The Kennesaw Journal of Undergraduate Research

Volume 5 | Issue 1 Article 1

June 2017

Autonomous Speed Control for KIA Optima

Andrew J. Combs
Kennesaw State University, acombs11@students.kennesaw.edu

Kyle Fugatt
Kennesaw State University, kfugatt@students.kennesaw.edu

Kevin McFall
Kennesaw State University, kmcfall@kennesaw.edu

Autonomous Speed Control for KIA Optima

Cover Page Footnote

We would like to give our heartfelt gratitude to Dr. McFall for his guidance during this project. Without him, this would not have been possible.

Autonomous Speed Control for KIA Optima

Andrew J. Combs, Kyle Fugatt, and Kevin McFall
Kennesaw State University

ABSTRACT

The standard method for speed control is the cruise control system built into most modern vehicles. These systems employ a PID controller which actuates the accelerator thus, in turn, maintains the desired vehicle speed. The main drawback of such a system is that typically the cruise control will only engage above 25 mph. The goal of this paper is to describe a system which we used to control vehicle speed from a stop to any desired speed using an Arduino microcontroller and a CAN BUS shield, from where autonomous features can be built upon. With this system, we were able to implement a proportional gain controller which maintains the speed at within ±1 mph with a 1s rise time.

Keywords: KIA Optima, Controls, Arduino, Speed Control, OBDII, CAN BUS, CAN, Serial, Controller Area Network, Automotive, Autonomous Vehicle

I. Introduction

THE cruise control system was invented by Ralph Teetor in 1950 as a way of limiting the maximum speed of a vehicle as it is driven. The patent explains the method of using the vacuum of the intake manifold to provide resistance on the accelerator [9]. Modern cruise control evolved from this initial method. In 1968, Daniel Wisner invented an electronic cruise control system which uses a voltage differential to control an actuator attached to the accelerator. A button is pushed once at the desired speed to lock the initial voltage and any change from that point engages the speed control mechanism [10]. The goal of this project is to further improve the existing cruise control in a 2012 KIA Optima. A similar project has been established by a group from Siegen University using a feed-forward PID in an AMOR autonomous mobile robot [2]. The system implemented by the Siegen group used R/C motor control and C++ programming; whereas, this paper proposes a direct control using an Arduino Mega and speed feedback through the vehicle controlled area network(CAN). With this method, a minimum speed is no longer required to engage the cruise control and a desired speed can be designating at any point in time. Further expanding on this idea, the system lays the groundwork to fully automate the vehicle control processes and allow implementation of autonomous control.

A. Controls System

The purpose of the cruise control is to allow the system to be self-regulating. The controller moves the acceleration and brake actuators independently, where the sum of the engine and brake related to the speed of the vehicle. The desired system is modelled in Figure 1.

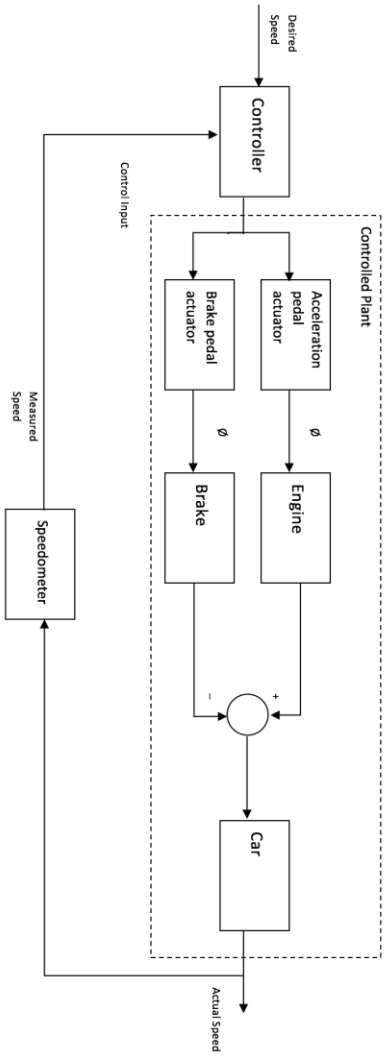

Figure 1: Cruise Control Diagram

[1]Andrew J. Combs, Kyle Fugatt, and Kevin McFall, Department of Mechatronics Engineering, Kennesaw State University. We would like to give our heartfelt gratitude to Dr. McFall for his guidance during this project. Without him, this would not have been possible. Correspondence concerning this article should be addressed to acombs11@students.kennesaw.edu, kfugatt@students.kennesaw.edu, or kmcfall@kennesaw.edu.

B. OBDII and CAN

Data acquired from the CAN is used to control speed. The CAN is a data bus where all the controllers for the car send and receive data. The primary data used in this paper is the speedometer information. This information is gathered from an encoder directly attached to the wheelbase.

The CAN, like every established network, has a distinct protocol that must be followed. To accomplish this, a serial communication module is installed in the vehicle [1]. Figure 2 Shows the protocol for every signal sent and received inside the CAN [11]. Connection to the CAN is simplest via the vehicle's OBDII port, with its CAN-H send line and CAN-L receive line.

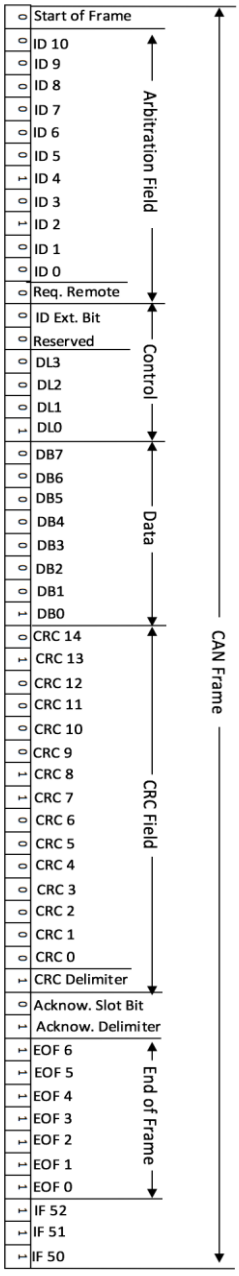

Figure 2:CAN Data Frame

II. HARDWARE

Before the car was modified, it was a standard KIA Optima from the factory donated to Southern Polytechnic State University, now Kennesaw State University. CAN is the protocol used throughout the car and can be established via the on-board diagnostic port, or OBD port, located under the dash. This is the port that mechanic shops use to diagnose the "Service Engine" light that appears on the instrument cluster when something is wrong in the car.

This can be used to communicate, while having an Arduino board controlling a few standalone motors. These motors were installed to control the steering, brakes, and gas pedal. The steering wheel was connected to the motor via a belt attached to a gear mounted on the steering column. The brake pedal was connected to the motor via a thin cable, coiling up around the motor's shaft when time to depress the brake, and unwinds when time to de-brake. Both of these motors are stepper motors and are 1200 oz-in, from Anaheim Automation, and classified in the 34Y series. See Table 1 for details.

Table 1 Stepper Motor Specifications

Model #	Torque	Current	Voltage	# Lead Wires	Weight	Length	Shaft Diameter
34Y207S-LW8	1200	5A	6.5VDC	8	8.4lbs	4.56in	0.5in

The gas pedal was controlled by two servo motors, operating in reverse direction from one-another. These 6.0VDC, 222.2oz-in motors were made by Savox. The motors were linked by a mechanical bolt linkage that went over the top of the pedal, securing it in a way that the pedal could not slip away. Table 2 contains the full details of the servos.

Table 2 Servo Motor Specifications

Model #	Torque	Current	Voltage	Weight	Length
SC-0251MG	222.2oz-in	350mA	6.0VDC	61g	40.7mm

Each of the two stepper motors were controlled by a motor driver. These drivers supply the correct voltage to each lead of the motor, in a certain "step" sequence. These are supplied with 24 VDC through a power supply, stepping up from 12 VDC. Table 4 contains full details for the motor drivers.

Table 3 Stepper Driver Specifications

Model #	Input Voltage	Output Current	Step Angle	Weight	Dimensions
CW250	20-60VDC	0-5A	1.8°	<500g	140x94x45mm

The servo motor was controlled directly from the Arduino controller by a pulse width modulated signal out, while being supplied with voltage by two separate power supplies, stepping down from 12VDC to 6VDC. Looking at the servos from the driver seat, the left servo's white wire, or signal wire, is connected to the Arduino on pin 27, and the right servo's signal wire is connected on pin 28. The black wires running to the servos are for ground and the red are for power.

The full electrical system is mounted on a plastic resin board in the trunk of the vehicle, which is mounted on a 3/4" plywood board frame. The carpet and what would be a spare tire were removed and the assembly was mounted to the body of the car. On the board, all electrical components were mounted strategically, with future implementations in mind. The drivers and power supplies are together with din rail and wire trace tunnels around the board to organize connections and hide connections. Each of the components are easily accessible from the raceway.

The circuit starts with the main 12 VDC supply running from the battery to a toggle switch mounted on the dash to the left of the steering wheel. From the toggle switch, the main supply runs in the door tracks, along with (3) eight conductor cables and the CAN bus cable, unseen, to the trunk of the car. After reaching the trunk, the wire is terminated through a battery protection fuse, to prevent drawing too much current, and is terminated in the terminal blocks. This source is distributed to everything that requires 12 VDC. Also, there are three cables that run from the front to the back which are dedicated for the motor connections, an emergency stop, and extra for future connections that may be used for future projects. The servo motors both require three connections each and the two steppers require four connections each. This totals to 14 connections total, accomplished with two cables. Also, the E-Stop requires two connections, resulting in only one cable left for open connections. Figure 3 shows the control wiring before the wiring paths are covered. Along the left side of the photo are the three cables for motor control.

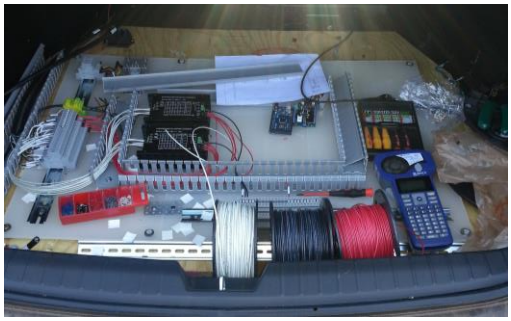

Figure 3 KIA Control Wiring

The servo motors were upgraded from the Savox SC models to Savox SB models. The new motors are the same voltage as the previous motors but have about 125 oz-in additional torque. The reason for the upgrade is when testing was being done, the motors were barely pushing the gas pedal down, needing assistance for initial depression. Table 4 shows the full specifications of the servos.

Table 4 Servo Motor Specifications

Model #	Torque	Current	Voltage	Weight	Length
SB-2270SG	347.2 oz-in	350mA	6.0V	69g	40.3mm

The stepper drivers, as discussed previously, provide the voltage needed to each motor's coil as necessary. This is done by the four connections made at the motor, but the driver is also connected to the Arduino. The Arduino sends voltage to both drivers by five wires: CP+, CP-, CW+, CW-, and H/S. These wires correlate to direction, speed, and position of each motor. For the steering motor driver, connections from the Arduino are not used. This project focused on accelerating, holding a constant speed, and decelerating autonomously. However, for the braking motor driver, connections from the Arduino to it are as follows; "CP+" is connected to pin 22, "CP-"to pin 23, "CW+" to pin 24, "CW-"to pin 25, and "H/S" is left blank. Both drivers are powered by 24VDC on terminal "VCC+" and -24VDC on terminal "GND- ". Earth Ground is connected via "GND". Figure 4 shows these connections.

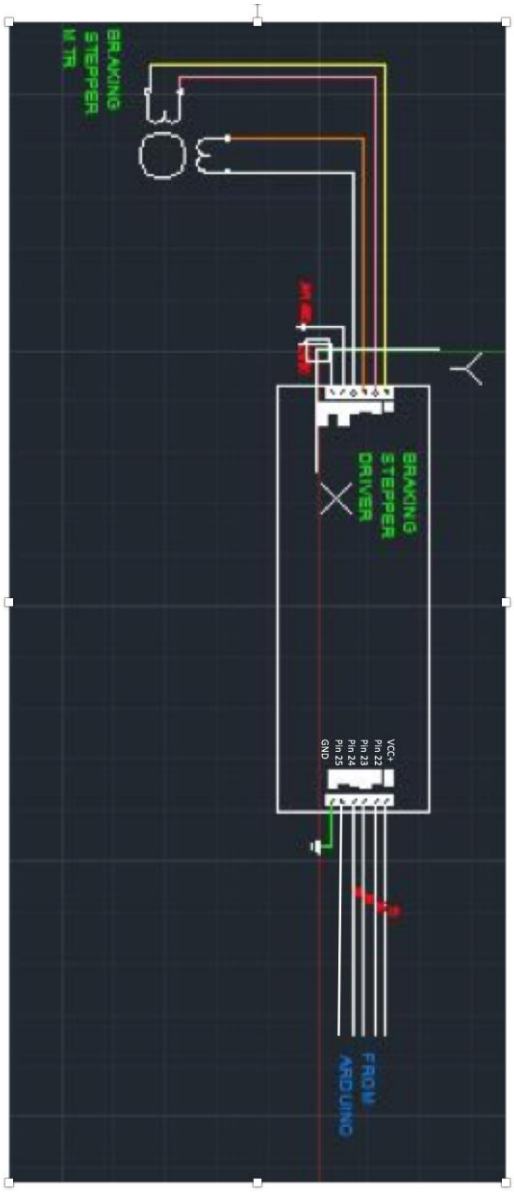

Figure 4 Motor Driver Diagram

These three connections can be made by jumping wires from one driver to the other.

The CAN-bus cable is converted from OBD in the door track before it is connected to the Arduino in the back. The OBD cable originally has 16 pin slots available for use. In the Can-Bus system, only 9 of these pins are used for the car. Therefore, to establish a connection, the 16 pin OBD cable is converted to a 9 pin DB9 cable connected to the Arduino Can-Bus Shield. This shield connects to the Arduino controller by stacking the Shield on top of the controller. The shield allows the Arduino to read information transmitted from the car.

III. METHODS

A. *Speed Data*

The most important data required to implement the speed control for the vehicle is the current vehicle speed. An Arduino Mega with the Sparkfun CANBUS shield was used to communicate with the CAN-bus. The shield is designed for use with an Arduino Uno; to make it work with a Mega the global.h header file must be modified. The default Uno SPI pins need to be changed to the Mega SPI pin locations for data to communicate between the module and the Arduino. The shield allows for direct serial communication through the OBDII using an OBDII to DB9 cable. The Sparkfun CANBUS library is used to read the data [1]. This allows the CAN bus to be sniffed and record the values read in through PUTTY then convert the output to a comma separated values (CSV) file. A graphical representation of the process is shown in Figures 5 and 6.

```
E4,4,"    0"," 0"," ",,,,,
F0, E4," 0"," ",,,,,,
F0,0,"    0"," 0"," 0"," 0"," 90","     10","     0"," "
90,0,"    0"," 0"," 0"," 0"," 0"," 0"," 0"," "
F0, E4," 0"," ",,,,,,
E4,4,"    0"," 0"," ",,,,,
A0,0,"    0"," 0"," 0"," 0"," 0"," 0"," 0"," "
A2,9,"    0"," 0"," 0"," 0"," 0"," 0"," 0"," "
F0,47,"   0"," 2"," C0","     0"," 90","     10","     0"," "
B0, FF," 7F","     FF","     0"," 80","     ",,,
F0,47,"   0"," 4"," C0","     0"," 90","     10","     0"," "
B0, FF," 7F","     FF","     6"," E0","     ",,,
```

Figure 5 CAN DATA BEFORE CSV

5	E4	4	0	0					
6	F0	E4	0						
7	F0	0	0	0	0	0	90	10	0
8	90	0	0	0	0	0	0	0	0
9	F0	E4	0						
10	E4	4	0	0					
11	A0	0	0	0	0	0	0	0	0
12	A2	9	0	0	0	0	0	0	0
13	F0	47	0	2	C0	0	90	10	0
14	B0	FF	7F	FF	0	80			
15	F0	47	0	4	C0	0	90	10	0
16	B0	FF	7F	FF	6	E0			

Figure 6 CAN DATA AFTER CSV

Now that the data in a form that is easily formatted, the information can be sorted and the order changed. This will allow the data to be more easily understood leading to the bit location of the speed and any other interesting data values. First, the commas need to be trimmed from the cells and then the hex values are converted into decimal values. The speed data will correspond to how speed was monitored during physical testing of the vehicle.

64	255	1	14	0	255	210	15	108
64	255	1	14	0	255	155	15	84
64	255	1	14	0	255	104	15	57
64	255	1	14	0	255	111	15	61
64	255	1	14	0	255	134	15	73
64	255	1	15	0	255	5	16	1
64	255	1	15	0	255	71	16	34
64	255	1	15	0	255	50	16	26
64	255	1	15	0	255	30	16	17
64	255	1	15	0	255	6	16	3
64	255	1	15	0	255	10	16	3
64	255	1	15	0	255	26	16	11
64	255	1	15	0	255	42	16	19
64	255	1	15	0	255	80	16	38
64	255	1	15	0	255	99	16	46
64	255	1	15	0	255	112	16	53
64	255	1	15	0	255	119	16	55
64	255	1	15	0	255	119	16	57
64	255	2	15	0	255	121	16	57
64	255	2	15	0	255	156	16	74
64	255	2	15	0	255	162	16	76

Figure 7 FORMATTED CAN DATA

In Figure 7, ID 40 byte 3 is the speed data with byte 2 being the current gear the engine is in.

B. *Arduino Code*

The Arduino processed the data gathered and compares the current speed with the desired speed. In doing this comparison the Arduino then controls the servers and steppers to regulate the vehicle speed. The flow of the program is written in Figure 9. The saw-tooth shape given by the system is indicative of a strict proportional gain control system. If the current speed of the vehicle is lower than the desired speed the brakes will always be in initial position and the servos will increment in position until the desired speed is reached. If the desired speed is lower than the current speed, the servos for the accelerator will move to the initial position and the brake will increment until the desired speed is reached.

IV. RESULTS

Using the Arduino and motor combination the team was able to successfully control the vehicle speed. If the desired input changes continuously, the accuracy of the system is limited to 2 mph; however, if the desired speed is steady over a prolonged period the system stays within 1 mph error.

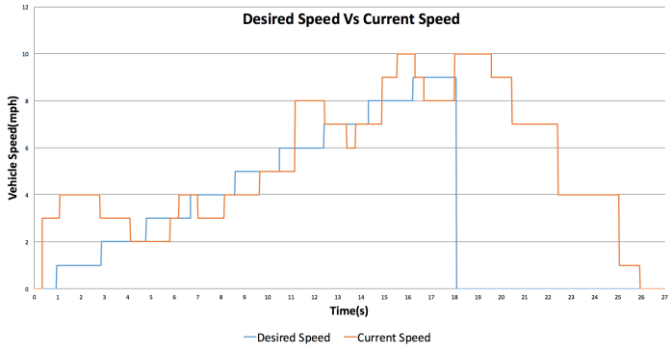

Figure 8 Speed Control Data

Figure 8 shows the visualization of the data received from the CAN control system. The hump in the beginning of Figure 8 is due to the natural speed of the vehicle without brakes or acceleration engaged. At the 4 second mark is when the desired speed and vehicle speed meet and the pairing begins. The desired speed is programmed to start at 0 and every 2 seconds increase by 1 mph until 10 mph is reached where the speed is set to 0 mph. The system accurately follows the desired speed until the drop off where the system slows to a complete stop.

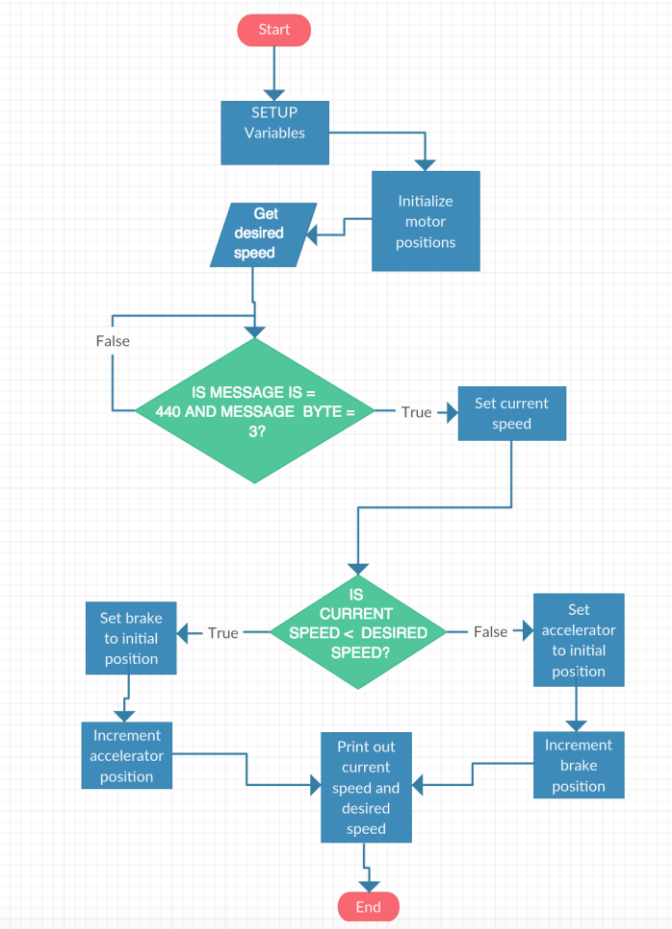

Figure 9 Flowchart for Arduino code

V. Conclusions

The system that has been implemented is very extensible and can easily be improved upon. The improvement that should be implemented first would be the addition of a full PID controller for the system. Additional variables need to be taken into consideration for this addition to make a sufficient impact on the control, such as the steering position and throttle position in the engine. Additionally, the control of steering would be an appropriate improvement on top of the current speed control.

References

[1] "sparkfun/SparkFun_CAN-Bus_Arduino_Library", GitHub, 2016. https://github.com/sparkfun/SparkFun_CAN-Bus_Arduino_Library.

[2] K. Sailan and K. Kuhnert, "Modeling and Design of Cruise Control System with Feedforward for all Terrian Vehicles", Computer Science & Information Technology (CS & IT), 2013.

[3] K. Osman, M. Rahmat and M. Ahmad, "Modelling and controller design for a cruise control system", 2009 5th International Colloquium on Signal Processing & Its Applications, 2009.

[4] A. Packard, Simple Cruise Control, 1st ed. UC Berkeley, 2005, pp. 24-52.

[5] M. Snare, "DYNAMICS MODEL FOR PREDICTING MAXIMUM AND TYPICAL ACCELERATION RATES OF PASSENGER VEHICLES", M.S., Virginia Polytechnic Institute and State University, 2002.

[6] G. CalderÃ³n-Meza, "A Simple Model of the Linear Speed of a Ground Vehicle Under Accelerating and Decelerating Forces", Embedded Systems Modeling Seminar, no. 501, 2003.

[7] N. Nise, Control systems engineering.

[8] M. Spong, S. Hutchinson and M. Vidyasagar, Robot modeling and control. Hoboken, NJ: John Wiley & Sons, 2006.

[9] R. Teetor, "Speed control device for resisting operation of the accelerator", US2519859 A, 1950.

[10] D. Wisner, "Speed control for automotive vehicles", US3511329 A, 1968.

[11] Robert Bosch GmbH; CAN with Flexible DataRate; Version 1.1; Date Aug. 2011, http://www.semiconductors.bosch.de/media/...pdf/canliteratur/can_fd.pdf