

Analog and Digital Control of an Electronic Throttle Valve

By
Tomás V. Martins

SUBMITTED TO THE DEPARTMENT OF MECHANICAL ENGINEERING
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF THE
DEGREE OF

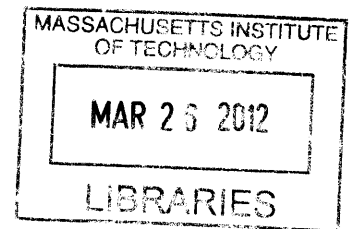
BACHELOR OF SCIENCE IN MECHANICAL ENGINEERING
AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

FEBRUARY 2012

©2012 Tomas V. Martins. All rights reserved

The author hereby grants to MIT permission to reproduce and to distribute publicly paper and electronic copies of this thesis document in whole or in part in any medium no known or hereafter created.

ARCHIVES



Signature of Author: _____

Department of Mechanical Engineering
January 23th, 2012

Certified by: _____

John B. Heywood
Sun Jae Professor, Emeritus
Department of Mechanical Engineering
Thesis Supervisor

Accepted by: _____

John H. Lienhard V
Samuel C. Collins Professor
Department of Mechanical Engineering
Undergraduate Officer

Analog and Digital Control of an Electronic Throttle Valve

by

Tomás V. Martins

Submitted to the Department of Mechanical Engineering
on January 24, 2012 in Partial Fulfillment of the
Requirements of the Degree of Bachelor of Science in
Mechanical Engineering

ABSTRACT

Two electronic throttler controllers were designed and implemented for an automotive throttle valve on a four-cylinder, spark-ignition gasoline engine. The first controller was designed using operational amplifiers and other analog componentry to realize a proportional-integral controller and feedback loop. The second controller utilized a programmable digital microcontroller to replace the analog componentry for signal processing. The use of analog to digital signal conversion by the microcontroller allows for the simple implementation of control logic and feedback loops through programming. Additionally, control architecture and characteristic gains implemented in the controller's code can be quickly changed and uploaded during testing. The digital controller was tested on the engine's throttle valve during motoring to demonstrate its actuation capabilities and response times.

The digital controller was programmed to quickly switch between different feedback signals like throttle angle, manifold pressure, and indicated mean effective pressure for control. The controller was designed for use in experimental testing of an experimental 2.0 liter, GM EcoTec engine in the Sloan Automotive Laboratory at MIT.

This study shows that rapid controller prototyping can be accomplished by using an inexpensive microcontroller for signal processing. This design concept greatly decreases implementation time and performance optimization time, increases controller flexibility and capabilities, and maintains favorable response characteristics.

Thesis Supervisor: John B. Heywood

Title: Professor of Mechanical Engineering

Acknowledgements

Although the purpose of this project is to learn about the design and implementation of throttle valve controllers, my involvement began after Kevin Cedrone built the analog PI throttle controller for use on his experimental engine. I would like to acknowledge both the work he put into designing the analog controller, and all the help he provided during the following stages of the project. His help with controller design, fabrication, and testing was invaluable.

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 5 |
| 2 | Design of Analog Throttle Controller | 9 |
| 2.1 | User Interface by Potentiometer | 10 |
| 2.2 | Analog Implementation of a Voltage Subtractor | 12 |
| 2.3 | Analog Controller Architecture | 13 |
| 2.4 | PWM Generation | 14 |
| 2.5 | Gain Tuning | 16 |
| 3 | Design of Digital Throttle Controller | 17 |
| 3.1 | Arduino Microcontroller | 17 |
| 3.2 | Digital Controller System Architecture | 20 |
| 3.3 | Implementation of Control Logic | 20 |
| 3.4 | Fast Gain Tuning | 23 |
| 3.5 | Additional Control Capabilities: MAP & NIMEP | 24 |
| 4 | Controller Features & Performance | 25 |
| 4.1 | Performance Validation | 27 |
| 5 | Conclusion | 30 |
| A | Arduino Code | 33 |

List of Figures

| | | |
|-----|---|----|
| 1.1 | Basic Carburetor Cross Section | 6 |
| 2.1 | Block Diagram of System with Analog Controller | 10 |
| 2.2 | Potentiometer Circuit | 11 |
| 2.3 | Differential Amplifier Circuit | 12 |
| 2.4 | Analog Proportional-Integral (PI) Controller | 14 |
| 2.5 | Absolute Value Circuit. The output signal is always a positive voltage with the same magnitude as the input signal. | 15 |
| 2.6 | Electronic circuitry of the analog controller implementation. | 16 |
| 3.1 | Front View of the Arduino Nano. | 19 |
| 3.2 | Block Diagram of System with Digital Controller | 20 |
| 4.1 | Diagram of Digital Controller User Interface | 26 |
| 4.2 | Throttle Body Step Response. | 28 |
| 4.3 | Manifold Absolute Pressure Step Response | 29 |

Chapter 1

Introduction

The internal combustion, spark ignition engine has been widely used as the dominant form of mechanical work production for consumer automobiles over the last century. Throughout this time, advancements in sensing, actuation, and fabrication have greatly improved the production, operation, and reliability of such automobiles. The development of electronic controlled solenoids and motors have greatly facilitated the implementation of engine air-fuel mixing with electronic fuel injection systems and electronic throttle valves. This paper discusses and compares the implementation of electronic throttle valve controllers through hard-wired analog circuits and soft-programmed digital microcontrollers¹.

The spark ignition (SI) engine uses a spark to ignite gasoline with air to release thermal energy for conversion to mechanical work. For the oxidation of gasoline to occur properly, a specific ratio between fuel and air must be preserved for optimal combustion. This ratio, called the air fuel ratio, asserts that for every unit by mass of gasoline, 14.7 units by mass of air must

¹While both valve controller designs may use analog, or variable voltage, and digital, or on/off, signals, the terms “analog” and “digital” are used to describe the components that make up the controllers themselves. That is, one design uses amplifiers and components for logic, while the other implements logic in code and programming.

also be present in the combustion chamber during ignition.[3] If there is too much air, the mixture is said to be lean. If there is not enough air in the mixture, it is said to be rich. The two components that directly control the amount of air and fuel per cycle in an SI engine are the fuel injectors and the throttle valve. They represent two of the most important components in the spark ignition (SI) engine. Fuel metering and intake air control were once performed together in a component called the carburetor. The carburetor was developed in the late nineteenth century and used as the predominant mechanism for air and fuel metering until the late 1980's. Shown in the figure 1.1, the carburetor was comprised of two main components: the butterfly valve and the fuel metering venturi and jet region.

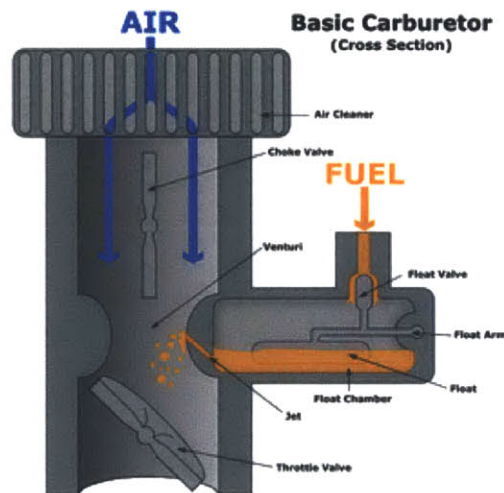


Figure 1.1: Basic carburetor cross section. Air enters from the top, is accelerated in the venturi, and mixed with fuel that is pulled from jet by lower static air pressure.³

¹Image courtesy of Wikipedia.org

The carburetor's operation relies on fluid mechanics to mix fuel with air. As air enters the carburetor, it will be forced into the narrow section, the venturi, which will accelerate the air. Bernoulli's principle indicates that the faster air travels, the lower its static pressure will be. As the accelerated air in the venturi passes the fuel jet, the lower pressure will pull fuel out of the jet for mixture. The amount of air that passes through the carburetor is controlled by the butterfly valve below. The more the valve is turned, the more air is pulled into the intake manifold by the pumping action of the pistons [3].

The development of solenoid actuation and electronic control in the 1980's made precise, reliable electronic fuel injection systems a reality. This led to the separation of air and fuel control through an intake air throttle valve and electronically controlled fuel injectors. The throttle valve assembly, or the throttle body has since become responsible for controlling the flow of air into the engine intake manifold. It accomplishes this using both an electric motor driven butterfly valve, as well as an angle sensing throttle position sensor (TPS). The signal from the throttle position indicates the angle of the butterfly valve and can be used to ensure proper actuation of the throttle valve based on desired angles.

The throttle body's electric motor and throttle position sensor must be coupled with a capable electronic controller for operation. The throttle controller consists of signal processing logic and components that use the throttle position sensor signal of valve angle as feedback when attempting to drive the valve to a desired angle. This controller can be composed of purely electronic components such as operational amplifiers, capacitors, and resistors, resulting in a purely analog architecture, or it can also use a microcontroller, with programming capabilities for a more digital control

architecture.

Both an analog and a digital throttle controller were built to compare the implementation and flexibility of each design. While the analog controller proved to be reliable and robust in many applications, an inexpensive microcontroller such as an Arduino, greatly decreases implementation and setup time and effort, increases controller flexibility, all the while maintaining reasonable performance and reliability.

This paper will show that improvements and advancements in electronic development platforms, such as the Arduino series, have made programmable control an excellent alternative to pure analog control. With an electronic, programmable development platform, implementation time is greatly decreased because many processing operations can be programmed into the microcontroller's processor. Flexibility is greatly increased as control schemes, architecture, and input can be be change by simply uploading a new program. Controller tuning and optimization time is significantly decreased as gains are adjusted on-the-fly in the code, instead of having to replace individual resistors or capacitors in the controller circuit. The digital controller built is to be used for testing purposes on a GM, turbocharged, four-cylinder EcoTec LNF engine.

Chapter 2

Design of Analog Throttle Controller

While transfer functions and control systems theory are both extensively discussed in core undergraduate mechanical engineering courses, one does not often learn about the realization of transfer functions through electronic circuits in such classes. Operational amplifiers can be conveniently used to implement lead-lag controllers, proportional, integral, derivative (PID) controllers, as well as nearly any other transfer function imaginable. In this manner, an analog PI controller was built to actuate the throttle valve to a desired user angle setpoint, while using the measured throttle position sensor signal as feedback. The subject matter covered in this chapter will relate to the design and implementation of the analog throttle controller.

The system's architecture, shown in figure 2.1, includes a desired throttle angle signal, an analog controller, the controlled plant, a feedback signal, and a subtracter that compares the voltage of the desired angle to that of the measured angle.

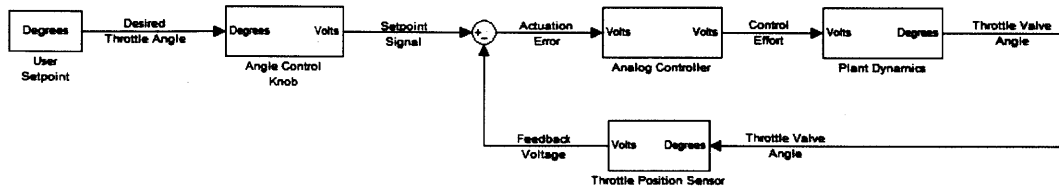


Figure 2.1: System architecture and block diagram. The throttle controller is implemented with exclusively analog components.

2.1 User Interface by Potentiometer

A potentiometer was used as a voltage divider to create a variable input voltage that the user controls. The potentiometer knob corresponds to the desired throttle angle, which is then converted to a signal from one to four volts. The voltage dividing range of the potentiometer was chosen to be the same as the sensing range of the throttle position sensor. This ensures that turning the potentiometer fully to the right will correspond to a wide open throttle and turning it all the way to the left corresponds to a completely closed throttle.

It was important to match the operational voltage ranges of the control knob potentiometer and the throttle position sensor so that the actuation error goes to zero when the throttle has reached the desired angle setpoint. Otherwise, if the potentiometer produces a setpoint voltage that the throttle position sensor cannot produce, the actuation error will always be nonzero and the controller might become unstable. This was accomplished by adding a $3\text{ k}\Omega$ resistor to either side of the $10\text{ k}\Omega$ potentiometer. This configuration is shown in figure 2.2.

Creating a control knob for the system was important to convert the

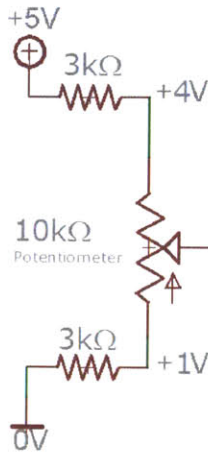


Figure 2.2: Voltage dividing potentiometer circuit used for user’s input signal. Allows for easy, quick control of the system command.

user’s desired behavior to an interpretable voltage signal. Other ways of accomplishing this would be to use a signal generator, or some form of computer controlled analog output. Generally, a signal generator will be able to produce a one to four volt analog signal, while also having other capabilities such as producing square waves or sinusoidal waves over a wide range of frequencies. While this is very useful for testing and validation, we wanted the user to have an easily accessible knob to control throttle angle. Similarly, a computer controlled analog output would provide various different capabilities with respect to throttle behavior control. However, its use would not be as straightforward and easily accessible as a simple input control knob. The control knob is also used in the digital controller implementation and will be discussed in Chapter 4.

2.2 Analog Implementation of a Voltage Subtractor

The feedback control loop requires the measured feedback signal to be subtracted from the input signal, so that an actuation error can be used for proper control of the plant. The simple voltage subtracter can be realized with an operation amplifier and four resistors, as shown in Figure 2.3. In this differential amplifier configuration, if all resistors are of the same value, the component becomes a unity gain amplifier with a characteristic equation of $V_3 = V_2 - V_1$ [5].

The differential amplifier outputs the actuation error signal. This represents, in volts, how far off the actual throttle angle is from the dialed-in desired angle. If the signal is negative, the throttle needs to open more. If the signal is positive, the valve needs to close to decrease the actuation error. This error is received and processed by the controller logic.

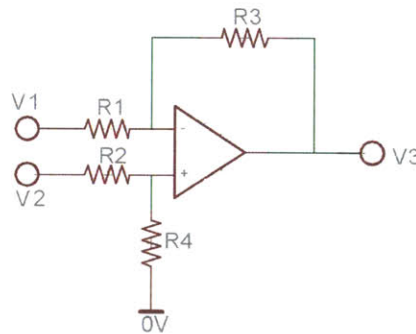


Figure 2.3: Differential Amplifier Circuit. This operational amplifier configuration subtracts two input voltages from each other, yielding the actuation error signal.

2.3 Analog Controller Architecture

The analog controller is responsible for processing the actuation error and sending a command signal to the plant to minimize the error. For a simple throttle controller, a proportional controller is sufficient for actuation and control. That is, the controller will send a command to the controlled plant that is proportional to the error and no more. However, the throttle valve uses a return spring to bring the valve back to the idle position when the drive motor is not applying any torque. This means that if the throttle is at the correct angle and there is no actuation error, a proportional controller will send no control signal, and there will be no torque to overcome the return spring and keep it at the desired angle. With such a controller, there will always be steady-state error and the valve will not reach its desired angle [4].

The inclusion of error integration in the analog controller eliminates steady-state error and improves valve actuation. Over time, actuation error is integrated so that a control effort may be applied to the valve motor even when the actuation error is zero. Such a controller is characterized by the following equation:

$$V_c = k_p * V_e + k_i * \int V_e dt \quad (2.1)$$

Where V_c is the control effort voltage sent to drive the motor, V_e is the actuation error voltage, while k_p and k_i are the controller amplification gains [5].

This was realized using three operational amplifiers. The actuation error signal is sent to an analog integrator and a simple amplifier, as shown in figure 2.4. The outputs of each operation amplifier is then summed to

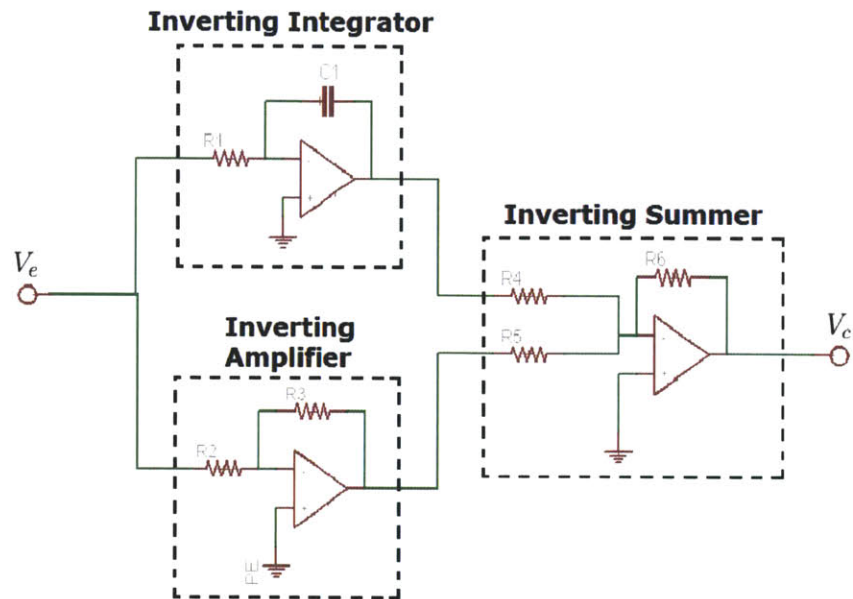


Figure 2.4: Analog Proportional-Integral (PI) Controller

produce the motor control effort.

2.4 PWM Generation

We cannot apply the motor control signal from the PI controller directly to the throttle body DC motor because the sensing and amplification circuit components cannot be exposed to the higher current flows that the motor experiences. Additionally, the coil loops of the motor effectively act as an inductor and must be isolated from the low current electronic componentry, so that voltage and current surges do not propagate and damage these components. The bi-directional control of the throttle dc motor using the PI controller's output signal is accomplished using an H-bridge. The H-bridge uses transistors to switch open and close both positive and negative twelve volts to the dc motor.

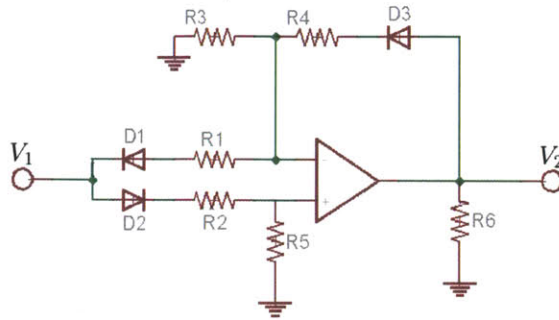


Figure 2.5: Absolute Value Circuit. The output signal is always a positive voltage with the same magnitude as the input signal.

The H-bridge requires a pulse-width modulation (PWM) command to switch the transistors and drive the motor at different torques. Converting the control effort signal to a PWM is accomplished by an absolute-value circuit, a sign comparator, and a 555 timer. The absolute-value circuit, as shown in figure 2.5, extracts the voltage magnitude from the control effort signal that is outputted from the PI controller [5]. This is passed into the 555 timer to produce a PWM signal of appropriate duty cycle.

The 555 timer is an integrated circuit that is used to produce a square wave whose duty cycle is determined by an input analog voltage, which comes from the absolute-value circuit in our controller implementation. This yields a PWM signal that depends on the controller output and is fed into the H-bridge to drive the valve motor. The control effort signal is also fed to a comparator to determine the direction of commanded motor torque.

Once the H-bridge receives a PWM signal and a direction signal, it applies a voltage to the motor for actuation. The throttle position sensor senses the valve angle and the loop is closed. Figure 2.6 illustrates the controller logic implemented in the analog design.

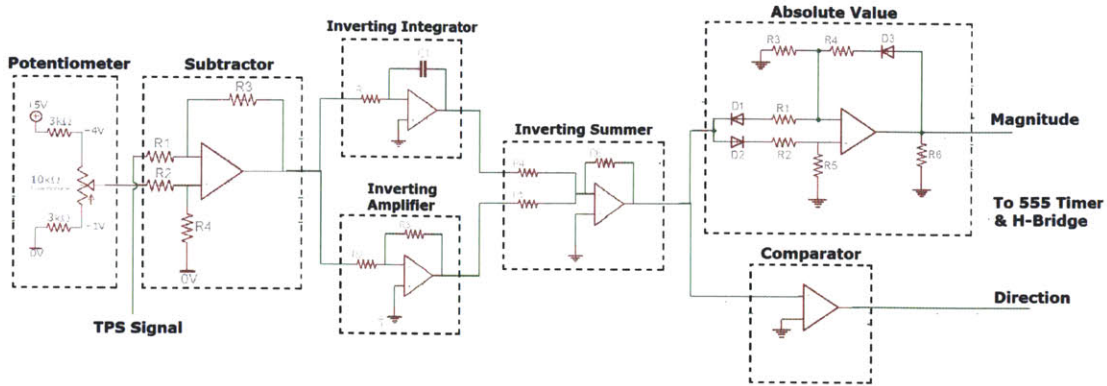


Figure 2.6: Electronic circuitry of the analog controller implementation.

2.5 Gain Tuning

Once the controller is assembled, it is necessary to validate its functionality, and if need be, change the controller gains to improve performance characteristics. Most component blocks in figure 2.6 have an effective amplification gain as set by the resistors and/or capacitors that are connected around the operation amplifiers. For simplicity, we have maintained the gain in all components but the PI control at unity. The effective amplification gain in the analog control as set by the PI controller componentry was $\frac{2}{477}$. This was the gain as set by Kevin that optimized actuation stability and performance.

The process by which the amplification gains are tuned for the analog controller is somewhat tedious, requiring the removal and installation of resistors and capacitors of different values. Controller performance is observed after the installation of the new components and the process is repeated until satisfactory performance and stability is achieved.

Chapter 3

Design of Digital Throttle Controller

The purpose of redesigning a throttle controller for the experimental 2.0L GM EcoTec engine was to not only learn about prototyping, controller design, and throttle actuation, but also provide a more capable engine control platform for experiments to be conducted. This was accomplished through the use of a microcontroller for signal processing and command control. While more expensive, creating a controller in such a manner cuts down on implementation time and effort, increases control capability and flexibility, all the while providing desirable performance characteristics. The microprocessor used for this project was an Arduino Nano, and was entirely responsible for signal processing and motor control.

3.1 Arduino Microcontroller

The Arduino Nano is a breadboard-friendly microcontroller carrier that is commonly used in hobby projects, and simple electrical engineering projects.

Its input and output capabilities, along with its compact size and processing speed make it an excellent candidate for use in a throttle controlling project. The board has over twenty input/output pins that can be used for sensing and control. Eight Arduino Nano pins are dedicated “analog input” pins that perform analog-to-digital voltage conversion and six pins are capable of pulse-width modulation signal generation. Two of the pins can act as external interrupts for low power operation. The board has an internal voltage regulator capable of producing a steady five and 3.3 volt supply. It also has a sixteen megahertz clock used for processing as well as PWM generation. As shown in figure 3.1, this entire package is available on a board less than an inch by two inches large and sold for under \$ 40 dollars [1].

Despite the many features and capabilities available with the Arduino Nano, only four input/output pins were needed to achieve throttle angle control. Two analog input pins were used to sense the user input signal and the throttle position sensor signal. The same control knob configuration from the analog controller discussed in section 3.1 was used with the digital controller setup for the user to dial in their desired angle. The analog input pins are capable of analog to digital conversion in a range of zero to five volts with an effective resolution of 0.005 volts. The other two pins used were “digital pins”, that can function as both a digital input and digital output. That is, each pin, once configured, can either sense zero or nonzero voltage signal or output a steady five volt or zero volt signal. One of the digital pins was configured to output a digital on/off signal for the direction pin on the H-bridge, while the other digital pin produced a PWM signal representing the torque command for the H-bridge [1].

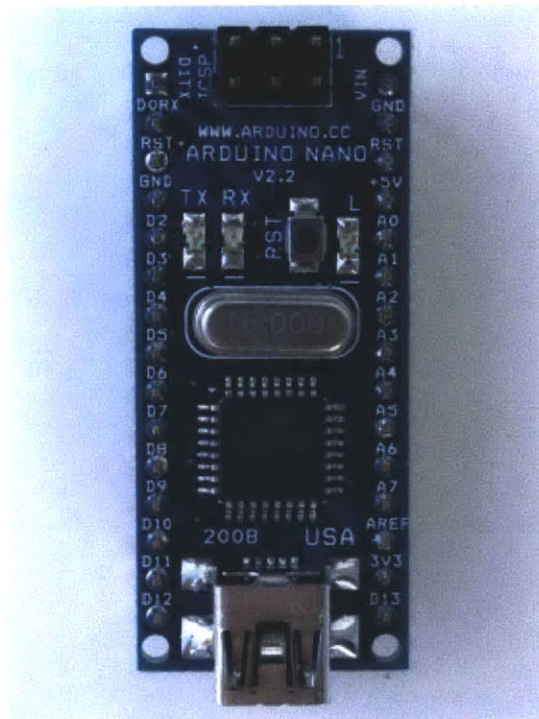


Figure 3.1: Front View of the Arduino Nano.

3.2 Digital Controller System Architecture

In the digital controller design, the Arduino Nano is responsible for receiving all the input and sensing signals as well as outputting the appropriate signals to control the motor. Figure 3.2 illustrates the new system block diagram, where the Arduino takes the place of the analog circuit's subtractor, PI controller, comparator, absolute value circuit, and 555 timer. With the Arduino Nano in place, no operational amplifiers need to be configured or used at all.

3.3 Implementation of Control Logic

Once the Arduino is connected to the potentiometer, the throttle position sensor, and the motor controlling H-bridge, the control logic must be written as an Arduino program that can be uploaded and executed within its processor. A basic knowledge of the Arduino code syntax is needed to write and implement scripts, however the language is very similar to C/C++ and the most common Arduino commands are well described on Arduino's reference website. Furthermore, due to its low cost and ease of use and implementation, the Arduino development platform has amassed a large following,

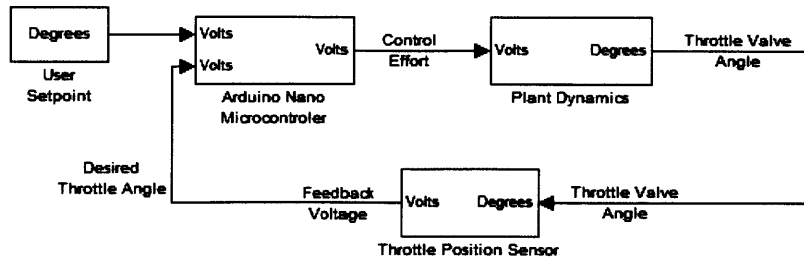


Figure 3.2: Block Diagram of System with Digital Controller

resulting in an abundance of documentation and support for almost any project one can think of with an Arduino board.

Here is a list of the most commonly used Arduino commands and their purpose [1]:

- `setup()` The initialization function for every arduino script.
- `loop()` The continuously running function that executes operational commands.
- `pinMode(Pin)` Configures the specified digital pin to either be a digital input or a digital output.
- `analogRead(Pin)` Commands the specified analog pin to measure the voltage applied to it and convert it to a digital value. This function will return a value from 0 to 1023 corresponding to zero and five volts, respectively.
- `digitalRead(Pin)` Commands the specified digital pin to read either a LOW (+0V) or a HIGH (5V) applied as its pin. This function will return a boolean 1 or 0.
- `digitalWrite(Pin,Value)` Commands the specified digital pin to output a LOW (0V) or HIGH (+5V) signal. The input value is either HIGH or LOW.
- `analogWrite(Pin,Value)` Commands the specified digital pin to simulate an analog voltage by outputting a pulse-width modulation signal. The input value is the representation of the desired duty cycle on a range from 0 to 255, where 0 is 0% PWM duty or zero volts, and 255 is 100% PWM or five volts

Once the `analogRead(Pin)` command is called to sense the voltages of the desired angle signal and tps signal, their values need to be subtracted from one another to obtain the actuation error. This error is then passed into a discrete PI controller to calculate the control effort. A simplified version of this command logic is written below for demonstration purposes. The actual implemented Arduino code can be found in Appendix A.

```
//READ SIGNALS AT ANALOG PINS

Here the setpoint and TPS signals are read.

User_Setpoint = analogRead(Pin_A1);

Read the potentiometer signal at the analog pin 1.

TPS_Signal = analogRead(Pin_A2);

Read the throttle position sensor signal at analog pin 2.

//CALCULATE ACTUATION ERROR

Actuation_Error = User_Setpoint - TPS_Signal;

This line implements the signal subtracter.

Error_Sum = Actuation_Error + Previous_Error_Sum;

Approximation of the integral of the error

//PI CONTROLLER

Discrete implementation of the PI Controller

Control_Effort = k_p*Actuation_Error + k_i * Error_Sum;

Kp represents the proportional gain and

Ki represents the integral gain.

//MAGNITUDE AND DIRECTION

The control effort signal is compared

to zero to determine direction.

if (Control_Effort > 0){

    If it is above zero,
```

```

Motor_Direction = 1;
    set the motor actuation direction,
Motor_Command = Control_Effort;
    and keep the positive magnitude.
}else{
    If its below zero,
Motor_Direction = 0;
    set the motor actuation direction the other way,
Motor_Command = -1*Control_Effort;}
    and take the positive magnitude.
//SEND COMMAND TO H-BRIDGE
    Actuate valve motor using H-bridge.
digitalWrite(Dir_Pin, Motor_Direction);
    Output the direction signal on the Direction Pin.
analogWrite(PWM_Pin, Motor_Command);
    Output the pwm magnitude signal on the PWM Pin.

```

3.4 Fast Gain Tuning

Unlike the analog controller, no components need to be disconnected or replaced to tune the controller or change its configuration. This can be achieved by simply writing a new script that has the new numerical gains or control logic to be used. Even simpler, for expedited tuning, a program can be written to interpret any numbers sent through the serial port to the Arduino as its gain parameters. For example, a user can configure the Arduino to expect a two digit number to be received in the serial port. Once the number is received, the Arduino would then set that number as the new controller gain, and resume its control operation with the new gain. With

this method, gains and control parameters can be changed in a few seconds, compared to the minutes or tens of minutes necessary to change those of the analog controller.

3.5 Additional Control Capabilities: MAP & NIMEP

The implementation of a throttle controller assembly with an easily programmable microcontroller provides the freedom to easily change the control architecture, or add new ones by simply writing and uploading a new Arduino program. We took advantage of this flexibility and supplemented the controller's angle control that uses TPS feedback with two new control schemes. The controller assembly was set up to also receive the manifold absolute pressure (MAP) signal from the MAP sensor on the engine and the nominal indicated mean effective pressure (NIMEP) signal from the engine's control computer.

The MAP and NIMEP signals are important in the characterization of the engine and its state of operation at any point in time. During engine testing, it may become more important to not keep the throttle angle constant, but rather to keep the engine's manifold pressure (MAP) or mean effective pressure (NIMEP) constant. Consequently, the controller assembly was assembled with a three way control switch, allowing the user to choose between TPS, MAP or NIMEP control. This switch toggles the input signal that is fed to the PI controller logic in the Arduino control program.

Chapter 4

Controller Features & Performance

The completed digital throttle controller, provides a simple and useful interface for operation and was tested on an experimental engine setup for performance validation. By installing the controller in a project enclosure, all components lay neatly hidden inside, while the small, 6" x 3" x 3" box can be placed near the engine's controls without taking up much space. The analog controller resided in a 6" x 6" x 10" box with multiple larger breadboards needed for the componentry around the many integrated circuits, and was clearly much more obtrusive than the smaller digital controller box. While these analog electric circuits can be greatly decreased in size by soldering the components onto a prototyping board, or designing a printed circuit board to be custom made, these options often require hours to accomplish, if not days.

The face of the project enclosure box has five main features that make the controller's operation easy and straightforward. The switch in the top center toggles the power supplied to the Arduino board, and the motor

controlling H-bridge. This switch is used to turn the entire system on and off. To the right of this switch, is an indicator light emitting diode (LED), which lights upon enabling the entire system. It indicates that the power is on and all components are at live voltage. In the bottom left corner is three-way switch used to toggle the operation mode the controller is in. The *up* position corresponds to the TPS-based, throttle angle control mode. The *middle* position corresponds to the NIMEP-based, nominal cylinder pressure control mode. The *down* position corresponds to the MAP-based,

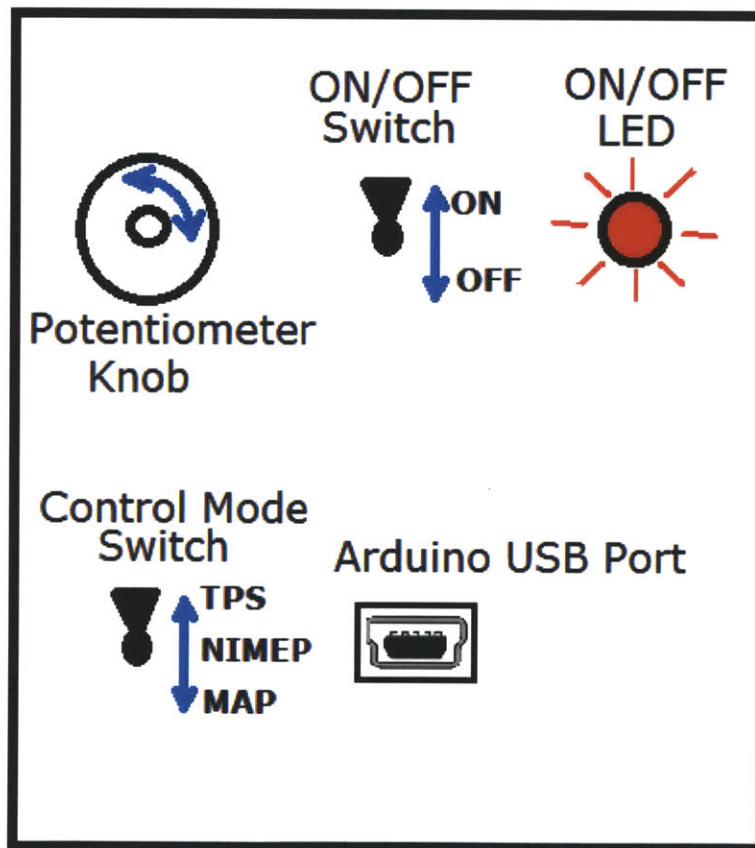


Figure 4.1: Diagram of Digital Controller User Interface

manifold pressure control mode. In the upper right corner, there is a blue knob connected to a potentiometer. This knob is used to dial in the control setpoint for any control mode. In TPS control mode, the knob controls the setpoint signal for the throttle angle and in other modes, the knob acts accordingly. At the bottom center of the controller's front panel is the Arduino Nano board's USB port. Without even opening the entire enclosure, a user can connect the Arduino to his computer to upload new programs or use the serial communication to tune gains or monitor variable behavior. Figure 4.1 illustrates the user interface and its features more clearly.

4.1 Performance Validation

The validation of the controller operation was performed on a GM 2.0L LNF EcoTec engine. This engine is a direct ignition, direction injection, turbocharged, 4-cylinder engine. The engine was operated at constant speed while the throttle controller command changed. Different input signals were applied to the controller to observe its response characteristics. A square wave was applied to the controller to observe its step response, shown in figure 4.2. The controller was also tested in its normal knob controlled operation.

The controller actuates the valve to its new angle setpoint within 780 crank angle degrees. At 1150 RPM, this translates to less than 0.1 seconds of response time. The remaining presence of steady-state error, which is under 10% indicates that the integral gain on the controller is not large enough. Designed to optimize response time and not steady state error, the current configuration sufficed for testing and use. However, as gain tuning progresses, some response time can be sacrificed to eliminate the small overshoot observed and to implement higher integral control for the

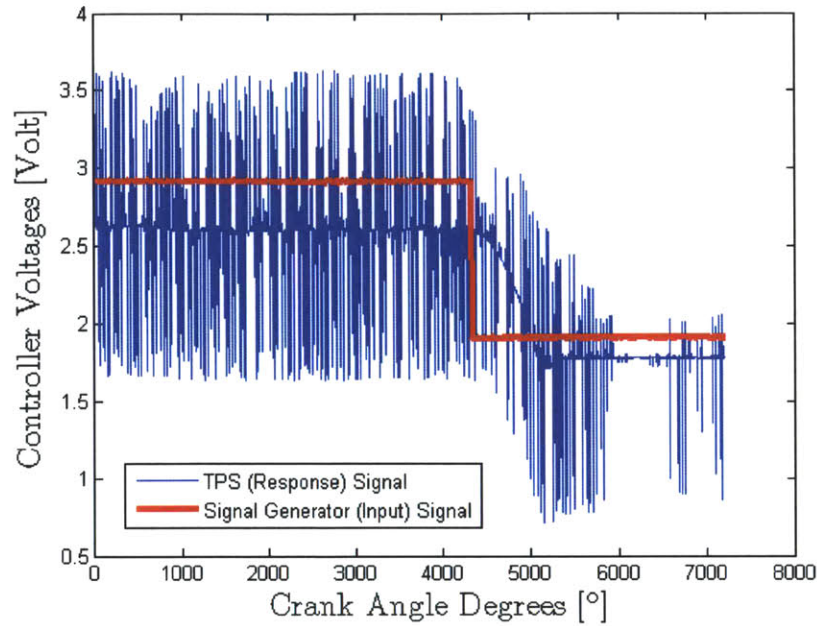


Figure 4.2: Throttle Body Step Response.

elimination of steady-state error.

The throttle position signal in figure 4.2 is accompanied with significant high frequency noise. While this noise can be minimized using a low pass filter, the integral term of the PI controller also serves to cancel out contributions from random noise.

Measurements of the manifold absolute pressure signal were also taken during the validation of the controller. Figure 4.3 show the immediate effect that the closing throttle valve has on the engine flow dynamics. Although the pressure fluctuates due to cylinder-to-cylinder intake pulses, the average pressure drops as the step control input is applied and as the valve closes.

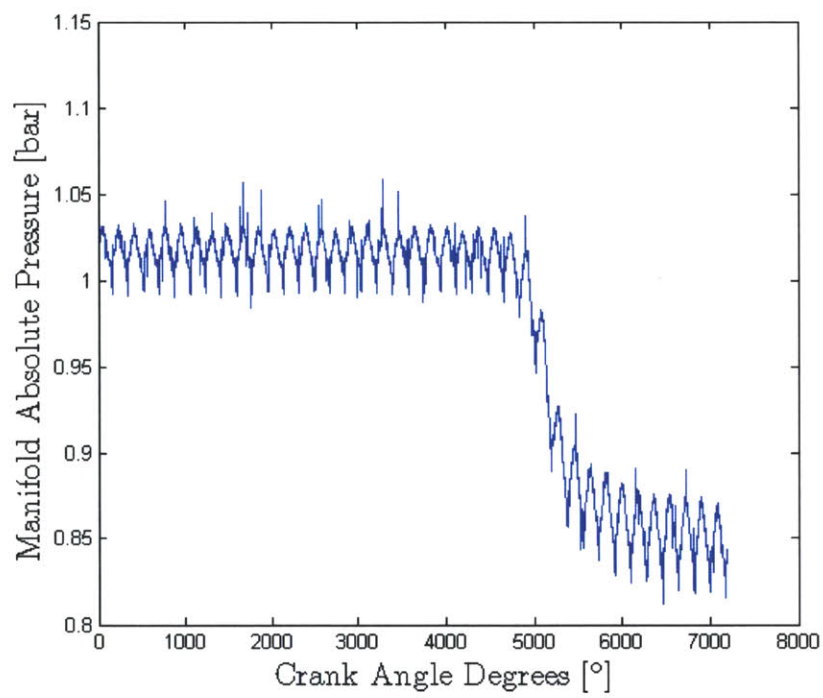


Figure 4.3: Manifold Absolute Pressure Step Response

Chapter 5

Conclusion

The Arduino microcontroller development platform opens a new door for inexpensive, fast, and easy controller prototyping for a large variety of applications. For example, using the analog inputs together with PWM generation for throttle control, ignition coil control, and fuel injector control signals is enough to quickly implement an engine control unit. At under \$40, the Arduino Nano may be considerably more expensive than the componentry used in the analog controller. However, taking into consideration implementation, tuning, and debugging time as well as capabilities, flexibility, and processing speed, \$40 is a small price to pay. Additionally, other capable versions of the Arduino development board are available at even lower prices. The Arduino Pro Mini is half the size of the Nano and costs half as much.

The Arduino's 16 megahertz clock allows for processing speeds that result in fast response times and complex control architectures. Using a simple programmed PI control scheme, the system was able to control the throttle valve with response times being as fast as 100 milliseconds. In under 10 minutes, dozens of control gains were used in the search for acceptable

controller behavior and valve responses. Additionally, the simple toggling of different control architectures made facilitated engine control for testing.

This project gave me a wonderful opportunity to learn more about electronic circuits, microcontroller usability and engine control methods, all the while contributing to the research efforts of a student in the Sloan Automotive Lab, Kevin Cedrone. It was possible to greatly reduce implementation effort and increase controller flexibility by using a programmable microcontroller such as an Arduino. Two throttle controllers were built and tested, and it was shown, that digital throttle controller is more than capable of controlling an engine throttle body with favorable performance characteristics.

Bibliography

- [1] Arduino Reference Homepage. Arduino. Web. 11 Dec. 2011.
- [2] Bosch Automotive Handbook. Plochingen, Germany. Warrendale, PA. Robert Bosch Society of Automotive Engineers. 2011.
- [3] Heywood, John. Internal Combustion Engine Fundamentals. New York: McGraw-Hill, 1988.
- [4] Nise, Norman. Control Systems Engineering. Hoboken, NJ: Wiley, 2011.
- [5] Trumper, David. *Operational Amplifiers*. 2.14 Course Notes. 2011.

Appendix A

Arduino Code

```
//Arduino Throttle Control Code
//Written by Kevin Cedrone and Tomas Martins

// *****
int pwmPin = 11;      // Output Pwm pin to H-Bridge
int dirPin = 2;      // Output Direction Pin to H-Bridge
// *****
int setPointPin = 1; // Input pot. voltage for set point
int tpsPin = 6;      // Input TPS2 voltage (0.5-4V for full closed to WOT)
int mapPin=5;        // Input MAP voltage (1-6V = 0-25psia ==> 1-3.94V = 0-1 atm)
int imepPin=4;       // Connected
int modePin1 = 7;    // Mode select pin 1
int modePin2 = 8;    // Mode select pin 2
// Physical mode switch up    [modePin1,modePin2]=[LOW, HIGH]
// Physical mode switch middle [modePin1,modePin2]=[HIGH, HIGH]
// Physical mode switch down  [modePin1,modePin2]=[HIGH, LOW]
// *****
int kpInv=1;
int kiInv=10;

int setPoint = 0;
int tps = 0;
int mapSig=0;

int error = 0;
int errorInt = 0;
```

```

int ctrlSig=0;
int actual=0;
boolean dir;
int pwm = 0;

int filtSP=0;
int filtTPS=0;
int filtMAP=0;

void setup()
{
  // set TIMER2 for PWM 32 kHz
  // clear all prescaler bits in TCCR2B = the last 3 Bits
  // leave other bits as set by arduino init() in wiring.c
  byte mask = B11111000;
  TCCR2B &= mask; // TCCR2B is now xxxxx000
  //
  // set CS22:20 in TCCR2B see p 156 of datasheet
  TCCR2B |= (0<<CS22) | (0<<CS21) | (1<<CS20); // same as TCCR2B |= B00000001; TCCR:

  pinMode(dirPin, OUTPUT);
  pinMode(pwmPin, OUTPUT);

  pinMode(setPointPin, INPUT);
  pinMode(tpsPin, INPUT);
  pinMode(mapPin, INPUT);
  pinMode(modePin1, INPUT);
  pinMode(modePin2, INPUT);
  Serial.begin(9600);
}

void loop()
{
  setPoint=analogRead(setPointPin);
  tps=analogRead(tpsPin);
  mapSig=analogRead(mapPin);
  // low pass filter to get rid of noise

  filtSP = ((filtSP << 3) - filtSP + setPoint ) >> 3;
  setPoint=filtSP;

  filtTPS = ((filtTPS << 3) - filtTPS + tps ) >> 3;
  tps=filtTPS;

```

```

filtMAP = ((filtMAP << 3) - filtMAP + mapSig ) >> 3;
mapSig=filtMAP;

//TPS mode, Physical mode switch up
if ( (digitalRead(modePin1) == LOW) && (digitalRead(modePin2) == HIGH) )
{
    actual = tps;
    kpInv=1;
    kiInv=10;
}

//MAP mode, Physical mode switch middle
if ( (digitalRead(modePin1) == HIGH) && (digitalRead(modePin2) == HIGH) )
{
    actual=mapSig;
    kpInv=100;
    kiInv=2;
}

//WOT Override, Physical mode switch low
if ( (digitalRead(modePin1) == HIGH) && (digitalRead(modePin2) == LOW) ){ setPoint=

//Physical mode switch error (impossible state)
if ( (digitalRead(modePin1) == LOW) && (digitalRead(modePin2) == LOW) ){ setPoint=(

errorInt += error;
//Anti-windup
if(errorInt > 400) { errorInt=400; }
if(errorInt < -400) { errorInt=-400; }

error=setPoint - actual;
ctrlSig=error/kpInv + errorInt/kiInv;

    if (ctrlSig >= 0)
    {
        dir = LOW;
        pwm = ctrlSig;
    }
    else
    {
        dir = HIGH;
        pwm = -1*ctrlSig;
    }
}

```

```
//Saturation limits on pwm output
if (pwm > 255) { pwm=255; }
if (pwm < 0)   { pwm=0;   }

// SEND OUTPUT TO H-BRIDGE
analogWrite(pwmPin, pwm); // analogRead values go from 0 to 1023, analogWrite va
digitalWrite(dirPin, dir);
```