University of Stuttgart
Institute of Industrial Automation and Software Systems
Prof. Dr.-Ing. M. Weyrich

Receivers
Prof. Andrey Morozov
MSc. Sheng Ding

Study Project

Generation of CAN data using Simulink and injection of anomalies in real time

Saiprasad Salkar

Study Project Final Report

Thesis No:	Type:	Class of Process Model:	Student:	Supervisor:	Begin:	End:
0	Study Project	Model for Software Development	Saiprasad Salkar	Sheng Ding	01/11/21	31/01/22

Document:	Version:	Author:	Date:	Status:
Study Project Final Report	1	Saiprasad Salkar	28.01.22	complete

Filename:	Pages:	Print Date:	Template:
Study Project Final Report	19	1/28/2022 9:04 PM	sw-project-final-report.dotm

Template Version Management

Version	Author	QA	Date	Status	Changes
1.0	Ringler	Kn	22.12.99	Submitted	Creation
1.0	Ringler	Gö	10.01.00	Accepted	
1.1	Keller	Ri	22.05.00	Accepted	New Logo
1.2	Eßlinger	Ri	13.03.00	Accepted	Layout
1.3	Wedel	Bt	31.08.04	Accepted	Layout, language

Document Version Management

Version	Author	QA	Date	Status	Changes
1	Saiprasad Salkar		24/01/22	complete	Creation
2	Saiprasad Salkar		28/01/22	complete	Creation

0 Table of Contents

1 Introduction into the Project

Connected and automated vehicles (CAVs) are expected to revolutionize the transportation industry, mainly through allowing for a real-time and seamless exchange of information between vehicles and roadside infrastructure. Although connectivity and automation are projected to bring about a vast number of benefits, they can give rise to new challenges in terms of safety, security, and privacy. To navigate roadways, CAVs need to heavily rely on their sensor readings and the information received from other vehicles and roadside infrastructure. Hence, anomalous sensor behavior/data points caused by either malicious cyber-attacks or faulty vehicle sensors can result in disruptive consequences, and possibly lead to fatal crashes. As a result, before the mass implementation of CAVs, it is important to develop methodologies that can detect anomalies and identify their sources seamlessly and in real-time.[1]

To do so first we need vehicle CAN data in simulation environment in real time. Which can be injected with fault or anomalous data.

In this project, we primarily focus on gathering the data generated by the sensors for speed, accel of automated vehicle. The data would be forwarded to Simulink model.

2 Objectives

The objective of the study project is to design MATLAB Simulink model for CAN Tx and Rx to observer CAN data in real time and injection of fault (anomalies) on the data**.**

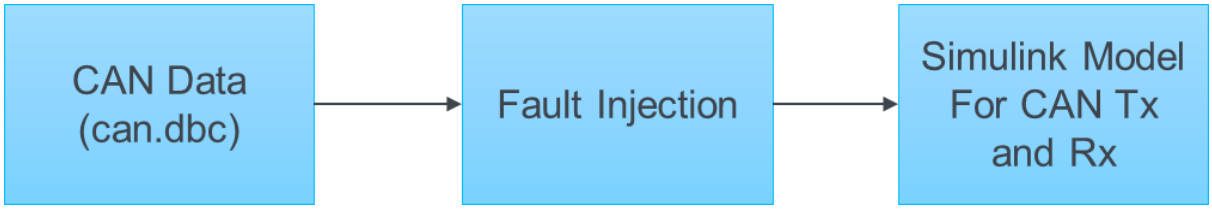


Figure 1: Project Overview

3 Operational Area

The result is to be used as a foundation for future research in context of anomalies detection and identification in automated vehicles. Analysis on data and find the source of anomalies rather be failure sensor or cyber-attacks.

4 Functional Requirements to the Conception

/UFR10/ Investigate the current system with focus on Simulation.
/UFR20/ Investigate type of sensor data to be used for Simulation.
/UFR30/ The implemented modules should be tested.
/UFR40/ Create a visualization to compare original and anomalies data.

5 Non-Functional Requirements to the Conception

/UNR10/ The efficiency of the Simulink model should not compromise on the performance.
/UNR20/ The database access should be as robust as possible.

6 Fundamental Design Decisions

Following design decisions regarding the data, MATLAB model and fault injection block selection.

6.1 Selection of CAN supported data format

The data to be send over the CAN communication should be in form of CAN.dbc format. The sensor data for automated vehicle was taken from [2]. The data was then converted to CAN supported format.

6.2 Selection of Simulink blocks

The Vehicle Network Toolbox provided by MATLAB is used for implementation of CAN communication in simulation environment.

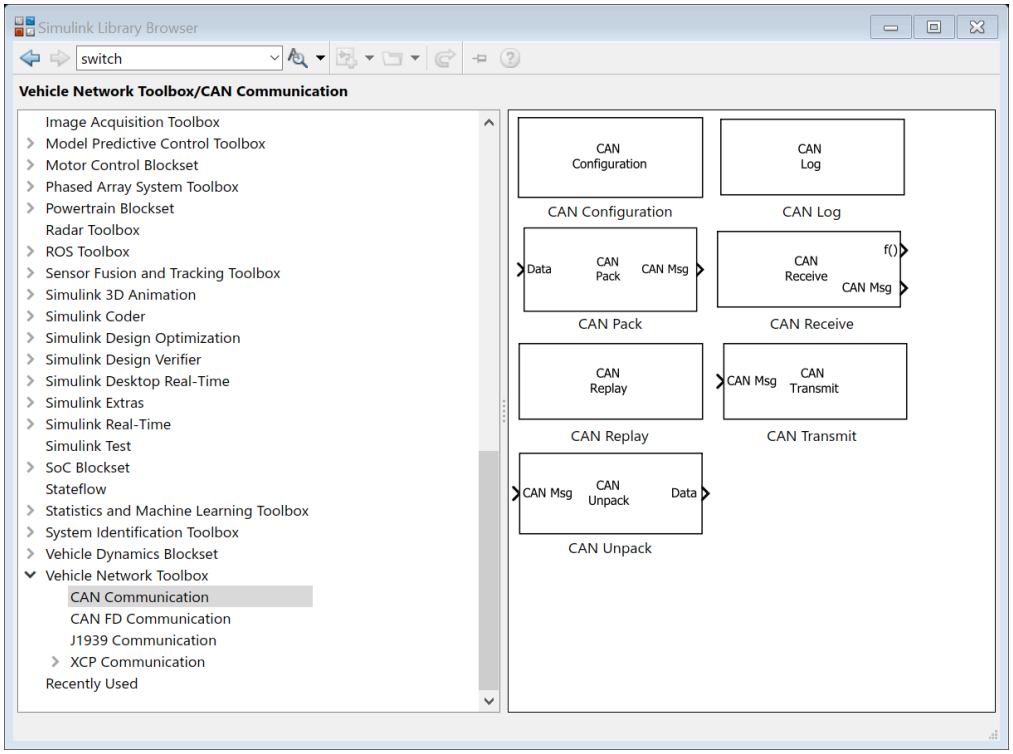


Figure 2: Vehicle Network Toolbox

6.3 Selection of Fault Injection Simulink block

The fault injection block developed by [3]. Has been used to introduce data anomalies in CAN communication.

7 Implementation

The implementation of the Simulation environment is described in detail.

7.1 Conversion of Data to the CAN supported format (.dbc)

A CAN DBC file (CAN database) is a text file that contains information for decoding raw CAN bus data to 'physical values'.

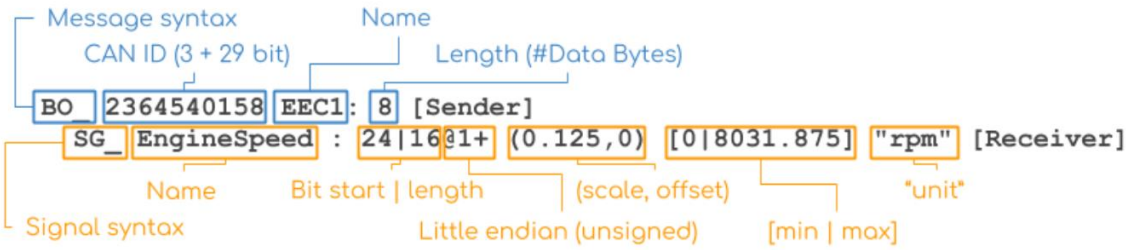


Figure 3: DBC message & signal syntax

At the heart of a DBC file are the rules that describe how to decode CAN messages and signals:

CAN Message syntax:

- A message starts with BO_ and the ID must be unique and in decimal (not hexadecimal)
- The DBC ID adds 3 extra bits for 29 bits CAN IDs to serve as an 'extended ID' flag
- The name must be unique, 1-32 characters and may contain [A-z], digits and underscores
- The length (DLC) must be an integer between 0 and 1785
- The sender is the name of the transmitting node, or Vector_XXX if no name is available

CAN Signal syntax:

- Each message contains 1+ signals that start with SG_
- The name must be unique, 1-32 characters and may contain [A-z], digits and underscores
- The bit start counts from 0 and marks the start of the signal in the data payload
- The bit length is the signal length
- The @1 specifies that the byte order is little-endian/Intel (vs @0 for big-endian/Motorola)
- The + informs that the value type is unsigned (vs - for signed signals)
- The (scale, offset) values are used in the physical value linear equation (more below)
- The [min|max] and unit are optional meta information (they can e.g., be set to [0|0] and "")
- The receiver is the name of the receiving node (again, Vector_XXX is used as default).[5]

Considering above syntax, CAN.dbc is created for data to be send over CAN bus.

7.2 CAN Communication Simulink Model

CAN communication Simulink Model is implemented using Vehicle Network Toolbox. Below is implementation of CAN communication in MATLAB Simulink.

Simulink Model:

Basic blocks that are useful for CAN Communication are: CAN Configuration, CAN Transmit, CAN Pack, CAN Receive, CAN Unpack and Function Block.

1) CAN Transmission Simulink Model:

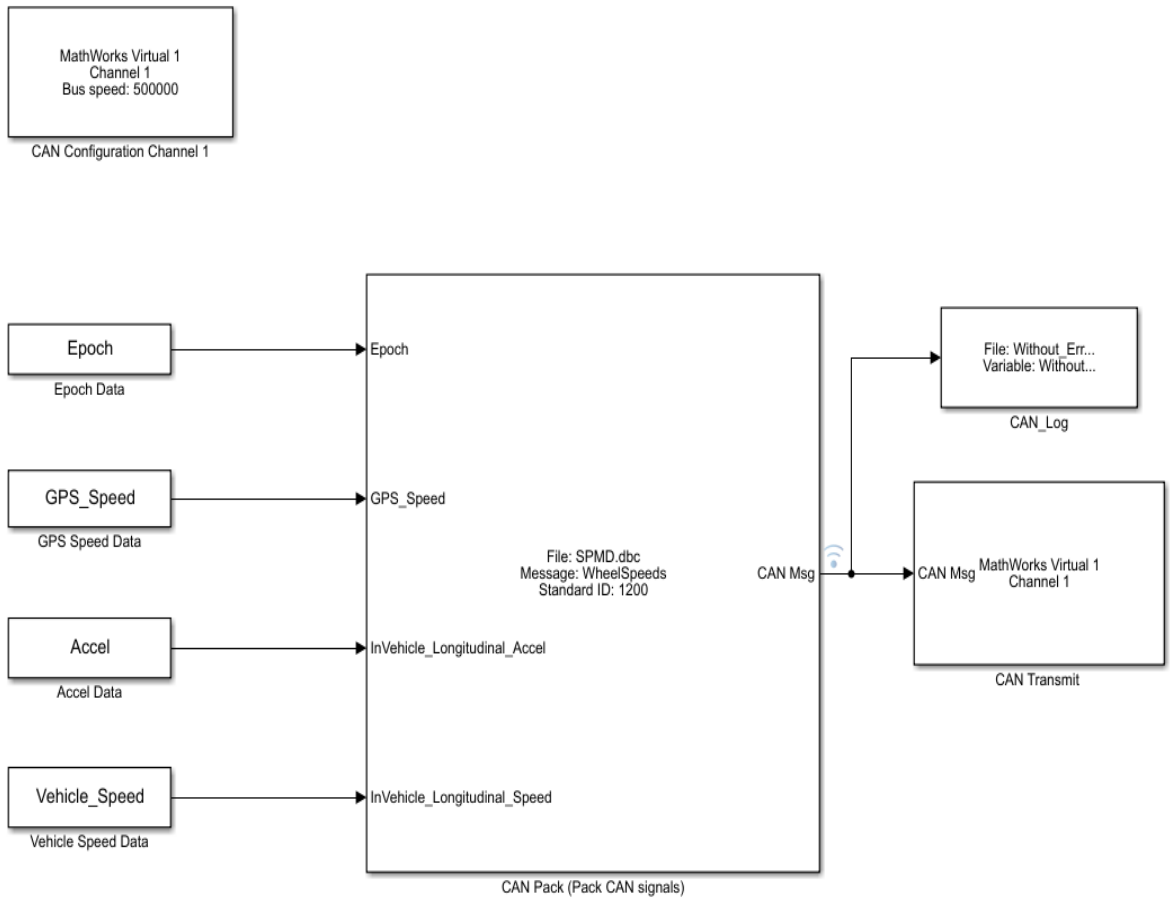


Figure 4: CAN Transmit Simulink Model

2) CAN Reception Simulink Model:

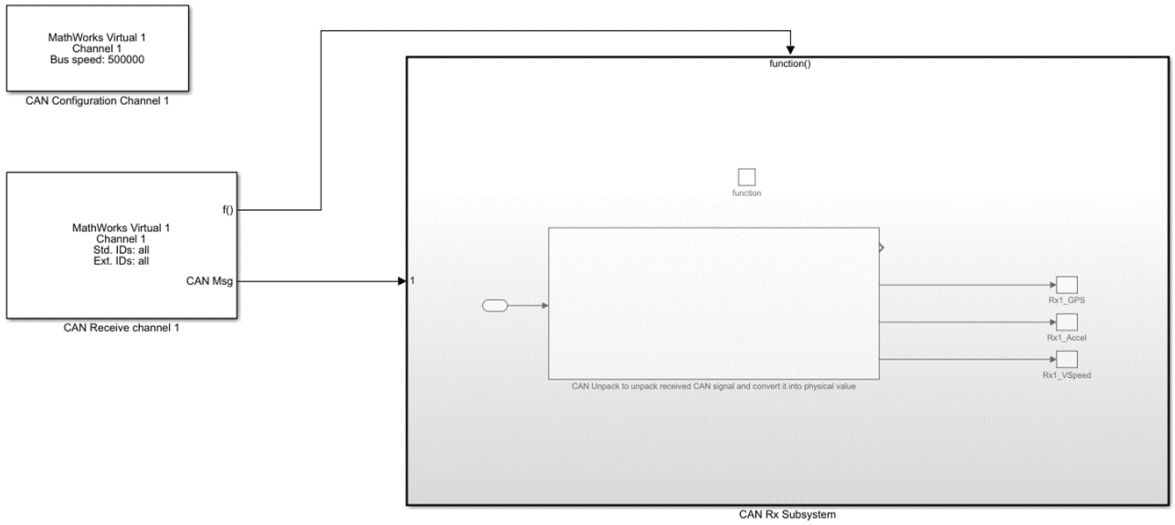


Figure 5: CAN Receive Simulink Model

7.3 Fault Injection

On the CAN bus, the raw CAN bus data consists of a list of (ID, data payload) pairs indexed with timestamps. The sequence of packet data payloads associated with a given ID can be viewed as a multivariate binary sequence. Generally speaking, streaming data in CAN network is highly regular. Thus, attacks usually include three performances: packets are added, packets are missing, or modified packets within a single ID's symbol stream. Effects on CAN bus traffic includes two major types of anomalies: frequency effects, and data effects. Frequency effects are in the context of all known IDs being periodic with fixed frequencies. They are defined as insertions of extra packets, or the erasure of expected packets. Data effects describe how data values can be changed in attack traffic. Advanced attacks usually have a malicious effect by setting control values in the data fields. Here we are concerned with pure data that does not involve additional packets.

All data field attacks can be described by the following parameters. Given an ID, a field within that ID's data protocol, for a duration d seconds that field is changed in one of the following ways:

Modification: the field is set to a constant value, such as the maximum or minimum possible value, or to an arbitrary constant.

Replay: the field is replaced with data from the same field captured at a different time. Only the data field is replaced. The rest of the data payload is left unchanged.[4]

1) **Data effect**: For Data effect fault injection block has been used. This block allows to conduct fault injection experiment.

Specify:

- name of the block instance.
- type of fault: Stuck-at, Package drop, Bias/Offset, Bit flips, Time delay, Noise.
- fault event: Failure probability, Mean Time to Failure, Failure Rate Distribution.
- fault effect: Once, Constant time, Infinite time, Mean Time to Repair.[3]

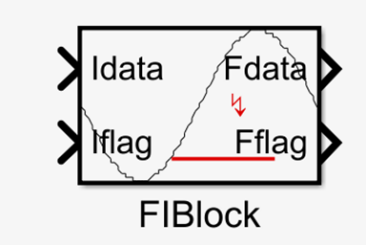


Figure 6: Fault Injection Block

2) **Frequency effect:** For frequency effect simulation time is set to 0.01 i.e., 10ms while timestamp for data sample is 0.005 i.e., 5ms. This will cause transmission of data at every 10ms over CAN bus skipping alternate data sample leading to frequency effect. Total data samples are 30000 and due to frequency effect data sample received are 15000.

8 Observations

The graph shows the comparison between original, data effect and frequency effect data.

Figure 7: Comparison between data

From above result we can see that the data transmitted (GPS speed) at time of 0.6 second is different in comparison with original data of 11.3 kmph. For data effect it is 11.2 kmph while for frequency effect is 6.5 kmph.

Thus, conclude injection of anomalies in CAN communication using Simulink model in real time using data and frequency effect.

1) Comparison between original and data effect data.

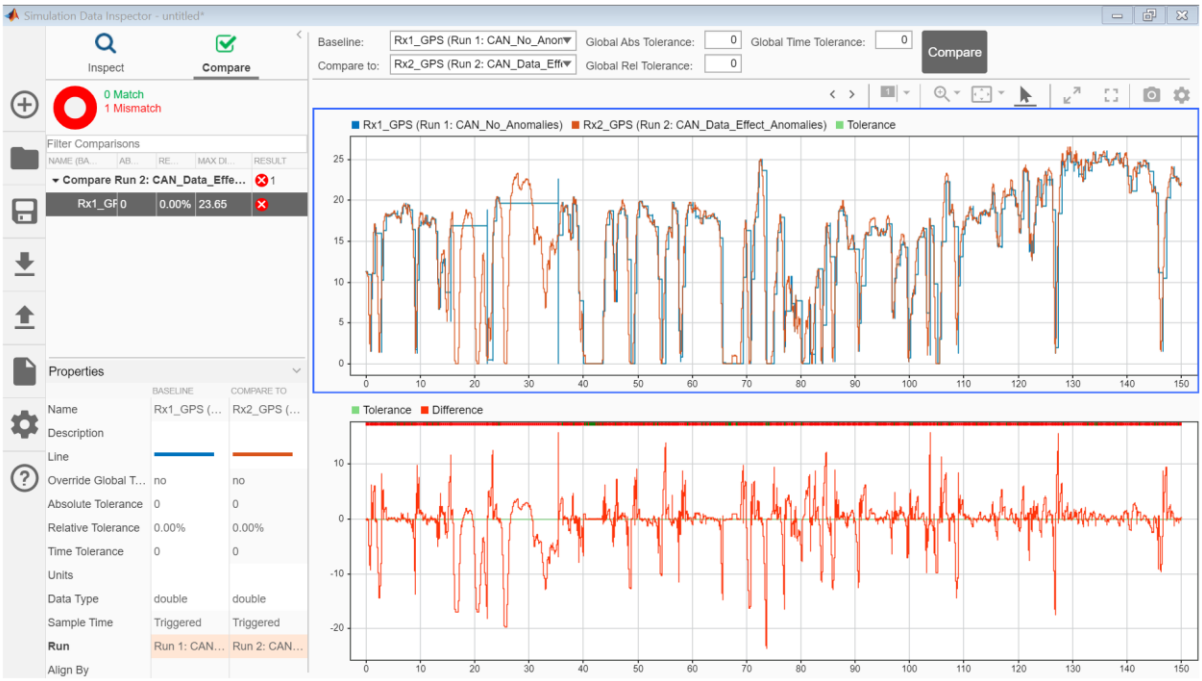


Figure 8: Comparison between original and data effect data

Data effects describe how data values can be changed in attack traffic. In this project, the fault is introduced using fault injection block. With fault type as bias/offset with different failure probability i.e., for GPS_Speed signal failure probability is 0.001, for Accel signal is 0.1 and for that of Vehicle_Speed signal is of 0.01. This causes introduction of fault while transmitting.

From above figure we can clearly see the difference between Rx1_GPS (indicated by blue line in graph (1)) i.e., CAN signal with no anomalies and Rx2_GPS (indicated by red line in graph (1)) i.e., CAN signal with data anomalies.

Multiple type of data fault can be introduced like bit flip, noise and so on by changing the setting of fault injection block and failure probability.

2) Comparison between original and frequency effect data.

Figure 9: Comparison between original and frequency effect data

Frequency effects are in the context of all known IDs being periodic with fixed frequencies. They are defined as insertions of extra packets, or the erasure of expected packet.

In this project, the fault is introduced by erasure of expected packet. To do so the data is transmitted at the timestamp of 10ms whereas the timestamp for transmission of signal is 5ms as mentioned in database file. Thus, erasing alternate packets while transmitting. The total packets to be transmitted are 30000 whereas frequency effect cause transmission of only 15000 packets.

From above figure we can clearly see the difference between Rx1_GPS (indicated by blue line in graph (1)) i.e., CAN signal with no anomalies and Rx3_GPS (indicated by orange line in graph (1)) i.e., CAN signal with frequency anomalies.

Frequency effect can be change by changing simulation time in MATLAB whereas keeping timestamp of data same i.e., of 5ms.

9 Usage Instructions

Following are the set of instructions that a user must follow for creation of CAN.dbc, MATLAB Simulink model for CAN communication.

1) CAN.dbc File

Make sure to have appropriate CAN.dbc file with all required signals and ID's. Refer part 7.1 for creation of CAN.dbc file.

2) CAN communication setup, using MATLAB and Simulink.

*Make sure main workspace is added to the path.
To “add to path” right click on workspace which include all the Simulink model and folder containing fault injection block taken from [3]. Then select add to path with folders and subfolders.

1) Check available CAN HW devices on system using MATLAB command:

```
canHWInfo
```

Command Window

```
>> canHWInfo

ans =

CAN Devices Detected

   Vendor    |  Device    | Channel | Serial Number |              Constructor
  ---------- | ---------- | ------- | ------------- | ------------------------------------
  MathWorks  | Virtual 1  | 1       | 0             | canChannel('MathWorks','Virtual 1',1)
  MathWorks  | Virtual 1  | 2       | 0             | canChannel('MathWorks','Virtual 1',2)
  Vector     | Virtual 1  | 1       | 0             | canChannel('Vector','Virtual 1',1)
  Vector     | Virtual 1  | 2       | 0             | canChannel('Vector','Virtual 1',2)

Use GET on the output of canHWInfo for more information.
```

This will give information regarding all available CAN device that can be selected for CAN communication.

2) Load Accel, Epoch, GPS Speed, and Vehicle Speed data sample into the workspace:

```
load Accel
load Epoch
load GPS_Speed
load Vehicle_Speed
```

This will load all the data samples along with its timestamp for transmission of data.
This data from workspace is used as data parameter in Simulink block “From Workspace” for continuous transmission of signal.

Simulink Setup:
CAN model is implemented using Vehicle Network Toolbox as shown in figure 4 and 5 respectively.
Below is the setting for each block used.

3) CAN Configuration Block:
- Device: MathWorks Virtual Channel 1
- Bus Speed: 500000
- Acknowledge mode: Normal

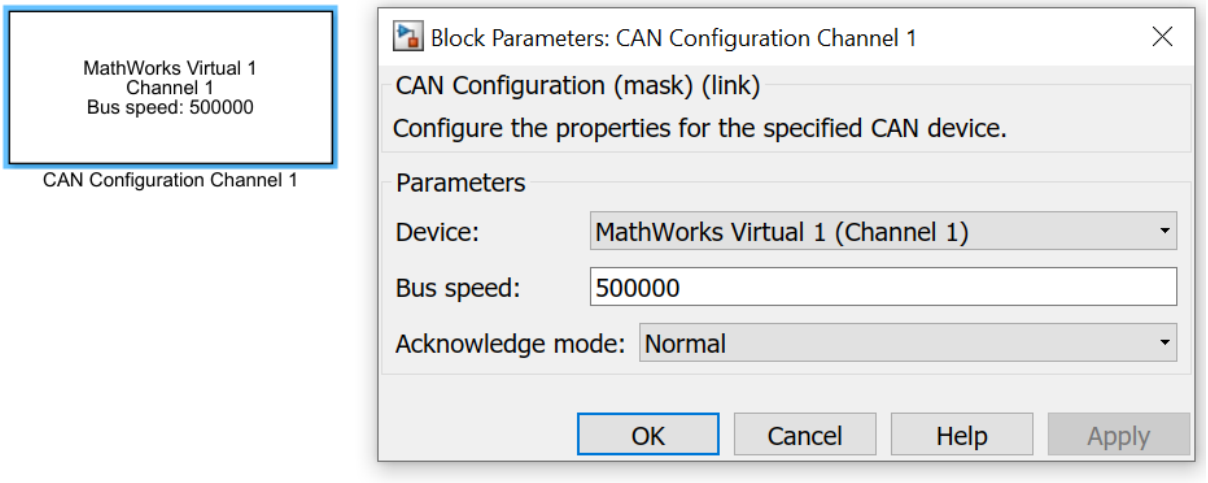


Figure 10: CAN configuration block settings

- **CAN Transmission Model**

4) From Workspace:
We already loaded data samples in workspace from point 2.
- Data: From workspace (Here it is Epoch, GPS_Speed, Accel, and Vehicle_Speed)
- Output data type: double
- From output after final data value by: Holding final value

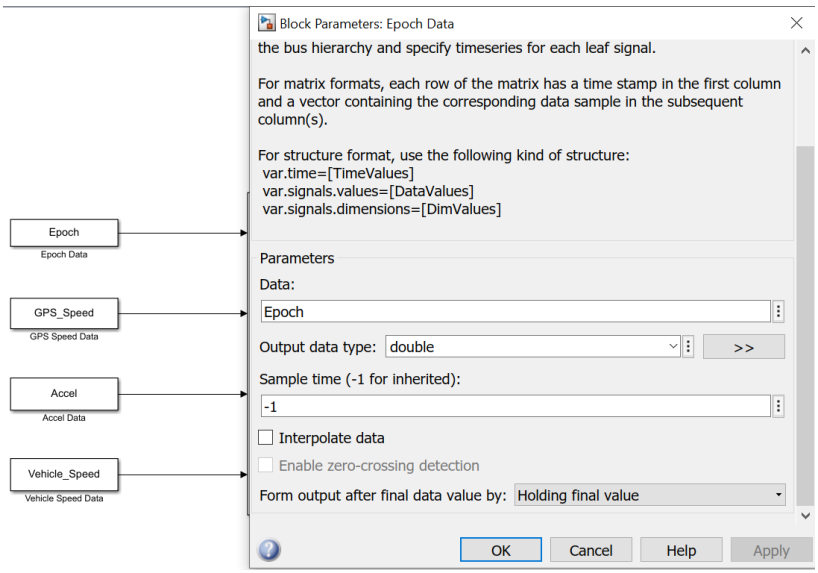


Figure 11: From Workspace block settings

5) CAN Pack Block:
Used to pack data into CAN message format

- Data is input as: CANdb specified signals
- CANdb file: select .dbc file in this case SPMD.dbc is used.
- Output as bus: check

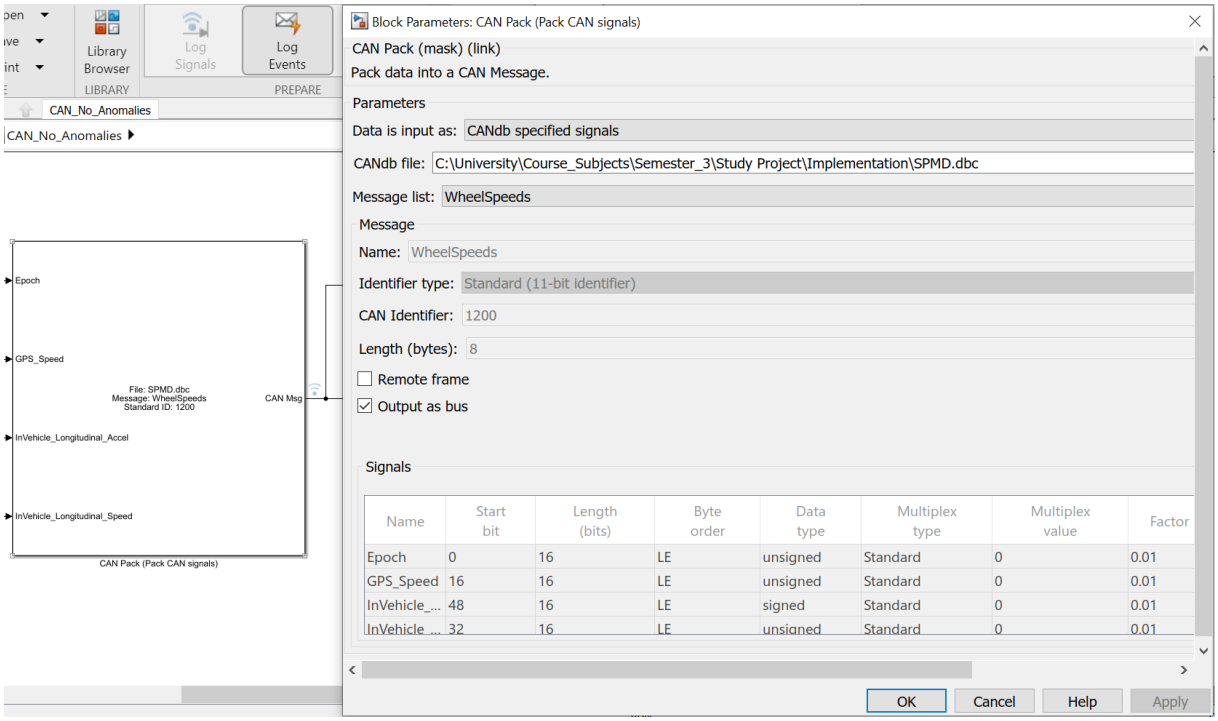


Figure 12: CAN Pack block settings

6) CAN Transmit Block:

- Device: MathWorks Virtual Channel 1

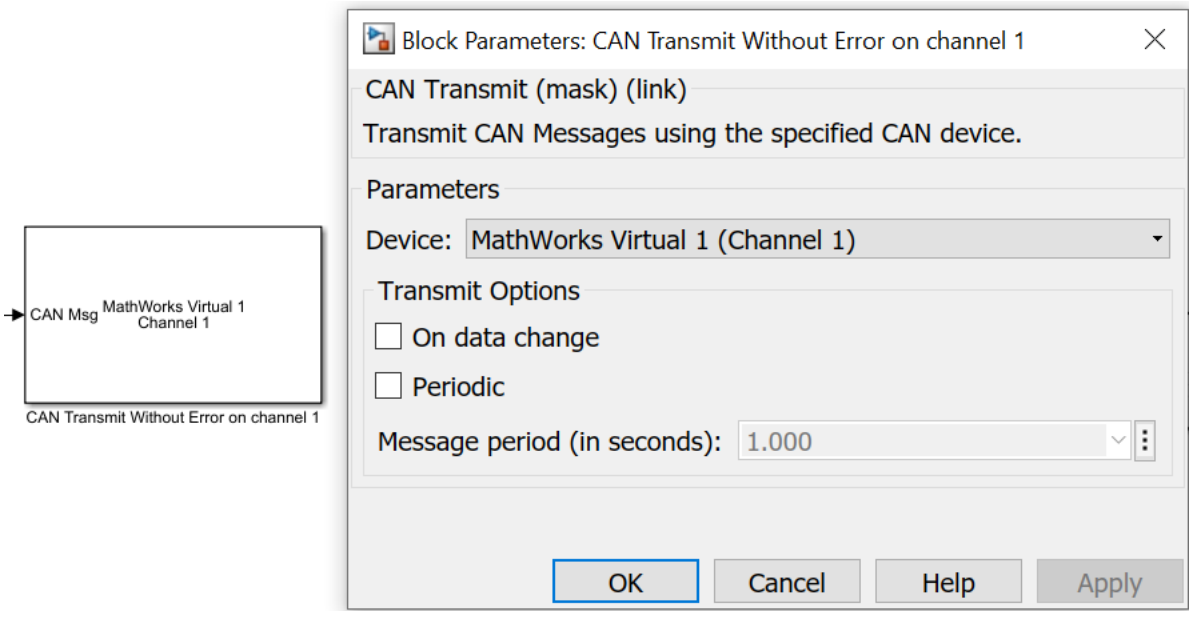


Figure 13: CAN Transmit block settings

- **CAN Reception Model**

7) CAN Receive Block:
- Device: MathWorks Virtual Channel 1
- If you want to filter out some specific IDs use IDs filter setting.
- Sample time to: -1

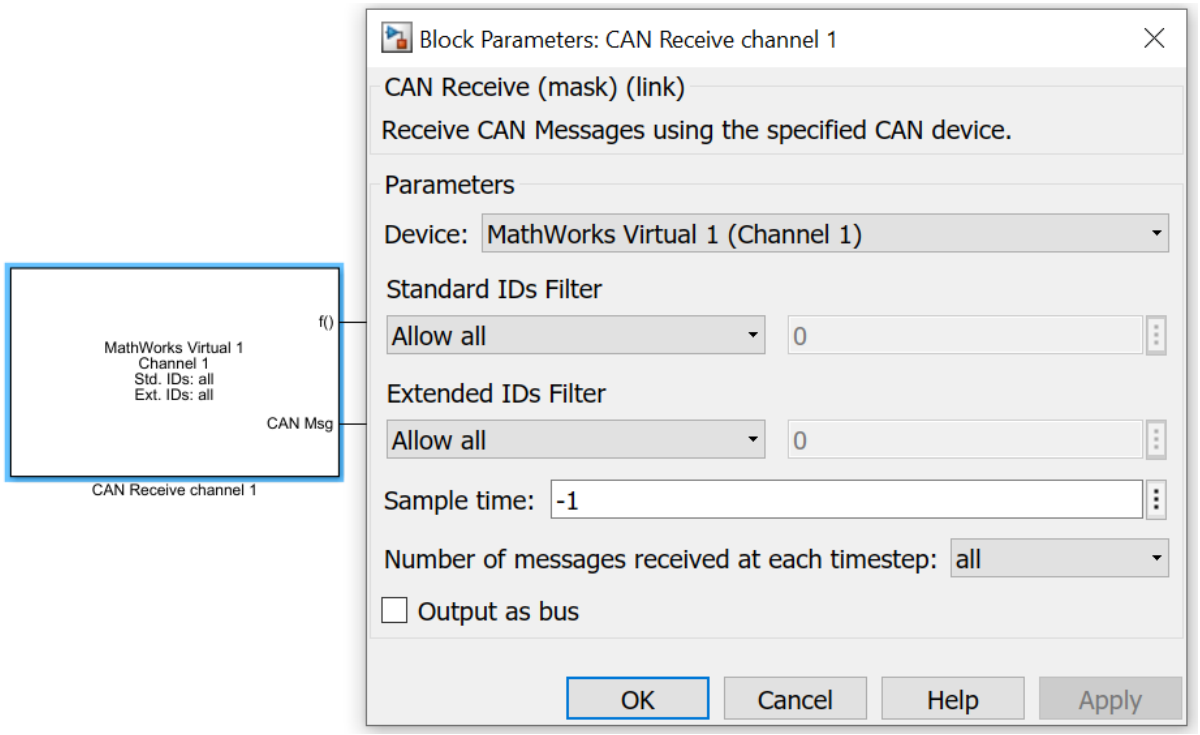


Figure 14: CAN Receive block settings

8) Add Function Block from Ports & Subsystems

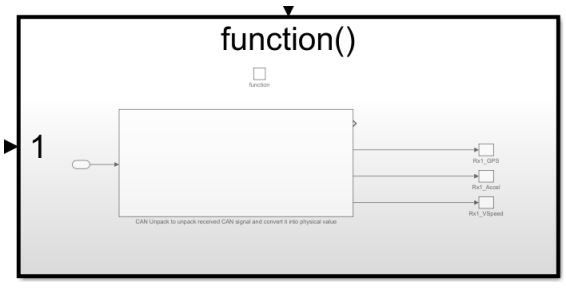


Figure 15: Function block

9) Add CAN Unpack Block inside function block.

- Data to output as: CANdb specified signals
- CANdb file: select .dbc file in this case SPMD.dbc is used.

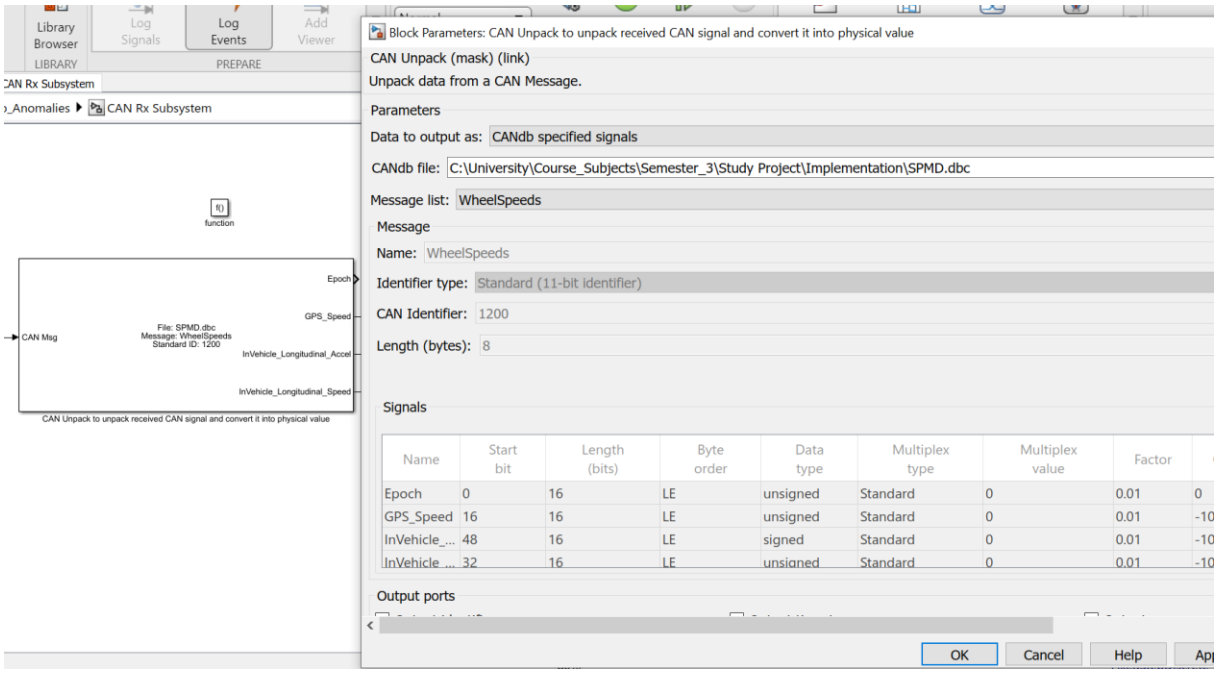


Figure 16: CAN Unpack block settings

10) Check all connections as show in figure 4 and 5.
11) Save and run Simulation.

For Data effect:
Get fault injection updated folder from MathWorks or GitHub of [3]. Include this folder in your workspace and add path for this folder.

Add this block to input signal before CAN pack block during CAN transmission.

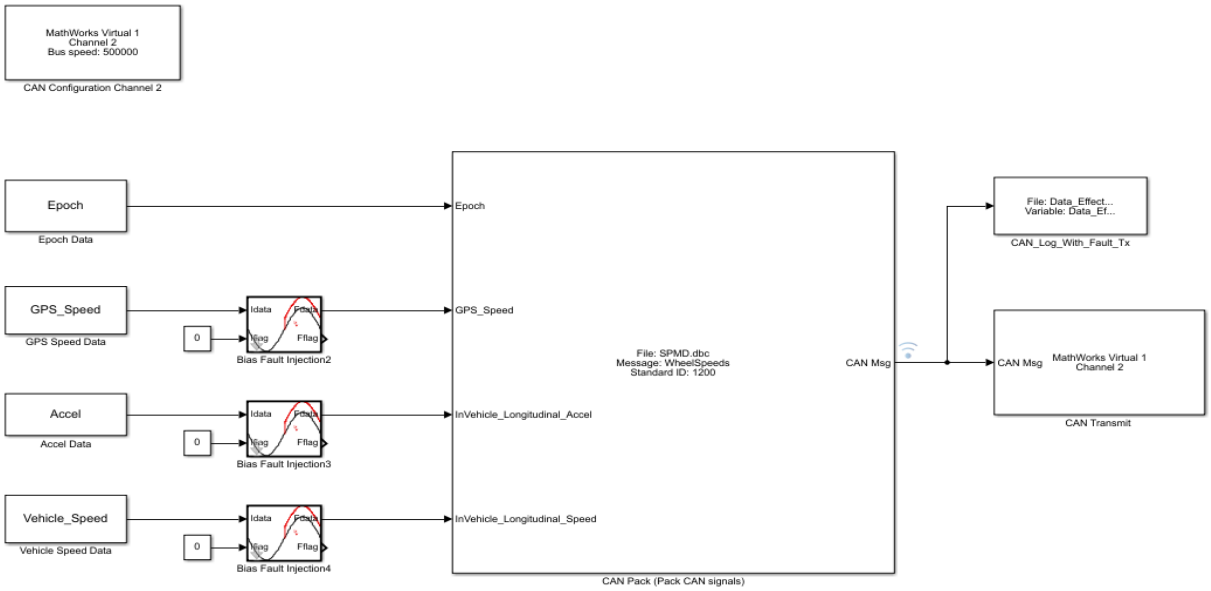


Figure 17: Data effect Simulink Model

Parameter:

- Numebr of the FI block instance (Name).
- Fault type: Stuck-at, Package drop, Bias/Offset, Bit flips, Time delay, Noise.
- Fault injection method: Failure probability, Mean Time to Failure, Failure Rate Distribution.
- Fault injection effect: Once, Constant time, Infinite time, Mean Time to Repair.[3]

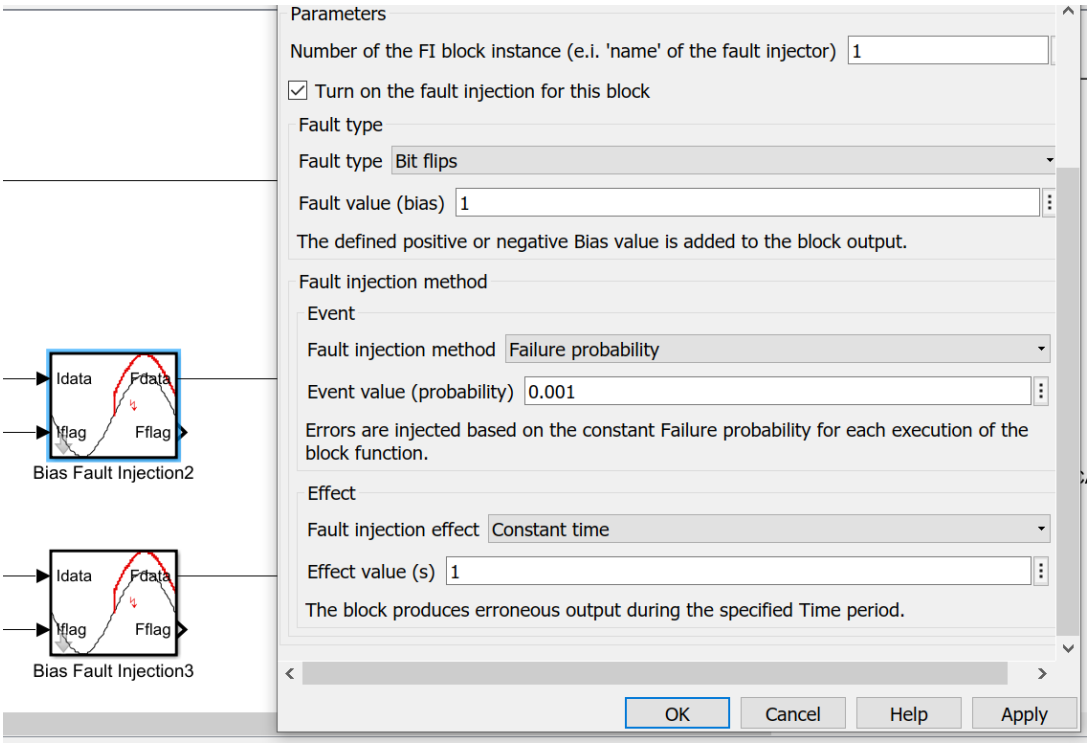
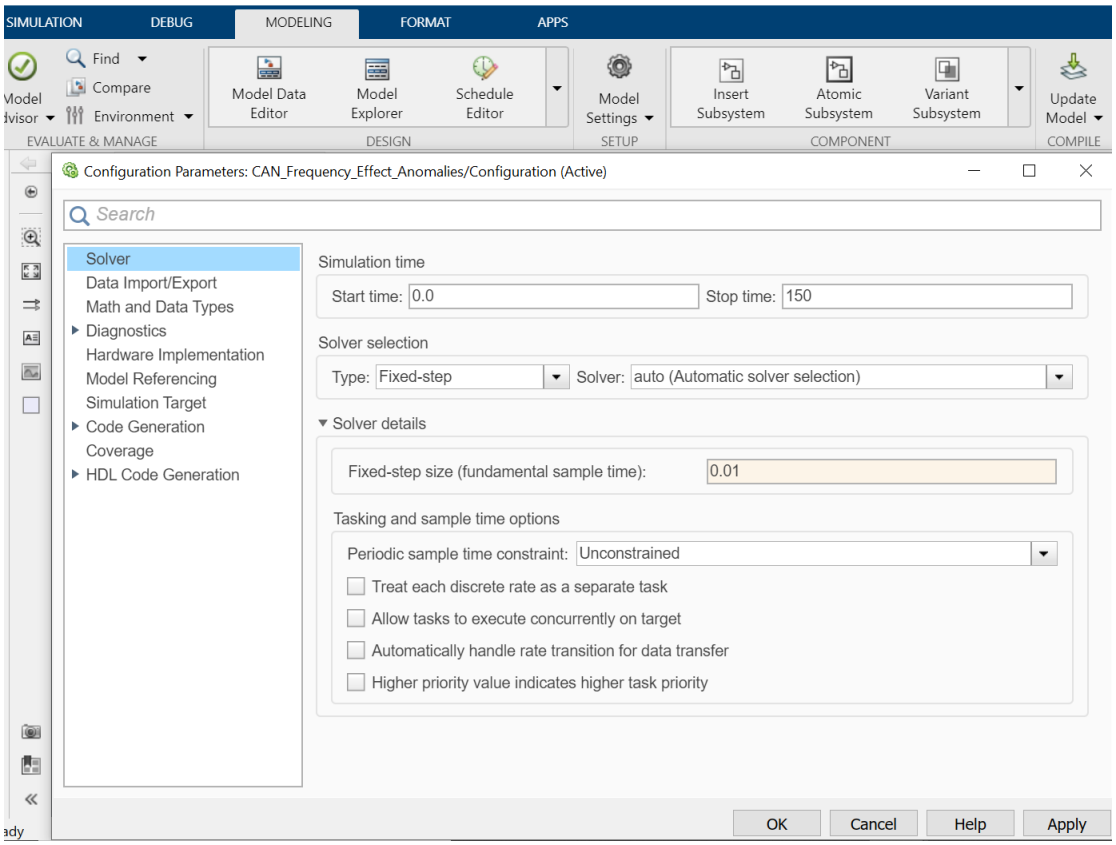


Figure 18: Fault Injection block settings

For Frequency effect:
CAN transmission and reception block same as figure 4 and 5.

In Simulink window inside Modeling tab, model settings change solver setting to Solver Selection: Type – Fixed Step, Solver – discrete, Fixed step size – 0.01.



10 Experiences

The overall experience of working on this project was a good one for me. Learnt more of MATLAB Simulink, few concepts related to CAN database creation, the CAN communication protocol, and how anomalies can occur in real world and its consequences. There were a few challenges in between, had to get all the things setup and working initially but it was a great experience overall. The faculty in IAS was very co-operating throughout the entire process and indeed it was a great fun working and learning in this project.

11 Problems

The idea of having the transmission of 30000 data continuously over CAN bus was initial hurdle faced and injection of fault was bit challenging.

12 Summary

The project focussed on transmission of sensor data over the CAN bus and injection of anomalies into that data in real time.

13 Future Scope

The fault injected data can further be used for anomaly detection and analysis. Analysis to find out source of fault/anomalies. This detection can be done using Kalman filter or CNN.

14 References

[1] C. Segler, S. Kugele, P. Obergfell, M. H. Osman, S. Shafaei, E. Sax, A. Knoll, "Anomaly Detection for Advanced Driver Assistance Systems Using Online Feature Selection," in IEEE Intelligent Vehicles Symposium (IV), June 2019, pp.2.

[2] F. Wyk, Y. Wang, A. Khojadi, "Real-Time Sensor Anomaly Detection and Identification in Automated Vehicles," in IEEE transactions on intelligent transportation systems, August 2018.

[3] Tagir Fabarisov, https://www.mathworks.com/matlabcentral/fileexchange/75539-fault-injection-block-fiblock.

[4] C. Wang, Z. Zhao, L. Gong, L. Zhu, Z. Liu, X. Cheng, "A Distributed Anomaly Detection System for In-Vehicle Network Using HTM," in IEEE Recent Advances on Radio Access and Security Methods in 5G Networks, January 2018, pp.3.

[5] https://www.csselectronics.com/pages/can-dbc-file-database-intro