Local Control Unit Design for Electric Vehicle Control System

ECSE 457 - ECSE Design Project 2

Under the supervision of

Benoit Boulet Ph.D.
Associate Professor – Department of Electrical and Computer Engineering
Associate Chair - Operations

Group 35

Shayan Ahmad
260350431

Alejandro Carboni Jimenez
260523638

Aditya Saha
260453165

McGill University
April 2016

1 Abstract

Transmission control systems increase the efficiency of motor driven vehicles. Electric cars have been mostly manufactured without complex transmission systems, relying on heavy motors with high torque capacities. Introducing a transmission control system optimized for electric vehicles and their motors is a complex task, requiring many software and hardware components working simultaneously to manage correct behaviour. Implementation of such systems using modern microcontrollers enable for greater overall system efficiency with higher prospect of granular tuning and customization.

This project is aimed towards developing a microcontroller based prototyping tool for distributed control system for an existing transmission test bed for electric vehicles. The final system network must be based on Arduino microcontrollers and is required to use the automobile standard CAN protocol, specifically the J1939 variant, for communication. The deliverable software is required to be extremely modular without having any external dependencies. The operations accomplished by the modules in the final design must be complete and should be readily deployable or require minimal customization in the event of future prototyping applications.

A two pronged strategy has been used to accomplish the final deliverable. The first phase of the design process involved acclimatizing with the required system development and verification tools. Based on a stringent set of requirements, various system design models have been explored and their selection criteria investigated. The second phase of the design process involved actual implementation of the system design and careful testing to validate each system component taken separately and the system taken as a whole.

2 Acknowledgement

We would like to thank Yingxuan Duan, PhD. for her continued guidance and support, which has been instrumental in every step of the design process. We also appreciate the kind support of Hossein Vahid Alizadeh, Mir Saman Rahimi Mousari and Abdeslam Medouar. The final outcome of this project would have never been realized without their continued guidance.

Contents

List of Figures

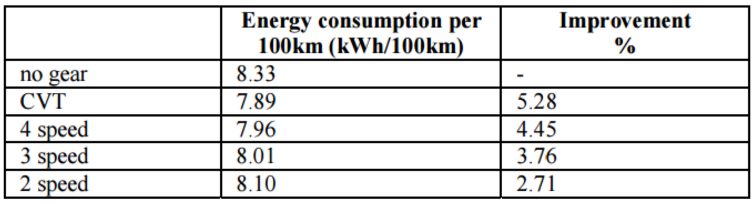
3 List of Abbreviations

CompactRIO - The CompactRIO is a real-time controller made by National Instruments (NI). The name comes from its compact nature and its Reconfigurable I/O capability.
CAN - Controller Area Network Vehicle bus standard design that allows inter-communication of devices and controllers in automobiles.
ECU - Electronic Control Unit Specialized microcontroller typically employed in embedded applications, including control of electrical subsystems in automobiles.
MCU - Microcontroller Control Unit Miniaturized computer on an integrated circuit designed for embedded applications.
CSMA - Carrier Sense Multiple Access Protocol Probabilistic media access control protocol employed in transmission through shared medium.
CD - Collision Detection Computational problem of detecting simultaneous intersection of multiple nodes in communication network.
AMP - Arbitration on Message Priority Set of rules defined in communication standard that specifies how bus arbitration will be carried out.
Kbps - Kilobits (1028 bits) per second Data transfer rate.
Mbps - Megabits (1028 X 1028 bits) per second Data transfer rate.
TCM - Transmission Control Module (also interchangeably used: TCU Transmission Control Unit) Device that controls modern electronic automatic transmissions.
VFS - Variable Force Solenoid Electro-hydraulic device that controls pressure proportionally or inversely proportionally to a driving signal, typically a voltage or current.
NEDC - New European Driving Cycle Standardized test designed to assess emission levels of car engines and fuel economy in passenger vehicles.
RPM - Rotations per Minute

4 Introduction

4.1 Motivation

The automotive industry, monopolized by internal combustion engine vehicles, has been going through a paradigm shift during recent years as the cost of fossil fuels has risen and their environmental impacts realized. This has brewed more interest in electric vehicles driving new research in areas of battery design with improved energy density and recharging properties. The electric motor exhibits a more desirable torque-speed characteristic spread in contrast to its internal combustion counterpart - providing maximum torque from zero up to low speeds, and subsequently releasing maximum power with increases in motor speed [1].



	Energy consumption per 100km (kWh/100km)	Improvement %
no gear	8.33	-
CVT	7.89	5.28
4 speed	7.96	4.45
3 speed	8.01	3.76
2 speed	8.10	2.71

Figure 1: *Efficiency improvements for electric motors for different transmission ratio over the NEDC cycle [1].*

The recent interest in electric vehicles has however been met with new challenges and trade-offs. The energy density of electric batteries is infinitesimal when compared to that of fossil fuels. Pure electric vehicles currently competing in the market are also equipped with single ratio gear transmission, intended to provide a trade-off between efficiency and dynamic performance. However recent research suggests that multi-speed transmission may be employed to provide better dynamic performance without compromising the prospect of efficiency in electric vehicles [2].

Recent advancements in technology can also be credited with the singular advantage of realizing the control units not only as hardware, but also as complete low-level functional software deployed on dedicated microcontroller units. All transmission sensors including temperature, rotational speed and path detection, are integrated, whereby a control unit does real time computations to drive the vehicle transmission. The central control unit takes charge of transmission control sequence and function, and relies on microcontrollers for gathering signals from the various sensors, information on bus and driver input patterns (including shifter positions, throttle state and pedal shifts). The computation of digital values occurs on-chip and is stored in nonvolatile memory.

4.2 Objective

A prototype for seamless clutch-less two-speed transmission for electric vehicles is already available. The aim of this project is to develop a prototyping tool for a distributed control system for the transmission, hence enabling a holistic approach in systematically driving motor speed during gear shifting by gathering sensor data pertinent to the immediate surrounding. The Arduino Uno has been chosen as the microcontroller platform of interest. The deliverable software should be highly modularized without having any external dependencies such that rapid prototyping for a functional system can be performed with minimal effort. The operations accomplished by the modules should be complete and require minimal customization for rapid deployment. The final system is required to have CAN compatibility for communication with the existing framework, specifically having support for the J1939 variant. A data logging module must also be implemented to aid in future applications in vehicle diagnostics and data acquisition.

4.3 Deliverable Breakdown

The following plan has been devised for the project timeline.

Term 1: Analysis and Literature Review

- Familiarization with system development and verification tools
- Familiarization with relevant communication protocols pertinent to the unit components of the final system design
- Exploration of system design models followed by investigation of their selection criteria
- Design and implementation of integrated microcontroller system for digital sensor data acquisition
- Implementation and validation of basic variant of CAN protocol

Term 2: Design and Validation

- Implementation and validation of J1939 variant of CAN protocol on Arduino platform
- Design and implementation of integrated microcontroller system for analog and digital sensor data acquisition
- Implementation and validation of data logging module
- Overall system integration and and validation using industry tools and typical application parameters
- Resolve and cleanup software modules for out-of-box readiness

5 Background

5.1 Controller Area Network

Controller Area Network (abbreviated as CAN) is an industry standardized in-vehicle communication protocol. It is a message-based serial protocol designed to establish intercommunication between electronic devices and/or micro-controllers in applications without a master controller, otherwise called a decentralized network. In automotive applications such devices include engine management system, anti-lock braking system, gear control system, active suspension, airbags etc. As an example of intercommunication in automobiles a spark ignition engine requires feedback from the engine control unit to control the precise timing of initiating the combustion chamber in order to provide better fuel efficiency while releasing more power. Similarly a transmission control unit leverages data passed by the engine control unit, the gearbox and various other sensors in the system to change the gear ratio at different speeds.

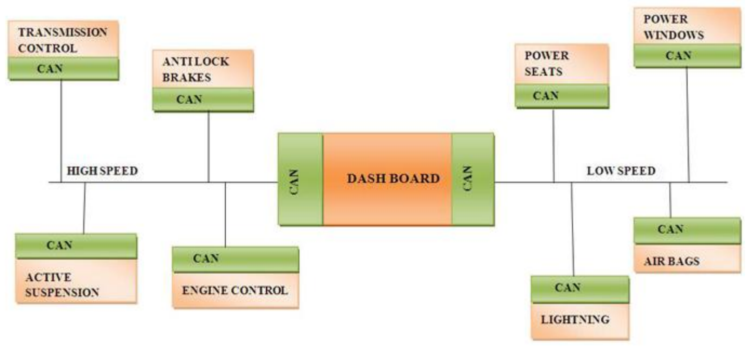
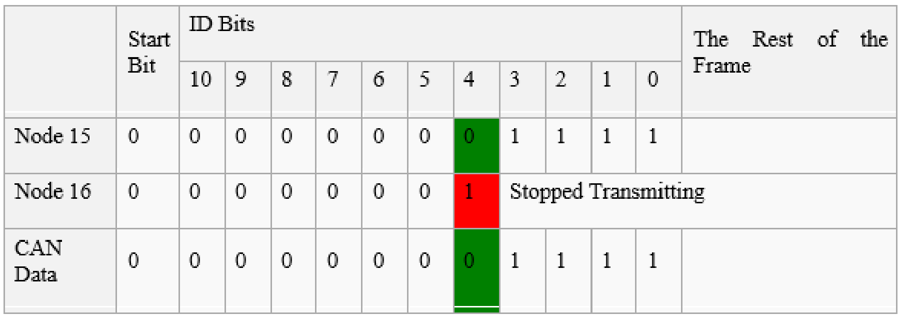


Figure 2: *CAN interconnect layout [3].*

CAN is characterized as CSMA/CD+AMP protocol [4]. CSMA, or carrier-sense multiple-access protocol means that each node on the bus is required to wait for a prescribed period of inactivity before it can prompt to send a message. CD+AMP, or collision detection and arbitration on message priority states that in the event of collision when multiple nodes prompt to get ownership of the bus, a bit-wise arbitration is carried out based on a pre-programmed priority of each message specified in the identifier field of the message. The CAN standard defines dominant and recessive transmission bit that corresponds to priority for bus access. Since bit-wise arbitration is carried out to resolve collision on the bus, all nodes on the network are required to sample every bit of data on the bus at the same pulse. The arbitration procedure is seamlessly carried out, ensuring least amount of lag in message transmission.



	Start Bit	ID Bits											The Rest of the Frame
		10	9	8	7	6	5	4	3	2	1	0	
Node 15	0	0	0	0	0	0	0	0	1	1	1	1	
Node 16	0	0	0	0	0	0	0	1	Stopped Transmitting				
CAN Data	0	0	0	0	0	0	0	0	1	1	1	1	

Figure 3: *Transmission sample of two nodes competing for bus arbitration, Node 15 winning bus access while Node 16 re-queues for later transmission [5].*

In order to establish CAN communication, each constituent node in the network is required to have a central processing unit, a CAN controller and a CAN transceiver.

- Central processing unit is the host processor that parses messages received over the bus and also decides what message is to be transmitted by the host node. Typically control devices, sensors and actuators are connected to such processing units.

- The CAN controller acts as an intermediary buffer which stores the received serial bits from the bus until an entire message worth of bits has been received. The controller then triggers an interrupt following which the host processor may fetch the received data. In the event of broadcasting the host processor sends the message to the controller, which initiates serial bit transmission onto the bus.
- The CAN transceiver acts as a message parser while receiving message, converting the received data stream from CAN bus levels to that understandable by the controller; and vice-versa when the node is transmitting. Transceiver also has protective circuitry that shields the controller from possible damage stemming from power surges on the bus line.

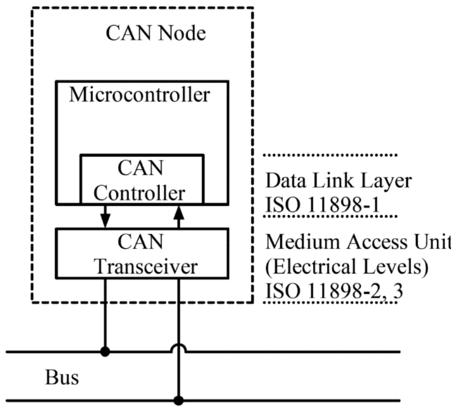


Figure 4: *Anatomy of a CAN node [5].*

5.2 J1939

J1939 is a variant of the CAN protocol which is the standardized communication protocol adopted by the International Society of Automotive Engineers (abbreviated as SAE). The main application focus of this standard variant is on the powertrains and chassis of commercial vehicles. Originally adopted in car and heavy-duty truck industry, this protocol is now widely adopted across a wide range of automobiles.

The J1939 protocol implements a decentralized multi-master communication network. It can support up to 254 logical nodes and 30 physical ECUs (abbreviated as Electronic Control Unit) per segment of transmission. The protocol defines specific hierarchical structures to moderate transmission. The information is interpreted as parameters that can be combined on 4 data pages in parameter groups (abbreviated as PG). Each parameter group can be identified via a unique number, the parameter group number (abbreviated as PGN). Irrespective of this structural hierarchy, each signal is assigned a unique suspect parameter number (abbreviated as SPN).

The J1939 protocol defines two modes of communication depending on the specific system implementation requirement. The majority of the communication occurs cyclically and can be received by all ECUs without explicit request of data, a way of communication referred to as broadcasting. In addition, the parameters groups are optimized to a length of 8 data bytes. This enables very efficient use of the CAN protocol. Particular information including configuration data or diagnostic data can be exchanged exclusively between two ECUs via peer-to-peer mode. The specification of the communication mode, broadcast or peer-to-peer, is a property of the parameter group used. Hence, the parameter group embeds the transmission type in addition to the definition of which parameters are transmitted.

J1939 also defines transport protocols for transmission of larger data quantities: Broadcast Announce Message (abbreviated as BAM) and Connection Mode Data Transfer (abbreviated as CMDT) [12]. BAM accomplishes transmission via broadcast, whereby control dataflow, or handshake, between the sender and receiver is non-existent. CMDT defines transmission protocol specifically between two ECUs. In the event of transmission error, CMDT allows a restarting of communication without complete repetition of the data transmission. CMDT also allows sending acknowledgement of data reception by the receiver.

To accomplish peer-to-peer communication, each ECU is required to be assigned a unique address in the range from 0 to 253. A network management protocol is also required to moderate network inconsistencies for instance, transmission between two ECUs with the same address. Network management is imperative to accomplish the following:

- Resolution of address conflicts
- Ongoing check as to whether ECU addresses are assigned in duplicate on a network
- Change of the ECU addresses dynamically at runtime
- Unique identification of an ECU with a name that is de facto worldwide

The specification of an ECU with a de facto standard name also serves to identify the component functionality in the network. With J1939, network management is decentralized. Hence each ECU implements network management logic to contribute to the overall robustness in the network.

5.3 Transmission Control

Control of modern automatic transmission system involves leveraging data gathered from the engine control module and various sensors across the automotive system to calculate the precise gear shift timings. The control system generates output signals that acts upon specialized actuators called transmission solenoids to accomplish this shifting. Modern transmission control systems are further optimized to contribute to improved vehicle performance, better shift quality and higher fuel efficiency thereby leading to overall reduced engine emissions and greater shift system reliability. Recent advancements in technology also provided for increased programmability of the control system, thereby allowing configurability of transmission characteristics for different automotive applications.

Transmission control systems operate in close integration with sensors and solenoids to provide a more holistic approach in gear shifting. In a typical control system the sensors register the throttle position, vehicle speed, gear position selection and other different parameters. The information gathered by these sensors are passed to the transmission control module (abbreviated as TCM) which adjusts the current supplied to the transmission solenoids and other specialized controllers to change the effective gear ratio and hence affect vehicle speed.

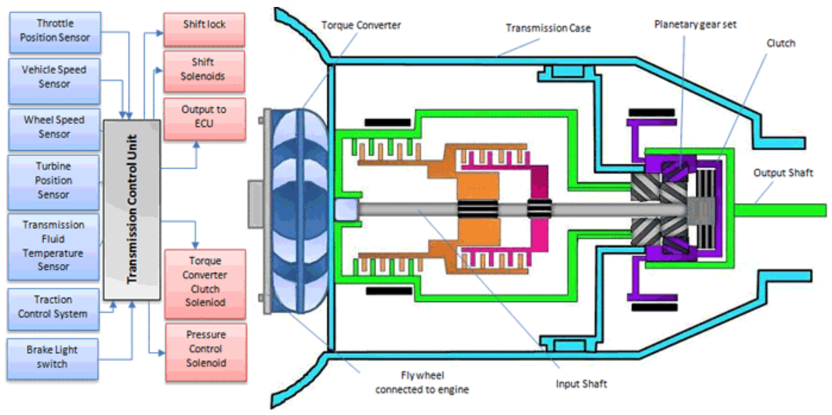


Figure 5: *Sensor-solenoid interfacing via transmission control module [6].*

The specific implementation is intended for a two-speed clutch-less transmission prototype comprising a dual-stage planetary gear set with common ring and common sun gears. Gear change is accomplished by actuating on two friction brakes that control the flow of power, hence the speed of corresponding gears. The friction brakes are actuated by variable force solenoids that can interface with electronic control units through system abstraction.

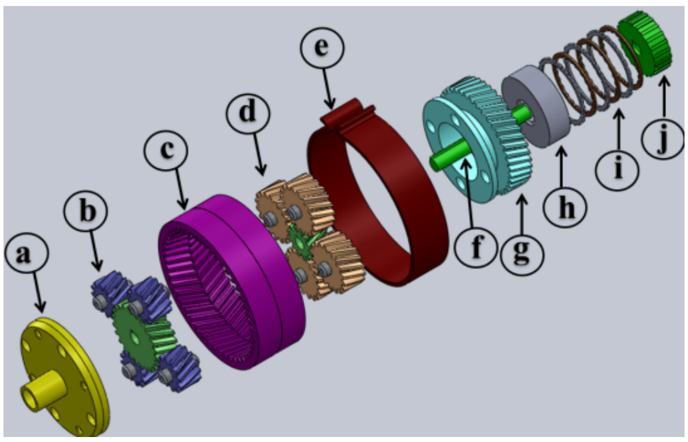


Figure 6: *Exploded view diagram of transmission prototype: a) input carrier b) first stage planetary gear set c) common ring gear d) second stage planetary gear set e) band brake f) common shaft for sun gears g) output carrier h) outer hub for the sun brake i) friction plates j) inner hub for the sun brake [7].*

5.4 Inter-Integrated Circuit

The inter-integrated circuit (abbreviated as I2C) is multi-master, multi-slave serial communication protocol, typically used for interfacing with low-speed peripheral devices over short distance. The

I2C bus consists of two lines: SCL and SDA. SCL is the system clock generated by the current master, and SDA is the data line for conveying actual information. Transmission messages are broken into an address frame and multiple data frames. The address frame is a means for the master to choose the slave of interest. The data frames contain 8-bit data messages passed from the master to slave, and vice versa. A transmission is initiated once the clock line goes low and data is sampled at every rising edge that follows.

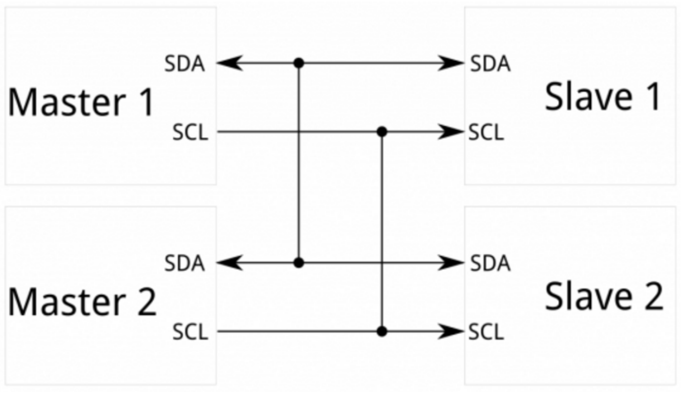


Figure 7: *Typical I2C network topolgy [13].*

The I2C read operation may be condensed into a sequence of instructions as in following:

- Sending a start sequence
- Sending slave address with R/W bit low
- Specifying internal address of the information bearing register
- Sending slave address with R/W bit high
- Reading data bytes
- Stopping sequence

Similarly, I2C write operation considers another sequence of instructions for transmission:

- Sending a start sequence
- Sending slave address with R/W bit low
- Specifying internal address of the register to be written to
- Sending data bytes
- Stopping sequence

Both instances outlined above consider transmission from the master.

5.5 Arduino

Arduino is an open-source platform for microcontroller prototyping. An Arduino board consists of a microcontroller and input/output pins and ports. The board includes a fixed voltage output, which can be used to power external circuits, as well as both analog and digital input and output pins. Analog outputs are acheived using Pulse Width Modulation (PWM) from the digital sources available to the Arduino board [14]. The board is extensible by the use of shields, which connect to the Arduino board by plugging into its power and I/O pins. Shields can provide physically different interfaces, such as a DB9 port that is not a part of the original board, or can integrate other devices, such as a CAN controller and transceiver.

Programming the board is done in the Arduino language, which is mostly comprised of C. Certain C++ features are also supported, but the limited resource run-time environment limits the C++ integration to mostly syntactical elements. Object-oriented programming is not recommended. Development can be done in any text editor, but the Arduino Integrated Development Environment (IDE) provides tools to compile Arduino-specific code and to execute the code on the Arduino board.

The external devices included on Arduino shields can be integrated using vendor-provided libraries, which are included using standard C/C++ practices.

6 Design and Implementation

The overall system design implements a multi-node network based on Arduino boards that communicate with each other via CAN J1939. Besides the microprocessors, the system also constitutes a diverse set of peripheral devices with unique roles, each adopting different protocols for communication with its corresponding Arduino controller. The peripherals include analog and digital sensors for sampling and shield devices for interfacing. The design and implementation of the system components have been accomplished incrementally with iterative testing and validation at each stage, hence requiring minimal revision when progressing through the later development phases.

6.1 CAN J1939

A CAN-BUS V1.2 shield has been directly plugged into the microprocessor boards. This shield adopts a CAN controller and transceiver that provides CAN bus capability. The shield also packages extensive APIs that greatly abstract low level operations required for establishing CAN communication.

Listing 1: CAN setup

```
if(canInitialize(CAN_250KBPS) == CAN_OK)
    Serial.println("CAN Init OK.\n\r\n\r");
else
        Serial.println("CAN Init Failed.\n\r");
```

The code snippet in Listing 1 shows initial CAN configuration steps. The method **canInitialize()** accepts the transmission baud rate specified by the user and configures the node as a CAN node. The method returns the status of the configuration procedure, which is used to verify the and validate the node initialization. Configuration is only done once during the system setup.

Listing 2: CAN transmit

```
byte nPriority = 0;
byte nSrcAddr = 0xff;
byte nDestAddr = 0xaa;
byte nData[8] = {data1, data2, data3, data4, data5,
data6, data7, data8};
int nDataLen = 8;
long lPGN = 72;
byte checker;

checker = j1939Transmit(lPGN, nPriority, nSrcAddr,
nDestAddr, nData, nDataLen);
Serial.println(checker);
```

The code snippet in Listing 2 explicates the steps for transmitting over CANJ1939. The routine **j1939Transmit()** accepts multiple parameters adhering to the J1939 specification. For the verification, dummy values have been used as the parameter group number, the message priority and the source and destination addresses. J1939 specifies its unique set of values for the corresponding parameters which can be replaced in the above function in the event of a real prototyping application. The data buffer is capable of holding bytes, hence any data of larger size must be decomposed into bytes to fit into the transmission buffer. The routine returns a boolean value of true when transmission is successful, hence validating the functionality.

Listing 3: Sensor setup

```
if(j1939Receive(&lPGN, &nPriority, &nSrcAddr, \
&nDestAddr, nData, &nDataLen) == 0){
  sprintf(sString, "PGN: 0x%X Src: 0x%X Dest: 0x%X ",
  (int)lPGN, nSrcAddr, nDestAddr);
  Serial.print(sString);
  if(nDataLen == 0 ){
      Serial.print("No Data.\n\r");
  }
  else{
    Serial.print("Data: ");
    for(int nIndex = 0; nIndex < nDataLen; nIndex++){
      sprintf(sString, "0x%X ", nData[nIndex]);
      Serial.print(sString);

    }
    Serial.print("\n\r");
  }
}
```

The code snippet in Listing 3 outlines the protocol for receiving CAN messages via J1939. Similar to the send routine, the **j1939Receive()** routine accepts multiple parameters as defined in the specification. The routine is put in an overall loop, so that the bus is continually polled for messages that are specifically intended for the recipient node. The received data is also in bytes, hence post-processing of the received data may be required in the event of transmission of more than one byte. The decomposition and recomposing of larger data sizes require predetermined protocols shared by sending and receiving nodes. For the purpose of this project, protocols were established within the team.

6.2 Digital Sensor

A dual digital atmospheric pressure and temperature sensor has been introduced to the CAN network to simulate the role of digital external device nodes in the network. The sensor is an external peripheral device in the context of the Arduino microprocessor and uses the I2C protocol for communication, whereby the Arduino is the master node that moderates the transmission and the sensor is the slave node that sends sampled data at regular intervals. The Arduino packages the extensive **Wire.h** library that has been used to establish the I2C communication.

Listing 4: Sensor setup

```
int Pressure_Init(void){
  int temp;
  int err = 0;

  Pressure_Write(_RES_CONF, 0x78);
  temp = Pressure_Read(_RES_CONF);
  Serial.println(temp);
  if(temp != 0x78){
    err++;
  }

  Pressure_Write(_CTRL_REG1, 0x74);
  temp = Pressure_Read(_CTRL_REG1);
  Serial.println(temp);
  if(temp != 0x74){
    err++;
  }

  Pressure_Write(_CTRL_REG1, 0xF4);
  temp = Pressure_Read(_CTRL_REG1);
  Serial.println(temp);
  if(temp != 0xF4){
    err++;
  }

  temp = Pressure_Read(_WHO_AM_I);
  Serial.println(temp);
  if(temp != 0xBB){
        err++;
  }

  return err;
}
```

The code snippet in Listing 4 demonstrates the initial configuration steps. This routine is intended to be run only once during system boot-up. The master first configures the resolution and the output data rate for the pressure gauge. Following the configuration, a slave reboot operation is carried out for the changes to take effect. As a means of error checking mechanism, several debugger statements have been intermittently placed to ensure only intended changes taking effect. Register read backs have also been implemented following boot-up procedure to check whether pertinent values match as previously specified. This routine references the sensor read and write registers that will be discussed next.

The code snippet in Listing 5 specifies the Arduino implementation of the I2C write protocol. This routine has been referenced in the code snippet in Listing 2 to write to intended control registers. The write protocol involves first starting the I2C transmission by selecting the bus clock as low. Following this write operation is carried out by simply selecting the intended register to be written to and then transmitting the original data. Each slave address encodes a R/W bit that determines the direction of transmission involved. Notably the initial slave register address is required to have the R/W bit set to low, which signifies data transmission from the master to slave.

Listing 5: I2C write

```
void Pressure_Write(int address, int value){
  Wire.beginTransmission(_DEVICE_ADDRESS);
  Wire.write(address);
  Wire.write(value);
  Wire.endTransmission();
}
```

The code snippet in Listing 6 specifies Arduino implementation of the I2C read protocol. Depending on the resolution and the data rate specified during sensor configuration, specialized slave register values are set that are read back periodically by the master to report the sensor reading. The read protocol first involves beginning the I2C transmission by selecting the bus clock as low. The slave register is also selected by writing out the specific address. For the read operation the direction of transmission should be reverted from the slave to the master, hence the R/W bit in the slave register address has been set to high.

Listing 6: I2C read

```
int Pressure_Read(int address)  {
  int tmp;

  Wire.beginTransmission(_DEVICE_ADDRESS);
  Wire.write(address);
  Wire.endTransmission();

  Wire.requestFrom(_DEVICE_ADDRESS, 1, true);
  while(Wire.available()){
    tmp = Wire.read();
  }
  return tmp;
}
```

6.3 Analog Sensor

A force sensor was provided for integration into the network. The force sensor was representative of the specific model that would be used for gathering transmission control system force output.

The force sensor provided was an analog sensor. The sensor operation was based on changing conductivity based on pressure applied to a sensing area, as shown in figure 8.

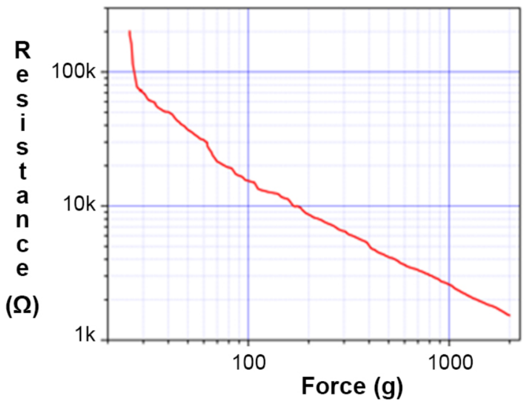


Figure 8: *Resistance of Force Sensing Resistor based on force applied*

A voltage divider was used to create an increasing output voltage as a function of force applied. The output was passed through an NPN transistor buffer to satisfy current requirements of the Arduino analog reading input. The analog to digital converter included in the Arduino analog read input pin converts the voltage read to an integer of scale 0 to 1023, corresponding to input voltage of 0 to 5V. This was then the data transmitted onto the CAN network using the similar logic as before.
The circuitry for this sensor was tested independently of the Arduino logic, using a multimeter. The range of the output voltage was verified for use in the Arduino logic testing.

6.4 Data Logging

The final component introduced to the system network is a data logging module. This functionality has been achieved with the aid of a data logging shield that interfaces with the microprocessor board. The data logging module enables storage of the transmission data into local permanent memory, making the data persistent. The data logging module also extends the functionality of the overall system network such that it can be implemented in applications of vehicle diagnostics and data acquisition. The data logging module used implemented supports using SD card for storage.

Listing 7: I2C read

```
Serial.print("Initializing SD card...");
pinMode(SS, OUTPUT);

if (!SD.begin(chipSelect)){
  Serial.println("Card failed, or not present.");
  while(1);
}
Serial.println("Card initialized.");

dataFile = SD.open("datalog.csv", FILE_WRITE);
if(!dataFile){
  Serial.println("Error opening datalog.txt");
  while(1);
}
```

The code snippet in Listing 7 demonstrates the initialization procedure for the SD card. The routine **SD.begin()** takes as input parameter the slave select line that interfaces with the SD card. The routine returns a boolean value depending on the availability of the card. The shield supports storing data in standard file formats; for the prototyping application in discussion a comma separated vector (abbrev as CSV) file has been used. The routine **SD.open()** accepts the name of the output and creates it in the event of a uniquely named file, or simply opens the file that may be persistent in the storage. The file is opened with write permissions. Error checks are placed at every atomic operations to ensure proper configuration of the settings.

Listing 8: I2C read

```
if(dataFile.println(dataString) == 0){
    Serial.println("Error writing to file!");
}
dataFile.flush();
```

The final code snippet in Listing 8 demonstrates the saving operation at every acquired sample of data. The routine **dataFile.println()** takes as input parameter the desired value to be saved. The routine accepts use of regular expressions, hence supporting multiple text stream editing options before being written to file. The data is continually flushed for ensured persistence, such that in the event of erroneous operation of the microprocessor, the final data is always preserved.

6.5 Graphical User Interface for Central Control Unit

The design of our GUI involves two layers:

- An interface that logs and displays real-time data being obtained by the sensors utilized, as well as a command input interface for control purposes.
- An interface that reads from CAN data frame files on the PC Hard-drive, interprets the frames and displays data obtained from the other controller.

For the first interface, an application was developed using LabVIEW that reads temperature values from sensors, logs them and displays them. A control feature was also developed that uses a thermostat to control temperature by means of outputting a command to set a target temperature by means of either turning a fan or a heater on. This application can be appropriately modified to function for values corresponding to the force sensors as well.

For the Read feature of this application, FPGA I/O Node function has been activated with three properties: Mod1/TC0, MOD1/Autozero, Mod1/CJC. The first is for the actual temperature, second for the autozero channels and third for the cold-junction compensation. The autozero and CJC data in the hot VI is used to convert data read from the thermocouple input channel from binary values to temperature.

For the control by means of a thermostat feature, the target temperature range is input by the target high and low values, and this range is compared to the actual temperature. The result of this comparison determines whether the fan or the heater will be turned on. The GUI explicitly displays this scenario.

A scaled unit host interface application was built to read and convert the actual temperature from un-scaled binary values to the Celsius scale as the engineering unit. Similarly, for the purpose of inputting the target temperature range in standard units, conversion to a binary value before the host VI would write the target temperature range to the FPGA VI had to be performed. For this purpose the NI 9211 Support Files library was used for design as work on this was being done remotely. An element was also added to read and display any errors from the FPGA VI.

The following diagram displays the LabVIEW application developed thus far. It shows the actual temperature observed, the low and high target temperatures input, as well as whether the target temperature forced the fan or the heater to turn on.

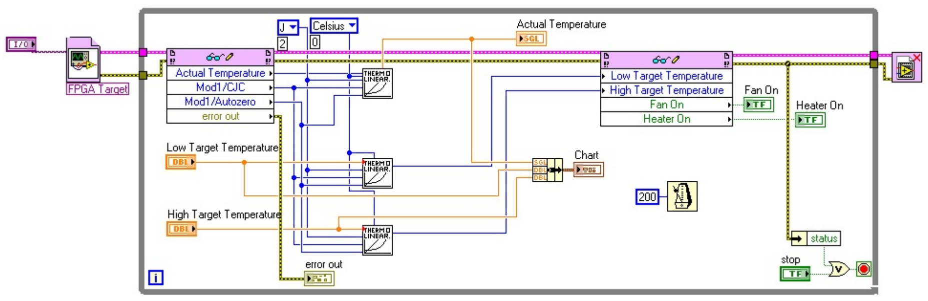


Figure 9: *GUI designed in LabVIEW*

7 Results and Validation

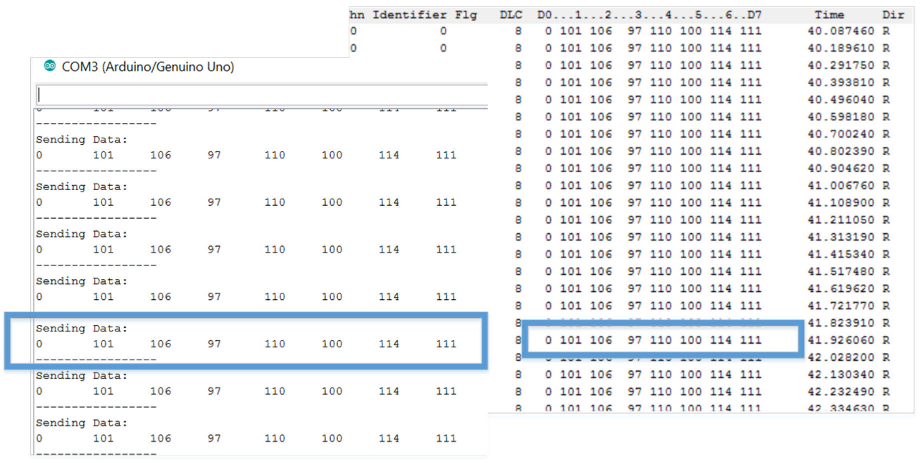
An iterative approach was taken to the system design and implementation. Principles of Agile development processes were used, where a given functionality was developed from design to validation and integration as one contained process, ending with a complete module ready for deployment. Programming was done in a pair, with a high level of collaboration. In this way, the system was built incrementally with little revision needed after implementation.

7.1 Network

Much of the system delivered consists of low-level software modules that interact with the network at the bit level. The transmitting modules process external data, arrange it into bytes within a message frame and send the frame with the CAN controller library provided. The receiving nodes read message frames on the bus, parse the frame to extract the data and display it or provide it to application level processes.

To verify correct transmission, the data encapsulation in a message frame was performed while not connected to the bus. The frame was inspected directly and checked for correctness against a manually created version for the same input. When the message frame was as expected, the transmission itself was inspected using the Kvaser Leaf Light tool, which can read message frames corresponding to many possible network configurations. The data was again compared to the expected value. This tool parses the frame and displays the different components, allowing a direct comparison with the original data.

Reception nodes were tested with a similar procedure. Off the bus, a verified message frame was passed to the node, which parsed it and displayed the original data. If the data was correct, the message frame was then transmitted to the node over the network. The resulting parsed data was verified, as shown in figure 10.



```
-----------------
Sending Data:
0       101     106     97      110     100     114     111
-----------------
Sending Data:
0       101     106     97      110     100     114     111
-----------------
Sending Data:
0       101     106     97      110     100     114     111
-----------------
Sending Data:
0       101     106     97      110     100     114     111

Sending Data:
0       101     106     97      110     100     114     111
-----------------
Sending Data:
0       101     106     97      110     100     114     111
-----------------
Sending Data:
0       101     106     97      110     100     114     111
-----------------
```

```
hn Identifier Flg   DLC  D0...1...2...3...4...5...6..D7       Time     Dir
0           0        8    0 101 106  97 110 100 114 111    40.087460 R
0           0        8    0 101 106  97 110 100 114 111    40.189610 R
                     8    0 101 106  97 110 100 114 111    40.291750 R
                     8    0 101 106  97 110 100 114 111    40.393810 R
                     8    0 101 106  97 110 100 114 111    40.496040 R
                     8    0 101 106  97 110 100 114 111    40.598180 R
                     8    0 101 106  97 110 100 114 111    40.700240 R
                     8    0 101 106  97 110 100 114 111    40.802390 R
                     8    0 101 106  97 110 100 114 111    40.904620 R
                     8    0 101 106  97 110 100 114 111    41.006760 R
                     8    0 101 106  97 110 100 114 111    41.108900 R
                     8    0 101 106  97 110 100 114 111    41.211050 R
                     8    0 101 106  97 110 100 114 111    41.313190 R
                     8    0 101 106  97 110 100 114 111    41.415340 R
                     8    0 101 106  97 110 100 114 111    41.517480 R
                     8    0 101 106  97 110 100 114 111    41.619620 R
                     8    0 101 106  97 110 100 114 111    41.721770 R
                     8                                     41.823910 R
                     8    0 101 106  97 110 100 114 111    41.926060 R
                     8                                     42.028200 R
                     8    0 101 106  97 110 100 114 111    42.130340 R
                     8    0 101 106  97 110 100 114 111    42.232490 R
                     8    0 101 106  97 110 100 114 111    42.334630 R
```

Figure 10: *Verifying Accurate Network Data Parsing*

Once individual nodes were tested, they were connected and a static data input was supplied. At this point, mismatched network configuration specifics were the source of any problems and could be quickly verified. The Kvaser Leaf Light tool was used here as well for diagnosis.

7.2 Testing for Different Hardware Platforms

Given the adherence to the CAN protocol, connectivity with other CAN-compatible nodes based on different hardware platforms would be trivial, consisting of network configuration only. The Kvaser Leaf Light tool provided an initial test of the compatibility.

7.3 Node Extension

Network connectivity was not the only goal for the Arduino nodes. Useful measurements should be made, transmitted, processed and logged on non-volatile memory.

The extra functionality was first designed and implemented using available tools and libraries. Every function was unit tested with typical input and boundary values.

For example, the node responsible for reading the force sensor relied on analog readings from an external circuit. Utility functions were unit tested as usual application software, disregarding the

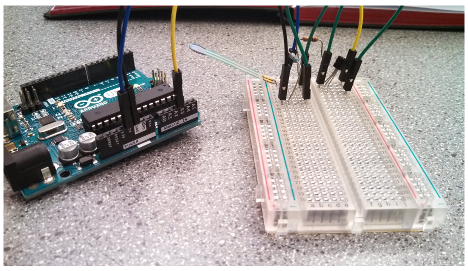


Figure 11: *Arduino Board With Force Sensor on Breadboard*

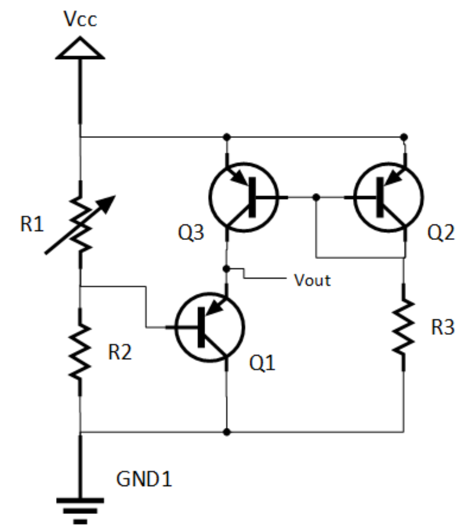


Figure 12: *Final Circuit for Force Sensor*

hardware connectivity. For the reading logic, the maximal and minimal expected values were determined and the node calibrated for this range. Typical values within the range, as well as both boundary values were generated separately and the data-gathering logic was verified. The circuit was designed and tested separately, using different tools, and then connected to the node. This subsystem was verified as a whole.

The node reading the digital sensor was tested in a similar way, with the reading logic replaced by message verification for the sensor-specific protocol.

The data logging module was verified by comparing real-time readings of input data as displayed on the serial monitor available for running Arduino programs with the data logged on the permanent memory.

Once the individual functionality of each node was verified and validated, network connectivity was returned to the node and the integration with other existing modules was tested. Essentially, the new functionality provided a dynamic source of input for the already tested connection, which could be monitored with the Kvaser Leaf Light tool or another receiving node. Following this method, integration tests were performed incrementally as the nodes were developed. Only system tests were required after all the nodes were completed.

8 Impact on Society and Environment

Political and technological developments, namely a requirement to control global emissions and the emergence of new battery designs with improved specific energy, energy density and the ability to recharge have thus far driven a great increase in the viability of electric vehicles to consumers.

As of yet, one of the main advantages of the electric motor has been its torque characteristic which provides maximum torque from zero to low speeds, and then is driven by the maximum power available as the motor speed increases. Significant benefits of this over the traditional torque-speed properties of the competing internal combustion engines include a more desirable characteristic speed of torque over the speed range, and the elimination of a need for any additional transmission clutch or gears.

However, research into energy efficient vehicles has made it necessary to pursue every possible avenue for efficiency gains, which eventually led to the basic conclusion that the overall energy consumption levels improved by up to 12% with the use of a variable ratio gearbox depending on the driving cycle used. In the simplest of terms, replacing a simple reduction gear with a transmission between the wheels allows the motor to operate more efficiently which in turn results in less of a burden on the batteries which results directly in a greater range of the vehicle.

With reference to the figures shown, the comparison shows that the performance scope of the motor can be expanded, with better load resistance performance than that with a single ratio gearbox [18] Improvement in performance characteristics to have been observed thus far include [8]:

- An increase in up to 20% in range of an electric vehicle by posing lesser of a burden on the battery when driving at highway-speeds
- The availability of a low-gear ratio for pull-away and for climbing gradients (especially important for laden commercial vehicles) allows the size, cost and weight of the motor to be reduced which allows for a smaller and lighter cooling system
- A dramatic increase of up to 30% in initial acceleration by means of using a lower-gear ratio for starting which multiplies torque
- An easily noticeable increase in top speed by the motor offering increased power in mid-range RPM
- A huge reduction in heat emissions as the motor moves to a more efficient RPM range in a much faster fashion which in turn reduces inrush current and energy losses in the form of motor heat

So far, the biggest limitation to have been observed in electric vehicles is their limited range on its limited battery supply. As a result of this, many consumers have been hesitant to use them, especially those who need to commute large distances at highway speeds. The advantages above tend to increase the viability of electric vehicles to such consumers.

As far as an increased load on power transmission systems is concerned, at this point, its pretty evident that electric cars are coming into vogue and that at least in the short term, research has shown that there wont be major difficulties absorbing them into the system. However, its also clear that they may have profound implications for the power grid in the longer-term, based on behavioural issues, tariff policies, emerging technology and economics. The development of electric vehicles has been realized as an asset to the grid and grid planning, and steps are being taken to place downward

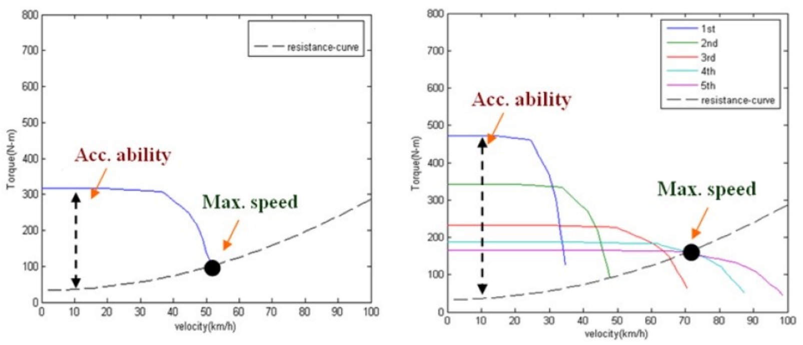


Figure 13: Difference in performance in EV in terms of acceleration and speed between single speed and multi-speed transmission with resistance taken into account

pressure on rates for all electric utility customers. [15]

Another challenge being faced includes the environmental impact of making electric vehicles being greater than that for making gas and diesel vehicles, but this is more than made up for by the greater impact of gas and diesel vehicles during the course of their use. This is true in terms of total energy combustion, use of resources, greenhouse gases, and ozone pollution. Studies have shown that this benefit is obtained even considering the factor of electric vehicles being assumed to be charged from a grid that includes a significant amount of fossil fuels. According to CAA, greenhouse gas emissions are reduced by 67-95% per vehicle (depending on how the electricity used to power the vehicle was generated), along with cheaper energy costs to run three to five times lower than internal combustion engine vehicles and lower overall maintenance [16].

There is no perfect transportation solution. Even electric vehicles have concerns such as higher up front cost for consumers along with variances in battery life and range anxiety. However, electric vehicles, coupled with clean and sustainable electricity, surely are important parts of the solution that meets the challenge of climate change and the increasing dependence on oil.

As shown in the figure below, in 2013, the highest greenhouse gas emissions in Ontario came from the transportation sector [17]. It is because of this reason that automotive manufacturers are taking electric vehicles much more seriously.

The Ontario government has already released its climate change strategy, with the goal to reduce transportation emissions by promoting the uptake of zero emission and plug-in hybrids. Our project goes side-by-side that goal in ensuring electric vehicles appeal to more consumers as without appealing to those who need their vehicles to efficiently commute on highways, these emissions cannot be reduced.

The safety of electric vehicles is obviously critical, for the same reasons as internal combustion engine based vehicles. Any electrical system used in the vehicle must be robust and thoroughly tested. In the case of low level components that interact directly with the vehicle mechanics, this is an absolute requirement. Following industry standards in protocol choice and implementation is a first step in this direction. The thorough testing of all components developed for this project was ensured with

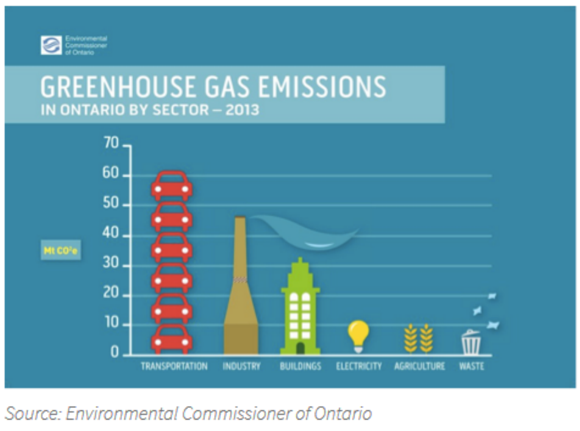


Figure 14: Ontario Emissions

this aim. Despite the fact that all components developed for this project are for prototyping, all designs were checked for possible points of failure to ensure these would not follow into the final versions.

9 Report on Teamwork

During the first semester of this project, the tasks were divided based on individual work. Each team member was assigned subtasks that were independent and could be made parallel. This semester, however, required a different approach. Two team members, Aditya Saha and Alejandro Carboni Jimenez, worked on the Arduino modules one at a time, together and in an iterative process. The type of work assigned required such an approach. Consequently, all work on the Arduino modules and their connectivity was equally shared between these two team members.

During this project, one team member, Shayan Ahmad, was unable to work in the lab space provided to us. Because of this, we attempted to find tasks he could take on remotely. After discussing the issue with our immediate supervisor, she very graciously created separate specifications and evaluated his work independently. He was given a task that he could achieve remotely and independently of the progress made on the Arduino platform work. He was able to submit relevant work and contribute to the group.

Based on our experience during these two semesters, the teamwork was best when tasks where shared and information silos were avoided. Project documentation was most efficient when written with each other in mind, avoiding overspecification and complete generalizations. Team trust was also increased with more task sharing. This more immediate and shared approach would be our first choice for similar projects.

10 Conclusion

This term, the design choices studied in the previous semester were implemented. Arduino microcontrollers were used to simulate Electronic Control Units for automotive industry application. These local control units are compatible with the CAN communication protocol, creating an extensible network of distributed sensors and data loggers which can be paired with a central control unit for holistic decision making and system control.

A lot of our time during this project was spent testing and verifying integration of different external hardware modules and software libraries. Correct integration was not trivial and building the system correctly depended heavily on it. Different CAN output and input modules had different DB9 pinouts, meaning they had to be coupled with individual crisscrossing jumper wires. Diagnosing problems in this type of mismatch required slow, incremental testing of individual components, which led to our iterative, non-parallel approach.

Given more time, direct integration with a CompactRIO-based central control unit would have been attempted. This system would then also be integrated with the existing testbed, which would provide a concrete system to control. Given this system, an Arduino CAN bus node could be developed as its actuator. The overall system could then be tested and optimized, fully implementing the distributed control network.

With more time, the software modules delivered would also be refactored to be more user-friendly and customizable. Configuration would be extracted from the modules, leading to a more cohesive set of modules for later use.

In summary, a model for Electronic Control Units was developed using the Arduino platform and industry standard techniques. The work accomplished during this project is a solid base for future use as a prototyping tool for the design of a transmission control system.

References

[1] Q. Ren, D. Crolla and A. Morris, *"Effect of Transmission Design on Electric Vehicle"*, in Vehicle Power and Propulsion Conference, Dearborn, MI, 2009

[2] Q. L. Y. X. L. G. a. H. C. B. Gao, *"Gear ratio optimization and shift control of 2-speed I-AMT in electric vehicle"*, in Mechanical Systems and Signal Processing, no. vol. 50-51, pp. 615-631, 2015

[3] B. Parikh, *"CAN Protocol - Understanding the Controller Area Network Protocol"*, i[Online]. Available: http://www.engineersgarage.com/article/what-is-controller-area-network

[4] S. Corrigan, *"Introduction to the Controller Area Network (CAN)"*, Texas Instruments, 2008

[5] *"CAN bus", Wikipedia, [Online].Available : https : //en.wikipedia.org/wiki/CAN_bus*

[6] T. C. U. V. E. Laboratory, *"Automotive Electronics - Transmission Control"*, [*Online*].*Available* : *http* : *//www.cvel.clemson.edu/auto/systems/transmission_control.html*

[7] B. B. Mir Saman Rahimi Mousavi, *"Seamless Clutchless Two-Speed Transmission for Electric Vehicles"*, Montreal, Quebec, H3A 0C3, Canada, 2015

[8] inMotive, *"Advantages"*, 2015. [Online]. Available: http://www.inmotive.com/ingear/advantages/. [Accessed 6 December 2015]

[9] BioAge Group, LLC, *"Green Car Congress: Renault"*, Texas Instruments, 2008

[10] S. Corrigan, *"Introduction to the Controller Area Network (CAN)"*, 2015. [Online]. Available: http://www.greencarcongress.com/2013/07/renault-20130711.html. [Accessed 6 December 2015]

[11] R. B. GmbH, *"CAN Specification Version 2.0"*, BOSCH, 1991

[12] *"SAE J1939", Wikipedia, [Online].Available : https : //en.wikipedia.org/wiki/SAE_J*1939

[13] *"I2C", Wikipedia, [Online].Available : https : //learn.sparkfun.com/tutorials/i*2*c*

[14] *"Arduino - Introduction"*, Arduino.cc, 2016. [Online]. Available: https://www.arduino.cc/en/Guide/Introduction.

[15] M. Grunig, M. Witte, B. Boteler, R. Kantamaneni, G. Etienne, D. Bennink, H. van Essen and B. Kampman, *"Assessment of the future electricity sector,"* CE Delft, 2011.

[16] *"Electric Vehicles: Overview"*, Electricvehicles.caa.ca, 2016. [Online]. Available: http://electricvehicles.caa.ca/electric-vehicles-overview/.

[17] L. O'Malley, *"Electric vehicle innovations: A market snapshot - MaRS"*, MaRS, 2016. [Online]. Available: https://www.marsdd.com/news-and-insights/electric-vehicle-innovations-a-market-snapshot/.

[18] Chih-Ming C, Jheng-Cin S, *"Performance Analysis of EV Powertrain system with/without transmission"*, World Electric Vehicle Journal Vol. 4, 2011.