California AHMCT Program
University of California at Davis
California Department of Transportation

Control of a Continuously Variable Transmission for Variable Geometry Mobile Equipment

Christopher R. Carlson, Richard W. Carlson, Andrew Frank

AHMCT Research Report
UCD-ARR-99-06-30-01

Report of Contract
65X875 T.O.96-7
65X875 T.O.99-7

June 30, 1999

This work was supported by the California Department of Transportation (Caltrans), and Advanced Highways Maintenance Construction Technology Program (AHMCT) at UC-Davis.

Table of contents:

Phase 1
0.0 Abstract

The purpose of this two year endeavor was to design, construct and demonstrate the servo-hydraulic control concepts proposed by Gear Chain Industrial and Dr. Andrew Frank for the control of a Continuously Variable Transmission (CVT).

In Phase I of the project, a 1.0L class Nissan CVT was fitted with a servo-hydraulic system under microprocessor control. The assembly was then experimentally characterized and closed loop compensators were developed to improve system response. Non-linear characteristics of the system made compensator design difficult, and techniques for the linearization of the system by feedback control were explored. A control system is proposed and demonstrated.

Phase II of the project involved the implementation of a 2.0L class CVT into a vehicle for evaluation and as a proof of concept. The vehicle chosen was the 1999 UC Davis Future Car and the CVT was an integral part of the parallel CVT Hybrid powertrain. The CVT proved to help increase efficiency while maintaining excellent driveability.

1.0 Introduction

In an effort to achieve the highest possible fuel economy in a hybrid-electric vehicle powertrain, UC Davis began researching the application of a Continuously Variable Transmission (CVT). This transmission could be used to precisely control the speed and torque on the powertrain through the continuous selection of gear ratio. This technology would then allow the vehicle powertrain to operate at its most efficient point for each power demand. By increasing the vehicle power efficiency in this way, it is possible to approach the highest theoretical fuel economy in a real mechanical system. The demonstration in a vehicle platform can easily be extended to other mechanical equipment requiring power transmission.

Up to this point in time, UC Davis had assembled a prototype servo-hydraulic CVT control system on a dynamometer. This paper explores the replacement of these motors with new, light weight and highly efficient DC brushless motors, the development of the control system compensators necessary precisely control a CVT, and explores the control concepts necessary to make CVTs a reality for electric power assist dual power plant vehicles and industrial equipment.

2.0 Setup

Figure 3.1 depicts the hardware setup on the dynamometer pad. The CVT is a 1.0L class Nissan transmission and is mounted on a stand with power introduced via the large electric motor in the back of the room. Power flows from the motor, through the CVT to a belt and pulley system, and then to the Eddy current dynamometer.

Figure 3.1 Dynamometer setup for CVT testing.

The ratio and clamping pressure of the CVT are controlled through the servo-pumps which are interfaced to a microprocessor. The processor is programmed via the Laptop in the bottom right of the picture.

Gear Chain Industrial and Prof. A. Frank inspired this research with the paper A New Control For Continuously Variable Transmissions. This paper specifies the replacement of the traditional mechanical control valve assembly and line pressure pump with the servo-hydraulic system shown in Figure 3.2.

Servo motors 1 & 2 drive small gear pumps which generate the pressure and flow of the hydraulic system. Pump 1 connects the two cylinders together. Actuation of this pump moves fluid back and forth between the two cylinders. This fluid movement changes the relative

displacement of the pulley faces, thereby changing the ratio of the transmission. The speed of servo 1 dictates the flow rate of the fluid and thus, the shift rate of the system.

Figure 3.2. Servo-hydraulic setup for Control of CVT

Pump 2 controls the line pressure of the system. The pressure generated by this pump, via cylinders 1 and 2, generates the force necessary to clamp the sides of the CVT belt and transmit torque.

The specification of the pumps and motors used appear in Appendix C.

3.0 Procedure

The CVT control system consists of two groups, the clamping pressure group and the shift group. Each group was characterized in the same way but at different frequency ranges. Each servo amplifier was forced by a function generator introducing a sine wave to the system. The output of the system and the function generator traces were then recorded on a multi-channel data acquisition system. A typical response curve would look something like Figure 4.1.

Figure 4.1. Sample data for frequency response of CVT control systems.

The gain and phase lag for each frequency was then recorded to generate the Bode plots analyzed in sections 4.2 and 4.3.

4.0 Results

4.1 Mean Power Draw

The benefits of an electronically controlled CVT are two fold. The first is that the CVT may then be used in a vehicle which does not have an engine idling at all times. Conventionally, a CVT generates primary line pressure via a pump on the main engine shaft. The pump size is such that the engine idling provides the maximum required power and flow rate to the hydraulic system. This means that the pump is oversized for normal operation therefore reducing efficiency. Electric and hybrid-electric vehicles do not need such a continuous supply of mechanical power because, their battery packs a capable of running the servo-motors while the vehicle is stopped. The second, and far more significant benefit, is the drastic reduction in the mean power draw of the hydraulic system. A Nissan CVT engineer quoted the mean power draw of Nissan's 2.0 liter class transmission as 3 kW. The same system controlled with servo-hydraulics draws 0.35 kW peak. That is an improvement of 757% of the peak power draw over the previous mean power draw. Mean power draw should be approximately half the peak in the real system.
That is an improvement of 757% of the peak power draw over the previous mean power draw.

	P1=35 bar	P2=17 bar
Flow1=1.0 l/min	$f1 \cdot p1 \cdot \frac{1}{Peff \cdot EMeff} = 0.098kW$	$f1 \cdot p2 \cdot \frac{1}{Peff \cdot EMeff} = 0.048kW$
Flow2=2.5 l/min	$f2 \cdot p1 \cdot \frac{1}{Peff \cdot EMeff} = 0.245kW$	$f2 \cdot p2 \cdot \frac{1}{Peff \cdot EMeff} = 0.119kW$

$EM_{eff} = 0.85$ $P_{eff} = 0.70$

The following table shows the current used by the pressure servomotor at 165V and a fixed ratio for the 1.0L modified CVT test rig.

Voltage = 165	P=35 bar	P=17 bar
I (Amps)	0.55	0.25
Power (kW)	0.091	0.041

4.2 Clamping Pressure Control Algorithm Development:

Once the mechanical system was built, the first control task was to develop the clamping pressure of the primary pump assembly. The control variable was chosen to by following the power transfer variables in the system. The pressure exhausted from the constant displacement pump is proportional to the torque on the pump shaft. The shaft torque is driven by the torque on the electric motor shaft, which is proportional to the current commanded through its windings. Ultimately, line pressure is proportional to the current commanded. For this reason, the motor controller was selected as a "dumb torque drive" type.

Once a control variable was specified, the system was run in "open loop" mode. Meaning that although the current controlled inverter had its own internal feedback controls, there was no feedback between the pressure of the system and the pressure command. This produced a stable that tracked a pressure proportional a command voltage. The system, however, oscillated at low frequencies and rejected disturbances poorly.

To improve system response characteristics, an analog, proportional, feedback controller was designed and implemented using readily available LM741 op-amps. The small DC offset induced by these inexpensive op-amps was not an issue as a small adjustment to the command variable countered this effect.

Figure 4.2.1. Analog Controller Block Diagram

Figure 4.2.2. Analog Controller Schematic

This controller scheme reduced the amplitude of low frequency oscillations of the system and improved disturbance rejection. However, the system suffered from steady state error and did not reject disturbances well enough.

The oscillations of the system were mostly due to a non-linear characteristic of the electric motor and pump near the point of zero velocity. Motor cogging as well and the discontinuity between kinetic and static friction caused the motor shaft to bind whenever the pump shaft velocity went through zero. Figures 4.2.3 and 4.2.4 illustrate this issue.

Figure 4.2.3. Nonlinear friction Vs. velocity graph

The system would behave as follows: The pressure of the system would start out initially at zero and the command would apply a step input. The error would command the electric motor torque which would accelerate the pump shaft and bring up the system pressure. The system would then reach a point where the feedback pressure and the command signal were the same, yielding an error signal of zero and therefore pump command signal of zero torque. The pump would then stop and its static frictional effects take hold. The system pressure would then slowly fall, effectively ramping the torque signal to the electric motor without actually turning the shaft due to the friction on the shaft. Finally the error signal becomes so large that the shaft brakes free, accelerating to the target pressure due to the now lower kinetic friction. The error then quickly reaches zero and the cycle begins anew.

Figure 4.2.4. Oscillating characteristic of closed loop system

Under the advice of a Nissan controls engineer, Masahiro Yamamoto, a feed-forward controller was implemented to help alleviate this non-linear effect and reduce the steady state error. The following controls diagram in figure 4.2.5. was implemented with the same inexpensive op-amps used previously.

Figure 4.2.5. Feed forward block diagram

This controller introduced a constant command to the servo controller even when the error of the system was zero, thereby eliminating the zero torque command situation. In addition, this system benefited tremendously from the introduction of a deliberate power draw to the system. By placing a bleed from the high pressure side of the system back to the low pressure side a constant flow was introduced to the otherwise nearly static pressure system. This modification kept the pressure pump rotating constantly, effectively avoiding the non-linear zero velocity region. At the typical pressures the system operates at with this low flow rate, it is estimated that the power draw of this bleed is approximately 0.015 kW. In a production system, this bleed could be introduced as a slight allowance in the gear faces.

Due to their increasingly low cost and flexibility the control system was then moved to a digital controller. Programmed in the high level language, C, the digital controller allowed many

```
/*Interrupt at 100 Hz */
int interrupt(void){
   command = sample_command();
   feedback = sample_feedback();
   error = command – feedback;
   output = gain * error;
   clip(output);
   output_to_DAC(output);
   return(1);
}
```

Manufacturer:	Z-World
Processor:	Z180
Clock Speed:	~18 MHz
I/O:	4 serial ports 16 lines high current Digital I/O 16 lines normal digital I/O 10 12bit, ±1 lsb Analog to Digital lines 4 Digital to Analog lines
Memory:	32K SRAM 128K Flash EEPROM
Development Environment	Dynamic C, Assembler

Figure 4.2.6. Sample digital control routine.
Figure 4.2.7 Digital controller specification.

different control algorithms to be tested quickly and easily without changing the hardware. It was, on average, taking one to two days to completely re-wire and test each analog controller design before actually using it to control the system and evaluate system performance. With the digital controller, however, it often took less than one hour to dramatically change the digital control algorithm. In addition, non-linear gain concepts, input and output clipping, and hardware diagnostic services all became the simple task of writing a few lines of code. A sample digital routine and digital controller specification appear in figures 4.2.6 and 4.2.7.

All digital control algorithms were developed by moving from the s domain to the z domain via the bilinear transformation or, where possible, Euler's integration and differentiation. For example a PID controller would consist of P,I and D controllers calculated separately and superimposed. The following mathematics would replace the proportional gain K, in figure 4.2.1.

$$\text{Controller}_{PID} = Kp + \frac{Ki}{s} + Kd \bullet s$$

$$Kp \Rightarrow P = Kp \bullet \text{Error}$$

$$\frac{Ki}{s} \Rightarrow I = Ki \bullet \Delta T \bullet \text{Error} + I_{n-1}$$

$$Kd \bullet s \Rightarrow D = Kd \bullet \left(\frac{\text{Error} - \text{Error}_{n-1}}{\Delta T} \right)$$

$$\text{Controller Output} = P + I + D$$

The first key issue upon moving to a digital controller was the selection of the sampling frequency. If a frequency too high is chosen, the resolution of the digital to analog converters becomes incapable of measuring the extremely small changes from one sampling period to the next. On the other hand, if the sampling frequency is too low, the time between samples introduces a lag in the system which leads to oscillations. As a rule, the time between sampling frequency was chosen to be ten to twenty times the –3dB cutoff frequency of the system on a bode response curve.

A variable frequency sine wave was used to excite the open loop system to determine its

Figure 4.2.8. Model and data for clamping pressure plant

frequency response. Figure 4.2.8 shows the system response data and the model that represents them.

From this data in figure 4.2.8, a sampling frequency of 100 Hz was selected. This selection proved to be a rather unfortunate error which led to many hours of wrestling with a digital system which would oscillate at about 2 Hz despite it closely approximating a proven analog controller. Although the open loop cutoff frequency of the plant was ~3Hz, the close

Figure 4.2.9. Closed loop response of clamping pressure system model.

loop frequency response was more like 15Hz, which can be understood by incorporating a proportional controller into the model developed earlier. Figure 4.2.9 shows the model with a proportional controller closing the loop, which demonstrates the higher closed loop frequency response.

Several different control schemes were tested and deemed unusable at 100 Hz. These included PI, ID and PID controllers, all with varying degrees of feed-forward incorporated as well. Unfortunately, the lag induced by the low sampling frequency doomed all of these schemes before testing even began. I highly recommend further researching these control schemes in the future as the existing system still suffers from steady state error and could improve its disturbance rejection.

The digital, proportional, feed forward controller shown in figure 6 was chosen to control the CVT clamping pressure system. This choice was largely motivated by time constraints but does work well. There is still room for improvement, however. The disturbances to this system are a direct result of shifting the ratio of the system. Therefore, this known disturbance could also be fed forward to the pressure system to aid in pressure correction.

4.3 Shifting System Control Algorithm Development

After a suitable clamping pressure system was developed, the next step was to develop a system for controlling the ratio of the transmission. For reasons that will be explained in the following section, Developing a CVT Vehicle Control Strategy, it was necessary to control the rate of change of ratio, dR/dt ($\dot{R}$), in addition to the ratio, R, itself. This separate control loop was never really attempted in analog form, rather, it began as a digitally controlled system. Due to the availability, cost and excellent quality of the Z-World controllers specified above, identical microcontroller hardware shifted the of the CVT.

Vital to the beginning of any compensator design is the selection of the variable to be controlled. Unlike the pressure system, the shift system depends on the displacement of fluid rather than the effort exerted upon the fluid. In a static state, the force exerted by each pulley is constant and the ratio is dependent on how much fluid is contained in each piston cylinder. Moving fluid back and forth between the two cylinders changes the pulley ratio. The shift motor, therefore, operates in velocity mode. Through the constant displacement pump, this mode governs fluid flow rate. Hence, the rate of change of fluid volume in each piston is proportional to the velocity command signal. Figure 1 shows the mechanical layout of the shifting system.

The same setup used for exciting the pressure system excited the shift system as well. The resulting frequency response of the shift system resulted in the following open loop MATLAB model in figure 4.3.1.

Figure 4.3.1. Model and data for shifting system.

From a closed loop model we get a cutoff frequency of approximately 0.1Hz. This suggests a sampling frequency of 2 Hz would be sufficient for the shift system, however, in light of the previous issues with low sampling rate, 50 Hz was selected as the sampling frequency.

An unfortunate reality is that the feedback variable for the shift system is the ratio R. R can be obtained two ways, by sensing the position of the pulley faces with a linear potentiometer and by dividing the input shaft velocity by the output shaft velocity. By either method, the feedback variable is the integral of the command variable R•. This configuration brings two classical control system configurations to mind as shown in figure 4.3.2.

Figure 4.3.2. Two configurations for Rdot control.

From a theoretical standpoint both of these configurations are equally useful, however, each has specific implementation issues that can only be avoided by incorporating the two of them together. The algorithm which integrates the R• suffers from integrator overshoot and poor rate following.

Figure 4.3.3. Shift system oscillations under integral control.

Figure 4.3.3 shows the primary control system commanding a shift rate with the intention of moving down a constant power line to a target point. The system then integrates the commanded R• to get a target R. The system then proceeds down the power line with a steady state error. The target point is then reached, and the shift rate command is now zero, however, there is still accumulated R due to the steady state error in the transient portion of the command. So the system overshoots to a point that is too low on the power curve. The cycle now begins anew, only in the opposite direction. The system is too low on the power line, so it commands a negative R• which then overshoots to a location too high on the power curve. This overshoot characteristic is a well known effect of integral action upon a control system. The two ways it is classically dealt with are by increasing the proportional gain or adding derivative action to the system. Both of these avenues warrant further investigation in the future.

The second control configuration suffers from the resolution of the analog to digital converters (ADCs). The ADCs are operating at an 11 bit resolution over a 10V rage. This fixes the maximum resolution to 5mV per bit. With a sampling frequency of 50 Hz, that means that we can accurately detect no less that 0.250V per second of drift. The ratio sensor reads between 0V and 2.5V. The CVT can and does drift from lock to lock in about 10 seconds without this control system detecting it. At low shift rates an accumulator is necessary to make sure correct ratio is tracked as the resolution of the derivative term becomes too low .

The final control system was selected to be a combination of the two systems. At low shift rates, the first control system would accumulate the target ratio. At high shift rates, the ratio signal would be differentiated to provide the most accurate rate control. In addition, there is a provision that resets the integrator in the low shift rate system every time the system command goes through zero. This significantly reduces the effects of integrator induced overshoot. Figure 4.3.4 shows a program that combines all of these features into one program.

```
// Interrupt code body.  This code runs at 50 Hz.
int
interrupt(void){
// Sample the command and feedback variables
  command     = sample_command();          // range: ± 2.5
  feedback[0] = sample_R();                // range: 0 - 2.5
  R    = feedback[0];
  Rdot = (feedback[0] - feedback[1]) / sample_time;     // feedback[1] = feedback_n-1

// Allow for a dead band in the command for zero Rdot
  if( command < low_band && command > -1*low_band)
    command = 0;
// Low shift rate command
  if(command > 0 && command < command_low){       // 0 < command < command_low
    if(state != 0){                               // If we just changed states, reset
      state    = 0;                               // accumulator.
      R_target = R;
    }
    R_target = R_target + command *sample_time;   // Integrate Rdot command
    error  = R_target - R;
    output = low_gain * error;
  }
  else
  if(command < 0 && command > -1*command_low){    // -command_low < command < 0
    if(state != 1){                               // Reset accumulator
      state    = 1;
      R_target = R;
    }
    R_target = R_target + command *sample_time;   // Integrate Rdot command
    error  = R_target - R;
    output = low_gain * error;
  }
  else{                                           // Rate feed back control here
    state   = 2;                                  // Acknowledge state change
    command = command / (50.0);                   // Scale command variable so that
                                                  // Max command signal is 2.5 V per
                                                  // second.
    error  = command - R_dot;
    output = high_gain * error;
  }
// Send output to DACs with level checking
  clip(output);
  output_dac(output);
// update history variables
  feedback[1] = feedback[0];
```

Figure 4.3.4. Sample R, R˙ control algorithm.

This algorithm reads the Rdot and R feedback variables from the external analog to digital converters. It then stores the R value read and calculates Rdot using Euler's method of differentiation. All external information has been read and processed.

Next the program decides weather the command is small enough to warrant ignoring the rate variable calculation. If so, it moves into states 0 or 1. Upon first entering a state, the accumulator R_target is reset. The program then integrates the command variable and tracks a constant ratio. Upon exiting this portion of the routine, the program clips the output variable and sends it to the digital to analog converters.

If the command is large enough to use rate feedback, the command variable is scaled and the feedback rate subtracted to produce an error for proportional control. The output is then clipped and output to the DACs as with the above code portion.

The simplicity afforded by the digital controller is magnificent. The design of such an analog system would not only be arduous, but it would require extensive work to make simple modifications.

The above code fragment accurately controls the CVT shift rate at high speeds and does not drift at low speeds where rate feedback looses its resolution.

4.4 Power Control Theory and Development

The addition of a CVT to a conventional powertrain allows continuous control of power loading where only discrete control was available before. Taking advantage of this feature evolved a new power management strategy with pitfalls as well as benefits.

First, it becomes necessary to define what powertrain variable we want to control directly with each of the inputs available to the driver. A conventional car with a manual transmission provides a basis for this analysis, where the driver can control the accelerator, the brakes and the transmission gear selection. Suppose for a moment that a driver wishes to accelerate to 60 mph from a stop. He would engage the engine at a high throttle setting in first gear, commanding torque from the engine through the transmission to the wheels. Upon reaching the maximum engine speed in the gear he would then shift to second gear and continue at his high throttle setting until again the engine approached its maximum speed. And so on, until 60 mph was reached. At this point, he would cut back on the throttle until the torque output from the engine equaled the road load and aerodynamic losses. He would then continue at a steady state speed of 60 mph.

The above logic chain lead to the first power management strategy. This strategy consisted of interpreting the accelerator pedal and brake pedal as 0 to 100% positive and negative torque from the powertrain. A simulation modeling the CVT dynamics and the Federal Urban Driving Cycle (FUDS) was written in the C programming language. The target ratio of the CVT was chosen as a function of the torque commanded. The powertrain would seek the speed at which the torque setting was the most efficient. The simulation performed well and proved that the torque control concept could effectively control the car. However, the simulation failed to show that the driver became a dependent feedback variable as well as a command variable. So while the car was controllable, it was not like driving a traditional car. The missing link is that the driver is simultaneously commanding the speed of the engine through the selection of the gear ratio and the torque of the engine through throttle setting. Thus, the driver really commands power output of the powertrain and not just the torque. The simulation was then re-written with this new insight.

The new control routine took the following form in figure 4.4.1:

Figure 4.4.1. Power control algorithm

The dotted line box represents the system model, everything external to the box is the control system. The power command signal is derived from the accelerator pedal and sent through the summing junction. It is then subtracted from the ideal operating power to generate a power error signal. This error is multiplied by a gain, K1, and sent through a low pass filter with time constant T. The result of which is the Rdot command which dictates the shift rate of the transmission.

The Rdot command is then fed into the dynamic model of the CVT automobile. The equation used governs the acceleration of the drive shaft $\alpha_d = (\omega_d)'$. There are two terms to this equation. The second merely divides the net torque of the powertrain minus the road losses by the inertia of the vehicle to find the net angular acceleration. The first represents the torque produced by adding inertia to the powertrain by changing gear ratio of the transmission. This effect is normally burned up between clutch surfaces when shifting speeds in a discrete transmission. In the CVT, however, this term to decelerates or accelerates the car when shifting the transmission up or down.

The model then goes on to integrate all the necessary variables to keep track of the states in the system. These states are then fed into an ideal operating line for the powertrain. This line is a equation that represents the ideal power setting for each engine speed setting. Ideal meaning that the powertrain operates the most efficiently at x power for each speed. The control system does not use this ideal line to control the power output, rather it uses this ideal power to generate an Rdot command the seek the point that is the most efficient for each power line.

Regardless of the ideal power output by the system IOL, the powertrain always provides the desired power commanded through the top branch of the block diagram through two terms. The first term calculate the torque required by the powertrain by dividing the power command by the speed of the powertrain. The second term is compensating for the inertia discussed above term in the dynamic equation that governs the CVT.

Figure 5.4.2 shows how the control system transitions between two power lines.

Figure 4.4.2. Power control transition between two power lines.

This control system works well at speeds above about 5 miles and hour. However, at low speeds, the power commanded divided by engine speed can yield extremely high values of torque to satisfy the power setting. Although the computer simulation runs fine with these huge values of torque, the drivability of the automobile is not acceptable for human drivers.

$$torque = \frac{Power}{Speed}$$

At the low speeds where the high values of torque are an issue the transmission will always be in the lowest possible gear. This feature allows the control system to command a torque proportional to the accelerator command for low speeds. This control policy does not allow for optimization of power transfer at extremely low speeds, however, once the cutoff speed is reached for power control, the system then transitions from torque control and begins shifting the transmission to seek the highest efficiency under power control.

Phase 2

5.0 Introduction

Phase 1 proved the proposed control method can robustly control a CVT in an effective and efficient manner. Phase 2 will incorporate the knowledge that was learned in Phase 1 and apply it in a real vehicle application. The vehicle chosen for this CVT research was the 1999 UC Davis Future Car. It is a parallel hybrid electric vehicle that is base on a Mercury Sable chassis.

The main goals for the Future Car are high fuel economy and low emissions. These goals parallel the long term goals of this CVT research.

Up to this point, UC Davis had assembled and tested a prototype servo-hydraulic CVT on a dynamometer. This paper explores the integration and use of a similar CVT in a vehicle. Shown below is the 1999 UC Davis Future Car.

The 1999 UC Davis Future Car
CVT Parallel Hybrid Electric Vehicle

6.0 Setup

The CVT, which was tested on the dynamometer, was a Nissan 1.0L class CVT which can transmit a maximum of 110 Nm of torque. For this vehicle application a much larger amount of torque capability is required. A larger CVT is therefore required. The selection of a CVT was limited by available production CVT's. Transmissions produced by Honda, Volvo, and Nissan were investigated for use in this vehicle. Table 6.1 shows the maximum allowable torque of each transmission, which indicates its acceleration potential.

Table 6.1 Torque capacities of production CVT's.

	Class	Maximum Allowable Torque
Honda	1.6 L	140 Nm
Volvo	1.8 L	145 Nm
Nissan	2.0 L	290 Nm

The Nissan CVT was chosen because of its high maximum allowable torque. This Nissan CVT although twice as large, has numerous similarities to the 1.0L CVT. This CVT is used as part of a parallel hybrid electric powertrain, which combines the torque from an electric motor and a gasoline engine. Shown below is the belt and pulleys of the 2.0L CVT.

Figure 6.1 – 2.0L CVT Belt and Pulley System

The Nissan CVT was modified to improve efficiency and mechanical simplicity. Mechanical changes to the CVT were made with the goal of removing unnecessary components while retaining as many stock components as possible. The bi-directional rotation of the electric motor allowed removal of the reversing gear set and all of its related clutch components. The electric motor's high torque at low speed and the lack of engine idle in the control strategy eliminated the need for a torque converter. These modifications reduce the mechanical complexity of the CVT and the operating losses caused by these systems.

Figure 6.2 shows the high pressure pump, shifting valve body, and reversing components removed from the Nissan CVT and the equivalent replacements added by UC Davis. And in Figure 6.3 the stock Nissan valve body is shown on the left were as on the right is the UC Davis servo motor / pump system.

Nissan Components **UC Davis Components**

Figure 6.2 - Internal CVT mechanical simplification.

Figure 6.3 - The Nissan Mechanical Hydraulic Control System (left)
The custom UC Davis Computer Controlled Servo Hydraulic System (right)

The 1999 UC Davis Future Car uses an in-line, CVT, parallel hybrid powertrain which integrates the latest technologies in internal combustion engines, transmissions, and electric

motors. A small displacement engine and permanent magnet brushless DC motor are directly coupled to the CVT. A diagram of the powertrain is shown in Figure 6.4. This powertrain design focuses on simplicity, reliability, and manufacturability.

Figure 6.4 - In-line Powertrain Schematic.

The Nissan CVT has only one torque input which is from the engine. For use in this hybrid powertrain system it had to be modified. An internal spline was machined into the end of the input pulley. This spline was used to couple the electric motor to the transmission. This spline was design for an operating life greater than 10,000 miles, if machining and assembly error are less than 0.005 inches. A custom bracket was also manufactured to mount the motor to the case.

Figure 6.5 - Drawing of Cross Section of the CVT and the Electric Motor (left)
Custom Electric Motor Mount and View of Internal Spline (right)

The stock hydraulic pump and electro-mechanical control system were replaced with computer controlled servo hydraulic pumps to improve system efficiency. The stock Nissan pump is sized to supply the flow rate required for shifting, lubrication, and maximum clamping pressure at low engine speeds. As engine speed increases, the flow rate from the positive displacement pump increases. Clamping pressure depends only on torque demand instead of engine speed, so pressure is maintained by increasing fluid bleed-off. Therefore, system losses increase with engine speed. The servo hydraulic pump system decouples flow rate from engine speed. This system provides the necessary flow rate and clamping pressure on a demand basis with no bleed-off. Initial testing has shown an order of magnitude reduction in parasitic hydraulic losses. The hydraulic circuit including pumps, pressure regulators, and pressure transducers is shown in Figure 6.6.

Figure 6.6 CVT hydraulic system schematic.

7.0 Results

After months of designing, manufacturing, and assembly, the CVT was successfully implemented into the 1999 UC Davis Future Car. The vehicle was driven from Detroit to Washington D.C. The calculated fuel economy for this trip was 40 mpg which is a considerable increase over the stock vehicle's fuel economy. The control system for the CVT proved to be very robust in many operating conditions such as extremely hot weather, hill climbing, and even performance driving. Operation of the CVT was incredibly smooth and seamless as it shifted, and noise levels were very low even when transmitting high torque and high power.

Figure 7.1 - CVT Parallel Hybrid Powertrain in Vehicle

Figure 7.2 - CVT Parallel Hybrid Powertrain

8.0 Conclusion

The concepts proposed by Dr. A. Frank and Gear Chain Industrial are feasible and servo-hydraulic control of a CVT has been implemented. The advantages of such an implementation are increased energy efficiency, and precise control of the powertrain power loading.

The closed loop control algorithms needed for a servo-hydraulic CVT have been demonstrated. It is necessary to incorporate non-linear control concepts into these control algorithms to achieve the desired performance, however, with the availability of high performance, low cost, digital controllers today, the concepts are simple and inexpensive to implement.

The replacement of a discrete transmission with a CVT necessitates a control structure to take advantage of the new gear ratio flexibility. One such structure has been designed and simulated which uses the CVT to move to powertrain to its point of highest efficiency for each power setting commanded by the driver. This strategy could be easily modified to optimize fuel economy or exhaust emissions.

The 1999 UC Davis Future Car has proven that a CVT hybrid powertrain can greatly improve fuel economy. Further improvements though can be made in continued optimization of powertrain control strategies and energy management.

9.0 References

1. N.H. Beachley, A. A. Frank, "Principles and Definitions for Continuously Variable transmission, with Emphasis on Automotive Apliications" ASME 80-C2/DET95.
2. C. Chan T. Volz, D. Breitweiser, A. Frank, F.S. Jamzadeh, T. Omitsu, "System Design and Control Considerations of Automotive Continuously Variable Transmissions" SAE 840048.
3. G. Ellis, "Control System Design Guide: Using Your Computer to Develop and Diagnose Feedback Controllers" Academic Press, Incorporated. © 1992.
4. J. Vonroij, "A New Control for Cotinuously Variable Transmissions." Gear Train Industrial.
5. D. Yang, A. A. Frank, "An Optimization technique for the Design of a Continuously Variable Transmission Control System for Automibiles." Int. J. of Vehicle Design, vol. 6, no. 1, pp. 41-54.
6. D. Yang, A. Frank, "On the Use of Engine Modulation for Deceleration Control of Contunously Variable Transmission Vehicles." SAE 850490
7. Di Yang, Z. Guo, A. A. Frank, "Control and Response of Continuously Variable Transmission Vehicles" University of Wisconsin, Dept. Electrical and Computer Engineering, Madison, Wisconsin.
8. M. Alexander, B. Johnston, D. Funston, B. Huff, R. Carlson, N. Meyr, B. Moran, B. Momsen, Dr. A. Frank, and Dr. A. Burke; " A Mid-Sized Sedan Designed for High Fuel Economy and Low Emissions: The 1999 UC Davis Future Car " University of California at Davis, Dept. of Mechanical Engineering, Davis, California.

Appendix A

Sample pressure control compensator

```
// This program is a digital feedforward compensator for the clamping pressure loop.
// Sample rate = 300 Hz.


#use ezlob117.lib

#use ezlocmmn.lib

#use ezlopbdv.lib

#use ezlopic2.lib


// function prototypes
int PRT1_init( int tc );                    // Initialize PRT1
int DAC2_init( void );
void init_global(void);

// Define Constants
#define ON 1
#define OFF 0
#define PRT1_VEC 0x06                       // interrupt vector for timer 1
#define TDB1_1                              // PRT Ch1 down count enable
#define TIB1 5                              // PRT Ch1 interrupt enable
#define ticks  3070                         // the number the counter counts to
                                            // before actuating the routine
                                            // (18.42MHz / 20)*.010 sec
                                            // 9200 -100 4600 -200 18400 - 50
                                            // 3070 - 300 Hz

//#define Kp 4.83
//#define Ki 60.8
#define Kp 0.350
#define Ki 1.0
#define Kd 0.0
#define Kff 1.0
#define Ksh 1.50
#define wc_fb 188.0
#define wc_cmd 1.0
#define sample_time 0.00333
#define Sat_neg (-4.0)
#define Sat_pos ( 4.0)

// Global Variable accessable to interupts

shared float analog_read0,
            analog_read1,
            analog_read2,
            error[3],
            feed_back[3],
            command[3],
            fb_filter[3],
            cmd_filter[3],
            P[3],
            I[3],
            D[3],
            out[3];


shared int dij,                             // interupt counter
```

```
            go,                                  // do not start sampeling untill adc's are
initialized!
                        led_ctr,
                        DAC2_addr,               // DAC2 address number
                        analog_write,       // value to be written to DAC2
                        flag;                         // For a triangular test wave


void
main(){
  int i;
  i = 0;

  init_global();
  PRT1_init(ticks);                              // Initialize the count down timer interrupt
  if( DAC2_init() == -1)                    // initalize analog out board
    printf("Could not initialize the DAC2 board\n");
  eioBrdInit(0);                                        // Initialize the analog in
  hitwd();                                                   // Hit Wd timer
  go = 1;
  for(;;){                                                   // outside loop of joy,
interrupt runs
                                                                           // like crazy
in here.
        if(i<25){
    printf("error[0]=%+6.3f P[0]=%+6.3f out[0]=%+6.3f A2=%+6.3f\n"
                 ,error[0],P[0],out[0],analog_read2);
//    printf("out0=%+6.3ferror[0]=%+6.3fP[0]=%+6.3fI[0]=%+6.3f A2=%+6.3f\n"
//               ,out[0],error[0],P[0],I[0],analog_read2);
//    printf("AN0%+6.3f  AN1[0]=%+6.3f    A2=%+6.3f\n"
//               ,analog_read0,analog_read1,analog_read2);
    i = 0;
//    i++;
    }
  }
}

// The count down timer interupt routine
// needs an initialization routine to be called first in
// main()==> PRT1_Init(ticks);
#INT_VEC PRT1_VEC rupt              // create a interrupt vector
interrupt reti rupt(){
  inport( TCR);                                  // ??
  inport( TMDR1L);                          // Clear timer register
  EI();                                                 // re-enable the interrupt again
  hitwd();                                              // hit the watch dog timer

  if( go != 1) return;
//*********************************************************************
//  Interrupt code body:
//*********************************************************************
// analog read
// analog_read0 = pressure feed back signal
// analog_read1 = pressure command signal
// analog_read2 = current feed forward command signal from shift
  analog_read0 = eioBrdAI(0);                           // Sample analog channel 0
  analog_read1 = eioBrdAI(1);                           // Sample analog channel 1
  analog_read2 = eioBrdAI(2);                           // Sample analog channel 2

  analog_read0  = analog_read0 - .23;        // remove sensor bias
  analog_read2  = analog_read2 - 1.697;
  feed_back[0]  = analog_read0;
  fb_filter[0]  = wc_fb*sample_time*feed_back[1] +
                (1 - wc_fb*sample_time) * fb_filter[1];
  command[0]    = analog_read1;
  cmd_filter[0] = wc_cmd*sample_time*command[1] +
                 (1 - wc_cmd*sample_time) * cmd_filter[1];


// process the data, herein lies the compensator transfer function
   error[0] = (analog_read1 - fb_filter[0]) ;
```

```
    p[0] = error[0] * Kp;

    out[0] = p[0] + analog_read1*Kff - analog_read2 * Ksh;

    if(out[0] > Sat_pos){
      out[0] = Sat_pos;
    }
    if(out[0] < Sat_neg){
      out[0] = Sat_neg;
    }

    analog_write = (out[0]) * 409.6 ;

    // analog write

    if(analog_write >= 0){
    plCXP86Out(DAC2_addr,analog_write);        // output to DAC2 channel 1

    // channel 2 = DAC2_addr +1
    plCXP86Out(DAC2_addr+1,0 );                // output to DAC2 channel 1
    }
    else{
    // channel 2 = DAC2_addr +1
    plCXP86Out(DAC2_addr+1,analog_write*-1);// output to DAC2 channel 1

    // channel 2 = DAC2_addr +1
    plCXP86Out(DAC2_addr,0 );                  // output to DAC2 channel 1

    // channel 2 = DAC2_addr +1
    }

    // update the history variables just in case we need them later
    //   feed_back[2] = feed_back[1];
    feed_back[1] = feed_back[0];
    fb_filter[2] = fb_filter[1];
    fb_filter[1] = fb_filter[0];
    //   command[2]   = command[1];
    command[1]   = command[0];
    //   cmd_filter[2] = cmd_filter[1];
    cmd_filter[1] = cmd_filter[0];
    //   error[2] = error[1];
    error[1] = error[0];
    //   out[2] = out[1];
    out[1] = out[0];

    //*****************************************************************

    /* The human i/o portion of the deal: */
    led_ctr++;
    if( led_ctr >= 50){
    led_ctr = 0;
      if(dij == 0){
        switchLED(0);
        dij = 1;
      } else {
        switchLED(1);
        dij = 0;
      }
    }
  }

int
PRTl_init ( int tc){                       // tc = ticks defined above
  IRES( TCR, TDE1 );                       // Disable count down
  IRES( TCR, TIE1 );                       // Disable interrupts
  outport(TMDR1L, tc       );     //
  outport(TMDR1H, tc  >> 8); //Set data register
  outport(RLDR1L, tc       );     //
  outport(RLDR1H, tc >> 8);   //set reload counter
  ISET( TCR, TDE1 );                       // enable count down
```

```
    ISET( TCR, TIE1 );                  // enable interrupts
    EI();
    return(0);
}

int
DAC2_init(void){
    int i, j, test_addr;

    hitwd();
    eioResetPlcBus();                   // reset the plcbus
    hitwd();

    for(i = 0; i<=15000; i++){  // lag a bit so the DAC can
        hitwd();                                        // equalize
        for (j=0; j<10; j++);
    }

    //printf("I am so excited about salad sensations!!!\n\n");

// Locate XP8600s; Save the highest addr found
    DAC2_addr = -999;
    for(test_addr = 0; test_addr <= 7; test_addr++){
        hitwd();
        if(plcXP86Init(test_addr) != -1 ){
            DAC2_addr = test_addr * 2;
            //printf("Channel located = %d\n",test_addr);
        }
    if(DAC2_addr == -999)
        return( -1 );
    else
        return(DAC2_addr);
}

void
init_global(void)
{
    dij = 0;
    led_ctr = 0;
    flashled
    go = 0;
    error[0] = 0.0;
    error[1] = 0.0;
    error[2] = 0.0;
    P[0] = 0.0;
    P[1] = 0.0;
    P[2] = 0.0;
    I[0] = 0.0;
    I[1] = 0.0;
    I[2] = 0.0;
    D[0] = 0.0;
    D[1] = 0.0;
    D[2] = 0.0;
    out[0] = 0.0;
    out[1] = 0.0;
    out[2] = 0.0;
    feed_back[0] = 0.0;
    feed_back[1] = 0.0;
    feed_back[2] = 0.0;
    fb_filter[0] = 0.0;
    fb_filter[1] = 0.0;
    fb_filter[2] = 0.0;
    cmd_filter[0] = 0.0;
    cmd_filter[1] = 0.0;
    cmd_filter[2] = 0.0;
    command[0] = 0.0;
    command[1] = 0.0;
    command[2] = 0.0;
    analog_read0 = 0.0;
    analog_read1 = 0.0;
```

// led flasher state variable
// the number of interrupts for

```
  analog_read2 = 0.0;
}
```

Appendix B

Sample Rdot Compensator Routine

```
// Rdot controller.
// Samples data at 200 Hz.  But only processes the data at 50 Hz.  Averages 4 //analog reads each
time.


#use eziob117.lib

#use eziocmmn.lib

#use eziopbdv.lib

#use eziopic2.lib


// function prototypes
int  PRT1_init( int tc );      // Initialize PRT1
int  DAC2_init( void );
void init_PID( void );
void init_global( void );

// Define Constants
#define ON 1
#define OFF 0
#define PRT1_VEC 0x06                  // interrupt vector for timer 1
#define TDE1 1                         // PRT ch1 down count enable
#define TIE1 5                         // PRT ch1 interrupt enable
#define ticks  4600                    // the number the counter counts to
                                       // before actuating the routine
                                       // (18.42MHz / 20) * ).010 sec
                                       // 9200 -100 4600 -200 18400 - 50

#define integral_time 1
#define derivitive_time 1
#define r_derivitive_time 1
#define r_derivitive_alpha 4
#define r_dot_gain 36.0
#define r_gain 12.0
#define r_min 4.3
#define r_max 4.9
#define gain 18.0
#define integral_alpha 10.0
#define derivitive_alpha 0.10
#define sample_time 0.02


// Global Variable accessable to interupts

shared double analog_read,
                analog_read0[2],
                analog_read1,
                r_desired,
                error[3],
                out[3],
                P[2],
                I[2],
                D[2],
                gain1,
                gain2,
                dither,
```

```
        x,
        p1,                                 // PID calculation coefficients
        I1,
        I2,
        I3,
        I4,
        D1,
        D2,
        D3,
        D4,
        RD1,
        RD2,
        RD3,
        RD4,
        a0_temp,
        a1_temp,
        r_dot[2],
        analog_read0_old;

shared int dij,                             // interupt counter
           DAC2_addr,                       // DAC2 address number
           analog_write,      // value to be written to DAC2
           flag,                            // For a triangular test wave
           avg_ctr,
           hold;

void
main(){
  int i;

// Initialize local variables
  i = 0;

// Initialize global variables
  init_global();

// Initialize PID coefficients
  init_PID();

  PRT1_init(ticks);                         // Initialize the count down timer interrupt
  if( DAC2_init() == -1)                    // intialize analog out board
    printf("Could not initialize the DAC2 board\n");
  eioBrdInit(0);                            // Initialize the analog in
  hitwd();                                  // Hit Wd timer
  dij = 0;                                  // initialize interrupt counter
  for(;;){                                  // outside loop of joy, interrupt
runs
                                            // like crazy in here.
    if(i<25){

//    printf("out[0]=%.3f P=%.3f RD_c=%.3f Err= %.3f R=%.3f rdot=%.3f\n",
//           out[0],p[0],analog_read1,error[0],analog_read0[0],r_dot[0]);
    printf("RD_c=%6.3f rdot=%6.3f R=%6.3f P=%6.3f hold=%d d=%6.3f out=%6.3f\n",
           analog_read1,r_dot[0],analog_read0[0],p[0],hold,D[0],out[0]);

    i = 0;
//    i++;
//    printf("a0 %f a1 %f e0 %f\n",analog_read0,analog_read1,error[0]);
  }
}

// The count down timer interrupt routine
// needs an initialization routine to be called first in
// main()==> PRT1_Init(ticks);
#INT_VEC PRT1_VEC rupt                      // create a interrupt vector
interrupt reti rupt(){
  inport( TCR);                             // ??
  inport( TMDR1L);                          // Clear timer register
  EI();                                     // re-enable the interrupt again
  hitwd();                                  // hit the watch dog timer
```

```
//**************************************************************************
//  Interrupt code body:
//**************************************************************************
// analog read, average four sample values for error removal
  if(avg_ctr <  3){
    a0_temp += eioBrdAI(0);                    // Sample analog channel 0
    a1_temp += eioBrdAI(1);                    // Sample analog channel 1
    avg_ctr++;
    return;
  }
  else{
    analog_read0[0] = a0_temp/avg_ctr;
    analog_read1 = a1_temp/avg_ctr;
    a1_temp = 0;
    a0_temp = 0;
    avg_ctr = 0;
  }

// process the data, herein lies the compensator transfer function
// analog_read0 = ratio feed back signal
// analog_read1 = ratio command signal

  analog_read1 = (analog_read1 - 2.5) / 15;  // Rdot command
  if( analog_read1 < 0.01 && analog_read1 > 0) analog_read1 = 0;
  if( analog_read1 > - 0.01 && analog_read1 < 0) analog_read1 = 0;

  if (analog_read1 < 0.03 && analog_read1 > 0){
    if(hold !=1 ){
      hold = 1;
      r_desired = analog_read0[0];
    }
    r_desired = r_desired + analog_read1*sample_time;
    if(r_desired > r_max) r_desired = r_max;
    error[0] = r_desired - analog_read0[0];
    p[0] = error[0];
    out[0] = p[0] * r_gain;
    r_dot[0] =(analog_read0[0] - analog_read0[1]
              + r_dot[1]*r_derivitive_alpha    )
              /(r_derivitive_alpha+sample_time);
  }
  else if(analog_read1 > -0.03 && analog_read1 <= 0 ){
    if(hold !=2 ){
      hold = 2;
      r_desired = analog_read0[0];
    }
    r_desired = r_desired + analog_read1*sample_time;
    if(r_desired < r_min) r_desired = r_min;
    error[0] = r_desired - analog_read0[0];
    p[0] = error[0];
    out[0] = p[0] * r_gain;
    r_dot[0] =(analog_read0[0] - analog_read0[1]
              + r_dot[1]*r_derivitive_alpha    )
              /(r_derivitive_alpha+sample_time);
  }
  else{
    hold = 0;
    r_dot[0] =(analog_read0[0] - analog_read0[1]
              + r_dot[1]*r_derivitive_alpha    )
              /(r_derivitive_alpha+sample_time);
    error[0] = analog_read1 * 1.6 - r_dot[0];
    p[0] = error[0];
    out[0] = p[0] * r_dot_gain - 0.6;
  }

  if(out[0] > 4.95) out[0] = 4.95;
  if(out[0] < -4.95) out[0] = -4.95;
  if(analog_read0[0] > r_max && out[0] > 0.0)  out[0] = -0.5;
  if(analog_read0[0] < r_min && out[0]< 0.0) out[0] = 0.5;
```

```
    if(out[0] > 0.0){
      analog_write = (out[0] ) * (205);
      if(analog_write > 3686) analog_write = 3686;
      plcXP860ut(DAC2_addr + 1, 0);                                  // output to DAC2
channel 1

      plcXP860ut(DAC2_addr,analog_write);                            // output to DAC2 channel 1
    }
    else { // must be negative                                       // channel 2 = DAC2_addr +1
      analog_write = (out[0]) * (-205);
      if(analog_write > 3686) analog_write = 3686;
      plcXP860ut(DAC2_addr , 0 );                                    // output to
DAC2 channel 1

    plcXP860ut(DAC2_addr + 1,analog_write);                          // output to DAC2 channel 1
    }                                                                // channel 2 = DAC2_addr +1

// update the history variables
    error[1] = error[0];
    out[1]       = out[0];
    P[1]         = P[0];
    I[1]         = I[0];
    D[1]         = D[0];
    r_dot[1] = r_dot[0];
    analog_read0[1] = analog_read0[0];

//*****************************************************************

/* The human i/o portion of the deal:  DIO for the scope*/
    if(dij == 0){
      eiobrdDO(BankB(0), 1);                                         // 1 is for on, 0 is for off
//
BankX(Y), Y = bit of port X
      dij = 1;
    } else {
      eiobrdDO(BankB(0), 0);                                         // 1 is for on, 0 is for off
//
BankX(Y), Y = bit of port X
      dij = 0;
    }
}

int
PRT1_init ( int tc){                                                 // tc = ticks defined above
  IRES( TCR, TDE1 );                                                 // Disable count down
  IRES( TCR, TIE1 );                                                 // Disable interrupts
  outport(TMDR1L, tc          );     //
  outport(TMDR1H, tc  >> 8);  //Set data register
  outport(RLDR1L, tc          );     //
  outport(RLDR1H, tc >> 8);   //set reload counter
  ISET( TCR, TDE1 );                                                 // enable count down
  ISET( TCR, TIE1 );                                                 // enable interrupts
  EI();
  return(0);
}

int
DAC2_init(void){
  int i, j, test_addr;

  hitwd();
  eioResetPlcBus();                                                  // reset the plcbus
  hitwd();

  for(i = 0; i<=15000; i++){  // lag a bit so the DAC can            // equalize
  hitwd();
  for (j=0; j<10; j++);
```

```
}

//printf("I am so excited about salad sensations!!!\n\n");

// Locate XP8600s; Save the highest addr found
  DAC2_addr = -999;
  for(test_addr = 0; test_addr <= 7; test_addr++){
      hitwd();
      if(picXP86Init(test_addr) != -1 ){
          DAC2_addr = test_addr * 2;
          //printf("Channel located = %d\n",test_addr);
      }
  }
  if(DAC2_addr == -999)
    return( -1 );
  else
    return(DAC2_addr);
}


void
init_global(void)
{
    x = 0;
    flag = 0;
    gain1 = .45;
    gain2 = 1;
    r_desired = 0.0;
    error[0] = 0.0;
    error[1] = 0.0;
    error[2] = 0.0;
    out[0] = 0.0;
    out[1] = 0.0;
    out[2] = 0.0;
    P[0] = 0.0;
    P[1] = 0.0;
    I[0] = 0.0;
    I[1] = 0.0;
    D[0] = 0.0;
    D[1] = 0.0;
    analog_read = 0.0;
    analog_read0[0] = 0.0;
    analog_read0[1] = 0.0;
    analog_read = 0.0;
    dither = 1.0;
    avg_ctr = 0;
    a0_temp = 0;
    a1_temp = 0;
    hold = 0;
    analog_read0_old = 0;
    r_dot[0] = 0;
    r_dot[1] = 0;
}

void
init_PID(void){
  float
controller coefficient
                Ti,Td,Ts,Alpha_i,Alpha_d,K; // calculation variables.
//
PID's are declared globally
  //D1,P1,I1....

  Ti = integral_time;
  Td = derivitive_time;
  Ts = sample_time;
  Alpha_i = integral_alpha;
  Alpha_d = derivitive_alpha;
  K = gain;

  D1 = Ts + Td;
```

```
D2 = -1 * Td;
D3 = Td*Alpha_d;
D4 = 1 / ( Td*Alpha_d + Ts) ;
I1 = Ts + Ti;
I2 = -1 * Ti;
I3 = Td*Alpha_i;
I4 = 1 / ( Ti*Alpha_i + Ts) ;
P1 = 1;

printf("The PID's: %.3f %.3f %.3f %.3f %.3f %.3f %.3f %.3f %.3f %.3f\n",
                                    D1,D2,D3,D4,I1,I2,I3,I4,P1);

Ts = sample_time;
Td = r_derivitive_time;
Alpha_d = r_derivitive_alpha;

RD1 = Ts + Td;
RD2 = -1 * Td;
RD3 = Td*Alpha_d;
RD4 = 1 / ( Td*Alpha_d + Ts) ;
}
```

Appendix C

Specification of the servomotor / pump characteristics for a CVT clamping pressure and shift system.

1. General:
Medium: CVT Traction Fluid (Similar to ATF)
Temperature Range: -40°C min, +140°C max

2. Clamping Pump
Pressure: 35 bar (508 psi) Max
Flow: ± 1liter/min (± 0.264 gallons/min)
Max speed: 6000 rpm
Displacement: 0.2 cm^3 / rev (0.012 in^3 /rev)

3. Shift Pump
Pressure: ± 20 bar (290 psi)
Flow: ± 2.5liter/min (± 0.66 gallons/min)
Max speed: 6000 rpm
Displacement: 0.5 cm^3 / rev (0.031 in^3 /rev)

The above characteristics led to the selection of these specific components for the actual system:

1. Clamping Pump, Shift Pump
Max Pressure: 207 bar (3000psi)
Max speed: 6000 rpm
Displacement: 1.0 cm^3 /rev (0.065 in^3 /rev)
Manufacturer: John S. Barnes

2. Clamping Servo, Shift Servo
Peak torque: 19.2 N•m (11.9 ft•lb)
Continuous: 1.28N•m (0.8 ft•lb)
Max speed: 4000 rpm
Manufacturer: Hathaway Motion Control

In addition to the above characteristics, it is important to specify a zero cogging torque for the motor. This was overlooked during the first design iteration and proved to be a serious problem for the control of the system. Hathaway motion control has many lines of servomotors with zero cogging torque available.

Appendix D
Development of CVT Dynamic Equations

Dr. Andrew Frank first developed this equation, which governs the torque produced by a CVT coupled to an inertia, for his work on flywheel cars. This equation was used to model the vehicle dynamics for all driving cycle simulations.

$$\dot{\omega}_d = \frac{-\dot{R} I_e \omega_e + T_e R - T_{rl} - T_{loss}}{(I_e R^2 + I_d)}$$

This equation is developed by looking at a free body diagram about the drive shaft of the vehicle model shown in figure D.1.

Figure D.1. CVT vehicle model.

The differential of the transmission is omitted for simplicity. Ie and Id are inertias in the system. Ie represents the engine or electric motor inertia, and Id represents the mass of the car transformed into a rotating inertia about the drive shaft (m•R_{wheel}^2).

Figure xx. Free Body Diagram at the drive shaft.

Summing the torques and setting them equal to I•α yields:

$$T_e R - R I_e \dot{\omega}_e - T_{rl} - T_{loss} = I_d \dot{\omega}_d$$

Only,

$$\omega_e = R\,\omega_d \quad \Longrightarrow \quad \alpha_e = R\,\alpha_d + \dot{R}\,\omega_d$$

Substituting the chain rule result into the above equation with a little massaging gives:

$$(I_d + I_e R^2)\dot{\omega}_d = T_e R - \dot{R} I_e \omega_e - T_{rl} - T_{loss}$$

Dividing both sides by the inertia term yields the dynamic equation used for our vehicle simulations.

This equation behaves somewhat intuitively, a positive electric motor or engine torque gives a positive acceleration. And road load and aerodynamic losses

This equation has somewhat surprising consequences. It states that shifting the CVT such that the Rdot term is positive will contribute a negative acceleration effect to the drive shaft, and a negative shift rate will contribute a positive acceleration to the drive shaft. This effect on the drive shaft creates a somewhat counter intuitive feel to an automobile. For instance, when one is braking the power demand decreases and engine speed typically slows down. This results in a negative Rdot term which would actually increase the powertrain torque to the wheels!