Code walkthrough by Ethan Park (4/13/18)

Notes written by Kirby Julian and Akash Borde

Includes

* CAN library
* timer
* Arduino library
* Not sure if we use steering (was at some point)
  + Definitions for pin numbers for steerings

Debug

* Basically always set to true
* Enables serial prints
* Prints to the serial monitor
* Serial baud
  + Maximum rate possible
  + Baud rate for serial monitor (arduino display)
  + Could be arbitrary

Pins

* Ignition switch and others
* They are literally pin numbers in the arduino
* Hardwired because we made a PCB
* Most of them are legacy from SC6 and previous cars
* The only ones that still matter are
  + BMS\_STROBE\_PIN
  + IGNITION\_PIN
  + (INTERRUPT\_PIN?)
  + (CS\_PIN) - chip-select? Maybe in use
  + Not BOARD\_LED

CAN parameters

* CAN BAUD rate
  + Been fucking around with it
  + 250 was for testing
  + 500 is what the motor controllers can accept so we will probably bump it back up
* RX then other numbers and letters
  + Parameters that will be used as inputs when the CAN object is initiated
  + CAN packet IDs
  + All defined in Can library
  + Masks says

Timer Intervals (in ms)

* Lots of em
* The numbers are the period of sending out the specified CAN packet
* Each one corresponds to a different important thing in the while loop of the arduino
* This allows us to check for certain things in the while loop and not check every single thing during every single loop of the while loop

Driver Parameters

* Constants
* Some are used for
* Charge and discharge temperatures are already implemented
* Still relevant ones:
  + None of the pedal stuff
  + None of the accel/velocity stuff
  + Maybe the velocity conversion constants could be useful if we’re printing to the serial monitor while the car is moving

Steering wheel parameters

* Probably useless since the steering wheel doesn’t send things out anymore

BMS parameters

* Still relevant!
* These are important values from the battery pack that could tell the car that the batteries are in a dangerous state
  + Too much current

Driver control errors

* BMS\_TIMEOUT still relevant
* Perhaps Motor Controller timeout is still relevant
* These are all within the driver controls code (the 0x01, 0x02, 0x04…) are all defined later

Temp Sensor Object

* There’s some sort of object we’re setting up. Might correspond
* The 50 in the OneWire ds(50) corresponds to Pin 50
* Dont fuck with

Enums

* Gives possible values that different states can hold
* Example
  + Gearstate can be in:
    - Forward
    - reverse
    - …
* They each have values, which you will see on the monitors

Struct CarState

* A data structure that holds the date of a car
* Outlines what is in the state of the car
* No functions
* Resort to comments to explain all the variables

Driver control status

* Numbers that could be in the CAN packet

Temperature

* Max\_temp is the most important thing

CAN variables

* CAN\_IO
  + Creates the CAN io object which has a bunch of things

Debugging variables

* Experimental

Temp sense

* Useful things
* In theory if the new wires are plugged in everything should work

Helper functions (majority of code)

* Readinputs
  + Reads inputs
  + Brake section
    - This is for brake lighting
  + Accel and regen
    - We might need it tbd
  + Ignition switch
    - Turns the car on
  + Steering wheel controls
    - Won’t be necessary
  + Lights
    - No software involved
  + Horn
    - No software needed
* ReadCAn
  + Reads CAN
  + Safety count
    - Number of CAN packets we can physically handle per loop
  + Frame
    - Basically a packet
  + All the if statements and elseif statements pretty much just read the CAN id (read from frame) and creates an object of the class that represents the specific packet
  + Get rid of all steering wheel data because it is now mechanical
    - And don’t tell anyone about it
* Checktimers
  + Checks metro (heartbeat) timers
* Updatestate
  + Redefines different parts of CarState
  + Called continuously to keep updating the state of the car
* Trip conditions
  + They will freak out if different conditions are met
  + They provide warnings, information, and perform actions to respond to inputs
* regen Current (probably not necessary)
* Current Buffer is an array that stores 10 values in it
  + Useful for checking if we’ve crossed the current limit more than the allowed 7 times
* Write outputs
  + Checks state, and if it m
  + digitalWrite(HORN\_PIN, state.horn ? HIGH : LOW)
    - Means check if state.horn is HIGH and if so, the HORN\_PIN is set to LOW
* writeCAn
  + canControl.Send(DC\_Drive(num1, num2), TXBANY)
    - Sending an object of class DC\_DRIVE()
    - And TXBANY is a buffer you can send it with
  + Checks conditions and then sends a CAN packet
  + timer.expired() is very useful for activating certain code at the necessary time interval
  + Motor control request packets are new for SC7
  + They work!
* checkErrors
  + Checks errors
* ReadTempSensor
  + 26 temp sensors
  + All chips are the same but each have their own address
  + The program needs to get all the addresses
  + Instead of hardcoding all the addresses in the program, the driver controls scans everything
  + The protocol then fills all the addresses
  + tempCount //refers to which address you’re on
  + All the temperature conversion data is done on the temperature control bits
  + Can select what resolution of temperature you want (resolution is proportional to the time it takes)

The rest

* The actual code that does the things
* Loop
  + <https://www.youtube.com/watch?v=9x8NcsTXI6s>
* 100s of serial prints in a big ass switch

DONE