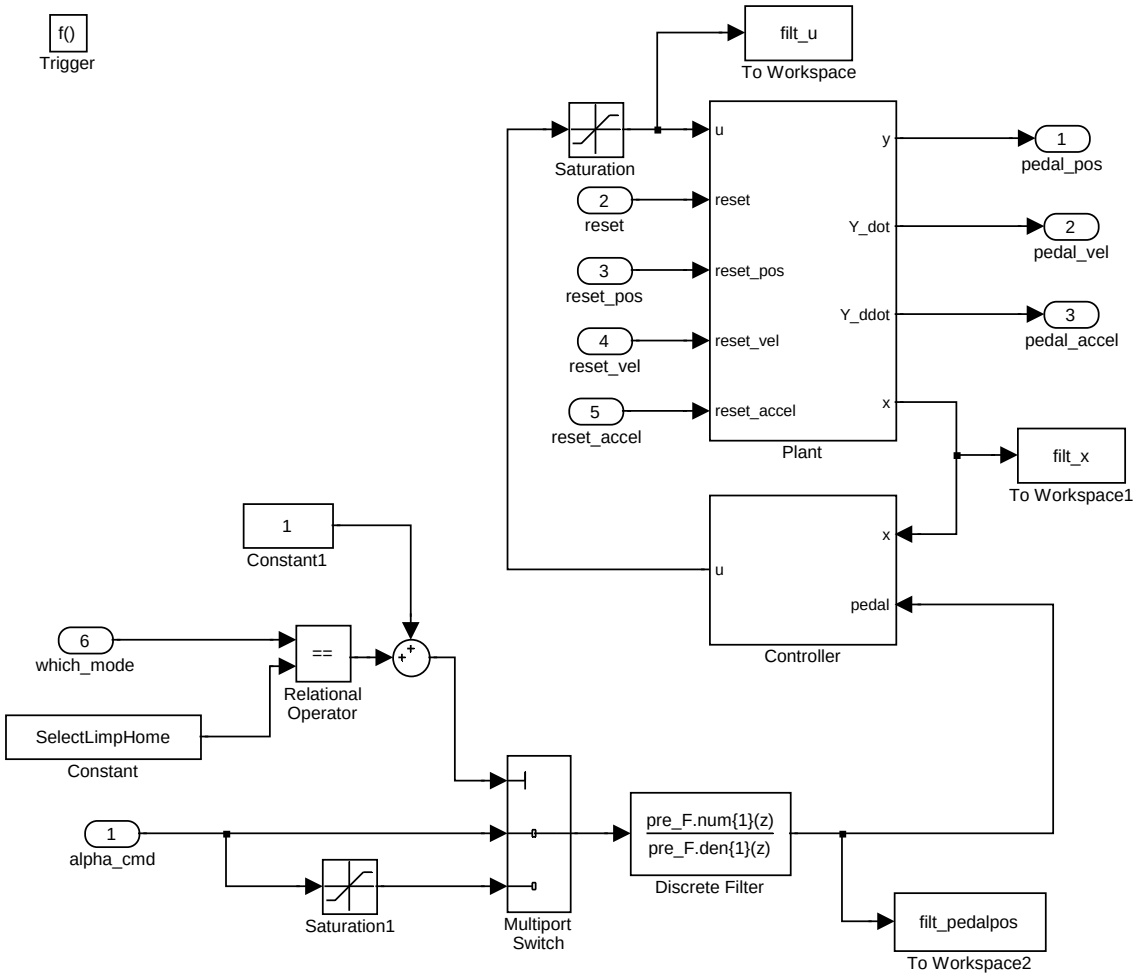


Figure 7.20: This is a resettable filter for the pedal position.

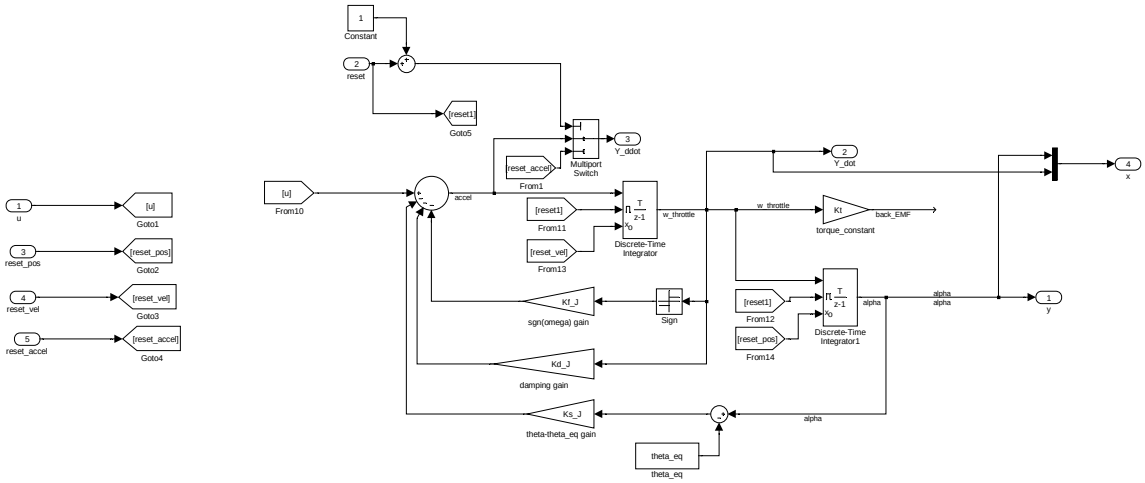


Figure 7.21: This model of the plant is embedded in the filter for the pedal position.

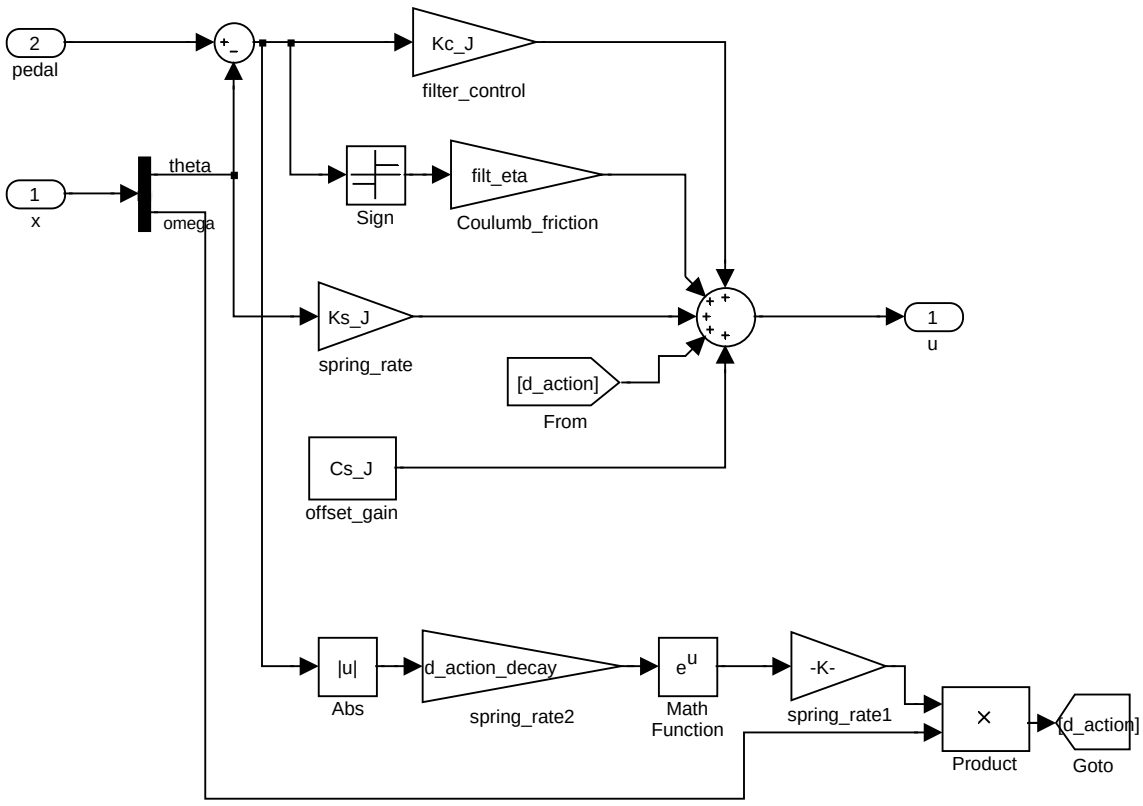


Figure 7.22: This simplified version of the real controller is inside of the pedal position filter.

controller allows the adaption portion of the controller to be disabled. In order to meet the persistency of excitation requirement, the input signals have a sine wave impressed upon them whenever the manual switch has selected the adaption law and the velocity of the throttle is near zero. The flip in sign of the reference velocity seems to be a key element of a sufficiently rich reference signal so that the friction parameter converges. The adaption law is disabled by setting the derivative of $\frac{\hat{K_f}}{J}$ to zero. This avoids the adverse parameter drift that can occur when the reference signal is not sufficiently exciting the system.

7.6 Driver Model

The driver model shown in figure 7.25 represents the software to hardware interface for actuating the system. The controller provides a desired current and a combination of hardware and low-level software implement either a current regulating PWM (figure 7.26) or a standard PWM with a static duty-cycle (figure 7.27). The shown model is configured to model the simple PWM.

7.7 Actuator Model

The actuator model is a continuous time model of the electrical dynamics of the motor. The dynamics and parasitic losses of the H-bride and double fly-back diode circuit are not included. The system is essentially a hybrid automaton with two states. In the off state the power supply voltage is zero and in the on state the power supply voltage is 12 volts minus the loss through the internal resistance.

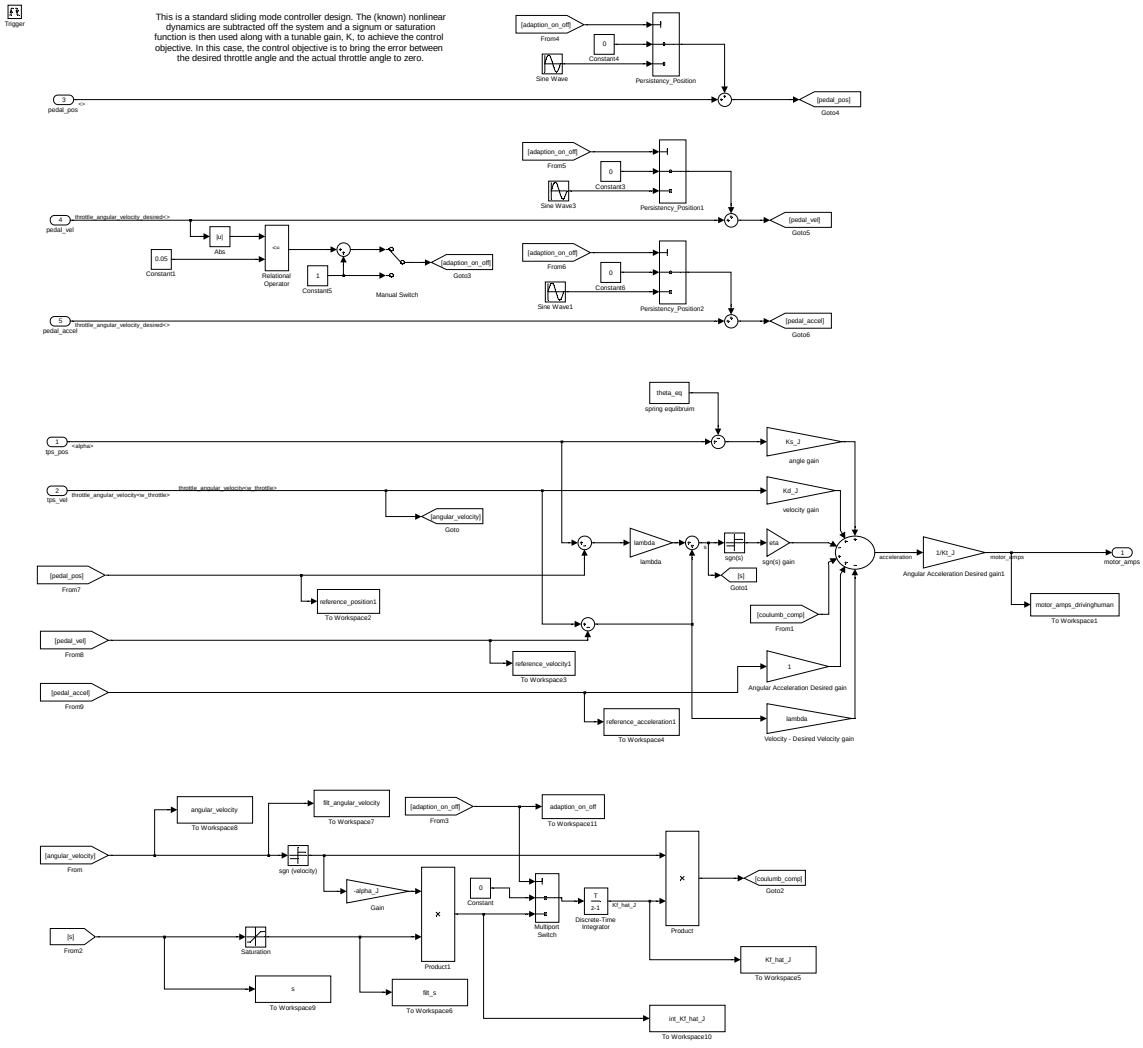


Figure 7.23: This is a model of the sliding mode controller.

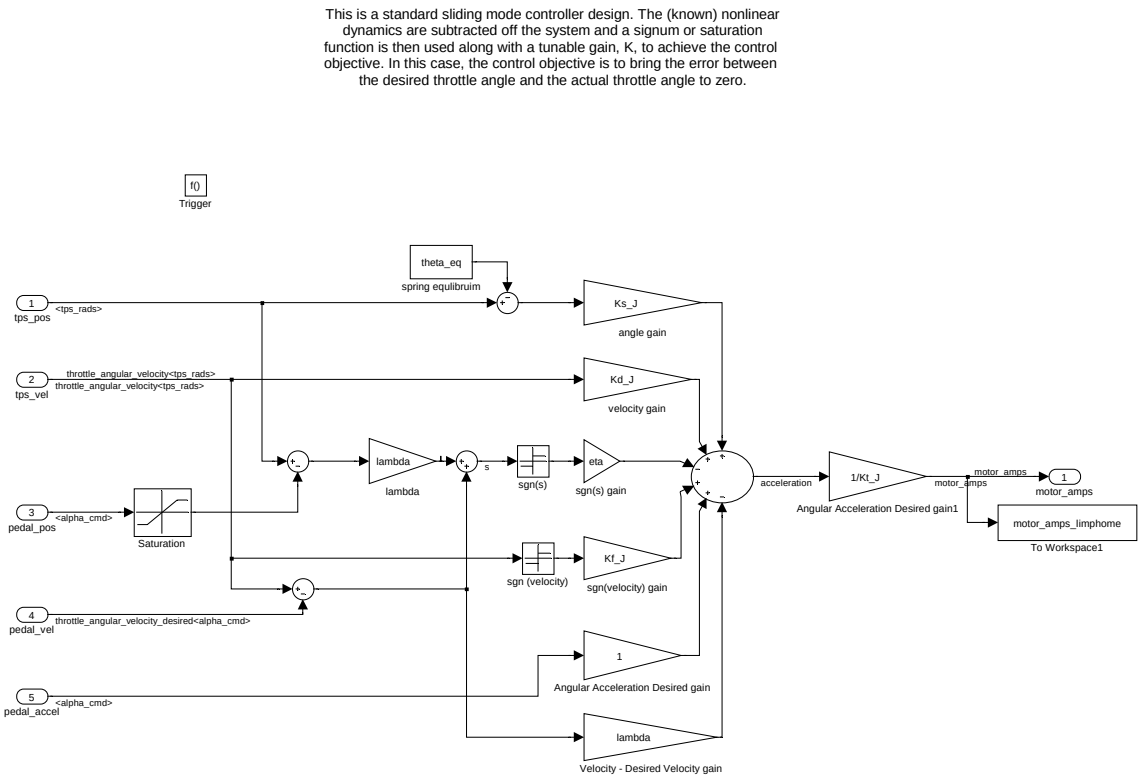


Figure 7.24: This is the same control as the sliding mode controller, but the commanded position is limited for the limp-home mode.

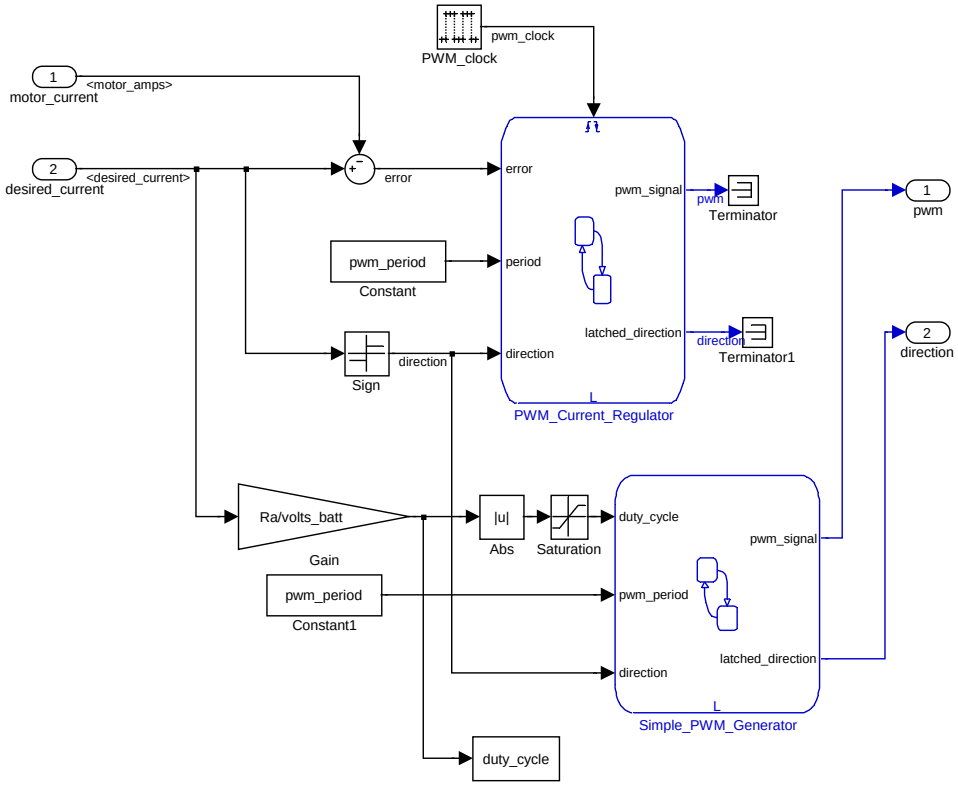


Figure 7.25: Model of Drivers & PWM

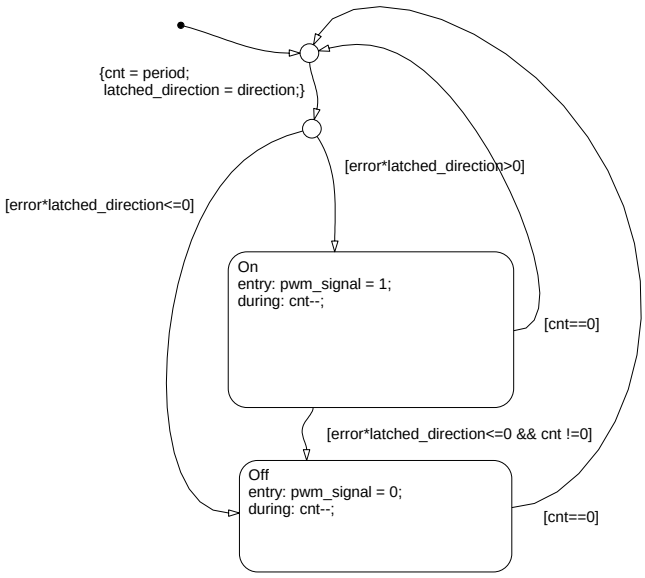


Figure 7.26: Model of the current regulating PWM

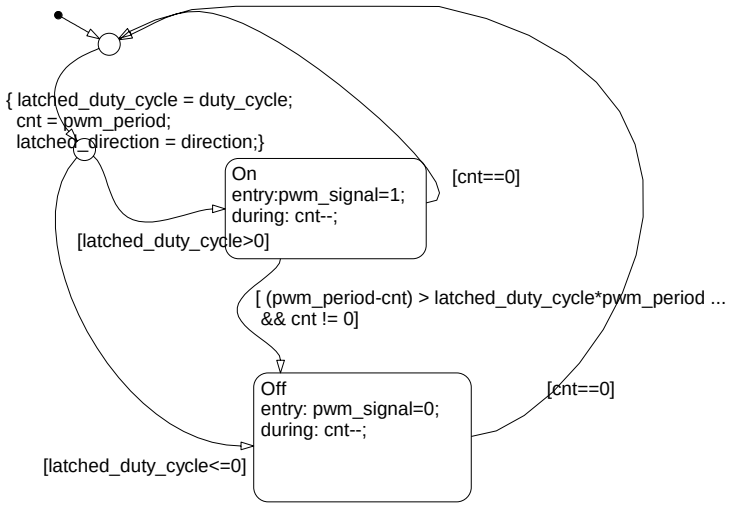


Figure 7.27: Model of a simple PWM

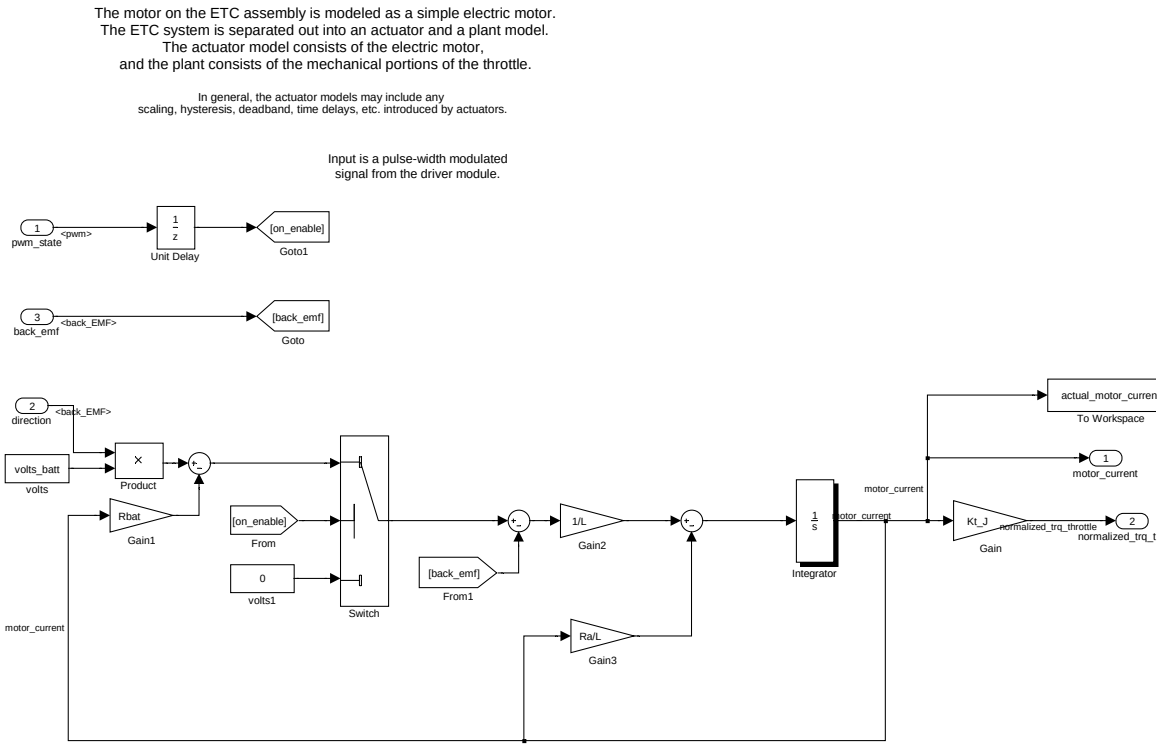


Figure 7.28: Model of the power electronics and the motor electrical dynamics

Chapter 8

Simulations & Experimental Results

8.0.1 Simulation Results

The controller defined by $u(t)$ in equation 4.10 is first verified on a simplified model of the plant. The PWM is replaced with a mean value approximation. The noise and delay due to data acquisition are also removed from the model. What is left is the adaptive sliding mode controller and the electrical and mechanical dynamics of the electronic throttle body. In order to compare simulation results, $\lambda = 70$ and $\eta = 100$ were used for all the simulations.

As a first check of the sliding mode controller, a simulation was run with the adaption law disabled. The simulation trace of the throttle angle and desired throttle angle are shown in figure 8.1. Note that the response does not appear to meet the step response requirement. This poor performance is a result of removing the resettable pedal filter. Whenever the controller is switched into a position tracking mode and the pedal filter states are reset to the current plant states. As the filter states approach the desired values, they provide both position and velocity reference values that can be tracked by the controller. From the simulation trace, it can be seen that because the desired signal was a sine wave with a relatively small velocity, the error in the velocity signal acted as a drag. This problem is addressed in the complete controller with pedal filter.

In the next simulation shown in figure 8.2, the adaption law is enabled. Whenever the reference velocity is within a small ϵ of 0, the adaption law is activated and a small, high-frequency sine wave is impressed upon the reference signal. This signal provides the needed persistent excitation for the friction parameter to converge. Figure 8.3 shows the simulation trace of the friction parameter estimate, $\frac{\hat{K_f}}{J}$. Note that the estimated value, although it appears to converge, it converges with steady-state error.

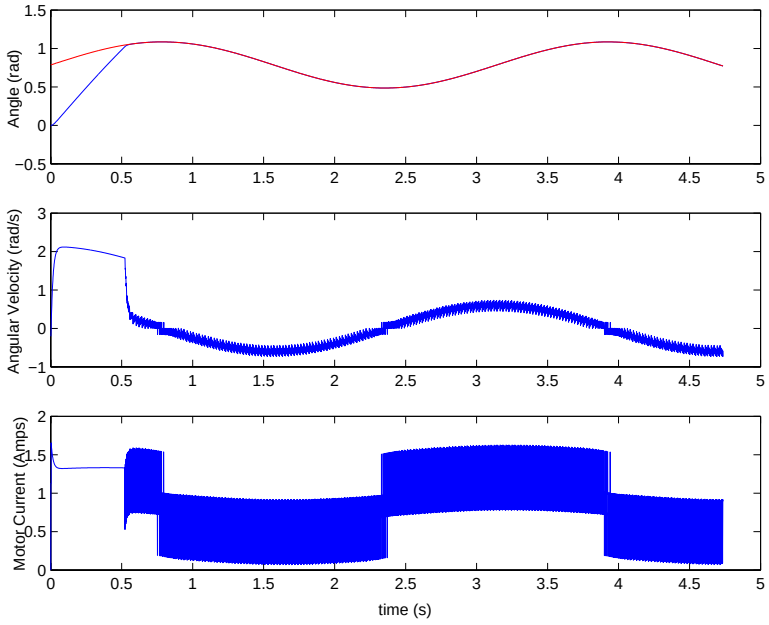


Figure 8.1: Simulation of closed-loop system with the adaption law disabled; (Desired throttle angle is sine wave, which starts above the actual angle)

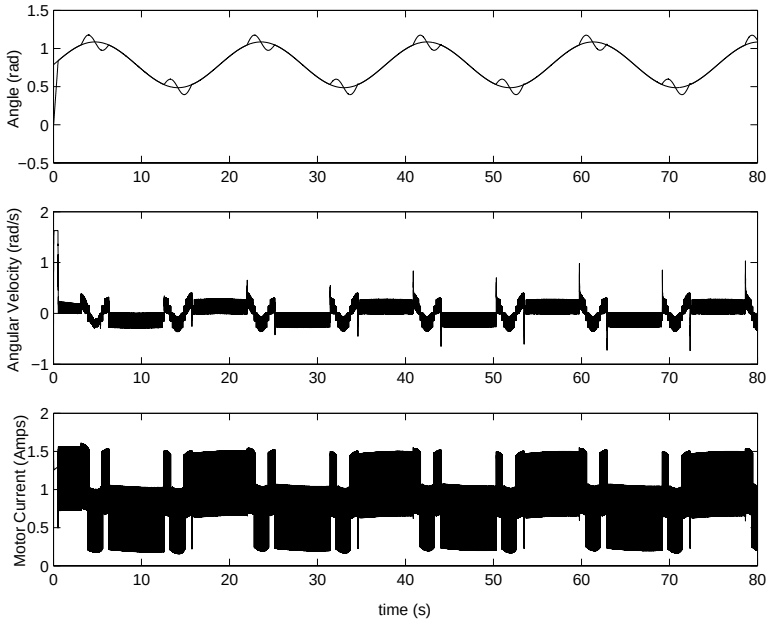


Figure 8.2: Simulation of the closed-loop system with the adaption law enabled; (The desired throttle angle contains small amplitude impressed sine waves at the peaks and troughs of the slow reference sine wave)

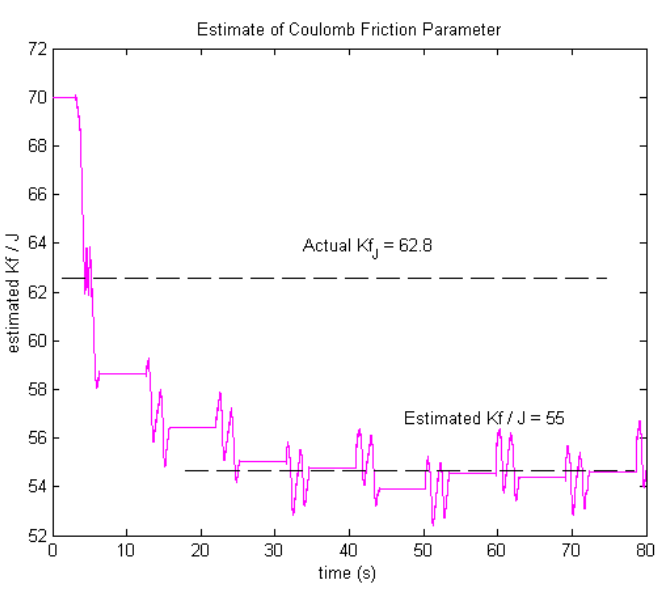


Figure 8.3: Simulation of the adaption of $\frac{\hat{K_f}}{J}$, where the actual plant parameter is given by $\frac{K_f}{J} = 62.8$.

In order to investigate the nature of the steady-state error in the estimated parameter, several other values of the actual plant parameter were tried. In figure 8.4 and 8.5 a smaller value of $\frac{K_f}{J}$ was selected. The simulation trace of the throttle position was included for this case because in this case the reduction in the estimated parameter has a visible effect on the control output; the up and down shifts of the control are attenuated as the estimated parameter is reduced. Figure 8.6 and figure 8.7 show the adaption only and the Coulomb friction plant parameter is tried at two larger values. For the values of the plant parameter between 60 and 120, the steady-state error appears to be linear. This generalization does not work for the smaller plant parameter.

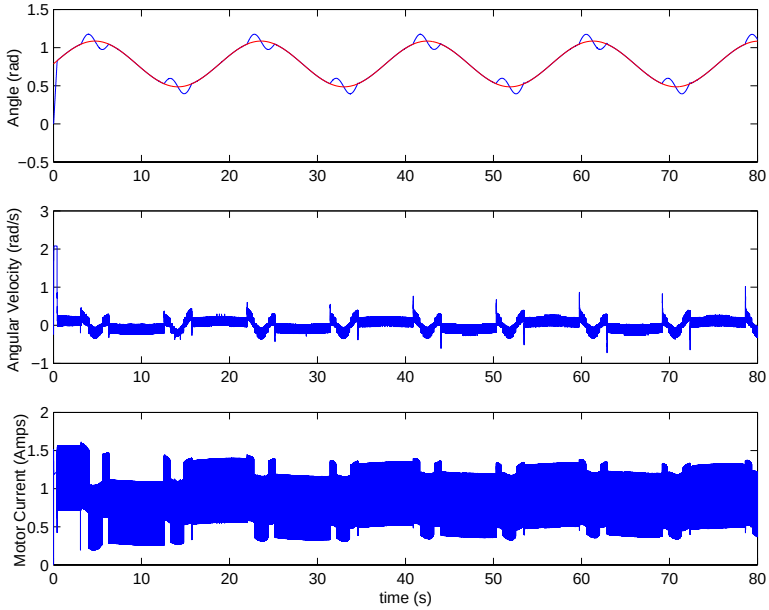


Figure 8.4: Simulation of the closed-loop system with a smaller value of $\frac{K_f}{J}$

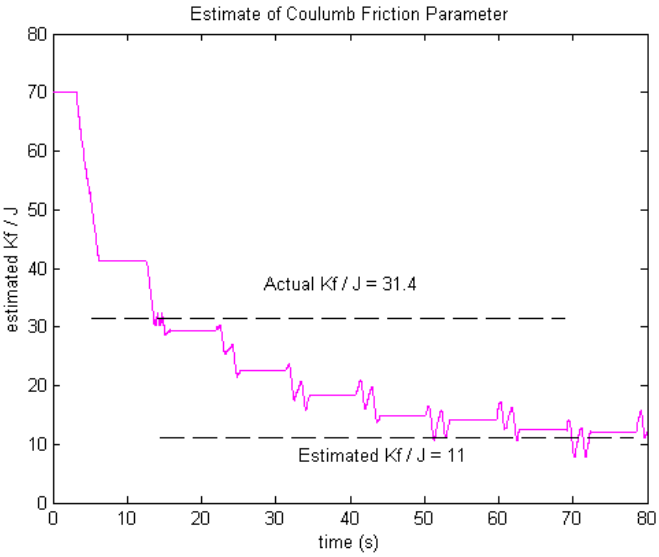


Figure 8.5: Simulation of the adaption of $\frac{\hat{K_f}}{J}$, where the actual plant parameter is given by $\frac{K_f}{J} = 31.4$.

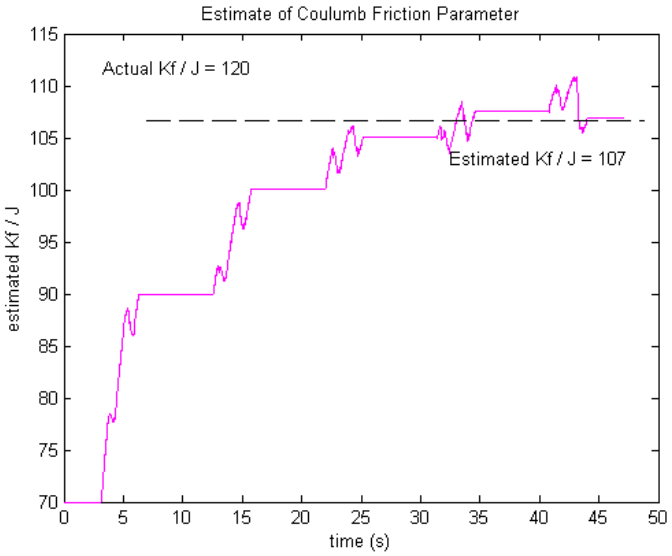


Figure 8.6: Simulation of the adaption of $\frac{\hat{K_f}}{J}$, where the actual plant parameter is given by $\frac{K_f}{J} = 120$.

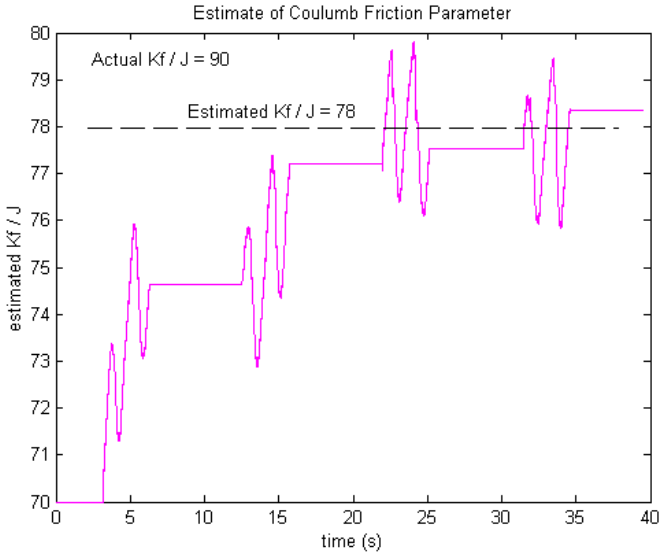


Figure 8.7: Simulation of the adaption of $\frac{\hat{K_f}}{J}$, where the actual plant parameter is given by $\frac{K_f}{J} = 90$.

8.0.2 Conclusions

The problems with the adaptive portion of the controller arise from the discrete implementation of the continuous control law. The Lyapunov stability theory guarantees that if derivative of the Lyapunov function, $V(t)$, is negative definite (and V and the system satisfy certain properties), then the system will be globally, asymptotically stable and the estimated parameter will converge to the correct value of the plant parameter with zero error. The problem with the discrete implementation can be seen if $V(t)$ and $\dot{V}(t)$ are examined. There are intervals where $V(t)$ is increasing yet $\dot{V}(t)$ is still negative.

There are two ways to design discrete controllers for continuous plants. The controller can be designed as a continuous time equation with the continuous time plant and the controller is discretized for the implementation. This is the approach taken here, and this design method was sensitive to the discrete approximation. (The steady-state error disappears as the sampling time is reduced.) The other design method is to discretize the plant and then design a discrete controller for the discrete plant. This design can have problems with inter-sample behavior of the plant, but this method might solve the difficulty with the adaptive portion of the controller.

Due to the problems with the adaptive portion of the controller in simulation, which did not include noise, delay and actuator dynamics, the controller that was implemented and tested on the physical hardware only implemented the sliding mode controller.

8.0.3 Experimental Results

Implementation

The controller consists of three hand-written C functions. Each function corresponds to one of the three tasks: the manager, monitor or servo-control task. A hardware abstraction layer consists of a set of ETC specific driver functions, which call low-level functions that access the hardware or provide simulated values for hardware that is not accessible (e.g. vehicle speed). A Giotto program specifies the rates of the tasks, the data initialization, the data dependencies of tasks inputs on each others outputs and sensor inputs, and the connection of task outputs to actuators. At run-time, the Giotto framework calls the functions of the ETC hardware abstraction layer and passes data from sensors to task inputs, from task outputs to task inputs, and from task outputs to actuators.

The controller is compiled with the Diab compiler and targeted for the MPC555 with WindRiver's OSEKWorks operating system. Using WindRiver's Tornado IDE, the controller is compiled, linked and loaded into the target's RAM. When the system is started, the entry point is the Giotto framework and that framework is responsible for initializing the system, calling the tasks at the correct rate and moving data. Controller data was buffered up while the controller was running and then uploaded for analysis after the controller was disabled. The only link through which this data can be retrieved is through a serial port connection.

Verification

Figure 8.8 shows the states of the throttle under closed-loop control with the sliding mode controller.

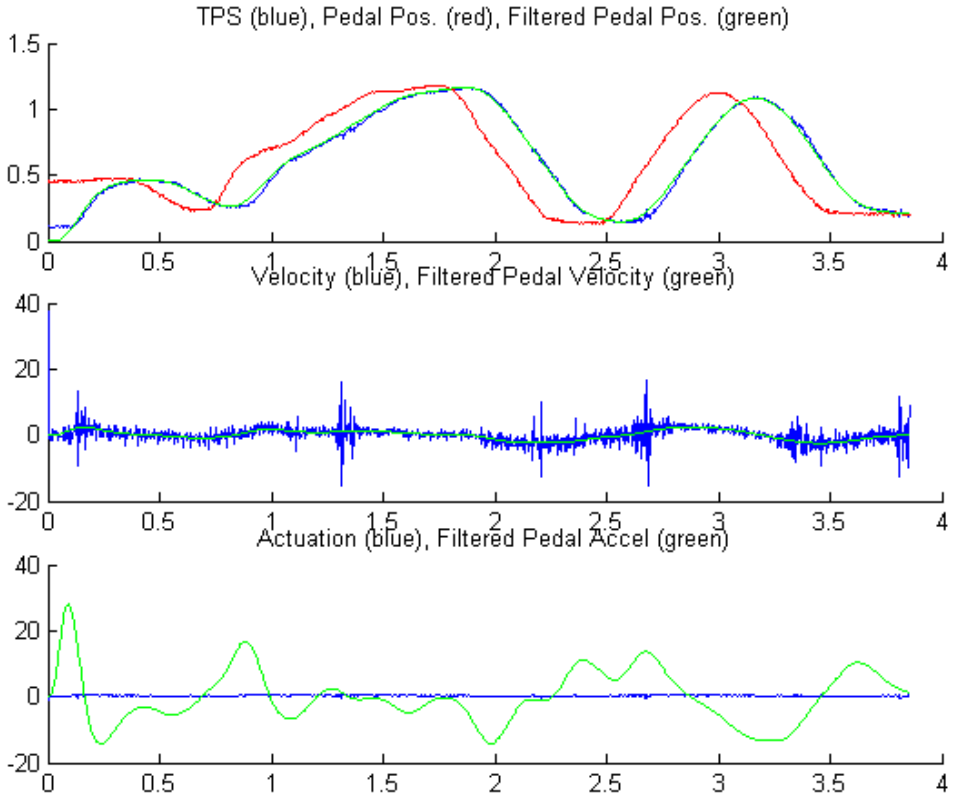


Figure 8.8: Experimental results of the sliding-mode controller

Figure 8.9 shows one cycle of the Giotto schedule. The first three tasks listed on the vertical axis make up the embedded machine (a type of virtual machine), which interprets the Giotto program. Over the 30 ms cycle, the servo-control runs 10 times, the manager runs 3 times and the monitor runs only once. It is important to note that

although the monitor was finished within 3 ms of the beginning of the cycle, its output is not made available to the manager or servo-control tasks until the end of the allotted 30 ms over which the monitor must complete its calculations. This provides deterministic performance because the availability of the output is determined by the passage of time instead of unknown factors within the processor, operating system, etc. If this program is run on a slower processor, the embedded machine will make use of preemption to ensure that the Giotto program is correctly executed if possible. This would mean that if the monitor started before 3 ms, but did not finish before 3 ms (as it does in the figure), then the embedded machine will interrupt the monitor and run the servo-control task again. When that task completes, it will return to the monitor task to complete its calculations.

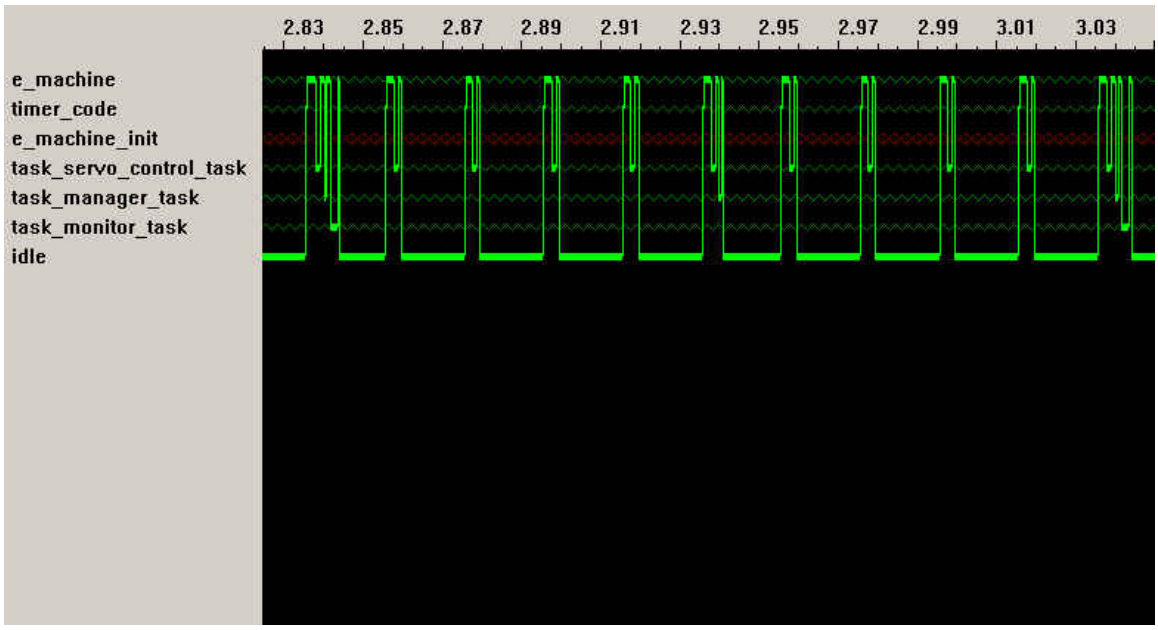


Figure 8.9: Timing data taken with WindRiver's WindView tool, which shows the execution of tasks in the Giotto program.

Bibliography

[1] Ken Butts, Dave Bostic, Alongkrit Chutinan, Jeffrey Cook, Bill Milam, and Yanxin Wang. Usage scenarios for an Automated Model Compiler. In Thomas A. Henziger and Christoph M. Kirsch, editors, *Embedded software : proceedings ; first international workshop*, pages 66–79. Spring-Verlag, October 2001.

[2] J. Christian Gerdes and J. Karl Hedrick. Hysteresis control of nonlinear single-acting actuators as applied to brake/throttle switching. In *Proceedings of the American Control Conference*, pages 1692–1696, 1999.

[3] Peter L. Goddard. Software FMEA techniques. In *2000 Proceedings, Annual Reliability and Maintainability Symposium*, pages 118–123, 2000.

[4] Derek J. Hatley and Imtiaz A. Pirbhai. *Strategies for real-time system specification.* Dorset House Pub., New York, NY, 1998.

[5] Thomas A. Henziger, Benjamin Horowitz, and Christoph Meyer Kirsch. Giotto: A Time-Triggered Language for Embedded Programming. In Thomas A. Henziger and Christoph M. Kirsch, editors, *Embedded software : proceedings ; first international workshop*, pages 166–184. Spring-Verlag, October 2001.

[6] Ian Kendall. The safety assurance of the AJV8 electronic throttle. In *IEE Colloquium on The Electrical System of the Jaguar XK8*, pages 2/1–8, 1996.

[7] K. Kimseng, M. Hoit, N. Tiwari, and M. Pecht. Physics-of-failure assessment of a cruise control module. *Microelectronics Reliability*, 39:1423–1444, 1999.

[8] Asaka Kitahara, Akiko Sato, Masatoshi Hoshino, Nobuo Kurihara, and Seichi Shin. LQG based electronic throttle control with a two degree of freedom structure. In *Proceedings of the 35th Conference on Decision & Control*, pages 1785–1788, 1996.

[9] Herman Kopetz. *Real-time systems : design principles for distributed embedded applications*. Kluwer Academic Publishers, Norwell, Massachusetts, 1997.

[10] Daniel McKay, Gary Nichols, and Bart Schreurs. Delphi Electronic Throttle Control Systems for Model Year 2000; Driver Features, System Security, and OEM Benefits. ETC for the Mass Market. Technical report, Delphi Automotive Systems, 2000.

[11] Carlo Rossi, Andrea Tilli, and Alberto Tonielli. Robust control of a throttle body for drive by wise operation of automotive engines. In *IEEE Transactions on Control Systems Technology*, volume 8, pages 993–1002, 2000.

[12] Paul Smith, Shailesh Patel, Weiqian Sun, Rajeev Ramanan, Hank Donald, Steve Toeppe, Scott Ranville, Dave Bostic, and Ken Butts. CACSD in Production Development: An Engine Control Case Study. Technical report, Ford Motor Company, 2000.

[13] Jae-Bok Song and Kyung-Seok Byun. Throttle actuator control system of vehicle traction control. *Mechatronics*, 9:477–495, 1999.

[14] Jeffrey T. Spooner and Kevin M. Passino. Fault-tolerant control for automated highway systems. In *IEEE Transactions on Vehicular Technology*, volume 46, pages 770–785, 1997.

[15] A. Stotsky, B. Egardt, and S. Eriksson. Variable structure control of engine idle speed with estimation of unmeasurable disturbances. In *Proceedings of the 38th Conference on Decision & Control*, pages 322–327, 1999.

[16] M. Yokoyama, K. Shimizu, and N. Okamoto. Application of sliding-mode servo controllers to electronic throttle control. In *Proceedings of the 37th Conference on Decision & Control*, pages 1541–1545, 1998.

Appendix A

Hardware Reference

A.1 Wiring

Wire Color	Pin #	Name	Description
Black	7 & Shield	AGRND	Analog Ground
Green	2	TPS1	Throttle Position Sensor 1
Green/Black	3	TPS2	Throttle Position Sensor 2
Red	1	IN1	H-Bridge Input 1
Red/Black	4	IN2	H-Bridge Input 2
White	6	DI1	H-Bridge Disable 1
White/Black	5	DI2	H-Bridge Disable 2
Orange	9	IMTR	Motor Current

Table A.1: This table provides a mapping between the various signals from the throttle driver electronics and pin numbers and wire colors.

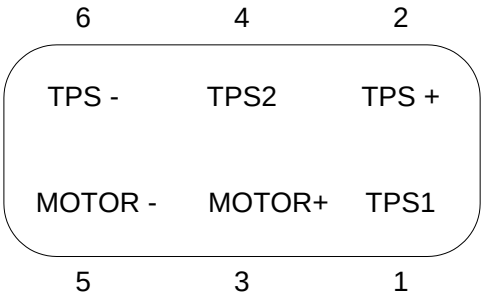


Figure A.1: Pin-out of the BMW throttle electrical connector

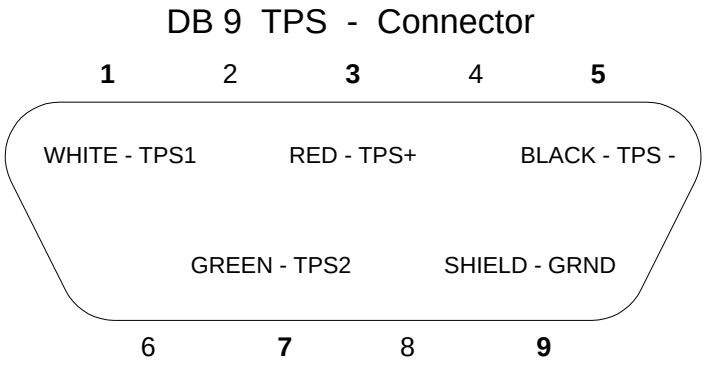


Figure A.2: Pin-outs of the DB9 connector for the TPS signals

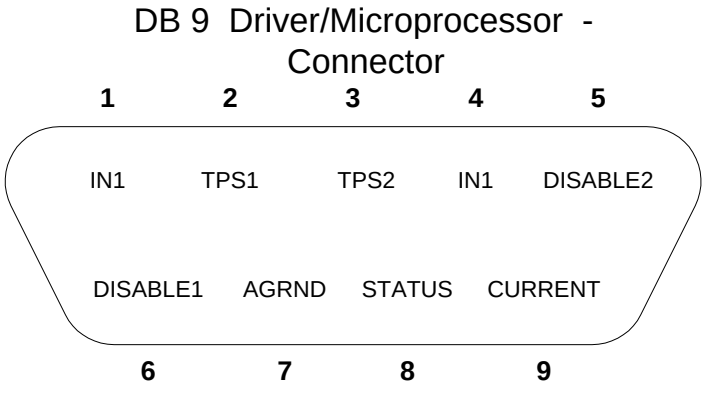


Figure A.3: Pin-outs of the DB9 connector between the driver electronics and the microprocessor