FACULTY OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

EE 401 – Senior Project I

< 2019 / 2020 FALL >

FINAL REPORT

Design and Real-Time Simulation of an Energy-Efficient Control Algorithm on OpenECU with HIL Simulators for Plug-in Electric Vehicles

MEHMET BATU ÖZMETELER
S009924

Jury Members

Supervisor:	Prof. H. Fatih Uğurdağ
Jury member 2:	Assistant Prof. Evşen Yanmaz Adam
Jury member 3:	Assistant Prof. Cenk Demiroğlu

Table of Contents

Table of Figures

Design and Real-Time Simulation of an Energy-Efficient Control Algorithm on OpenECU with HIL Simulators for Plug-in Electric Vehicles

1. Introduction

1.1. Problem Statement

Global warming emissions and air pollution have become major threats for our planet. According to NASA: Climate Change and Global Warming website, the current warming trend considered to be crucial due to the majority of it is quite likely (greater than 95 percent probability) to be the consequence of human activity since the mid-20th century and advancing at a rate that is remarkable over decades to millennia [1]. Since carbon dioxide emissions from the combustion of fossil fuels are one of the primary causes for both issues [2], reducing oil consumption can help avoid such hazards.

Reducing dependence on petroleum which has turned into a scarce resource is a difficult task. However, thanks to progresses in electrically powered transportation technology, electric vehicles (EVs) are conceived to be a relevant solution. As reported by Bloomberg New Energy Finance, electric vehicles are expected to discard two million barrels of oil a day by 2028 [3].

The success of electric vehicles can be connected to them having fewer moving parts meaning lower operating and maintenance costs. The weight is ideally distributed front-to-rear below the car's center of gravity making it more stable and safer to drive. Reduced noise pollution is another benefit for the greater good. Additionally, the instantaneous torque response and full torque from standstill is practical. However, the long recharge time, the limited speed and cruising range are probably the most critical drawbacks of EVs.

1.2. Project Description

For the purpose of increasing the popularity of EVs to rapidly shift into a greener era, this project aims to help enhance the driving range of PHEVs. Improving this competence can be achieved by only two possibilities: develop a battery technology which allows higher capacity and lower weight or come up with an energy management strategy that influences the driving style in order to optimize the energy consumption. Since soon improvements on batteries is uncertain [4], second option seems to be more suitable to address this challenge. But how can this issue be solved when testing on automobiles is costly, unsafe and not easily feasible?

2. Literature Review

2.1. Predicting the Remaining Driving Range for EVs

The limited driving range has been revealed to be one of the main technical factors affecting the acceptance of EVs. In 2013, Javier A. Oliva, Christoph Weihrauch and Torsten Bertram have developed an approach that predicts the RDR under a stochastic framework since many stochastic factors such as the driving style, the traffic situation, the road conditions or the weather largely influence it [5]. The basic theoretical foundation of this study is based on the work introduced by Daigle and Goebel (2011) [6], in which a model-based approach is seen to be applied for predicting the remaining useful life (RUL) of pneumatic valves. In the figure below, their RDR prediction architecture is depicted.

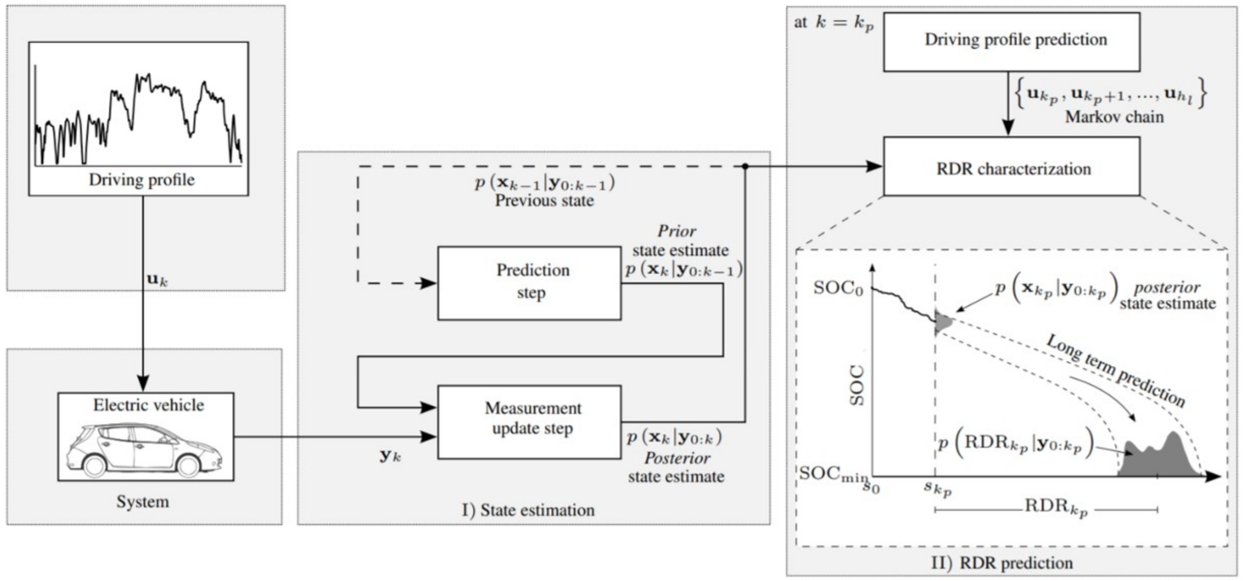

Driving profile
$\mathbf{u}_k$
Electric vehicle
System
$p(\mathbf{x}_{k-1}|\mathbf{y}_{0:k-1})$ Previous state
Prediction step
Prior state estimate $p(\mathbf{x}_k|\mathbf{y}_{0:k-1})$
Measurement update step
$\mathbf{y}_k$
$p(\mathbf{x}_k|\mathbf{y}_{0:k})$ Posterior state estimate
I) State estimation
at $k = k_p$
Driving profile prediction
$\{\mathbf{u}_{k_p}, \mathbf{u}_{k_p+1}, \ldots, \mathbf{u}_{h_l}\}$ Markov chain
RDR characterization
SOC_0
$p(\mathbf{x}_{k_p}|\mathbf{y}_{0:k_p})$ posterior state estimate
Long term prediction
SOC
$p(\mathrm{RDR}_{k_p}|\mathbf{y}_{0:k_p})$
$\mathrm{SOC}_{\min}$
s_0
s_{k_p}
RDR_{k_p}
II) RDR prediction

Figure 1: RDR Prediction Architecture [5]

A model-based approach for predicting the RDR by combining unscented filtering and Markov chains is introduced in this paper. It is also included in their article that at the beginning of the estimation and prediction steps; the system states, and especially the initial state of charge, are unknown. To obtain an accurate prediction of the RDR, they state that the prediction module needs an initial starting point that is as accurate as possible. For that reason, the state estimation must converge to the true value before the prediction is carried out. Thus, they use state estimation algorithms for nonlinear systems.

2.2. Designing a Real-Time Implementable Energy-Efficient Model-Predictive Cruise Controller for EVs

T. Schwickarta, H. Voosa, J. R. Hadji Minagloub, M. Darouach and A. Rosicha has presented the design of a novel energy-efficient model-predictive cruise controller for electric vehicles as well a simulation model of the longitudinal vehicle dynamics and its energy consumption in 2013 [7]. Both, the controller and the dynamic model they designed meets the properties of a series-production electric vehicle whose characteristics are identified and verified by precise measurements. This eco-cruise controller involves the minimization of a compromise between terms related to driving speed and energy consumption which are in general both described by nonlinear differential equations.

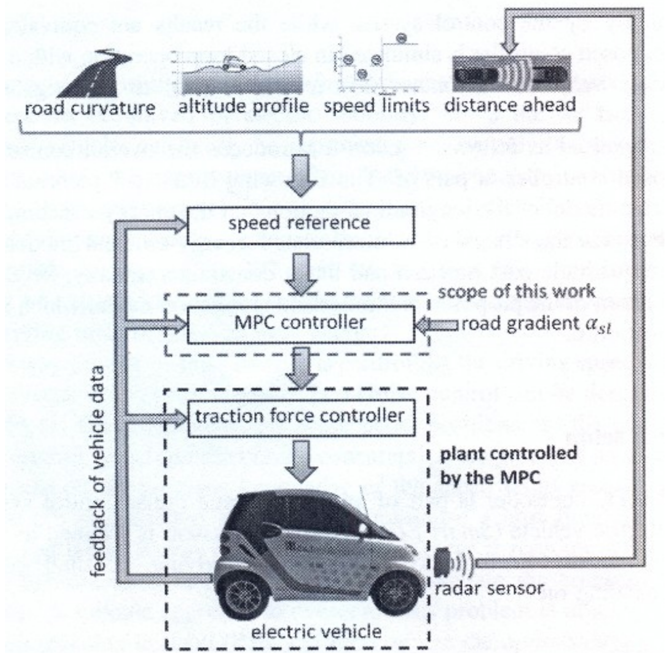
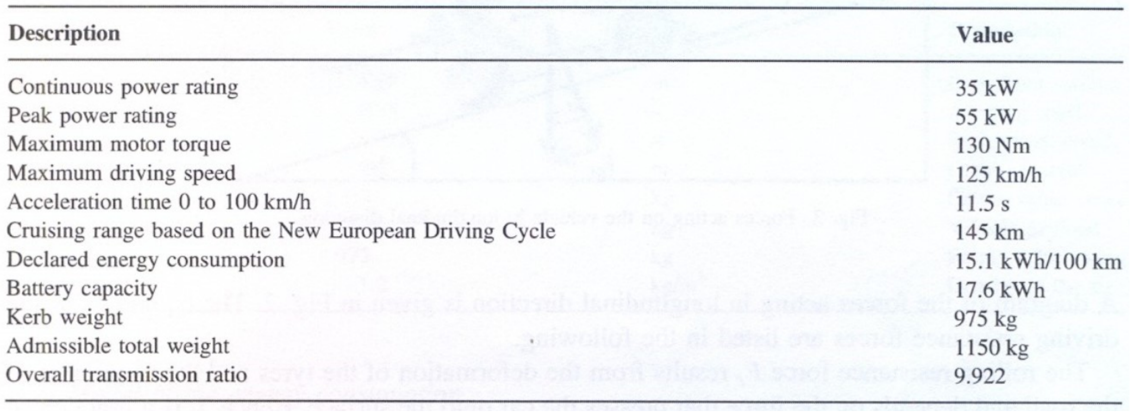


Figure 2: Predictive Cruise Control System for Energy-Efficient Driving [7]

In their study, considering the nonlinearities is said to be essential for a proper prediction of the system states over the prediction horizon to achieve the desired energy-saving behavior.



Description	Value
Continuous power rating	35 kW
Peak power rating	55 kW
Maximum motor torque	130 Nm
Maximum driving speed	125 km/h
Acceleration time 0 to 100 km/h	11.5 s
Cruising range based on the New European Driving Cycle	145 km
Declared energy consumption	15.1 kWh/100 km
Battery capacity	17.6 kWh
Kerb weight	975 kg
Admissible total weight	1150 kg
Overall transmission ratio	9.922

Figure 3: Main Specifications of the Test Vehicle Smart ED [7]

In this work, it's mentioned that the vehicle motion equation is reformulated in terms of the kinetic energy of the moving vehicle which leads to a linear differential equation without loss of information. The energy consumption is modeled implicitly by exploiting the special form of the optimization problem. The reformulations finally lead to a model-predictive control approach with quadratic cost function, linear prediction model and linear constraints that corresponds to a piecewise linear system behavior and allows a fast real-time implementation with guaranteed convergence. Simulation results of the MPC controller and the simulation model in closed-loop operation finally provide a proof of concept.

3. Methodology

3.1. Suggested Solution

The answer suggested in this project for this complication lies in having a specific energy-efficient control algorithm for longer driving ranges by manipulating the driving speed according to several factors. The main deliverable for this project concerning the first term is a reciprocal working demonstration that can be elaborated like so: running the modified algorithm on an electronic control unit while receiving the directly provided necessary data inputs and meanwhile having the process being simulated on a model thoroughly by a real-time HIL simulator in order to show how the simulation provides testing feasibility and enhances coverage. This is a beginning step for concluding a real-time optimization problem which corresponds that the range can be extended by minimizing a cost function that is going to be studied on the next semester.

During this current stride, essential software tools and hardware technicalities will be grasped profoundly and experimented on. A methodology of trial and error and I/O manipulation via a calibration tool will be used to achieve the desired outcome from the simulation process. The block diagram for ECU software development supported with a HIL Simulator for testing, debugging and tracking is given below:

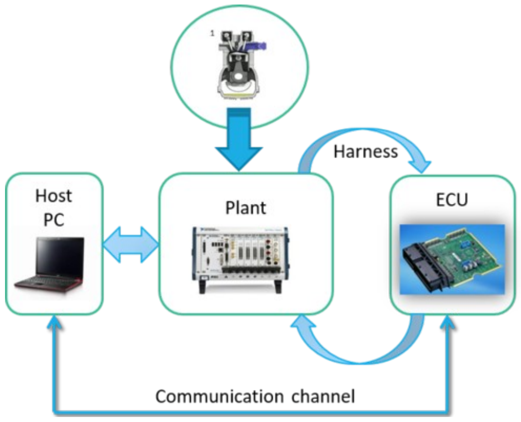


Figure 4: Block Diagram of the Setup [8]

3.2. Utilized Hardware

3.2.1 Electronic Control Unit (ECU)

Figure 5: Pi Innovo M250 Module [9]

ECU is a name given to a device which controls one or more electrical systems in a vehicle. It provides instructions for various electrical systems, guiding them on what to do and how to operate. In this case, it will be carrying a distinct control algorithm as its set of rules. For this project, M250 module from Pi Innovo is selected due to its useful features instead of other products from the M-series.

The M250 module is a compact electronics module that is suited to computationally intensive applications requiring a chassis mounted, sealed metal housing with IP67 environmental protection. It has a NXP MPC5534 as a 32-bit MCU (80Mhz), 2 CAN interfaces along with a 46-pin connector. The code space is 512 KB, RAM space is 64 KB and the calibration space is 256 KB. [9]

3.2.2 Hardware-In-The-Loop Simulator (HIL)

The instructions provided by the ECU will be followed by a HIL (Hardware-in-the-Loop) simulation which mimics the physical part of the system, the plant. It is a technique to test control systems, hereby representing the car that has many actuators and sensors. Hereby, Opal-RT OP4200 is used due to it offering Hardware-in-the-Loop (HIL), Rapid Control Prototyping (RCP), data acquisition and I/O expansion capabilities in a desktop-friendly package. [10]

Figure 6: Opal-RT OP4200 [10]

3.2.3 Power Source

A power source of AATech APS-3303DD is used to power the whole system with the required energy.

Figure 7: AATech APS-3303DD

3.2.4 USB-CAN Interface

In order to connect host PC(s) to the CAN interface, a KVASER Leaflight cable is used as an USB-CAN interface. The alternatives are CANable, PEAK-System etc. This product is preferred since its installation is quick and easy (plug-and-play). It supports both 11-bit (CAN 2.0A) and 29-bit (CAN 2.0B active) identifiers while being 100% compatible with applications written for other Kvaser CAN hardware with Kvaser CANlib. A high-speed CAN connection (compliant with ISO 11898-2), up to 1 Mbit/s is obtainable. Furthermore, it is fully compatible with J1939, CANopen, NMEA 2000R and DeviceNet. [11]

Figure 8: Kvaser Leaflight HS-V2 [11]

3.2.5 Host PC(s)

Host PCs have the required software installed on them. They run Win 7 SP1 64-bit operating systems for third-party tool compatibility.

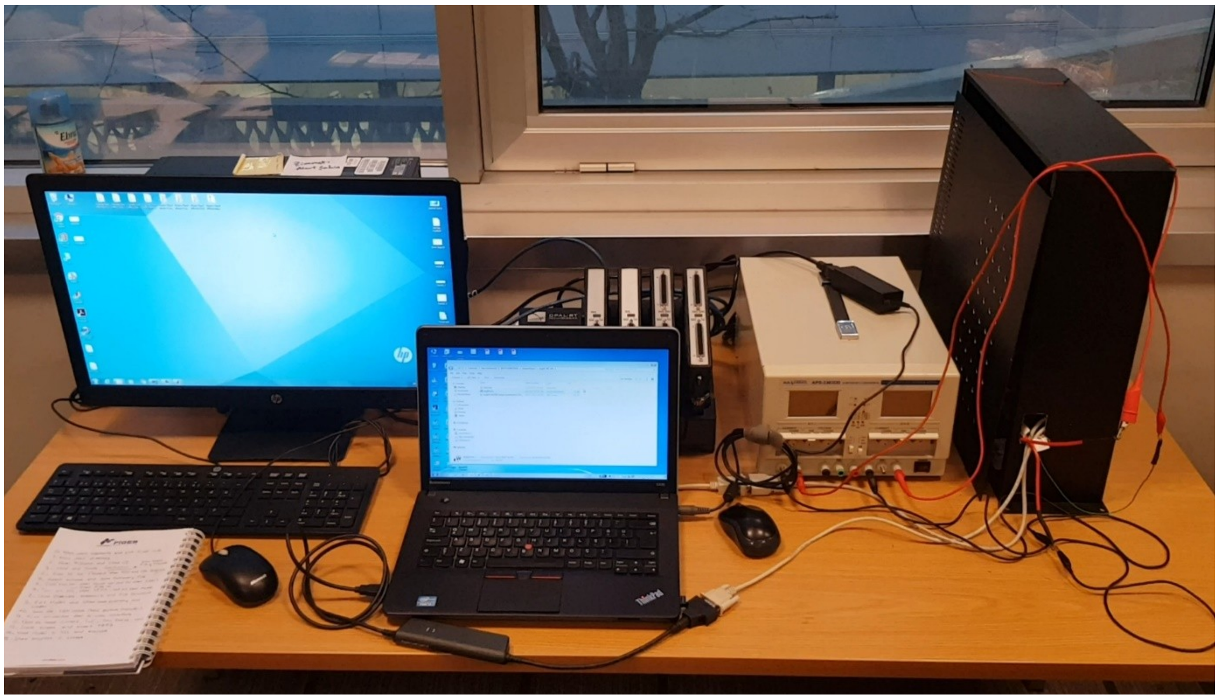


Figure 9: The Hardware Setup

3.3. Utilized Software

3.3.1 MATLAB, Simulink and Other Related Toolboxes

MATLAB is a programming platform which is based on a matrix-based language that allows the most natural expression of computational mathematics. Implementing, testing and debugging algorithms is easy with the help of the large database of built-in functions. External libraries can be called, integrated to perform extensive data analysis and visualization. Working with OpenECU library which is the compatible software to our hardware is much simpler due to the effortless integration. Also, using Simulink for model-based design is advantageous. For these reasons, MATLAB is preferred instead of other high-level languages.

3.3.2 Calibration Tool and its Related Libraries

One of the essential third-party tool requirements for OpenECU is a calibration tool. This is used to interact with software while it runs in real-time on an embedded system or an ECU. It makes use of a communication link to access the memory for read/write operations, loads information (memory layout, variables, functions) about the application. Among other possible options like ETAS INCA, Vector CANape, ATI Vision; Pi Snoop Professional is chosen. Having a tool which helps for software debugging, testing, flash reprogramming, calibration editing, CAN bus work and diagnostics is more than enough. **[12]**

3.3.3 Real-Time Simulation Software

RT-LAB is OPAL-RT's real-time simulation software which is fully integrated with MATLAB/Simulink. It offers the most complex model-based design for interaction with real-world environments. In order to turn models into interactive real-time simulation applications for automotive industry, RT-LAB has the necessary flexibility and scalability. It handles everything, including code generation, with an easy-to-use interface. **[13]**

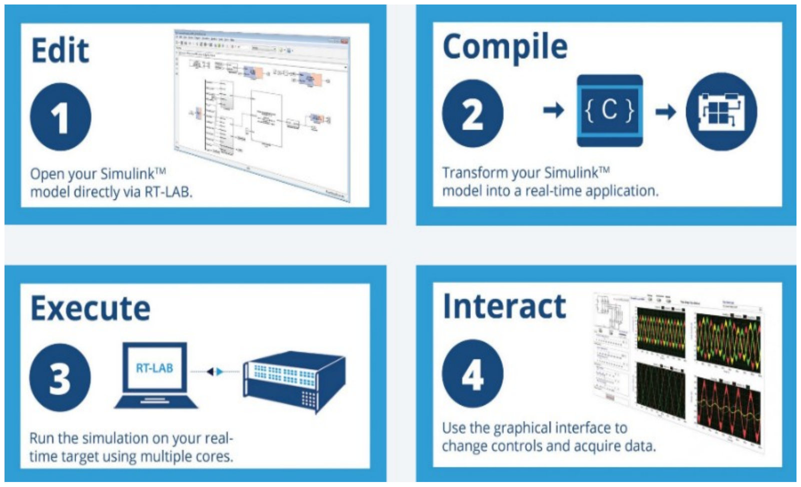

Edit
1
Open your Simulink™ model directly via RT-LAB.
Compile
2
{ C }
Transform your Simulink™ model into a real-time application.
Execute
3
RT-LAB
Run the simulation on your real-time target using multiple cores.
Interact
4
Use the graphical interface to change controls and acquire data.

Figure 10: 4 Steps to Real-Time Simulation [13]

3.4. Milestones

1. Profound Research on Related Subjects (papers, books, products etc.)
2. Getting Familiar with OpenECU library and Simulink Through Examples
3. Getting Familiar with HIL Testing and Modifying the Control Algorithm
4. Modifying the Vehicle Model and Setting the Algorithm Up for Testing
5. HIL Simulation for the Energy-Efficient Control Algorithm on the Given Model
6. Fixing Bugs
7. Practicing for the Demonstration
8. Gathering Material and Writing the Final Report

3.5. Gantt Chart

TASK NAME	START DATE (Initially Planned)	END DATE (Initially Planned)
Profound Research on Related Subjects (papers, books, products etc.)	4/9/2019	18/9/2019
Getting Familiar with OpenECU library and Simulink Through Examples	18/9/2019	31/10/2019
Getting Familiar with HIL Testing and Modifying the Control Algorithm	31/10/2019	15/11/2019
Modifying the Vehicle Model and Setting the Algorithm Up for Testing	15/11/2019	6/12/2019
HIL Simulation the Energy-Efficient Control Algorithm on the Given Model	6/12/2019	16/12/2019
Fixing Bugs	16/12/2019	23/12/2019
Practicing for the Demonstration	23/12/2020	27/12/2020
Gathering Material and Writing the Final Report	27/12/2019	14/12/2019

Figure 11: Gantt Chart

3.6. Days Spent on Tasks

TASK NAME	DURATION (Initially Planned)	DURATION (Actual Load)
Profound Research on Related Subjects (papers, books, products etc.)	14	30
Getting Familiar with OpenECU library and Simulink Through Examples	13	20
Getting Familiar with HIL Testing and Modifying the Control Algorithm	15	15
Modifying the Vehicle Model and Setting the Algorithm Up for Testing	21	15
HIL Simulation the Energy-Efficient Control Algorithm on the Given Model	10	7
Fixing Bugs	7	4
Practicing for the Demonstration	4	5
Gathering Material and Writing the Final Report	17	7

Figure 12: Days Spent on Tasks

3.7. Deliverables

DELIVERABLES	IS IT ACHIEVED?
Modified Vehicle Model	YES
Modified Energy-Efficient Control Algorithm	YES
Functioning HIL Simulator (Real-Time) with a Vehicle Model	YES
Functioning ECU with an Energy-Efficient Control Algorithm	YES
Functioning CAN Communication	YES

Figure 13: Deliverables

4. Work Done

The work done will be summarized progressively in the latter chapters. In the first place, a lot of profound reading and research was done about the associated domain. Coming next was to understand how the ECU worked with a compiled model that represents the control algorithm.

In the hope of grasping the idea better, a few examples such as:

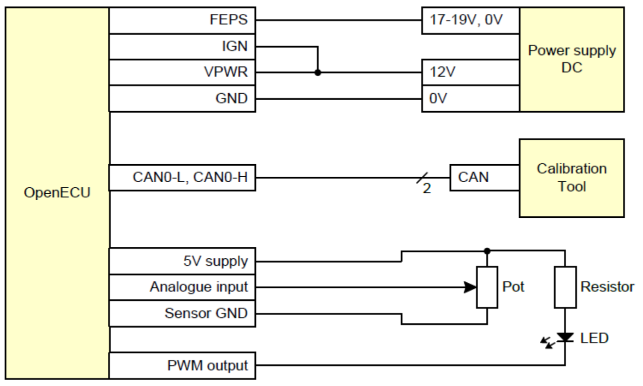
- an application that monitors a temperature sensor and activates a warning lamp when a threshold is breached
- a model which describes how blocks can be put in different tasks by setting different sample rates where values are passed between tasks of different rates by using unit-delay and zero-order hold blocks
- a CAN receiver/transmitter
- I/O block manipulation

were done from the OpenECU developer platform user guide (C-API) to practice with the OpenECU library and M-250 module. For brevity, only the first mention on the list will be explained. This exercise is chosen because it describes how to build, program, and test an application in the simplest manner possible. Also, an alternative workflow can be visually pursued from a Youtube video as well. [14]

Then, the control algorithm was modified obeying the constraints. Respectively, the vehicle model modification and setup came after which was succeeded by the HIL simulation to complete the demonstration.

4.1. A Sample Experiment

The application uses a single analogue input to measure the temperature, and a single PWM output to activate a warning lamp connected to the power supply. The pinouts and the circuit diagram for this task is depicted below.



			M460	
	M220	M250	M461	M670
FEPS	A27	A27	A2	Y22
VPWR	A2	A2	A20	Y3
GND	A31	A31	C1	Y2
IGN	A26	A26	A12	Y25
CAN0-L	A43	A43	A37	Y11
CAN0-H	A28	A28	A36	Y12
5V supply	A25	A25	C8	Y8
Analogue input	A19	A19	A28	Y31
Sensor GND	A40	A40	C9	Y9
PWM output	A37	A18	A15	Y29

	CAN0-L	CAN0-H	Terminated?
M220	A43	A28	No
M250	A43	A28	No
M460	A37	A36	Yes
M461	A37	A36	Yes
M670	Y11	Y12	Yes

External CAN termination required on loom if CAN bus not terminated

Figure 14: Pinouts and the Circuit Diagram for the Quick Start Example [15]

In an automobile, the purpose of the ignition system is to generate a very high voltage from the car's 12V battery in order to ignite the fuel-air mixture in the engine's combustion chambers by sending power to each sparkplug in turn. The IGN pin represents the ignition sense which is a power source that turns on and off with the vehicle ignition to turn various devices onboard on and off.

A27 is the FEPS (Flash EEPROM Programming Signal) pin where a positive voltage is asserted before the ECU is powered up so that reprogramming mode is triggered on the bootloader. In the end, the ECU has to be power cycled while this pin is grounded to place it in the application mode.

There are many electronic control units (e. g. engine control unit, airbags, audio system etc.) onboard (up to 70 ECUs in a modern car) which need to exchange information to work properly. However, this communication is quite hard to put into practice. The purpose of a CAN (Controller Area Network) bus is to allow any ECU to communicate with the entire system without causing an overload to the controller computer.

It is very popular due to it being centralized which helps with the error diagnosis and configuration, resilient towards failure caused by electromagnetic interference and efficient since there are CAN IDs for prioritization and relevance for action. In this experiment, CAN0 was the preferred channel over the pins A43 for CAN-Low and A28 for CAN-High. The rest of the connections can be seen in the circuit diagram.

Under the directory named **[installation_directory]\examples\c_api** for the installed OpenECU software, there is a folder named **step1_quick_start** which contains provided code for our built-in application tutorial. Subdirectories have definitive content disclosed likewise:

- **build**

The **build** directory contains a batch file which will compile and link the completed step1 implementation.

- **interface_specification**

The **interface_specification** directory the initial implementation of two files: **step1_m250.capi** which describes supporting functionality the OpenECU library will provide to the application; and **step1.dde** which contains a description of the C variables in the step1 implementation.

- **loom_specification**

The **loom_specification** directory contains pinouts and a copy of the circuit diagram.

- **src**

The **src** directory contains the initial implementation of the step1 application.

step1_m250.c file under **src** directory is edited to add necessary constants and calibration variables along with the implementation of the sensor reading, the sensor conversion to temperature, the threshold comparison and the lamp driver. An important point to touch upon is the calibration potential. A calibration is a C variable which can be modified by a calibration tool, without having to recompile the application. ECUs come in two types: fleet units, developer units. The main difference in between is that the developer unit has the capability to adjust calibrations while the app is running. In our case, M250 is a developer unit therefore it is particularly convenient since this facilitates tuning. Next, having the functionality ready; the OpenECU library should be linked to the application. There are 2 aspects to this step:

- Editing the data dictionary file **(.dde)**

This file is used to explain the C variables that will be accessible by a calibration tool while the application is running on an ECU. It has several fields all separated by a tab character: Name, Units, Description, Class, Accuracy, Min, Max, Offset. All variables defined here can be viewed from the calibration tool in real-time.

- Editing the interface specification file **(.capi)**

This file is used to detail information about the application and how it will use the library. It contains information like what CCP settings are required to communicate with the calibration tool, target ECU specifics, application properties (e. g. **tasks** are functions that are called periodically, stack size to be allocated) etc.

After all is done, the build process is executed by a GCC Compiler via clicking the **build_m250_gcc_4_7_3.bat** file under **build** directory. This generates:

• Additional code files which must be linked with the application and OpenECU library to create a binary image to program an ECU with (only **step1_m250_image_small.s37** is used)
• An ASAP2 file (**step1.a2l**) to provide access to the application's C variables while the application runs on an ECU.

Instead of following this path of actions, starting from scratch and using MATLAB commands with Simulink software to build the application has a smoother, highly automated workflow since this environment is well-integrated via OpenECU toolbox and API. Firstly, **oe_create_model** command is run to create a Simulink model and designed as below with matching configurations accordingly.

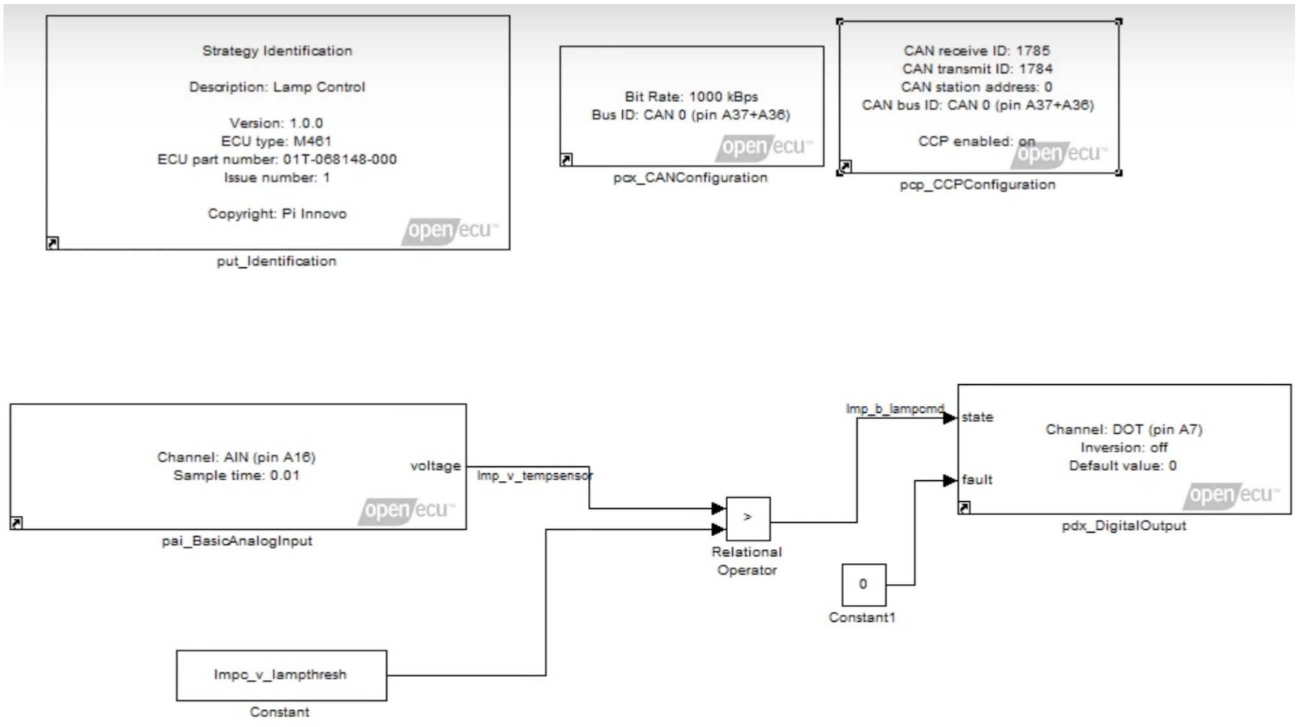


Figure 15: Simulink Model for the Quick Start Example

Later, the data dictionary file is created, filled out and **oe_read_build_list** command is run to obtain workspace data from each featured file. Build button from the Simulink interface is pressed to start real-time workshop build procedure for the specified model. This is basically where everything is handled automatically by MATLAB to generate **.s37** and **.a2l** files. Lastly, configuring the calibration tool so that it can communicate with the ECU for the reprogramming is same for both procedures. Illustrations below point out the roles of the subparts in Pi Snoop Professional's interface. Utilizing these functionalities, the ECU can be easily managed, programmed.

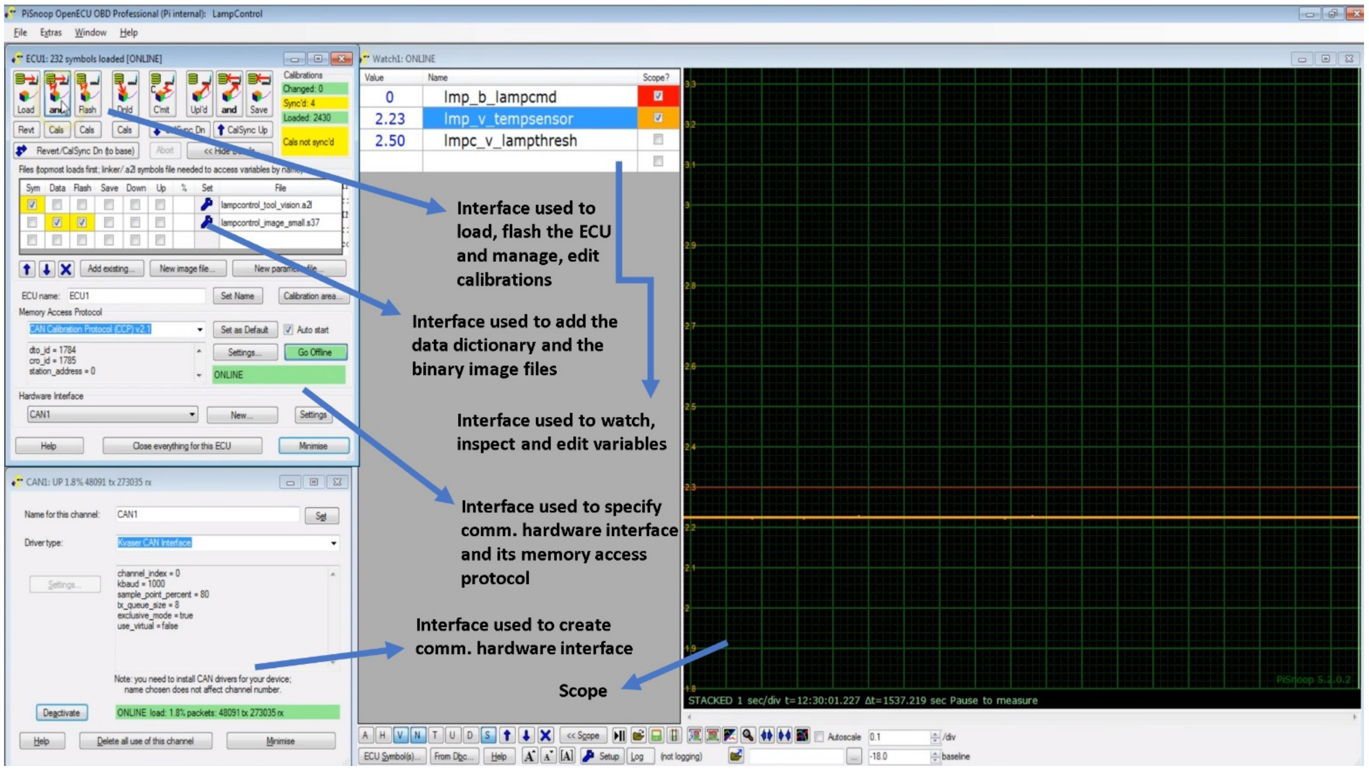

Interface used to load, flash the ECU and manage, edit calibrations
Interface used to add the data dictionary and the binary image files
Interface used to watch, inspect and edit variables
Interface used to specify comm. hardware interface and its memory access protocol
Interface used to create comm. hardware interface
Scope
PiSnoop OpenECU OBD Professional (Pi internal): LampControl
ECU1: 232 symbols loaded [ONLINE]
Watch1: ONLINE
CAN1: UP 1.8% 48091 tx 273035 rx
Imp_b_lampcmd
Imp_v_tempsensor
Impc_v_lampthresh
STACKED 1 sec/div t=12:30:01.227 Δt=1537.219 sec Pause to measure

Figure 16: Interfaces Used in Pi Snoop Professional

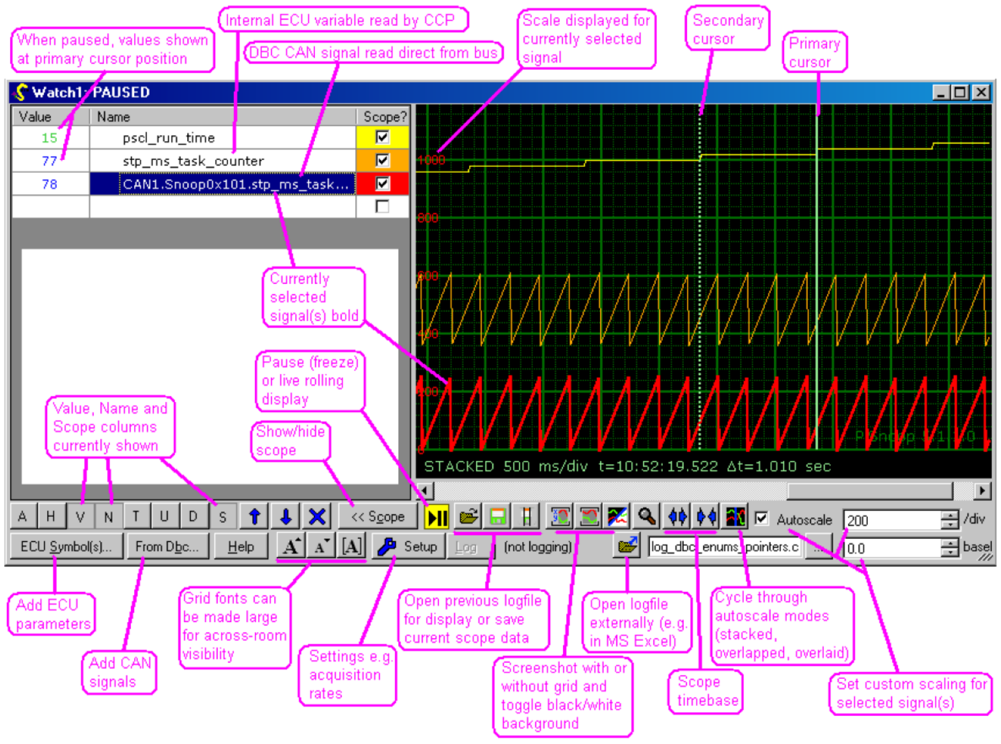
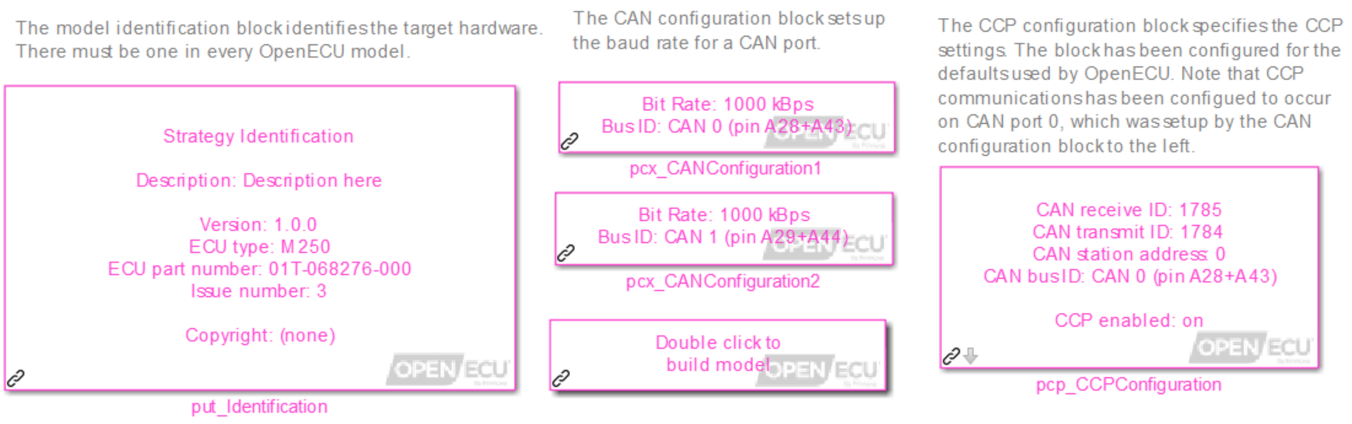

When paused, values shown at primary cursor position
Internal ECU variable read by CCP
DBC CAN signal read direct from bus
Scale displayed for currently selected signal
Secondary cursor
Primary cursor
Watch1 PAUSED
pscl_run_time
stp_ms_task_counter
CAN1.Snoop0x101.stp_ms_task...
Currently selected signal(s) bold
Pause (freeze) or live rolling display
Value, Name and Scope columns currently shown
Show/hide scope
STACKED 500 ms/div t=10:52:19.522 Δt=1.010 sec
Add ECU parameters
Add CAN signals
Grid fonts can be made large for across-room visibility
Settings e.g. acquisition rates
Open previous logfile for display or save current scope data
Screenshot with or without grid and toggle black/white background
Open logfile externally (e.g. in MS Excel)
Scope timebase
Cycle through autoscale modes (stacked, overlapped, overlaid)
Set custom scaling for selected signal(s)

Figure 17: A Detailed Diagram for the Watch Window

4.2. The Architecture of the Control Algorithm Model

The control algorithm model was modified on Simulink by following the same guideline as explained in the sample experiment. Used blocks have comments and metadata over them which describes their role, settings and behavior. The blueprint is illustrated below.



The model identification block identifies the target hardware. There must be one in every OpenECU model.

Strategy Identification

Description: Description here

Version: 1.0.0
ECU type: M250
ECU part number: 01T-068276-000
Issue number: 3

Copyright: (none)

The CAN configuration block sets up the baud rate for a CAN port.

Bit Rate: 1000 kBps
Bus ID: CAN 0 (pin A28+A43)

Bit Rate: 1000 kBps
Bus ID: CAN 1 (pin A29+A44)

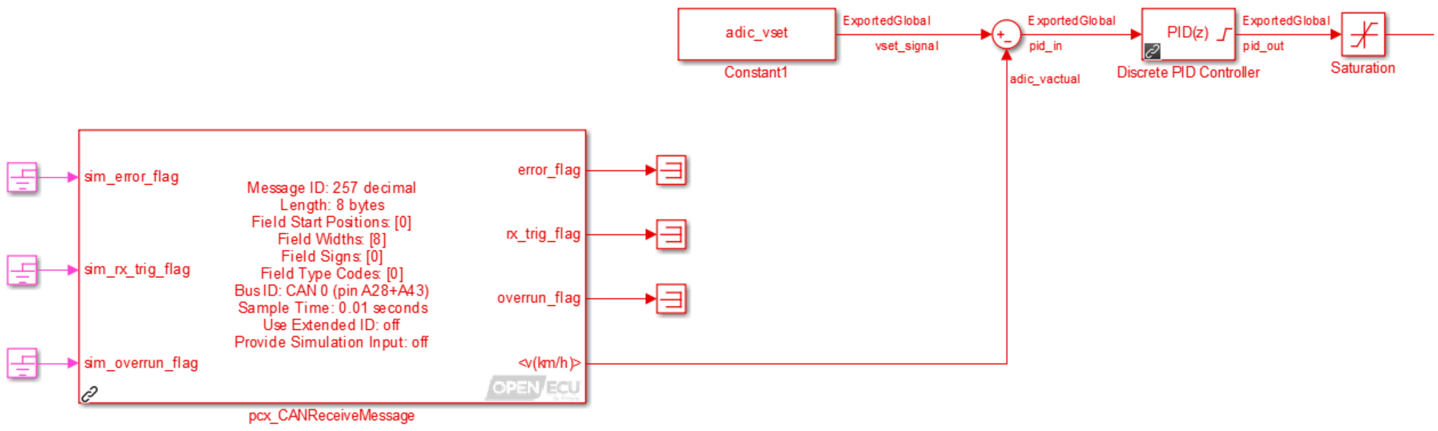
Double click to build model

The CCP configuration block specifies the CCP settings. The block has been configured for the defaults used by OpenECU. Note that CCP communications has been configued to occur on CAN port 0, which was setup by the CAN configuration block to the left.

CAN receive ID: 1785
CAN transmit ID: 1784
CAN station address: 0
CAN bus ID: CAN 0 (pin A28+A43)

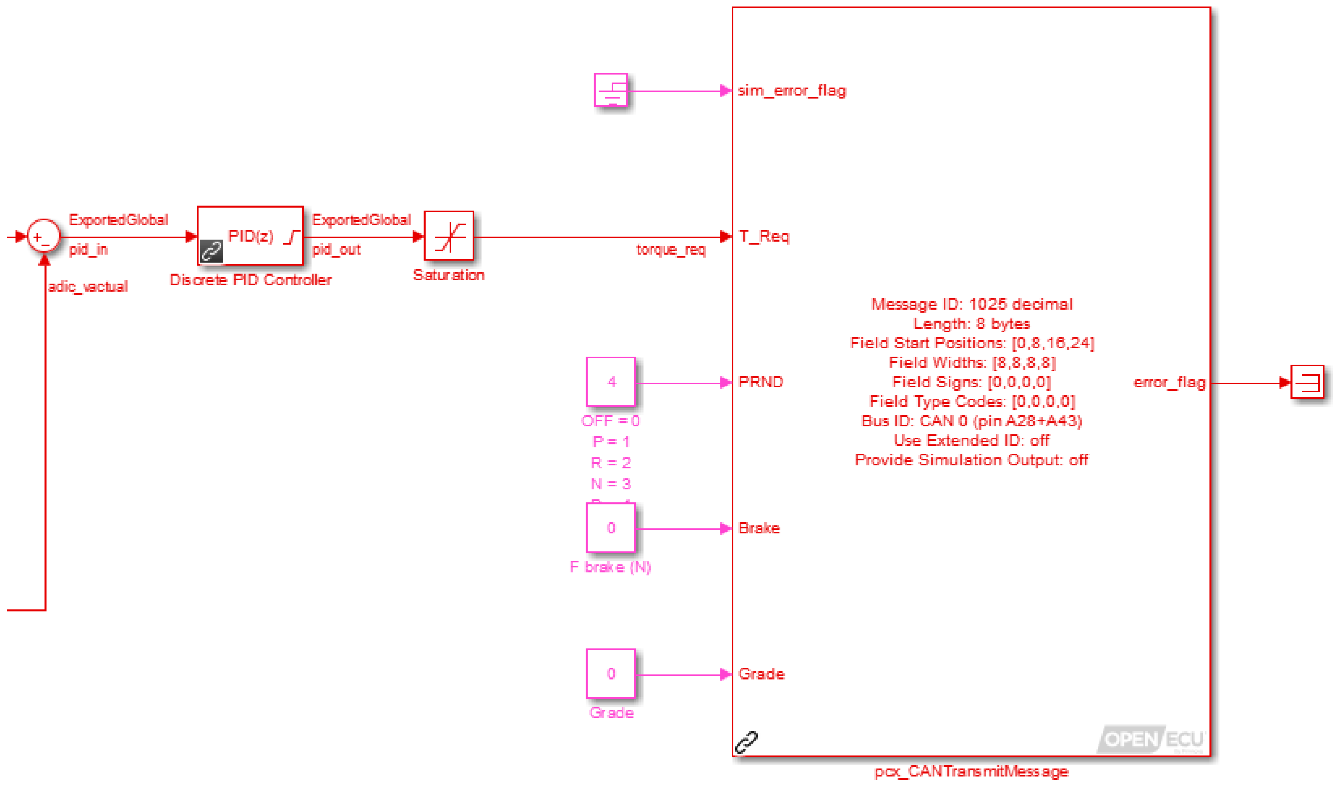
CCP enabled: on

Figure 18: Target ECU Details, CAN IDs, Bitrate and CCP Configurations



Message ID: 257 decimal
Length: 8 bytes
Field Start Positions: [0]
Field Widths: [8]
Field Signs: [0]
Field Type Codes: [0]
Bus ID: CAN 0 (pin A28+A43)
Sample Time: 0.01 seconds
Use Extended ID: off
Provide Simulation Input: off

Figure 19: CAN Message Receiver, Discrete PID Controller for Speed Manipulation with a Constant Reference Speed Input



Message ID: 1025 decimal
Length: 8 bytes
Field Start Positions: [0,8,16,24]
Field Widths: [8,8,8,8]
Field Signs: [0,0,0,0]
Field Type Codes: [0,0,0,0]
Bus ID: CAN 0 (pin A28+A43)
Use Extended ID: off
Provide Simulation Output: off

Figure 20: CAN Message Transmitter with a Torque Request Input

The vehicle speed v(km/h) is received from channel 0 on the CAN bus via the pcx_CANReceiveMessage block and input into a discrete PID controller. A PID controller is an instrument used in industrial control applications to regulate temperature, flow, pressure, speed and other process variables. PID (proportional integral derivative) controllers use a control loop feedback mechanism to control process variables by keeping the actual output from a process as close to the target or setpoint output as possible which gives accurate results. The discrete PID controller was tuned with the values: **P:** 25.42, **I:** 2.59, **D:** 2.72 Afterwards, the output goes through a saturation (to bound it in between -255 and 255) block and fed into the pcx_CANTransmitMessage block as a torque request. Besides, there are 3 other inputs i. e.

- PRND for vehicle's motion pattern (P=1 for parking, R=2 for reverse, N=3 for neutral, D=4 for drive) which is 4 so, the vehicle moves forward constantly
- F brake (N) for the braking force in Newtons which is 0 so, no braking involved
- Grade (%) for the road's slope which is 0

to be sent over to the HIL simulator.

4.3. The Concept of Real-Time Simulation

A simulation is a representation of the operation or features of a system through the use or operation of another. [16] A real-time digital simulation is based on discrete time-steps where the simulator solves model equations consecutively while ensuring coherent time responses.

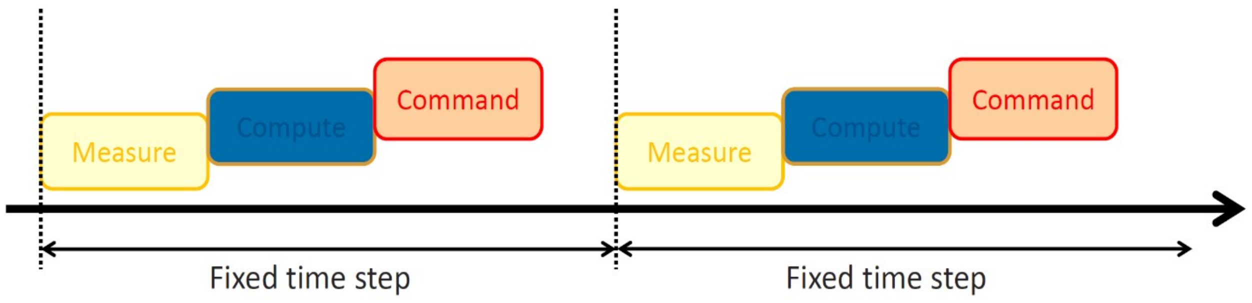


Figure 21: A Real-Time Simulation Working as Expected

During real-time simulation, the accuracy of computations not only depend upon precise dynamic representation of the system, but also on the length of time used to produce results. If simulator operations are not all achieved within the required fixed time-step, the real-time simulation is considered erroneous. This is commonly known as an “overrun”. [17]

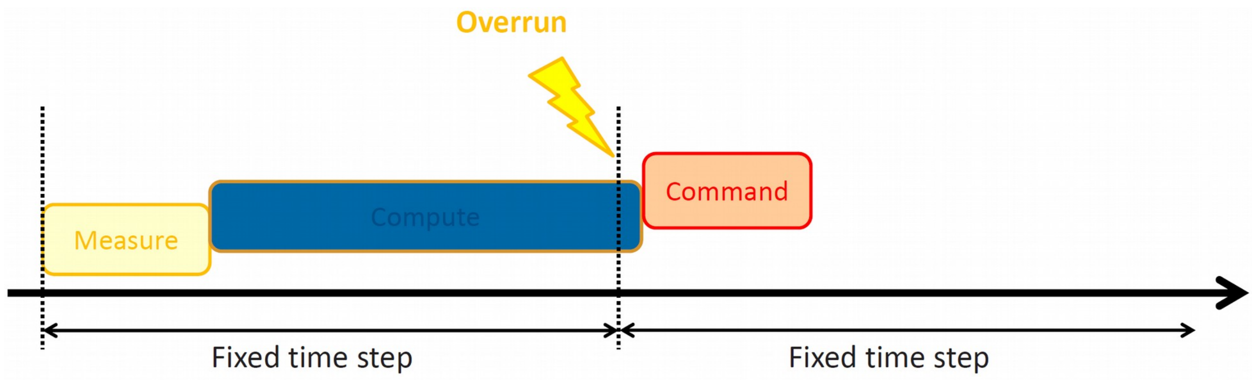


Figure 22: An Overrun

Consequently, it can be concluded that a real-time simulator is performing as expected if the equations and states of the simulated system are solved accurately, with an acceptable resemblance to its physical counterpart, without the occurrence of overruns. In our case, a Hardware-in-the-loop (HIL) simulation is used. HIL simulation is a technique for validating a control algorithm, running on an intended target controller, by creating a virtual real-time environment that represents a physical system to control. [18]

With a simulated plant, we can run tests which would destroy our plant (an automobile) or would be harmful for people in a real situation. Also, the correct action of a control system to various failure modes can effectively be tested since it provides testing conditions that are unavailable on real hardware, such as extreme events testing meaning increased test coverage. This makes HIL simulation an effective technique to increase the safety of machines and systems. In addition, more benefits can be listed as it saves time and money given the fact that it reveals errors which would otherwise have been detected at the final stages of a design where the control system and plant are integrated.

Main steps for HIL simulation are disclosed thus and so:

1. A virtual real-time implementation of physical components such as a plant, sensors, and actuators on a real-time target computer are created and simulated.

2. The control algorithm is loaded on an embedded controller and the plant or environment model is ran in real time on a target computer connected to the controller. The embedded controller interacts with the plant model simulation through various I/O channels.

3. Software representations of the components are refined and gradually the parts of the system environment are replaced with the actual hardware components.

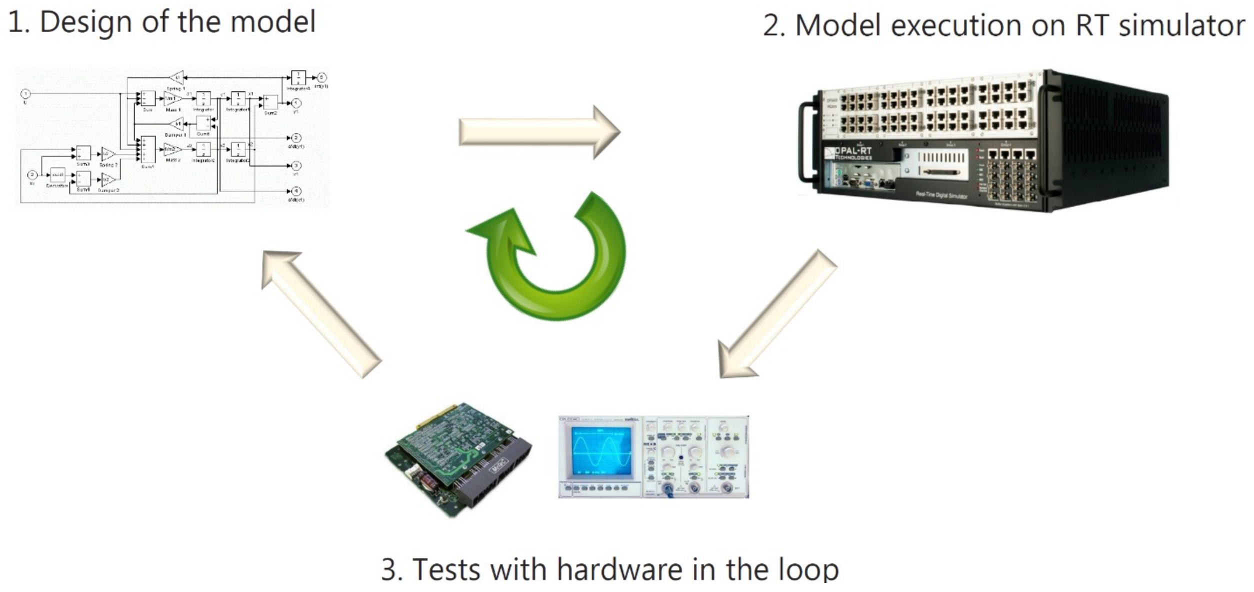

1. Design of the model
2. Model execution on RT simulator
3. Tests with hardware in the loop

Figure 23: HIL Testing

4.4. The Composition of the Vehicle Model

The vehicle model was modified on Simulink to fix the data transmission parameters thereby the communication is established with the ECU. The top module which only consists of 2 subsystems is illustrated below.

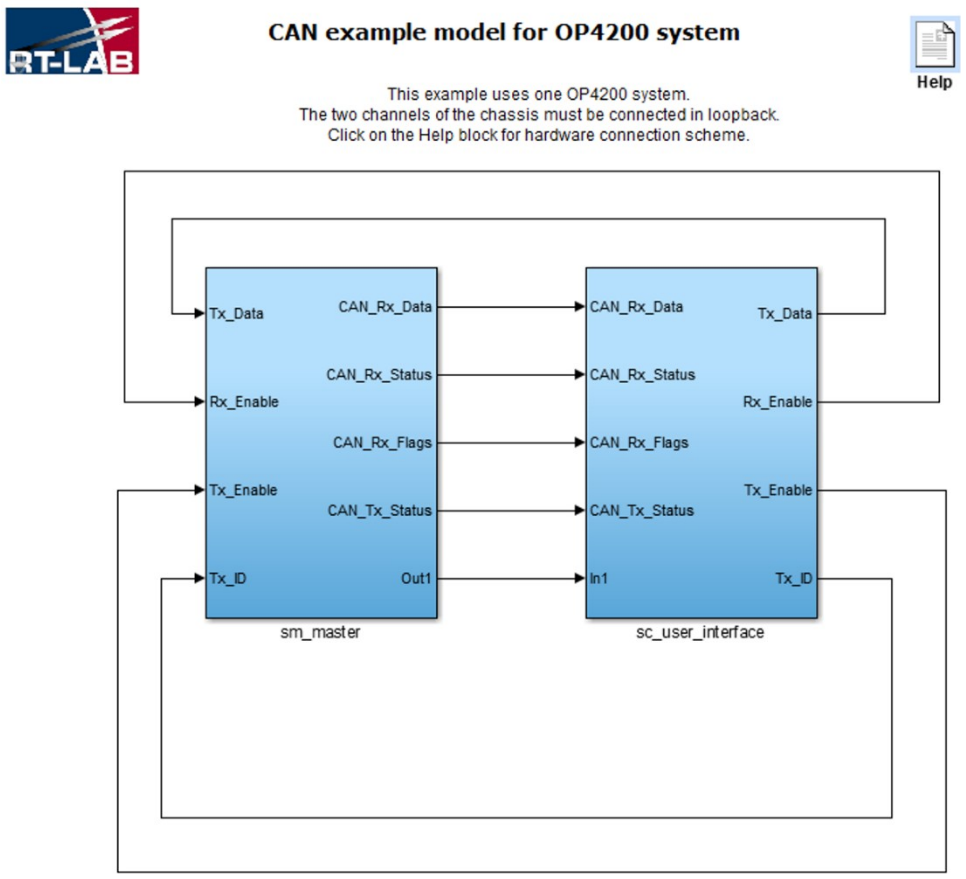


Figure 24: Top Block

Subsystems in Simulink are formed to simplify the model by grouping blocks, to establish hierarchical block diagram and to keep functionally related blocks together. In RT-LAB platforms, they have 2 objectives:

1. Distinguishing computation subsystems and GUI subsystem
    - The computation subsystem will be executed in real-time (or accelerated simulation mode) on one CPU core of the real-time target
    - The GUI subsystem will be displayed on the Host PC
    - The data between computation subsystem and GUI subsystem is exchanged asynchronously through the TCP/IP link
    - Each signal between subsystems (**wire** in Simulink) can be a scalar (single value) or a vector (multiple values), but it must be of type double

2. Assigning computation subsystems to different CPU cores
   - The computation blocks can be split into different computation subsystems
   - Each of the computation subsystems will be executed on one CPU core of the real-time target
   - The data between 2 computation subsystems is exchanged synchronously through shared memory

Here, **sm_mast**er block is the computation subsystem that communicates with the FPGA board and the physical system I/Os while the **sc_user_interface** block represents the GUI subsystem, contains only user interface blocks (scopes, displays, switches, constants) but no signal generation blocks, no mathematical operation blocks or no physical model. It only monitors and controls the algorithm.

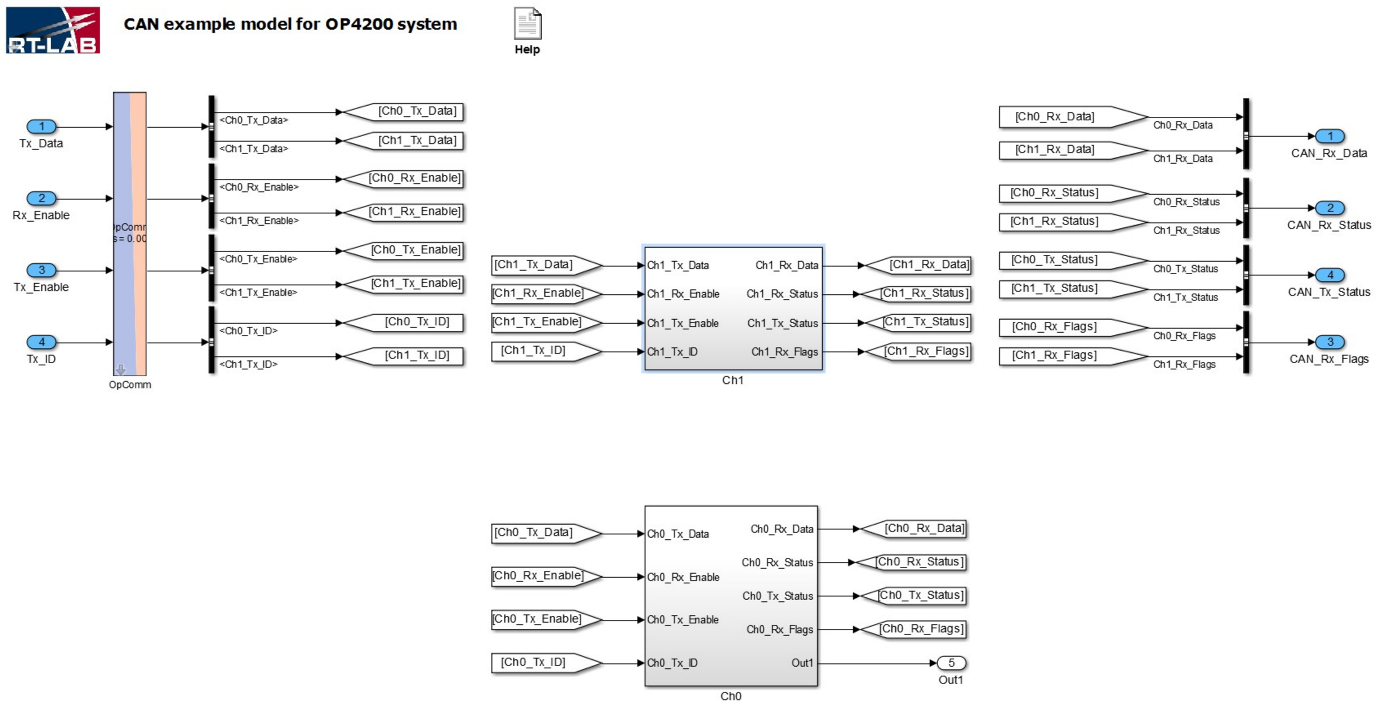


Figure 25: Top/sm_master Block

Both channels are setup but only Ch0 is being used by both sm_master and sc_user_interface blocks. Ch0 block in sm_master has 4 inputs coming from the sc_user_interface side which are Ch0_Tx_Data, Ch0_Rx_Enable, Ch0_Tx_Enable, Ch0_Tx_ID. The OpComm block in both subsystems is responsible for the communication. Every input in a subsystem must go through an OpComm block before any operations can be done on the signals they are associated with. Ch0_Tx_Data carries the transmitted data, Ch0_Tx_ID is the ID used while transmitting messages, Ch0_Rx_Enable and Ch0_Tx_Enable signals are to enable reception and transmission of the messages.

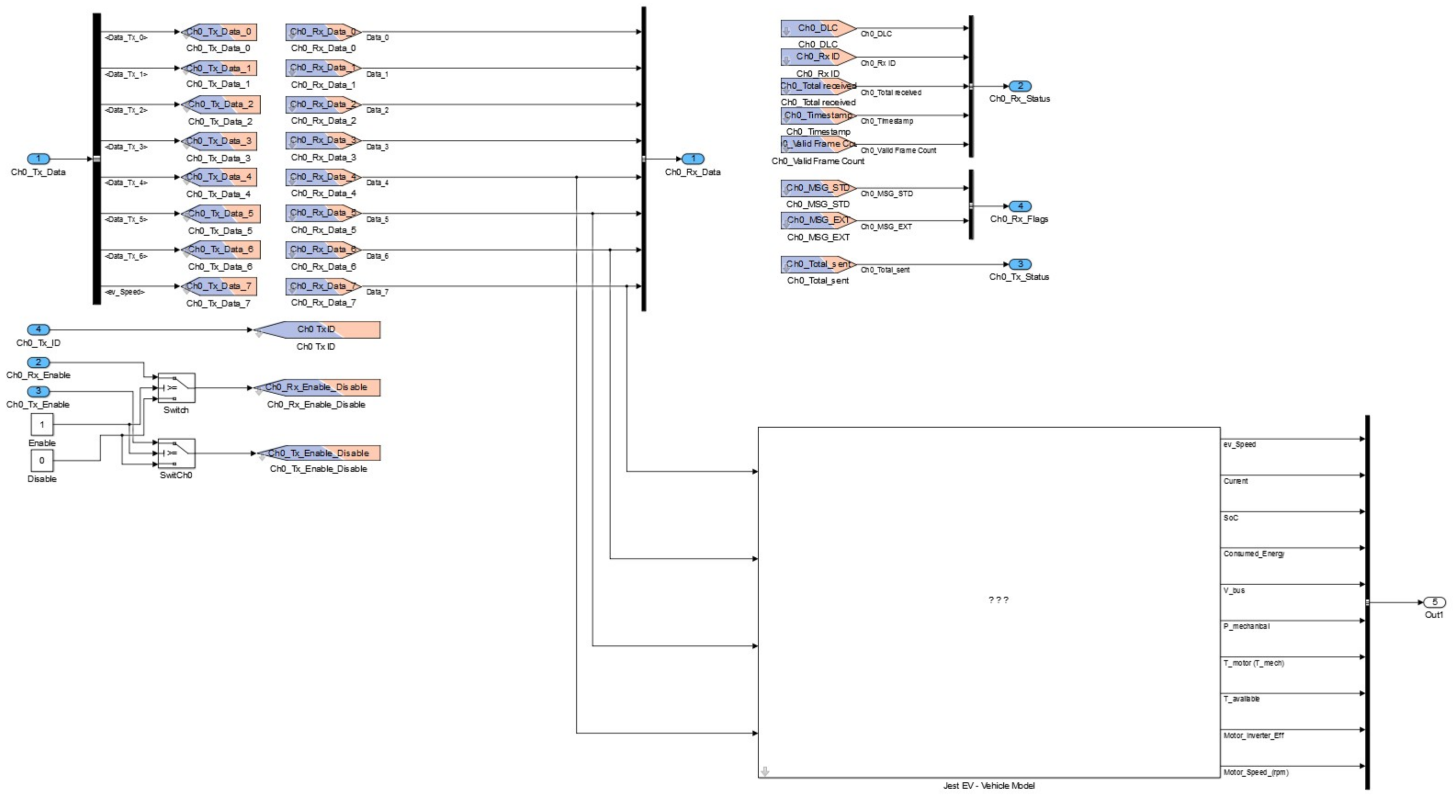


Figure 26: Top/sm_master/Ch0 Block

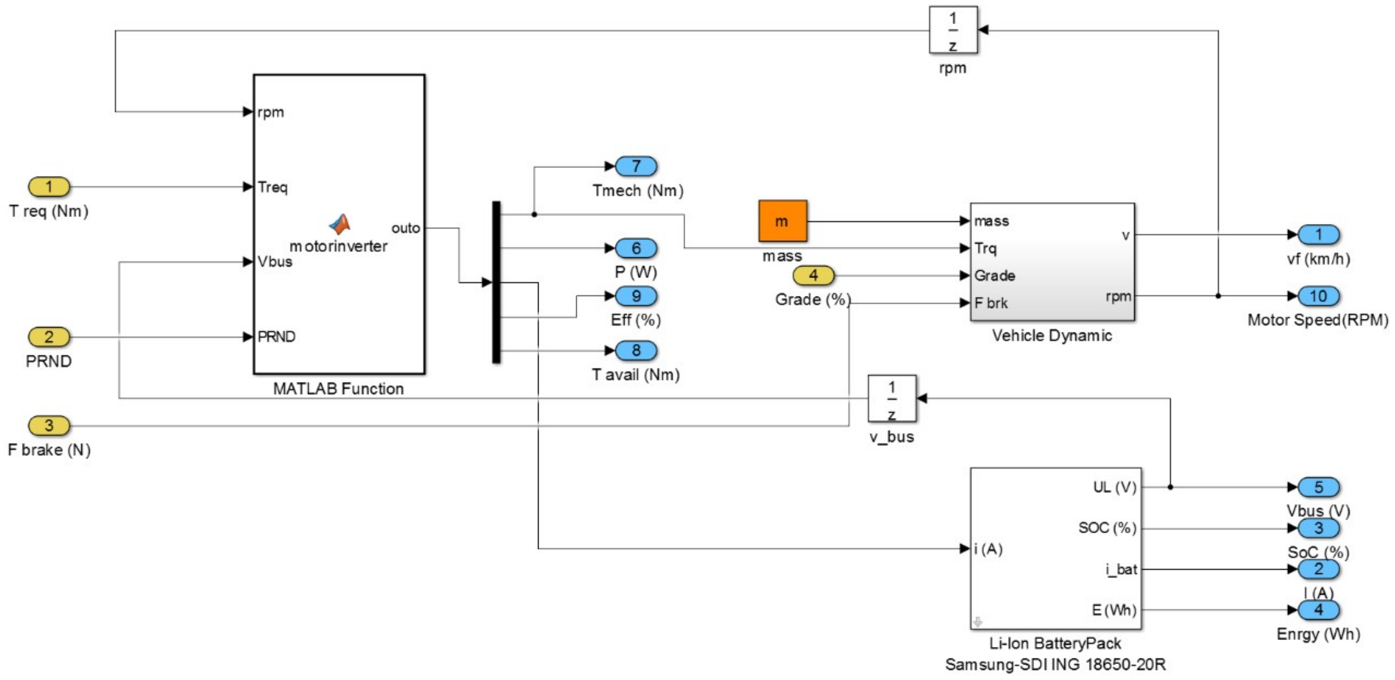


Figure 27: Top/sm_master/Ch0/Jest EV – Vehicle Model Block

In Ch0 block in sm_master, the last 4 slots of Ch0_Rx_Data[4-5-6-7] are occupied by the 4 inputs of the pcx_CANTransmitMessage block in the control algorithm model since it's received over the CAN0 Channel. These variables are input into the vehicle to compute 10 outputs: ev_Speed, Current, SoC, Consumed_Energy, V_bus, P_mechanical, T_motor(T_mech), T_available, Motor_inverter_Eff and Motor_Speed_(rpm).

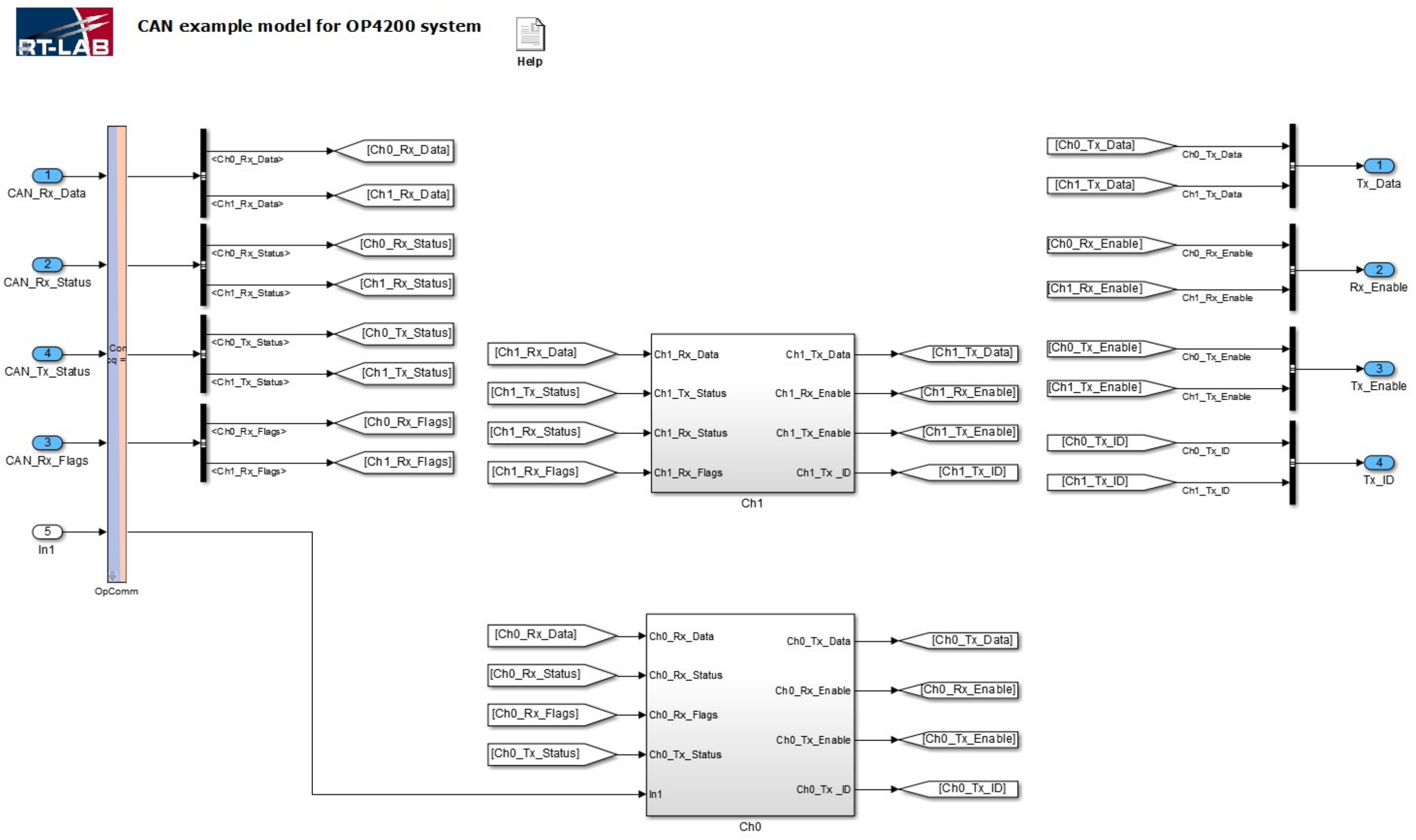
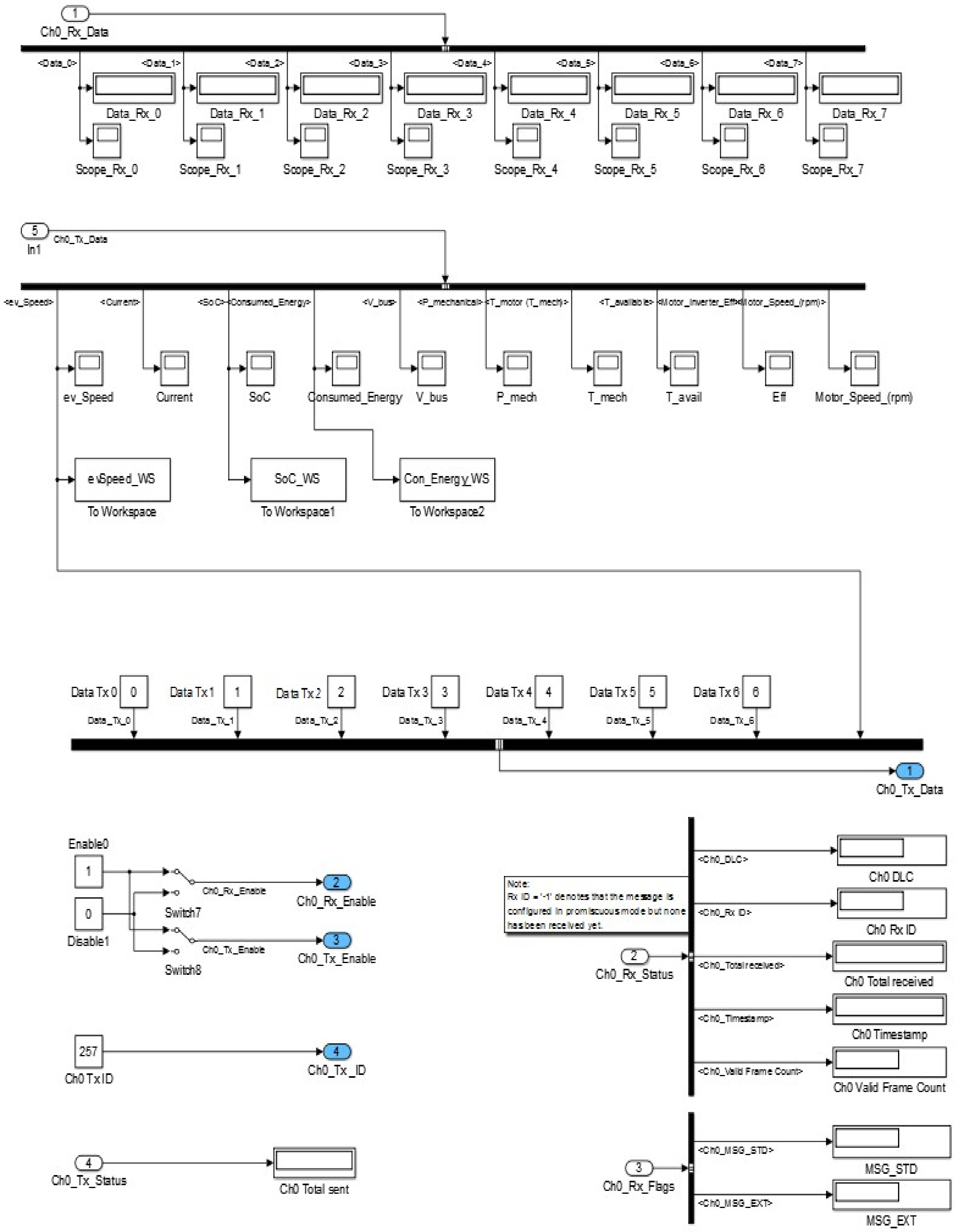


Figure 28: Top/sc_user_interface Block

The 10 outputs coming from the sm_master side are input into Ch0 block in sc_user_interface block to be monitored by the associated scopes. Also, the **ev_Speed** variable is directed back to Ch0_Tx_Data which forwards towards the control algorithm via CAN bus to ensure the calculation of the next-step value for the vehicle speed. This process continues in a loop and can be interacted with the RT-LAB software user interface.



Note:
Rx ID = '-1' denotes that the message is configured in promiscuous mode but none has been received yet.

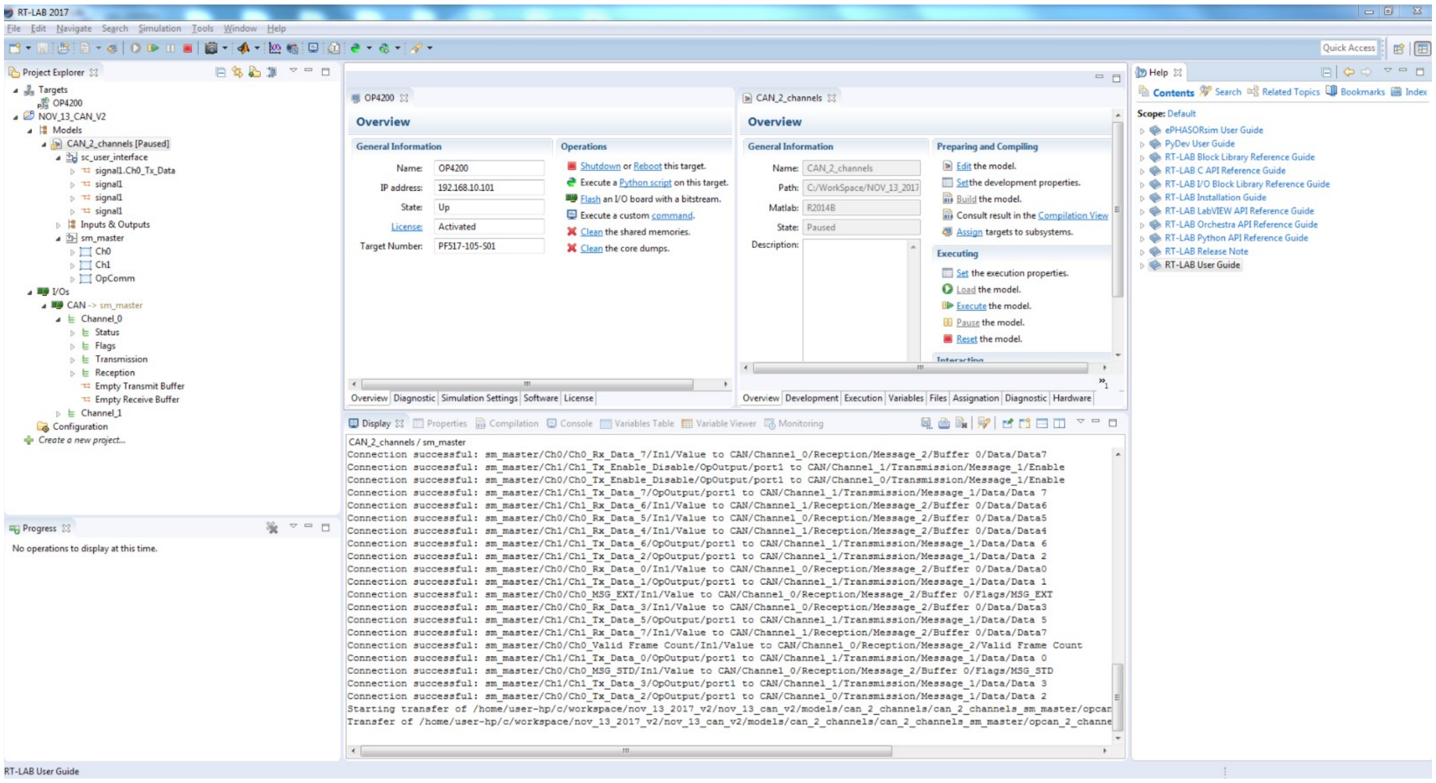
Figure 29: Top/sc_user_interface/Ch0 Block

4.5. Simulation Results

The simulation screenshots were carried out in Opal-RT with a vehicle model that has the parameter descriptions as follows:

Symbol	Value	Unit	Description
Rtire	0.354	m	Radius of tire
g	9.81	m/s2	Earth gravitational constant
mu	0.009	N	Road friction
Cp	0.55	-	Air drag coefficient
A	5.1	m2	Vehicle frontal area
mss	2100	kg	Vehicle mass
ng	0.85	-	Tire specific coefficient
Im	0.18	Kg.m2	Motor inertia
Gb	18.5	-	Total gear ratio

Figure 30: Vehicle Parameters Description



Name: OP4200
IP address: 192.168.10.101
State: Up
License: Activated
Target Number: PF517-105-S01

Shutdown or Reboot this target.
Execute a Python script on this target.
Flash an I/O board with a bitstream.
Execute a custom command.
Clean the shared memories.
Clean the core dumps.

Name: CAN_2_channels
Path: C:/WorkSpace/NOV_13_2017
Matlab: R2014B
State: Paused
Description:

Edit the model.
Set the development properties.
Build the model.
Consult result in the Compilation View
Assign targets to subsystems.
Set the execution properties.
Load the model.
Execute the model.
Pause the model.
Reset the model.

```
CAN_2_channels / sm_master
Connection successful: sm_master/Ch0/Ch0_Rx_Data_7/In1/Value to CAN/Channel_0/Reception/Message_2/Buffer 0/Data/Data7
Connection successful: sm_master/Ch1/Ch1_Tx_Enable_Disable/OpOutput/port1 to CAN/Channel_1/Transmission/Message_1/Enable
Connection successful: sm_master/Ch0/Ch0_Tx_Enable_Disable/OpOutput/port1 to CAN/Channel_0/Transmission/Message_1/Enable
Connection successful: sm_master/Ch1/Ch1_Tx_Data_7/OpOutput/port1 to CAN/Channel_1/Transmission/Message_1/Data/Data 7
Connection successful: sm_master/Ch1/Ch1_Rx_Data_6/In1/Value to CAN/Channel_1/Reception/Message_2/Buffer 0/Data/Data6
Connection successful: sm_master/Ch0/Ch0_Rx_Data_5/In1/Value to CAN/Channel_0/Reception/Message_2/Buffer 0/Data/Data5
Connection successful: sm_master/Ch1/Ch1_Rx_Data_4/In1/Value to CAN/Channel_1/Reception/Message_2/Buffer 0/Data/Data4
Connection successful: sm_master/Ch1/Ch1_Tx_Data_6/OpOutput/port1 to CAN/Channel_1/Transmission/Message_1/Data/Data 6
Connection successful: sm_master/Ch1/Ch1_Tx_Data_2/OpOutput/port1 to CAN/Channel_1/Transmission/Message_1/Data/Data 2
Connection successful: sm_master/Ch0/Ch0_Rx_Data_0/In1/Value to CAN/Channel_0/Reception/Message_2/Buffer 0/Data/Data0
Connection successful: sm_master/Ch1/Ch1_Tx_Data_1/OpOutput/port1 to CAN/Channel_1/Transmission/Message_1/Data/Data 1
Connection successful: sm_master/Ch0/Ch0_MSG_EXT/In1/Value to CAN/Channel_0/Reception/Message_2/Buffer 0/Flags/MSG_EXT
Connection successful: sm_master/Ch0/Ch0_Rx_Data_3/In1/Value to CAN/Channel_0/Reception/Message_2/Buffer 0/Data/Data3
Connection successful: sm_master/Ch1/Ch1_Tx_Data_5/OpOutput/port1 to CAN/Channel_1/Transmission/Message_1/Data/Data 5
Connection successful: sm_master/Ch1/Ch1_Rx_Data_7/In1/Value to CAN/Channel_1/Reception/Message_2/Buffer 0/Data/Data7
Connection successful: sm_master/Ch0/Ch0_Valid Frame Count/In1/Value to CAN/Channel_0/Reception/Message_2/Valid Frame Count
Connection successful: sm_master/Ch1/Ch1_Tx_Data_0/OpOutput/port1 to CAN/Channel_1/Transmission/Message_1/Data/Data 0
Connection successful: sm_master/Ch0/Ch0_MSG_STD/In1/Value to CAN/Channel_0/Reception/Message_2/Buffer 0/Flags/MSG_STD
Connection successful: sm_master/Ch1/Ch1_Tx_Data_3/OpOutput/port1 to CAN/Channel_1/Transmission/Message_1/Data/Data 3
Connection successful: sm_master/Ch0/Ch0_Tx_Data_2/OpOutput/port1 to CAN/Channel_0/Transmission/Message_1/Data/Data 2
Starting transfer of /home/user-hp/c/workspace/nov_13_2017_v2/nov_13_can_v2/models/can_2_channels/can_2_channels_sm_master/opcan
Transfer of /home/user-hp/c/workspace/nov_13_2017_v2/nov_13_can_v2/models/can_2_channels/can_2_channels_sm_master/opcan_2_channe
```

Figure 31: Opal-RT User Interface

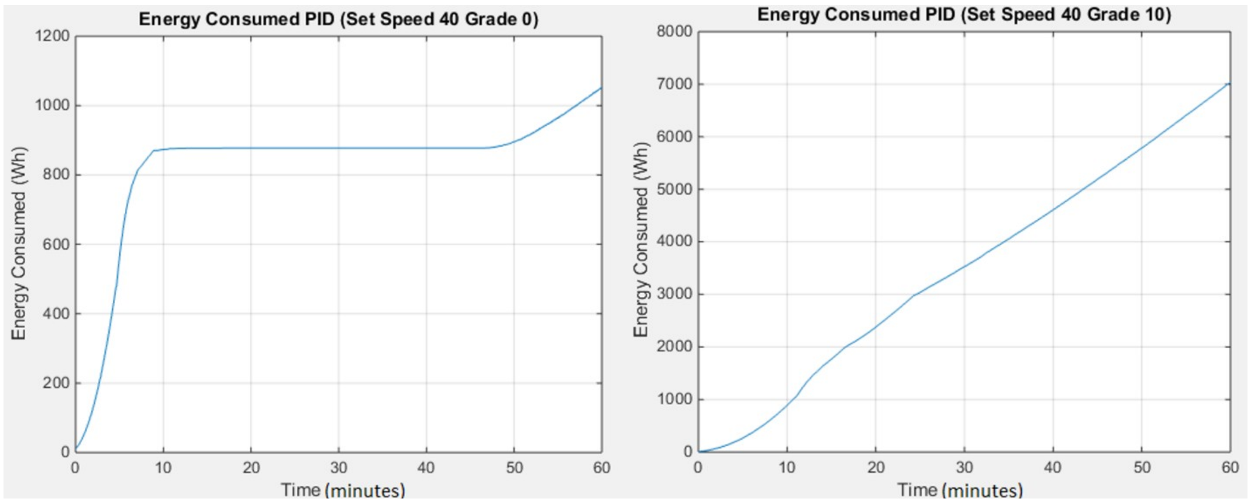


Figure 32: Energy Consumed (Set Speed 40 (Grade 0%-10% comparison))

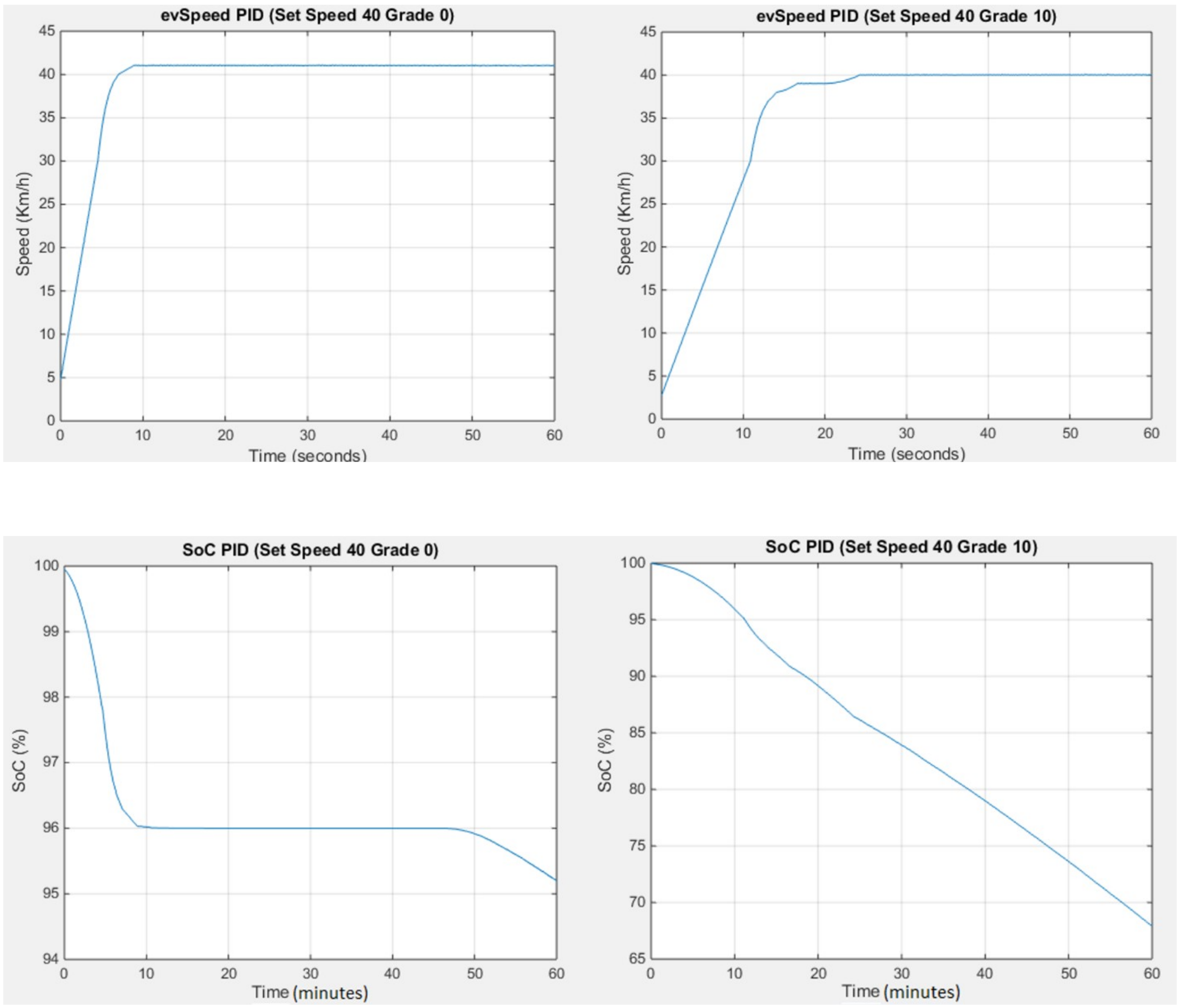


Figure 33: evSpeed (Set Speed 40 (Grade 0%-10% comparison))

Figure 34: SoC (Set Speed 40 (Grade 0%-10% comparison))

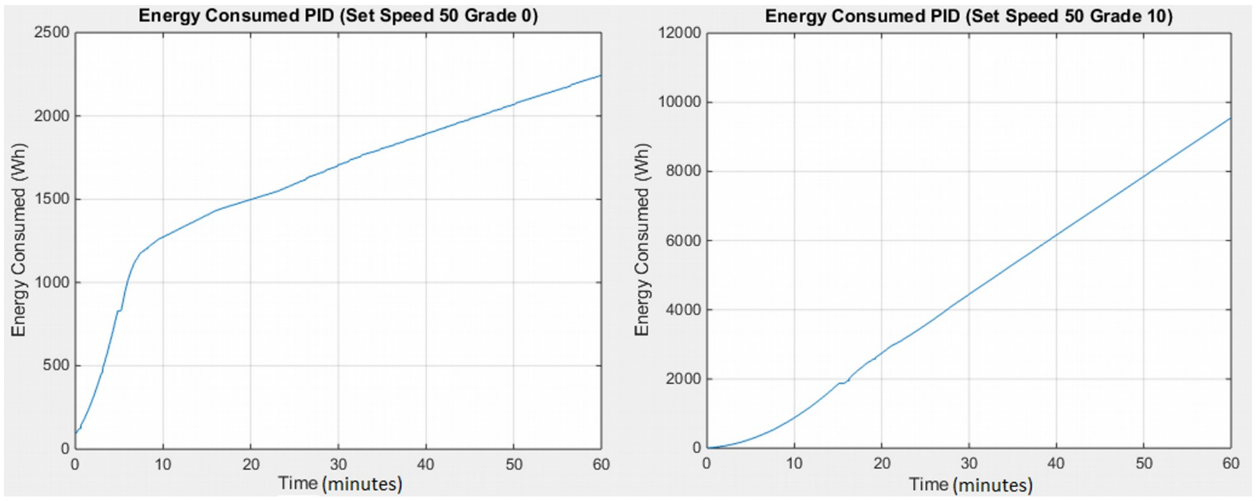


Figure 35: Energy Consumed (Set Speed 50 (Grade 0%-10% comparison))

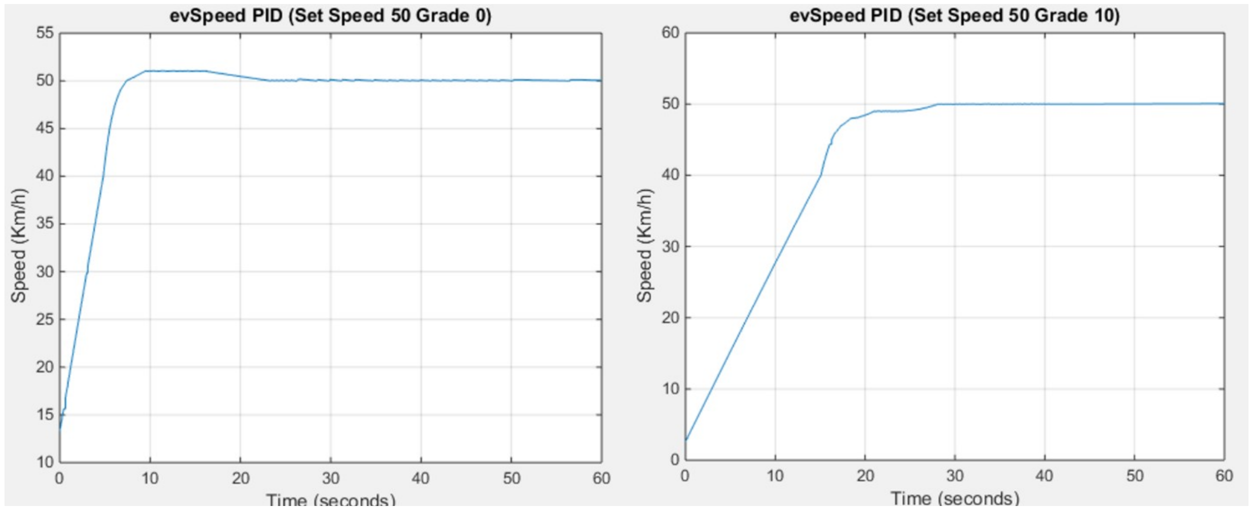


Figure 36: evSpeed (Set Speed 50 (Grade 0%-10% comparison))

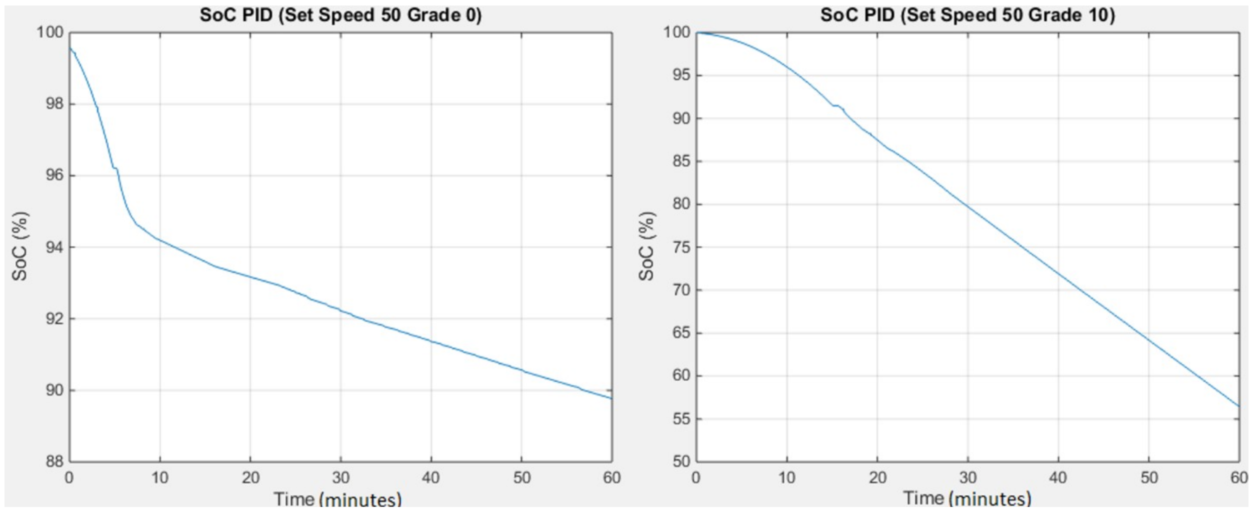


Figure 37: SoC (Set Speed 50 (Grade 0%-10% comparison))

References

[1] https://climate.nasa.gov/evidence/ (06/11/2019 05:29PM)

[2] https://www.c2es.org/content/international-emissions/ (06/11/2019 05:51PM)

[3] https://www.bloomberg.com/features/2016-ev-oil-crisis/ (06/11/2019 06:40PM)

[4] Bassett, Mike & Brods, Bruno & Hall, Jonathan & Borman, Stephen & Grove, Matthew & Reader, Simon. (2015). GPS Based Energy Management Control for Plug-in Hybrid Vehicles. SAE Technical Papers.

[5] Oliva, Javier & Weihrauch, Christoph & Bertram, Torsten. (2013). Model-Based Remaining Driving Range Prediction in Electric Vehicles by using Particle Filtering and Markov Chains. World Electric Vehicle Journal.

[6] Daigle, Matthew & Goebel, Kai. (2011). A Model-based Prognostics Approach Applied to Pneumatic Valves. International Journal of Prognostics and Health Management.

[7] Schwickart, Tim & Voos, Holger & Hadji-Minaglou, J.-R & Darouach, Mohamed & Rosich, Albert. (2014). Design and Simulation of a Real-Time Implementable Energy-Efficient Model-Predictive Cruise Controller for Electric Vehicles. Journal of the Franklin Institute. 352. 10.1016/j.jfranklin.2014.07.001.

[8] https://5.imimg.com/data5/VJ/BB/MY-6937640/automotive-hil-hardware-in-loop-testing-2c-500x500.png (06/11/2019 07:31PM)

[9] https://www.pi-innovo.com/product/m250/ (06/11/2019 08:10PM)

[10] https://www.opal-rt.com/op4200/ (06/11/2019 08:15PM)

[11] https://www.kvaser.com/product/kvaser-leaf-light-hs-v2/ (06/11/2019 08:22PM)

[12] https://www.pi-innovo.com/product/pisnoop/ (06/11/2019 08:35PM)

[13] https://www.opal-rt.com/software-rt-lab/ (06/11/2019 09:05PM)

[14] https://www.youtube.com/watch?v=SbsCdAC0l7E (01/12/2019 06:45PM)

[15]http://support.openecu.com/doc_user/openecu_user_guide_c_api/openecu_user_guide_c_api.html (02/12/2019 04:06PM)

[16] The American Heritage Dictionary of the English Language. Boston: Houghton Mifflin Company, 2000.

[17] Bélanger, Jean & Venne, P. & Paquin, Jean-Nicolas. (2010). The what, where, and why of real-time simulation. Planet RT. 37-49.

[18] https://www.mathworks.com/discovery/hardware-in-the-loop-hil.html (12/01/2020 10:48AM)

ISO 11898-2 Standard: https://www.iso.org/standard/67244.html (06/11/2019 08:32PM)

ISO 26262-1 Standard: https://www.iso.org/standard/68383.html (06/11/2019 08:47PM)

Project Enquiry Form: Standards and Constraints

Q1. What is the **project scope**?

It is a new project, but some material was provided by my supervisor. To be exact, the essential forms of the vehicle model and the control algorithm were provided. I modified both in some ways e. g. the communication parameters and connections, vehicle dynamics, PID parameters etc.

Q2. Did you formalize an **engineering problem** yourself and solve it?

I picked a project topic that my supervisor suggested. I want to pursue a career in automotive industry therefore in order to broaden my knowledge in this area, I chose this problem. I learned how the interaction in between an ECU and a real-time HIL simulator works by realizing a demonstration. I will engage for a design/solution/method through solving an optimization problem next semester.

Q3. Among the **knowledge and skill set** that you have acquired in your previous **courses**, which of them were utilized for this project?

I used my coding (MATLAB), computer architecture, microprocessors, embedded systems fundamentals, control systems essentials, simulation basics knowledge that I gained in CS240, EE362, EE321, EE301, EE201, EE302, CS100 coded courses.

Q4. What are the **engineering standards** that you either used or took into consideration?

ISO 26262-1:2018 Road vehicles — Functional safety — Part 1: Vocabulary
ISO 11898-2:2016: Road vehicles — Controller area network (CAN) — Part 2: High-speed medium access unit

Q5. Discuss the possible **effects of your project** under the following titles and explain how these were used as **constraints** guiding your project.

- ***a.*** **Security - Health - Sustainability - Economics - Impact on Environment**: My project aims to demonstrate how HIL simulation can be beneficial when doing testing on a physically integrated plant is costly and dangerous for the surrounding environment or people. Since the general idea is to increase the cruising range of EVs, it aspires to increase the acceptance of EVs which as a consequence provides a quicker shift to a greener era.
- ***b.*** **Manufacturability**: ---
- ***c.*** **Ethics**: ---
- ***d.*** **Social and political issues**: ---