Custom Data Acquisition System for the Cal Poly Racing Baja Team

A Senior Project Report

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science in Computer Engineering

by

Joe Keenan

June 2025

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

Baja SAE is an international collegiate competition run by the Society of Automotive Engineers (SAE) where teams design, build, test, and compete with offroad baja style vehicles. In the United States, there are three competitions held each year across the country for teams to compete in. There are three categories of events for which teams are scored: static events, dynamic events, and the endurance event. Static events include different challenges to test a team's ability to effectively communicate design choices and business aspects of making a vehcile. Dynamic events test the abilities of the vehicle to perform in different conditions. These events include an acceleration event, a manueverability event, a suspension event, and a traction event. Finally, the endurance event tests the vehicles and drivers ability to withstand rough terrain and wheel to wheel racing for a full four hours. At the end of the competition, all the event scores are added together to determine who the top three overall teams at the competition are. To win the competition, it is crucial to perform well in all three styles of events.

Cal Poly Racing competes in the Baja SAE series of competitions. Over the last several years, Cal Poly Racing has had a moderate amount of success, with several place tropies in dynamic events. In order to design a vehcile capable of withstanding all the harsh events in a Baja SAE competition, it is critical to fully understand how every system of the car is behaving. Additionally, undertstanding the load cases, such as impacts, is also essential to making informed design choices and considerations.

To understand the the vehicle and the load cases, a data collection system is needed to log and process sensor data.

Data Acquisition Systems (DAQs) are tools that allow for the logging and processing of data.

Chapter 2

BACKGROUND

2.1 CAN

Controller Area Network (CAN) busses are commonly used in automotive applications to connect different control or instrumentation nodes together developed by Bosch in the 1990s. This allows for any node to communicate with any other node on the bus. CAN utilizes a two wire asynchronous differential twisted pair signal to transmit across the bus. The asynchronous nature of this protocol reduces the number of wires required to transmit data. By utilizing a differential twisted pair, noise and interference are reduced improving reliability and robustness of the network. However, only utilizing a single differential pair means that a node can only transmit or only receive at any given time, reducing throughput.

CAN is an addressed based communication protocol. An address can correspond to a specific node or to a specific message. Since CAN only utilizes a signle differential signal, it must negotiate to determine which node is transmitting and which nodes are receiving. The lowest address trying to be transmitted wins the negotiation, meaning that priority can be assigned to messages by assigning a lower value for an address to the message. This addressing and need to negotiate also adds overhead to the transmission, reducing overall data throughput. Additionally, there are control bits, cyclical redundancy (CRC) bits, and end of frame (EoF) bits that all also contribute to overhead.

2.1.1 CAN 2.0

CAN 2.0 is the most commonly used CAN protocol in the automotive. This version of CAN uses an 11 bit identification or address section with maximum of 8 bytes of data transmitted. The bitrate for this version of CAN can be up to 1 Mega bit per second (Mbps) [4]. With an assumption of 1 byte of data transmission, the overhead can be computed as $\frac{48\text{bits}}{56\text{bits}}$. With the assumption of 8 bytes of data transmission, the overhead can be computed as $\frac{48\text{bits}}{112\text{bits}}$. The more data that is transmitted per frame, the less overhead impacts the total data throughput. An example data frame of CAN 2.0 can be seen in fig. 2.1.

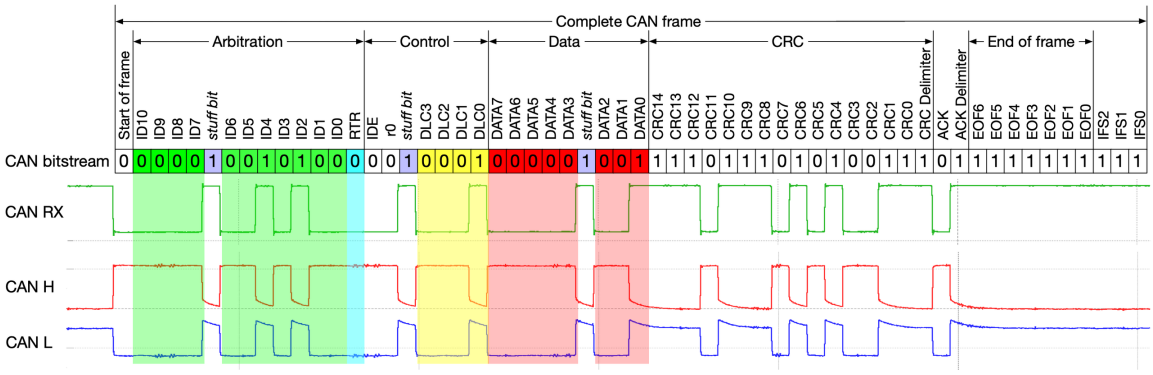


Figure 2.1: *Example CAN 2.0 Frame [2]*

2.1.2 CAN FD

In 2012, Bosch released a new version of CAN with a flexible data rate called CAN FD. This new version of CAN has several improvements including faster data rates and allowing for more data to be transmitted in each frame. The flexible data rate comes from CAN FD's ability to increase the data rate during the data section of the transmission. The protocol supports up to 8 Mbps of throughput during the data section of transmission and up to 1 Mbps of throughput during the beginning and ending parts of the frame [3]. This allows for significantly more

data throughput overall. In addition to the flexible data rate used by CAN FD, the maximum size of the data section was increased from 8 bytes to 64 bytes [3]. This decreases total overhead significantly making the transmission more efficient. These two new characteristics combined provide for a much more efficient transfer of data, particularly when transmitting high amounts of data very frequently.

In order to use CAN FD, it must be supported by the hardware, including both the CAN controller and the CAN transceiver. Additionally, to rely solely on CAN FD, each node on the bus must be using CAN FD. Otherwise, conflicts may occur impacting frames being sent or received. The system would theoretically operate without too many issue, but the control section of the frame would be different sizes to accomondate the additional control bits needed for the CAN FD frame.

2.2 AEM

A company that makes a similar product is AEM Electronics. AEM designs data loggers and dashboards for testing purposed for cars and other vehicles. These dataloggers connect to their different modules that connnect to various can sensors and breakouts for different types of sensors. Some of these modules also come with an inertial measurement units (IMU) and global positioning system (GPS) which are useful for understanding the entire vehicle state.

2.2.1 AEM Loggers

There are several different AEM loggers available for purcahse. The two major loggers available for purchase are the AEM CD-5L and the AEM CD-7L. The CD-5L costs $1,574.95 without the IMU or GPS and the CD-7L costs $2,081.95 without

the IMU or GPS [1]. This cost provides access to data logging features as well as a dashbaord display. These dashboards can be configured to display different information to a driver or engineer to get a snapshot of the current state of the vehicle. A computer can be easily connected to these devices to upload different configurations as well as to download the logged files.

Figure 2.2: *AEM CD5L Dashboard*

The loggers log files in a comma seperated file (csv) format for ease of use with tools such as Matlab or Python for visualizing the data. These files can also be visualized on the dashboard as well to give a quick glimpse at explaining what is happening.

2.2.2 AEM Add-Ons

In order to connect more sensors or instrumentation to the AEM loggers, additional add-ons must be purchased and configured. Some of these add-ons include a CAN module which allows the AEM logger to connect to and interface with a CAN bus and a thermocouple unit for measuring temperature. These expansion units, before

the cost of any sensors, are several hundred dollars on their own. In order to hook up any sensors, at least one of these expansion units must be purchased.

Figure 2.3: *AEM thermocouple expansion unit*

AEM also sells some sensors that are simple to integrate into their ecosystem. These sensors include thermocouples, air and water temperature sensors, and several styles of pressure transducers. Each of these sensors is somewhat expensive on their own, costing around $100 per individual sensor. The AEM loggers can also interface with traditional analog or digital sensors so long as the appropriate add-ons are purchased.

In addition to wired sensors, AEM is also compatible with some wireless sensors for things such as tire pressure monitoring systems (TPMS) or fuel sensors. In order to communicate with these wireless sensors, a wireless module in needed or a logger with wireless capabilities is needed. AEM utilies a propietary system call X-Wifi to connect to these wireless sensors that is already a part of their CD5 and CD7 loggers. AEM's sensors that can utilize the X-Wifi capabilities of the logger are priced Similarly to their more traditional wired sensors.

2.2.3 DBC Files

In addition to AEM specific sensors, the AEM dataloggers can interface with any custom hardware that is outputting it's data over CAN. In order to configure the AEM to read data from a CAN bus, including its own CAN capable sensors, a configuration file must be made to tell the logger which value corresponds to which bytes of a specific CAN frame. A CAN Database file (.dbc) is a standarized file format for configuring which is able to describe what information is contained within a CAN frame. An example .dbc file can be seen in listing A.1 in the appendix. AEM utilizes this file format for its dataloggers to interface with a CAN network so long as the proper hardware add-ons are purchased.

2.3 AiM

Another company that makes dataloggers to attach to a vehicle is AiM Sports. AiM makes dataloggers that are specifically intended for motorsport applications as well other products such as dashboards, steering wheels, and ECUs. One distinction between AiM and AEM is that AEM also makes and sells sensors for use with their logger while AiM sticks to exclusively making the logger unit. Similarly to AEM, AiM also makes combined dashes and loggers.

2.3.1 AiM Loggers

AiM develops several loggers with similar specs to the AEM loggers with some additional features. The MX series and MXM series loggers are comparable to th CD5 and CD7 logger dashes that are sold by AEM. These loggers also act as dash displays, with the MX series having a nicer display than the MXM series. The MX

series and the MXM series both support many different ECU communication protocols over CAN or RS-232 as well as several different direct sensor inputs.

AiM also develops loggers that do not serve as a dash, only containing the functionality for aggregating and logging data. These loggers offer most of the same features that the dash loggers offer with CAN connectivity, wireless connectivity, and different direct sensor inputs. There are two main products offered in this line, the ECULog and the XLog. The ECULog is the cheaper version with fewer features, only supporting external sensor inputs and communication. The XLog includes all of this and also includes and on board IMU and GPS, making it easier to monitor vehicle position and behaviour during a lap.

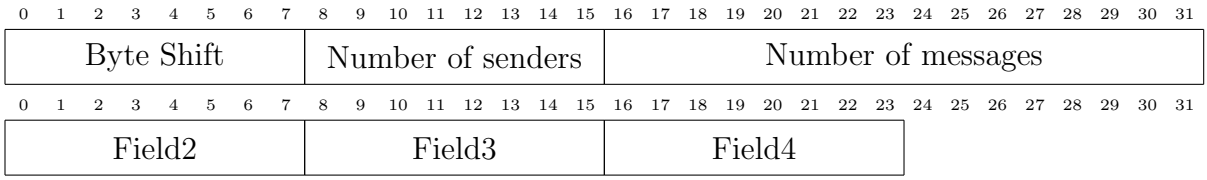
2.4 Motec

Chapter 3

FORMAL PROJECT DEFINITION

Chapter 4

SYSTEM DESIGN AND IMPLEMENTATION



4.1 System Level Design

At a high level, this system is designed to efficiently and reliably aggregate and store controller and instrumentation data from all over our vehcile. In order to accomplish this, the data collection solution needs to be desgined with the expandability and usability in mind. One of the challenges of collecting sensor data for analysis is that it often is the case that which sensors are wanted or needed are not known until the system wanting to be tested is designed and implemented.

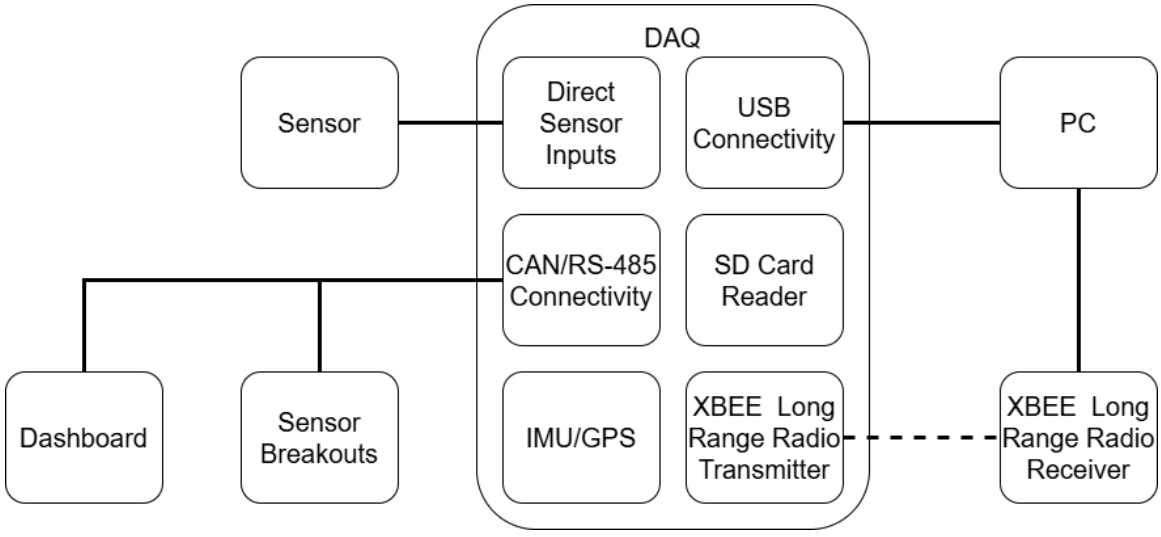


Figure 4.1: *System Level Block Diagram*

Chapter 5

SYSTEM TESTING AND ANALYSIS

Chapter 6

CONCLUSION

Chapter 7

REFLECTIONS

REFERENCES

[1] *AEM Electronics Official Website*. 2025. URL: https://www.aemelectronics.com/.

[2] *CAN bus — Wikipedia, The Free Encyclopedia*. 2025. URL: https://en.wikipedia.org/wiki/CAN_bus.

[3] R. B. GmbH. *CAN FD Specification*. Tech. rep. Robert Bosch GmbH, 2012. URL: https:/https://web.archive.org/web/20151211125301/http://www.bosch-semiconductors.de/media/ubk_semiconductors/pdf_1/canliteratur/can_fd_spec.pdf.

[4] R. B. GmbH. *CAN Specification Version 2.0*. Tech. rep. Robert Bosch GmbH, 1991. URL: https://web.archive.org/web/20221010170747/http://esd.cs.ucr.edu/webres/can20.pdf.

Appendix A

CODE IMPLEMENTATIONS

Code Listing A.1: *Example .dbc file for an IMU*

```
VERSION ""


NS_ :
        NS_DESC_
        CM_
        BA_DEF_
        BA_
        VAL_
        CAT_DEF_
        CAT_
        FILTER
        BA_DEF_DEF_
        EV_DATA_
        ENVVAR_DATA_
        SGTYPE_
        SGTYPE_VAL_
        BA_DEF_SGTYPE_
        BA_SGTYPE_
        SIG_TYPE_REF_
        VAL_TABLE_
        SIG_GROUP_
        SIG_VALTYPE_
        SIGTYPE_VALTYPE_
        BO_TX_BU_
        BA_DEF_REL_
        BA_REL_
        BA_DEF_DEF_REL_
        BU_SG_REL_
        BU_EV_REL_
        BU_BO_REL_
        SG_MUL_VAL_

BS_:

BU_: IMU_DBC
```

```
BO_ 1304 MESSAGE_4: 8 Vector__XXX
 SG_ IMU_TEMP : 48|16@1- (0.01,0) [-20|80] "C"  IMU_DBC
 SG_ EULER_Z : 32|16@1- (0.01,0) [-180|180] "deg/s"
    IMU_DBC
 SG_ EULER_Y : 16|16@1- (0.01,0) [-90|90] "deg/s"
    IMU_DBC
 SG_ EULER_X : 0|16@1- (0.01,0) [-180|180] "deg/s"
    IMU_DBC

BO_ 1303 MESSAGE_3: 8 Vector__XXX
 SG_ Raw_GyroII_Z : 48|16@1- (0.1,0) [-1000|1000] "deg/
    s"  IMU_DBC
 SG_ Raw_GyroII_Y : 32|16@1- (0.1,0) [-1000|1000] "deg/
    s"  IMU_DBC
 SG_ Raw_GyroII_X : 16|16@1- (0.1,0) [-1000|1000] "deg/
    s"  IMU_DBC
 SG_ ACC_RAW_Z : 0|16@1- (0.001,0) [-4|4] "g"  IMU_DBC

BO_ 1302 MESSAGE_2: 8 Vector__XXX
 SG_ ACC_RAW_Y : 48|16@1- (0.001,0) [-4|4] "g"  IMU_DBC
 SG_ ACC_RAW_X : 32|16@1- (0.001,0) [-4|4] "g"  IMU_DBC
 SG_ ANGULAR_VEL_Z : 16|16@1- (0.01,0) [-125|125] "deg/
    s"  IMU_DBC
 SG_ ANGULAR_VEL_Y : 0|16@1- (0.01,0) [-327.68|327.67]
    "deg/s"  IMU_DBC

BO_ 1301 MESSAGE_1: 8 Vector__XXX
 SG_ LIN_ACC_Z : 32|16@1- (0.001,0) [-4|4] "g"  IMU_DBC
 SG_ LIN_ACC_Y : 16|16@1- (0.001,0) [-4|4] "g"  IMU_DBC
 SG_ LIN_ACC_X : 0|16@1- (0.001,0) [-4|4] "g"  IMU_DBC
 SG_ ANGULAR_VEL_X : 48|16@1- (0.01,0) [-125|125] "deg/
    s"  IMU_DBC



BA_DEF_  "MultiplexExtEnabled" ENUM  "No","Yes";
BA_DEF_  "BusType" STRING ;
BA_DEF_DEF_  "MultiplexExtEnabled" "No";
BA_DEF_DEF_  "BusType" "CAN";
```