7166CEM: Automotive Software Engineering

Development & Testing

Coursework-Individual Report

Prathmesh A. Gardi
Student Id: 13355397
Email : gardip@uni.coventry.ac.uk

Abstract :

This is the report on the development of saftey controller software for an automomous racecar platform.This report contains the information about the procedures and methods used during the developement of the software.

The developed software reads the raw data signals with information about the input parameters such as Speed, Torque and Battery Voltages and using those inputs generates the current level values that need to be transmitted to the motor controller.

The report consist of the points such as:

Code written and its scope
Development tools used
Software information and Specifications
Guidlines and Code standards followed
Unit testing and its results
Limitations of the software in the current version and proposed solutions

Scope, Specifications and Requirments of the Software:

The scope of this coursework is

- To read the raw values for the Torque, Speed, Voltages from the CAN bus.
- To deocde those values and then use those values to Calculate the current values as per the follwoing formula:

c = (t/v) * (1 + (q * q * (t/100)) + (r * (b/100) * t)) and v != 0
Where,
c = Current in milli Ampere
v = Battery Voltage in Volts
t = Requested Torque in Nm
r = Wheel Speed in rpm
q = Motor Q = 0.879
b = Back Elelctromagnetic Field = 0.6721

- And then we have to transmitt these values over the CAN bus.

The code for these above steps is written in the file **get_data.cpp.**

Input values as CAN signal can be sent using the command ‘cansend’ in raw format.
OR
There is another code as **send_data.cpp** written in order to take the input values from user and then convert and pack those values in the CAN frame and transmit on the CAN bus automatically.

Before Calculating the Current value, the input values need to fullfill the few requirements,

The Maximum allowed torque needs to be limited as per following conditions:

Fastest Wheel RPM	**Max Allowed Torque(Nm)**
Greater than 700	50
Greater than 600	85
Greater than 500	100
Greater than 400	120
Greater than 300	150
Less Than 300	32767

If the Battery Voltage is less than 2.8 Volts, Maximum allowed torque should be 20 Nm.

Development stages and tools used:

- **Code Editor** :
  Microsoft Visual Studio (Version : 1.74.3 (Linux _Ubuntu))

- **Programming Language** :
  C++ (Version : Ubuntu 11.3.0)

- **Build Tool** :
  Cmake (Version : 3.22.1)

- **Linting Tools :**
  Clang – format (Version :14.0.0-1ubuntu1)
  Clang – tidy (Version : Ubuntu LLVM version 14.0.0)

- **Version Control :**
  git (Version : 2.34.1)

- **Unit testing Framework:**
  Catch2 (Version : 3.2.1)

- **Code Standards Used:**
  JPL Power of 10

- **Git repository for the project:**
  https://github.coventry.ac.uk/gardip/7166CEM-Coursework_Prathmesh_Gardi.git

Code Standard and Exception to them in the code:

While the development procedure, the 'JPL Power 10' standards were refered. There are total 10 rules that are required to be followed, which are as following:

Rule No.1:
Restrict all code to very simple control flow constructs – do not use goto statements, setjmp or longjmp constructs, and direct or indirect recursion.
(Gerard J. Holzmann,The Power of Ten – Rules for Developing Safety Critical Code)

Rule No. 2:
All loops must have a fixed upper-bound. It must be trivially possible for a checking tool to prove statically that a preset upper-bound on the number of iterations of a loop cannot be exceeded. If the loop-bound cannot be proven statically, the rule is considered violated.
(Gerard J. Holzmann,The Power of Ten – Rules for Developing Safety Critical Code)

Rule No.3:
Do not use dynamic memory allocation after initialization.
(Gerard J. Holzmann,The Power of Ten – Rules for Developing Safety Critical Code)

Rule No.4:
No function should be longer than what can be printed on a single sheet of paper in a standard reference format with one line per statement and one line per declaration. Typically, this means no more than about 60 lines of code per function.
(Gerard J. Holzmann,The Power of Ten – Rules for Developing Safety Critical Code)

Rule No.5:
The assertion density of the code should average to a minimum of two assertions per function. Assertions are used to check for anomalous conditions that should never happen in real-life executions. Assertions must always be side-effect free and should be defined as Boolean tests. When an assertion fails, an explicit recovery action must be taken, e.g., by returning an error condition to the caller of the function that executes the failing assertion. Any assertion for which a static checking tool can prove that it can never fail or never hold violates this rule. (I.e., it is not possible to satisfy the rule by adding unhelpful "assert(true)" statements.)
(Gerard J. Holzmann,The Power of Ten – Rules for Developing Safety Critical Code)

Rule No.6:
Data objects must be declared at the smallest possible level of scope.
(Gerard J. Holzmann,The Power of Ten – Rules for Developing Safety Critical Code)

Rule No.7:
The return value of non-void functions must be checked by each calling function, and the validity of parameters must be checked inside each function.
(Gerard J. Holzmann,The Power of Ten – Rules for Developing Safety Critical Code)

Rule No.8:
The use of the preprocessor must be limited to the inclusion of header files and simple macro definitions. Token pasting, variable argument lists (ellipses), and recursive macro calls are not

allowed. All macros must expand into complete syntactic units. The use of conditional compilation directives is often also dubious, but cannot always be avoided. This means that there should rarely be justification for more than one or two conditional compilation directives even in large software development efforts, beyond the standard boilerplate that avoids multiple inclusion of the same header file. Each such use should be flagged by a tool-based checker and justified in the code.
(Gerard J. Holzmann,The Power of Ten – Rules for Developing Safety Critical Code)

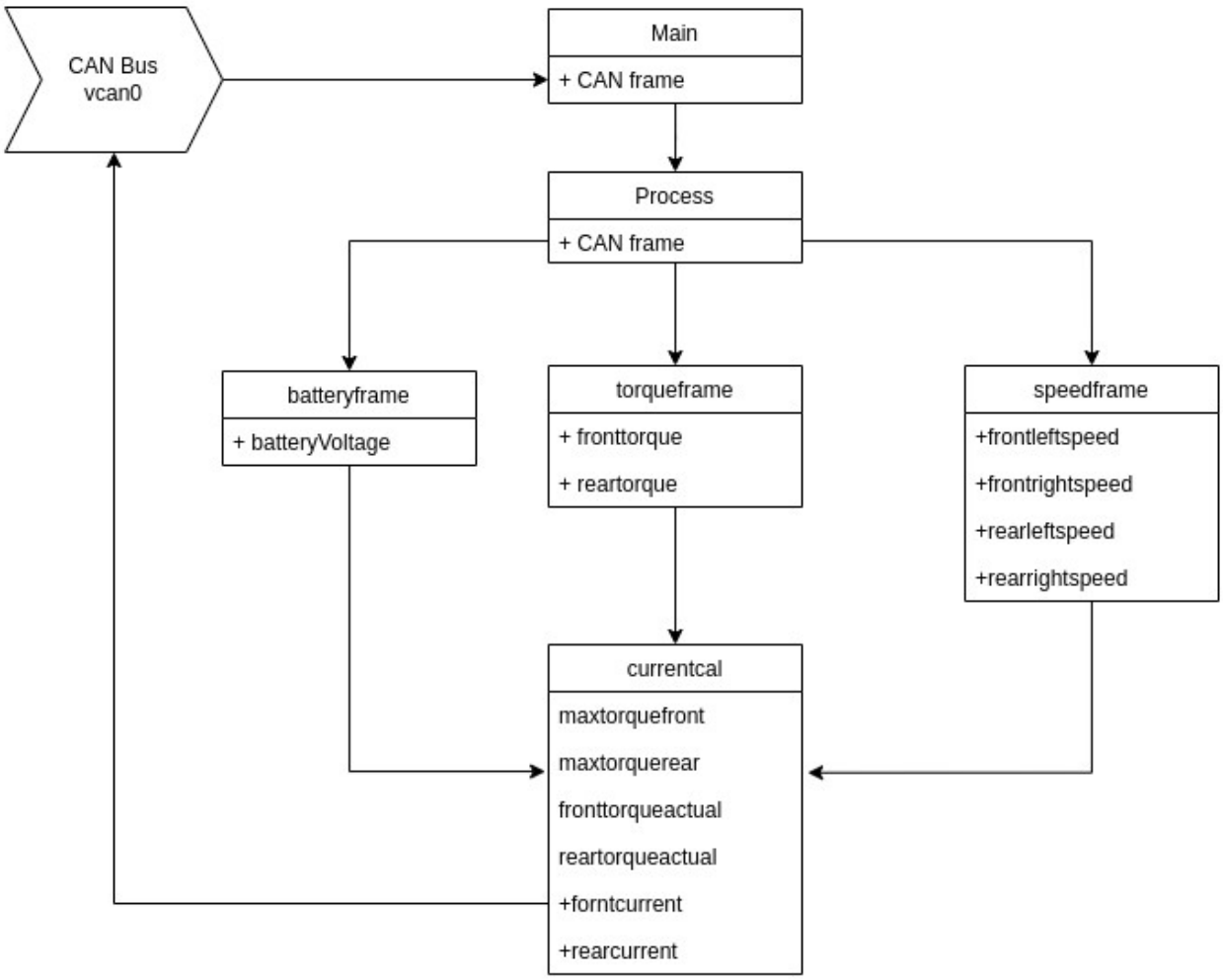
Rule No. 9 :
The use of pointers should be restricted. Specifically, no more than one level of dereferencing is allowed. Pointer dereference operations may not be hidden in macro definitions or inside typedef declarations. Function pointers are not permitted.
(Gerard J. Holzmann,The Power of Ten – Rules for Developing Safety Critical Code)

Rule No. 10:
All code must be compiled, from the first day of development, with all compiler warnings enabled at the compiler's most pedantic setting. All code must compile with these setting without any warnings. All code must be checked daily with at least one, but preferably more than one, state-of-the-art static source code analyzer and should pass the analyses with zero warnings.
(Gerard J. Holzmann,The Power of Ten – Rules for Developing Safety Critical Code)

Throughout the code at most of the occasions the rules have been adhered to, but there are few exception to the rules in the code. These instances are as follows:

No.	Rule Breahed	File and location	Reason	Documented in the code?
1.	Rule No.2	src/ get_data.cpp/ line:276	At this specific instnace the while loop that will run forever is used. The code inside loop reads the input values and calculates the current and then transmit it to CAN Bus. As the signal transmitted is used by motor controller to control the motors, we need to keep updating and transmitting the value of current as long as the vehicle is running , due to this the loop needs to run forever. Due to that the upper bound for the loop is not provided.	Yes
2.	Rule No.6	src/get_data.cpp	There are few variables that have been declared globally. As group of these variable are defined in single function ;so to return or access those variable we would have needed to create a structure or array and there pointers , which would have increased the complexity of the code and also that would have breached the Rule No. 9.	Yes

Data Flowchart for the Code:



Unit Testing and Results

During the unit testing we tested all the requirements relavant to the problem state to ensure that the code is working as per the expectations.
For the unit testing Catch2 framework has been used.
The details of the test cases and the results are as below:

1. Test on the Battery voltage :

Requirement: The software should be able to decode the raw value of the battery voltage to the decimal value. And if the Voltage is out of range it should throw an error signal to let the user know.

Test Case 1	Battery Voltage is within Range
Test Case Variant 1	Applying the Middle Value
Inputs	CAN frame: 0x0007
Expected Output	Batteryvoltage == 7 Return 0
Actuall Output	Batteryvoltage == 7 Return 0
Result	**Pass**

Test Case 1	Battery Voltage is within Range
Test Case Variant 2	Applying the Lower limit value
Inputs	CAN frame: 0x0000
Expected Output	Batteryvoltage == 0 Return 0
Actuall Output	Batteryvoltage == 0 Return 0
Result	**Pass**

Test Case 1	Battery Voltage is within Range
Test Case Variant 3	Applying the Upper limit value
Inputs	CAN frame: 0x000D
Expected Output	Batteryvoltage == 13 Return 0
Actuall Output	Batteryvoltage == 13 Return 0
Result	**Pass**

Test Case 2	Battery Voltage out of the Range
Inputs	CAN frame: 0x0A0A
Expected Output	Batteryvoltage == 2570 Return 1
Actuall Output	Batteryvoltage == 2570 Return 1
Result	**Pass**

2. Tests on the Wheel speeds

Requirement: The software should be able to decode the raw value of the wheel speeds to the decimal value. And if any of the wheels speed is out of the range specified in the DBC , throw an error to let the user know about it.

Note: There is no test case for the wheel speeds going out of range, beccasuse for sending the value of speed of each wheel there are total 16 bits given. So the highest possible value that can be stored in 16 bits is 65535. This is same as the upper limit of the value of the wheel speed. So its practically not possible to transmit the values greater than the upper limit.

Test Case 3	Wheel Speed is within the range
Test Case Variant 1	Applying the Mid value
Inputs	CAN frame: 0x01F401F401F401F4
Expected Output	frontleftspeed == 500 frontrightspeed == 500 rearleftspeed == 500 Rearrightspeed == 500 Return 0
Actuall Output	frontleftspeed == 500 frontrightspeed == 500 rearleftspeed == 500 Rearrightspeed == 500 Return 0
Result	**Pass**

Test Case 3	Wheel Speed is within the range
Test Case Variant 2	Applying the Lower limit value
Inputs	CAN frame: 0x00000000
Expected Output	frontleftspeed == 0 frontrightspeed == 0 rearleftspeed == 0 Rearrightspeed == 0 Return 0
Actuall Output	frontleftspeed == 0 frontrightspeed == 0 rearleftspeed == 0 Rearrightspeed == 0 Return 0
Result	**Pass**

Test Case 3	Wheel Speed is within the range
Test Case Variant 2	Applying the Upper limit value
Inputs	CAN frame: 0xFFFFFFFF
	frontleftspeed == 65535
	frontrightspeed == 65535
Expected Output	rearleftspeed == 65535
	Rearrightspeed == 65535
	Return 0
	frontleftspeed == 65535
	frontrightspeed == 65535
Actuall Output	rearleftspeed == 65535
	Rearrightspeed == 65535
	Return 0
Result	**Pass**

3. Test cases for the Requested Torque:

Requirement: The software should be able to decode the raw value of the requested torques to the decimal value. And if any of the requested torque is is out of the range specified in the DBC , throw an error to let the user know about it.

Test Case 4	Requested torrque value is within the range
Test Case Variant 1	Applying the Mid value
Inputs	CAN frame: 0x006400000064
Expected Output	fronttorque == 100
	Reartorque == 100
	Return 0
Actuall Output	fronttorque == 100
	Reartorque == 100
	Return 0
Result	**Pass**

Test Case 4	Requested torrque value is within the range
Test Case Variant 1	Applying the Upper limit value
Inputs	CAN frame: 0x7FFF00007FFF
Expected Output	fronttorque == 32767
	Reartorque == 32767
	Return 0
Actuall Output	fronttorque == 32767
	Reartorque == 32767
	Return 0
Result	**Pass**

Test Case 5	Requested torrque value is within the range
Test Case Variant 1	Front Torque is Greater than Upper Limit
Inputs	CAN frame: 0x9C4000007FFF
	fronttorque == 40000
Expected Output	Reartorque == 32767
	Return 1
	fronttorque == 40000
Actuall Output	Reartorque == 32767
	Return 1
Result	**Pass**

Test Case 5	Requested torrque value is within the range
Test Case Variant 2	Rear Torque is Greater than Upper Limit
Inputs	CAN frame: 0x7FFF00008000
Expected Output	fronttorque == 32767
	Reartorque == 32768
	Return 1
Actuall Output	fronttorque == 32767
	Reartorque == 32768
	Return 1
Result	**Pass**

4. Test cases for the determination of the Maximum Allowed Torque

Requirement: Software should determine the Maximum Allowed Torque based on the Wheel speeds as follow:

Fastest Wheel RPM	**Max Allowed Torque(Nm)**
Greater than 700	50
Greater than 600	85
Greater than 500	100
Greater than 400	120
Greater than 300	150
Less Than 300	32767

Test Case 6	Checking the Maximum torque for the Different wheel speed values
Test Case Variant 1	Checking the value for Wheelspeeds Greater than 700
Inputs	frontleftspeed = 800 frontrightspeed = 650 rearleftspeed = 750 Rearrightspeed = 600
Expected Output	Maxfronttorque == 50 Maxreartorque == 50
Actuall Output	Maxfronttorque == 50 Maxreartorque == 50
Result	**Pass**

Test Case 6	Checking the Maximum torque for the Different wheel speed values
Test Case Variant 2	Checking the value for Wheelspeeds Greater than 600
Inputs	frontleftspeed = 500 frontrightspeed = 650 rearleftspeed = 550 Rearrightspeed = 640
Expected Output	Maxfronttorque == 85 Maxreartorque ==85
Actuall Output	Maxfronttorque == 85 Maxreartorque == 85
Result	**Pass**

Test Case 6	Checking the Maximum torque for the Different wheel speed values
Test Case Variant 3	Checking the value for Wheelspeeds Greater than 500
Inputs	frontleftspeed = 500 frontrightspeed = 450 rearleftspeed = 580 Rearrightspeed = 440
Expected Output	Maxfronttorque == 100 Maxreartorque == 100
Actuall Output	Maxfronttorque == 100 Maxreartorque == 100
Result	**Pass**

Test Case 6	Checking the Maximum torque for the Different wheel speed values
Test Case Variant 4	Checking the value for Wheelspeeds Greater than 400
Inputs	frontleftspeed = 350 frontrightspeed = 450 rearleftspeed = 380 Rearrightspeed = 440
Expected Output	Maxfronttorque == 120 Maxreartorque == 120
Actuall Output	Maxfronttorque == 120 Maxreartorque == 120
Result	**Pass**

Test Case 6	Checking the Maximum torque for the Different wheel speed values
Test Case Variant 5	Checking the value for Wheelspeeds Greater than 300
Inputs	frontleftspeed = 350 frontrightspeed = 250 rearleftspeed = 380 Rearrightspeed = 240
Expected Output	Maxfronttorque == 150 Maxreartorque == 150
Actuall Output	Maxfronttorque == 150 Maxreartorque == 150
Result	**Pass**

Test Case 6	Checking the Maximum torque for the Different wheel speed values
Test Case Variant 6	Checking the value for Wheelspeeds Less than 300
Inputs	frontleftspeed = 150 frontrightspeed = 250 Rearleftspeed = 180 Rearrightspeed = 240
Expected Output	Maxfronttorque == 32767 Maxreartorque == 32767
Actuall Output	Maxfronttorque == 32767 Maxreartorque == 32767
Result	**Pass**

Limitations of the Software and Solution

Limitation:
There are few limitations to this software that were noticed during the development.
When we read the data from the CAN bus for the torque, there is no way to know if the transmitted value is positive or negative one.
This leads to error duirng the decoding. As while transmitting the negative values they are transmitted as the 2s complement of the value. So when we decode the value while calculating the current, it gives the wrong result. We can add provision to take a reverse 2s complement while decoding, bit this will produce wrong values case the positive value is received.

Solution:
Solution to this problem would be to include another one byte signal in the DBC which will indicate the polarity of the signal.
The depending upon the value of that signal , we can write a code to either peform normal decoding or do the reversal of the 2’s complement first.

Apendix A : Source Code:

GIT Directory for the Source code:
https://github.coventry.ac.uk/gardip/7166CEM-Coursework_Prathmesh_Gardi.git

File : src/get_data.cpp

```
#include <memory> // for allocator, shared_ptr, __shared_ptr_access
#include <string> // for char_traits, operator+, to_string

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include <chrono>
#include <iomanip>
#include <iostream>
#include <string>
#include <thread>

#include "can_wrap.hpp"
#include "candata.h"
using can::operator<<;

const std::string canChannel = "vcan0";
const int canSocket = can::connect(canChannel);

uint16_t batteryvoltage;  // battery voltage
uint16_t frontleftspeed;  // speed of the front left wheel
uint16_t frontrightspeed; // speed of the front right wheel
uint16_t rearleftspeed;   // speed of the rear left wheel
uint16_t rearrightspeed;  // speed of the rear right wheel
uint16_t fronttorque;     // requested torque for the front wheels
uint16_t reartorque;      // requested torque for the rear wheels
_Float32 frontcurrent;    // calculated current for the front wheels
_Float32 rearcurrent;     // calculated current for the rear wheels
uint16_t maxtorquefront;  // maximum allowed torque for the front wheels
uint16_t maxtorquerear;   // maximum allowed torque for the rear wheels

/*Autogenerated Code from the DBC file
Packs the given value into CAN frame ,
@param value:Variable that stores the value to be packed in the frame
@param shift: Number of bytes to be shifted by
@param mask: Bit mask thats ANDed with the input*/
static inline uint8_t pack_left_shift_u16(uint16_t value, uint8_t shift,
                                          uint8_t mask) {
  return (uint8_t)((uint8_t)(value << shift) & mask);
}
```

```
/*Autogenerated Code from the DBC file
Packs the given value into CAN frame ,
@param value:Variable that stores the value to be packed in the frame
@param shift: Number of bytes to be shifted by
@param mask: Bit mask thats ANDed with the input*/
static inline uint8_t pack_right_shift_u16(uint16_t value, uint8_t shift,
                                           uint8_t mask) {
  return (uint8_t)((uint8_t)(value >> shift) & mask);
}

/*Autogenerated Code from DBC file
Takes data frame as a input and deocode it from HEX to Decimal value */
static inline uint16_t unpack_left_shift_u16(uint8_t value, uint8_t
shift,
                                             uint8_t mask) {
  return (uint16_t)((uint16_t)(value & mask) << shift);
}

/*Autogenerated Code from DBC file
Takes data frame as a input and deocode it from HEX to Decimal value */
static inline uint16_t unpack_right_shift_u16(uint8_t value, uint8_t
shift,
                                              uint8_t mask) {
  return (uint16_t)((uint16_t)(value & mask) >> shift);
}

/* Takes two parameters
@param a,b - inputs
@return - Max of the two inputs*/
uint16_t maxof(int16_t a, int16_t b) {
  if (a > b)
    return a;
  else
    return b;
}

/* Takes two parameters
@param a,b - inputs
@return - Min of the two inputs*/
uint16_t minof(int16_t a, int16_t b) {
  if (a < b)
    return a;
  else
    return b;
}

/*Reconstructs the value of the battery voltage from the CAN frame
Also checks whether the value of the parameter is within the limits from
the DBC
file
@param frame : input can frame read from the CAN channel
```

```
@param batteryvoltage : stores the decodes value of battery voltage
*/
uint16_t batteryframe(const can_frame frame) {
  batteryvoltage = static_cast<uint16_t>(frame.data[0] << 8) +
                   static_cast<uint16_t>(frame.data[1]);

  if (batteryvoltage > 13) {
    std::cout << "Battery Voltage value Out of Range" << std::endl;
    return 1;
  }

  return 0;
}

/*Reconstructs the value of the wheel speeds from the CAN frame
Also checks whether the value of the parameter is within the limits from
the DBC
file
@param frame : input can frame read from the CAN channel
@param frontleftspeed : stores the decodes value of speed for the front
left
wheel
@param frontrightspeed : stores the decodes value of speed for the front
right
wheel
@param rearleftspeed : stores the decodes value of speed for the rear
left wheel
@param rearrightspeed : stores the decodes value of speed for the rear
right
wheel
*/
uint16_t speedframe(const can_frame frame) {
  frontleftspeed = static_cast<uint16_t>(frame.data[0] << 8) +
                   static_cast<uint16_t>(frame.data[1]);

  frontrightspeed = static_cast<uint16_t>(frame.data[2] << 8) +
                    static_cast<uint16_t>(frame.data[3]);

  rearleftspeed = static_cast<uint16_t>(frame.data[4] << 8) +
                  static_cast<uint16_t>(frame.data[5]);

  rearrightspeed = static_cast<uint16_t>(frame.data[6] << 8) +
                   static_cast<uint16_t>(frame.data[7]);

  if (frontleftspeed > 65535 || frontrightspeed > 65535 ||
      rearleftspeed > 65535 || rearrightspeed > 65535) {
    std::cout << "One of the Wheel speed value is Out of Range" <<
std::endl;
    return 1;
  }

  return 0;
```

```
}

/*Reconstructs the value of the requested torque from the CAN frame
Also checks whether the value of the parameter is within the limits from
the DBC
file
@param frame : input can frame read from the CAN channel
@param fronttorque : stores the decoded value of requested torque for the
front
wheels
@param reartorque : stores the decoded value of requested torque for the
rear
wheels
*/
int16_t torqueframe(const can_frame frame) {
  fronttorque = unpack_left_shift_u16(frame.data[0], 8u, 0xffu) +
                unpack_right_shift_u16(frame.data[1], 0u, 0xffu);

  reartorque = unpack_left_shift_u16(frame.data[4], 8u, 0xffu) +
               unpack_right_shift_u16(frame.data[5], 0u, 0xffu);

  if (fronttorque > 32767 || fronttorque < -32768 || reartorque > 32767
||
      reartorque < -32768) {
    std::cout << "One of the Torque Request is Out of Range" <<
std::endl;
    return 1;
  }

  return 0;
}

/*calculates the current based on the input parameters
Also determines the max torque limit for the wheels based on the speeds
of the
wheel*/
void currentcal() {
  uint16_t frontwheelspeed =
      maxof(frontleftspeed,
            frontrightspeed); // the speed for the front wheel which is
max of
                              // the speeds for the two wheels
  uint16_t rearwheelspeed = maxof(
      rearleftspeed, rearrightspeed); // the speed for the rear wheel
which is
                                      // max of the speeds for the two
wheels
  _Float32 qfactor = 0.879;
  _Float32 BEFF = 0.6721;

  if (frontwheelspeed > 700) {
    maxtorquefront = 50;
```

```
  } else if (frontwheelspeed > 600) {
    maxtorquefront = 85;
  } else if (frontwheelspeed > 500) {
    maxtorquefront = 100;
  } else if (frontwheelspeed > 400) {
    maxtorquefront = 120;
  } else if (frontwheelspeed > 300) {
    maxtorquefront = 150;
  } else {
    maxtorquefront = 32767;
  }

  if (rearwheelspeed > 700) {
    maxtorquerear = 50;
  } else if (rearwheelspeed > 600) {
    maxtorquerear = 85;
  } else if (rearwheelspeed > 500) {
    maxtorquerear = 100;
  } else if (rearwheelspeed > 400) {
    maxtorquerear = 120;
  } else if (rearwheelspeed > 300) {
    maxtorquerear = 150;
  } else {
    maxtorquerear = 32767;
  }

  if (batteryvoltage < 2.8) {
    maxtorquefront = 20;
    maxtorquerear = 20;
  }


  int16_t fronttorqueactual = minof(fronttorque, maxtorquefront);
  int16_t reartorqueactual = minof(reartorque, maxtorquerear);


  frontcurrent = (fronttorqueactual / batteryvoltage) *
                 (1 + (qfactor * qfactor * (fronttorqueactual / 100)) +
                  (frontwheelspeed * (BEFF / 100) * fronttorqueactual));
  rearcurrent = (reartorqueactual / batteryvoltage) *
                (1 + (qfactor * qfactor * (reartorqueactual / 100)) +
                 (rearwheelspeed * (BEFF / 100) * reartorqueactual));
}

/*@param frame: Input can frame read from the can channel
Identifies the type of the frame based on the can frame id and the sends
the
frame to neccesary function to decode the value */
void process_frame(const can_frame frame) {
  switch (frame.can_id) {
  case 0x526:
    batteryframe(frame);
```

```
    break;

  case 0x525:
    speedframe(frame);
    break;

  case 0x521:
    torqueframe(frame);
    break;

  default:
    break;
  }
}

int main(int argc, char *argv[]) {
  // const std::string canChannel = "vcan0";

  can_frame Current;
  std::memset(&Current, 0, sizeof(Current));
  Current.can_id = 0x320;
  Current.can_dlc = 4;
  Current.data[0] = pack_right_shift_u16(frontcurrent, 8u, 0xffu);
  Current.data[1] = pack_left_shift_u16(frontcurrent, 0u, 0xffu);
  Current.data[2] = pack_right_shift_u16(rearcurrent, 8u, 0xffu);
  Current.data[3] = pack_left_shift_u16(rearcurrent, 0u, 0xffu);

  try {

    while (1) {
      const can_frame frame = can::read(canSocket);
      process_frame(frame);
      currentcal();

      Current.data[0] = pack_right_shift_u16(frontcurrent, 8u, 0xffu);
      Current.data[1] = pack_left_shift_u16(frontcurrent, 0u, 0xffu);
      Current.data[2] = pack_right_shift_u16(rearcurrent, 8u, 0xffu);
      Current.data[3] = pack_left_shift_u16(rearcurrent, 0u, 0xffu);

      std::cout << Current << std::endl;
      can::write(canSocket, Current);

    }

    can::close(canSocket);
  } catch (const std::runtime_error &e) {
    std::cerr << e.what() << std::endl;
  }

  return 0;
}
```

File: send_data.cpp (Code for taking input from user)

```
#include <memory> // for allocator, shared_ptr, __shared_ptr_access
#include <string> // for char_traits, operator+, to_string

#include "ftxui/component/captured_mouse.hpp" // for ftxui
#include "ftxui/component/component.hpp"      // for Slider, Renderer,
Vertical
#include "ftxui/component/component_base.hpp" // for ComponentBase
#include "ftxui/component/screen_interactive.hpp" // for
ScreenInteractive
#include "ftxui/dom/elements.hpp" // for separator, operator|, Element,
size, text, vbox, xflex, bgcolor, hbox, GREATER_THAN, WIDTH, border,
HEIGHT, LESS_THAN
#include "ftxui/screen/color.hpp" // for Color

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include <chrono>
#include <iomanip>
#include <iostream>
#include <string>
#include <thread>

#include "can_wrap.hpp"
#include "candata.h"
using can::operator<<;
using namespace ftxui;

Element Display(int batV, int frontL, int frontR, int rearL, int rearR,
                int torqueF, int torqueR) {
  return text(
      "Battery Voltage = " + std::to_string(batV) + "\n"
+                 //
      "Wheel Speeds \n" + "Front Left" + std::to_string(frontL) + "\n"
+ //
      "Front Right" + std::to_string(frontR) + "\n" + "Rear Left" +
      std::to_string(rearL) + "\n" + "Rear Right" + std::to_string(rearR)
+
      "\n" + "Requested Torque \n" + "Front = " + std::to_string(torqueF)
+
      "\n" + "Rear = " + std::to_string(torqueR) + "\n" //
  );
```

```
}

/*Autogenerated Code from the DBC file
Packs the given value into CAN frame ,
@param value:Variable that stores the value to be packed in the frame
@param shift: Number of bytes to be shifted by
@param mask: Bit mask thats ANDed with the input*/

static inline uint8_t pack_left_shift_u16(uint16_t value, uint8_t shift,
                                          uint8_t mask) {
  return (uint8_t)((uint8_t)(value << shift) & mask);
}

/*Autogenerated Code from the DBC file
Packs the given value into CAN frame ,
@param value:Variable that stores the value to be packed in the frame
@param shift: Number of bytes to be shifted by
@param mask: Bit mask thats ANDed with the input*/

static inline uint8_t pack_right_shift_u16(uint16_t value, uint8_t shift,
                                           uint8_t mask) {
  return (uint8_t)((uint8_t)(value >> shift) & mask);
}

/* @param signame : Input
   @param value : Buffer to store the input from the user
   @param binvalue: function returns this value */
uint16_t getinputs(char signame) {
  std::string signalname;
  switch (signame) {
  case 'a':
    signalname = "Battery Voltage(0 to 13)";
    break;

  case 'b':
    signalname = "Speed of Front Left Wheel(0 to 65535)";
    break;

  case 'c':
    signalname = "Speed of Front Right Wheel(0 to 65535)";
    break;

  case 'd':
    signalname = "Speed of Rear Left Wheel(0 to 65535)";
    break;

  case 'e':
    signalname = "Speed of Rear Right Wheel(0 to 65535)";
    break;

  case 'f':
    signalname = "Requested Torque for Front(-32768 to 32767)";
```

```
    break;

  case 'g':
    signalname = "Requested Torque for Rear(-32768 to 32767)";
    break;

  default:
    break;
  }
  uint16_t value;
  uint16_t binvalue;
  std::cout << "Please enter the " << signalname << std::endl;
  std::cin >> value;
  binvalue = value;
  return binvalue;
}

/*@param ---> battVolt: Batery voltage
  @param ---> Battery: structure that stores the voltage values*/
void get_dataBatt(int battVolt, struct candata_vcu_battery_t *battery) {
  // function to create structure of the data for battery
  battery->voltage = battVolt;
  size_t batSize = 2;
}

/* @param : ---> FL: Speed of the front left wheel
   @param : ---> FR: Speed of the front right wheel
   @param : ---> RL: Speed of the rear left wheel
   @param : ---> RR: Speed of the rear right wheel
   @param : ---> speed: Structure to store the values of in */
void get_dataSpeed(uint16_t FL, uint16_t FR, uint16_t RL, uint16_t RR,
                   struct candata_vcu_wheel_speeds_t *speed) {
  speed->fl_wheel_speed = FL;
  speed->fr_wheel_speed = FR;
  speed->rl_wheel_speed = RL;
  speed->rr_wheel_speed = RR;
}

/*
   @param : ---> TF: Requested torque of the front wheels
   @param : ---> TR: Requested torque of the rear wheels
   @param : ---> torque: Structure to store the values of in */
void get_dataTorque(int16_t TF, int16_t TR, int16_t steer,
                    struct candata_ai_drive_request_t *torque) {
  torque->front_trq_request = TF;
  torque->rear_trq_request = TR;
  torque->steering_request = steer;
}
int main(int argc, char *argv[]) {
  const std::string canChannel = "vcan0";
  const int canSocket = can::connect(canChannel);
```

```
  // data inputs and structure for battery voltage
  int bat = getinputs('a');
  struct candata_vcu_battery_t battery;

  // data inputs and structure for speeds
  int speedFL = getinputs('b');
  int speedFR = getinputs('c');
  int speedRL = getinputs('d');
  int speedRR = getinputs('e');
  struct candata_vcu_wheel_speeds_t speed;

  // data inputs and structure for requested torque

  int torqueF = getinputs('f');
  int torqueR = getinputs('g');
  int steering = 0;
  struct candata_ai_drive_request_t torque;

  while (true) {
    // canframe defination for battery voltage
    can_frame Battery;
    std::memset(&Battery, 0, sizeof(Battery));
    get_dataBatt(bat, &battery);
    Battery.can_id = 0x526;
    Battery.can_dlc = 2;
    Battery.data[0] = pack_right_shift_u16(battery.voltage, 8u, 0xffu);
    Battery.data[1] = pack_left_shift_u16(battery.voltage, 0u, 0xffu);

    // canframe defination for wheel speeds
    can_frame Speeds;
    std::memset(&Speeds, 0, sizeof(Speeds));
    Speeds.can_id = 0x525;
    Speeds.can_dlc = 8;
    get_dataSpeed((uint16_t)speedFL, (uint16_t)speedFR,
(uint16_t)speedRL,
                  (uint16_t)speedRR, &speed);
    Speeds.data[0] = pack_right_shift_u16(speed.fl_wheel_speed, 8u,
0xffu);
    Speeds.data[1] = pack_left_shift_u16(speed.fl_wheel_speed, 0u,
0xffu);
    Speeds.data[2] = pack_right_shift_u16(speed.fr_wheel_speed, 8u,
0xffu);
    Speeds.data[3] = pack_left_shift_u16(speed.fr_wheel_speed, 0u,
0xffu);
    Speeds.data[4] = pack_right_shift_u16(speed.rl_wheel_speed, 8u,
0xffu);
    Speeds.data[5] = pack_left_shift_u16(speed.rl_wheel_speed, 0u,
0xffu);
    Speeds.data[6] = pack_right_shift_u16(speed.rr_wheel_speed, 8u,
0xffu);
    Speeds.data[7] = pack_left_shift_u16(speed.rr_wheel_speed, 0u,
0xffu);
```

```
    // canframe definaion for torque
    can_frame Torque;
    std::memset(&Torque, 0, sizeof(Torque));
    Torque.can_id = 0x521;
    Torque.can_dlc = 6;
    get_dataTorque((int16_t)torqueF, (int16_t)torqueR, (int16_t)steering,
                   &torque);
    Torque.data[0] = pack_right_shift_u16(torque.front_trq_request, 8u,
0xffu);
    Torque.data[1] = pack_left_shift_u16(torque.front_trq_request, 0u,
0xffu);
    Torque.data[2] = pack_right_shift_u16(torque.steering_request, 8u,
0xffu);
    Torque.data[3] = pack_left_shift_u16(torque.steering_request, 0u,
0xffu);
    Torque.data[4] = pack_right_shift_u16(torque.rear_trq_request, 8u,
0xffu);
    Torque.data[5] = pack_left_shift_u16(torque.rear_trq_request, 0u,
0xffu);

    try {
      // std::cout << RPM << torque << std::endl;
      std::cout << Battery << std::endl;
      can::write(canSocket, Battery);

      std::cout << Speeds << std::endl;
      can::write(canSocket, Speeds);

      std::cout << Torque << std::endl;
      can::write(canSocket, Torque);

    } catch (const std::runtime_error &e) {
      std::cerr << e.what() << std::endl;
    }

    std::this_thread::sleep_for(std::chrono::seconds(1));
  }

  can::close(canSocket);

  return 0;

  // GUI code

  /*auto slider_1 = Slider("Battery Voltage  :", &bat, 0, 13, 1);
  auto slider_2 = Slider("Speed Front Left :", &speedFL, -65535, 65535,
10);
  auto slider_3 = Slider("Speed Front Right:", &speedFR, -65535, 65535,
10);
  auto slider_4 = Slider("Speed Rear Left  :", &speedRL, -65535, 65535,
10);
```

```
  auto slider_5 = Slider("Speed Rear Right :", &speedRR, -65535, 65535,
10);
  auto slider_6 = Slider("Front Torque     :", &torqueF, -32768, 32767,
10);
  auto slider_7 = Slider("Rear Torque      :", &torqueR, -32768, 32767,
10);

   auto container = Container::Vertical({
    slider_1,
    slider_2,
    slider_3,
    slider_4,
    slider_5,
    slider_6,
    slider_7,
});

  auto renderer = Renderer(container, [&] {
  return hbox({
              //ColorTile(red, green, blue),
              separator(),
              vbox({
                  slider_1->Render(),
                  separator(),
                  slider_2->Render(),
                  separator(),
                  slider_3->Render(),
                  separator(),
                  slider_4->Render(),
                  separator(),
                  slider_5->Render(),
                  separator(),
                  slider_6->Render(),
                  separator(),
                  slider_7->Render(),
                  separator(),
                  Display (bat, speedFL,
speedFR,speedRL,speedRR,torqueF,torqueR),
              }) | xflex,
          }) |
          border | size(WIDTH, LESS_THAN, 80);
});
auto screen = ScreenInteractive::TerminalOutput();
screen.Loop(renderer);*/
}
```

file: tests_get_data.cpp (Unit Test Code)

```
#define CATCH_CONFIG_MAIN
#include <catch.hpp>
#include <cstdint>

int storeV;
can_frame testframe;
TEST_CASE("Baterry Voltage in range") {
  SECTION("Applying the Mid Value") {
    testframe.can_dlc = 2;
    testframe.data[0] = 0x00;
    testframe.data[1] = 0x07;
    storeV = batteryframe(testframe);
    REQUIRE(batteryvoltage == 7);
    REQUIRE(storeV == 0);
  }

  SECTION("Appying Minimum value") {
    testframe.data[0] = 0x00;
    testframe.data[1] = 0x00;
    storeV = batteryframe(testframe);
    REQUIRE(batteryvoltage == 0);
    REQUIRE(storeV == 0);
  }

  SECTION("Applying the Maximum Value") {
    testframe.data[0] = 0x00;
    testframe.data[1] = 0x0D;
    storeV = batteryframe(testframe);
    REQUIRE(batteryvoltage == 13);
    REQUIRE(storeV == 0);
  }
}

TEST_CASE("Battery Voltage out of range") {
  testframe.data[0] = 0x0A;
  testframe.data[1] = 0x0A;
  storeV = batteryframe(testframe);
  REQUIRE(batteryvoltage == 2570);
  REQUIRE(storeV == 1);
}

TEST_CASE("Wheel Speeds in the range") {
  SECTION("Applying the Mid Value") {
    testframe.data[0] = 0x01;
    testframe.data[1] = 0xF4;
    testframe.data[2] = 0x01;
    testframe.data[3] = 0xF4;
```

```
    testframe.data[4] = 0x01;
    testframe.data[5] = 0xF4;
    testframe.data[6] = 0x01;
    testframe.data[7] = 0xF4;
    storeV = speedframe(testframe);
    REQUIRE(frontleftspeed == 500);
    REQUIRE(frontrightspeed == 500);
    REQUIRE(rearleftspeed == 500);
    REQUIRE(rearrightspeed == 500);
    REQUIRE(storeV == 0);
  }

  SECTION("Applying Minimum Value") {
    testframe.data[0] = 0x00;
    testframe.data[1] = 0x00;
    testframe.data[2] = 0x00;
    testframe.data[3] = 0x00;
    testframe.data[4] = 0x00;
    testframe.data[5] = 0x00;
    testframe.data[6] = 0x00;
    testframe.data[7] = 0x00;
    storeV = speedframe(testframe);
    REQUIRE(frontleftspeed == 0);
    REQUIRE(frontrightspeed == 0);
    REQUIRE(rearleftspeed == 0);
    REQUIRE(rearrightspeed == 0);
    REQUIRE(storeV == 0);
  }

  SECTION("Applying the Maximum Values") {
    testframe.data[0] = 0xFF;
    testframe.data[1] = 0xFF;
    testframe.data[2] = 0xFF;
    testframe.data[3] = 0xFF;
    testframe.data[4] = 0xFF;
    testframe.data[5] = 0xFF;
    testframe.data[6] = 0xFF;
    testframe.data[7] = 0xFF;
    storeV = speedframe(testframe);
    REQUIRE(frontleftspeed == 65535);
    REQUIRE(frontrightspeed == 65535);
    REQUIRE(rearleftspeed == 65535);
    REQUIRE(rearrightspeed == 65535);
    REQUIRE(storeV == 0);
  }
}

TEST_CASE("TORQUE Values in Range") {

  SECTION("Applying the Mid Values") {
    testframe.data[0] = 0x00;
    testframe.data[1] = 0x64;
```

```
    testframe.data[2] = 0x00;
    testframe.data[3] = 0x00;
    testframe.data[4] = 0x00;
    testframe.data[5] = 0x64;
    storeV = torqueframe(testframe);
    REQUIRE(fronttorque == 100);
    REQUIRE(reartorque == 100);
    REQUIRE(storeV == 0);
  }

  SECTION("Applying the Maximum Values") {
    testframe.data[0] = 0x7F;
    testframe.data[1] = 0xFF;
    testframe.data[2] = 0x00;
    testframe.data[3] = 0x00;
    testframe.data[4] = 0x7F;
    testframe.data[5] = 0xFF;
    storeV = torqueframe(testframe);
    REQUIRE(fronttorque == 32767);
    REQUIRE(reartorque == 32767);
    REQUIRE(storeV == 0);
  }
}

TEST_CASE("TORQUE Values are out of range") {

  SECTION("Front Torque is Greater than upper limit") {
    testframe.data[0] = 0x9C;
    testframe.data[1] = 0x40;
    testframe.data[2] = 0x00;
    testframe.data[3] = 0x00;
    testframe.data[4] = 0x7F;
    testframe.data[5] = 0xFF;
    storeV = torqueframe(testframe);
    REQUIRE(fronttorque == 40000);
    REQUIRE(reartorque == 32767);
    REQUIRE(storeV == 1);
  }

  SECTION("Rear Torque is Greater than upper limit") {

    testframe.data[0] = 0x7F;
    testframe.data[1] = 0xFF;
    testframe.data[2] = 0x00;
    testframe.data[3] = 0x00;
    testframe.data[4] = 0x80;
    testframe.data[5] = 0x00;
    storeV = torqueframe(testframe);
    REQUIRE(fronttorque == 32767);
    REQUIRE(reartorque == 32768);
    REQUIRE(storeV == 1);
  }
```

```
}

TEST_CASE("Checking the Maximum torque for the different wheel speed
values") {
  SECTION("Checking the value for Wheelspeeds Greater than 700") {
    frontleftspeed = 800;
    frontrightspeed = 650;
    rearleftspeed = 750;
    rearrightspeed = 600;
    currentcal();
    REQUIRE(maxtorquefront == 50);
    REQUIRE(maxtorquerear == 50);
  }

  SECTION("Checking the value for Wheelspeeds Greater than 600") {
    frontleftspeed = 500;
    frontrightspeed = 650;
    rearleftspeed = 550;
    rearrightspeed = 640;
    currentcal();
    REQUIRE(maxtorquefront == 85);
    REQUIRE(maxtorquerear == 85);
  }

  SECTION("Checking the value for Wheelspeeds Greater than 500") {
    frontleftspeed = 550;
    frontrightspeed = 450;
    rearleftspeed = 580;
    rearrightspeed = 440;
    currentcal();
    REQUIRE(maxtorquefront == 100);
    REQUIRE(maxtorquerear == 100);
  }

  SECTION("Checking the value for Wheelspeeds Greater than 400") {
    frontleftspeed = 350;
    frontrightspeed = 450;
    rearleftspeed = 380;
    rearrightspeed = 440;
    currentcal();
    REQUIRE(maxtorquefront == 120);
    REQUIRE(maxtorquerear == 120);
  }

  SECTION("Checking the value for Wheelspeeds Greater than 300") {
    frontleftspeed = 350;
    frontrightspeed = 250;
    rearleftspeed = 380;
    rearrightspeed = 240;
    currentcal();
    REQUIRE(maxtorquefront == 150);
    REQUIRE(maxtorquerear == 150);
```

```
  }

  SECTION("Checking the value for Wheelspeeds Smaller than 300") {
    frontleftspeed = 150;
    frontrightspeed = 250;
    rearleftspeed = 180;
    rearrightspeed = 240;
    currentcal();
    REQUIRE(maxtorquefront == 32767);
    REQUIRE(maxtorquerear == 32767);
  }
}

TEST_CASE("Checking the Maximum allowed torque for the battery voltage
less "
          "than 2.8 V") {
  testframe.data[0] = 0x00;
  testframe.data[1] = 0x02;
  storeV = batteryframe(testframe);
  currentcal();
  REQUIRE(maxtorquefront == 20);
  REQUIRE(maxtorquerear == 20);
}
```

While Running the executables:

`bin/send_data` : for transmitting the data b user on the CAN Bus.
`bin/get_data` : For reading the inputs from the CAN bus, Calculate the current value and transmit it back on CAN bus
`bin/tests_get_data` : To run the unit tests on the code.

References

1. “The Power of Ten – Rules for Developing Safety Critical Code” By Gerard J. Holzmann .