A document to explain better what the Arduino code in our car is for, since comments can often be hard to read.

**Temp\_Sense\_Slave**

This code is the code for the Arduinos in the battery boxes that are not connected to the central mega. It communicates with the temperature sensors via I2C. The sensors only have six addresses (72-77), and since each Arduino only has one I2C channel, each Arduino can only communicate with six sensors at a time. It sends the data it gathers to an intermediate Arduino via UART serial.

Part-by-Part

Part 1

Here, the variables for the program are defined.

sensorReadings is where the data from the sensors is going to be stored.

variablePlaceWriteTo specifies where in “sensorReadings” to write data to.

readFailure increments each time the Arduino fails to communicate with the sensors in a row

i is used for the various “for” loops

highestTemp holds the highest temperature recorded that is to be sent to the intermediate Arduino.

commFailure indicates if the Arduino fails to communicate with the sensors during the current run of the loop

I include the library for I2C, and in “setup” I prepare the Arduino for I2C and UART serial communication. I will implement a small delay in the code in one arduino to de-synchronize the two lesser Arduinos so that they don’t send data to the intermediate at the same time.

Part 2

The loop begins. I call a small function to reset “sensorReadings”, “variablePlaceWriteTo”, and “commFailure”. Then, I begin a “for” loop to gather data from the sensors. I begin the loop by clearing the I2C buffer, so only new data is gathered. Then, I request two bytes of data from the sensor with the specified address. I put the data given to me in the “sensorReadings” variable, in the spot indicated by “variablePlaceWriteTo”, and incrementing this variable afterward. 12 bytes of data total are given to me.

Part 3

The “variablePlaceWriteTo” is incremented each time I recieve data, so it will be 12 after receiving al of the data. If it is not 12, that means I have not received all of the data. If this occurs, I turn the “commFailure” variable to TRUE, and increment the readFailure variable. If the “variablePlaceWriteTo” is equal to 12, then a possible streak of non-communication has been broken, so I set “readFailure” back to 0.

Part 4

If communicaton has not failed, the “commFailure” variable will be true. If this is the case, I have all the data I need, and I need to extract the data from the bytes I received. The two least significant bits of the temperature values are the two most significant bits of the second byte given to me by each sensor. I implement some logic to extract these final bits to get the true sensor readings.

Part 5

I only need to send the highest temperature reading to the main Arduino, so I find the maximum temperature from all the temperatures I have and send it via UART to the intermediate Arduino. I implement a delay so that I am not constantly throwing data at the intermediate.

If communication with the sensors had failed five times in a row, I send the value 255 to the intermediate Arduino, to be recognized as an error.

**Temp\_Sense\_Intermediate**

This Arduino not only gets temperature values from temperature sensors, it also receives values from two other arduinos via serial and then sends the maximum of these values to the main Mega via Serial.

Part 1

I define the variables for the code.

sensorReadings holds the raw bytes received from the temperature sensors via I2C

variablePlaceWriteTo specifies where in “sensorReadings” to write data to.

readFailure increments each time the Arduino fails to communicate with the temperature sensors in a row

i is used for the various “for” loops

highestTemp holds the highest temperature recorded that is to be sent to the central Arduino.

commFailure indicates if the Arduino fails to communicate with the sensors during the current run of the loop

timenotGottenSerial1/timenotGottenSerial2 indicates the last time that the Arduino received data from slave ½

serial1Absence/serial2absence indicates how long (in ms) it has been since the arduino has received data from slave ½

notGottenSerial1/notGottenSerial2 indicates wether or not the srduino has gotten serial data from slave 1/2 on this loop.

I then include the libraries for I2C and Software Serial, so that the Arduino can have more than one set of rx-tx pins. I set which pins are going to be the rx and tx pins for the software serial ports.

Part 2

I begin the loop. resetVars resets “highestTemp” , “commFailure”, and “vairablePlaceWriteTo” variables. The function exact\_same\_as\_temp\_sense\_slave does the same function as the TempSenseSlave program, communicating with the temperature sensors. Among the differences is that, if communication fails 5 times with the sensors, commFailure is declared false, so that parts 3 and 4 of the program run and give a temperature value of 255 to the central Arduino.

Part 3

This part only runs if there has not been a communication failure with the temperature sensors, or if the temperature sensors have failed to communicate more than 5 times. If there is serial data ready from slave 1, that value is read and given to “highTemp1”, and the “notGottenSerial1” flag is set to false. If there is not data ready from slave 1 and “notGottenSerial1” is false, the “timenotGottenSerial” is set equal to the number of milliseconds that the program has been running so far, and the “notGottenSerial1” flag is set to true. If there is no serial data and “notGottenSerial1” is true, “serial1Absence” is set to the total number of milliseconds minus the number of milliseconds that had passed since an absence in serial data from slave 1 had been realized. Essentially, it records how long it has been since it last got data from slave 1. An identical process occurs for slave 2.

Part 4

This part only runs as well if there has not been a communication failure with the temperature sensors, or if the temperature sensors have failed to communicate more than 5 times. If the Arduino has not received data from either slave for more than 1000 milliseconds, “highestTemp” is set to 255. highestTemp is then set to the maximum of the value received from the temperature sensors and the values received from slaves 1 and 2. This value is then sent to the main Arduino.

**Dashboard\_Arduino**

This Arduino is going to be a Mega. It has several tasks- read the pedal potentiometers, receive statuses from the main Arduino, and display information on the dashboard LCD and LEDs. I really want to connect this to the main Arduino via I2C, but it might have to be UART :/ oh well… enough sadness. On to the code!