Project Title: System Verification and Validation Plan for Mechatronics

Team 28, Controls Freaks
Abhishek Magdum
Dharak Verma
Jason Surendran
Laura Yang
Derek Paylor

November 1, 2022

Revision History

Date	Version	Notes
Nov 1, 2022	1.0	Initial Revision
April 7, 2023	2.0	Final Revision: Updated and added unit and integration tests

Contents

List of Tables

List of Figures

1 Symbols, Abbreviations and Acronyms

See the below table for abbreviations and technical lingo relevant to this project and document.

MFE	Mac Formula Electric
ECU	Electronic Control Unit
BMS	Battery Management System
CAN	Controller Area Network
EV	Electric Vehicle
SAE	Society of Automotive Engineers
PCB	Printed Circuit Board
SoC	State of Charge
TMS	Thermal Monitoring System
AMK	Arnold Müller Kirchheim GmbH; supplier of the vehicle's motor, inverter, controller kit
APPS	Accelerator Pedal Position Sensor
AMS	Accumulator Management System - FSAE lingo, alias for BMS
IMD	Insulation Monitoring Device, installed in the Tractive system.
BSPD	Brake System Plausibility Device, nonprogrammable circuit to check for simultaneous braking and high power output.
PWM	Pulse Width Modulation
G/Gov	Governor module
BM	Battery Monitor module
MI	Motor Interface module
VD	Vehicle Dynamics module

2 General Information

This document provides an overview of the methodology our team will employ to conduct verification and validation for the system, beginning with background context for this project.

2.1 Summary

The purpose of this project will be to design, test and implement a software control system for a quarter-scale Formula 1 style electric vehicle, in cooperation with the McMaster Formula Electric team, who is the primary project stakeholder. The control system can be compartmentalized into the following subsystems, to be developed in a model-based environment via MATLAB/Simulink. All subsystems will need to be tested:

- **Driver Interface**: Takes sensor inputs and computes steering angle and pedal mapping. Provides those filtered driver inputs to the Vehicle Dynamics system.
- **Battery Monitor**: Monitors the battery and HV contactor status, and relays the information to the Governor.
- **Vehicle Dynamics**: Determines desired motor torques based on driver inputs, vehicle kinematics (eg. speed, acceleration), and current motor states (eg. speed, torque).
- **Motor Interface**: Communicates with the 2 AMK motors to execute the torque requests from the vehicle dynamics system, returning motor state data.
- **Governor**: As the name suggests, this module governs the entire control system. The Governor monitors the vehicle state as well as the state of each control subsystem, and determines mode commands for each subsystem. For example, the Governor may command a startup or shutdown sequence for specific subsystems. The Governor communicates only with other software subsystems, and not over IO to other hardware elements.

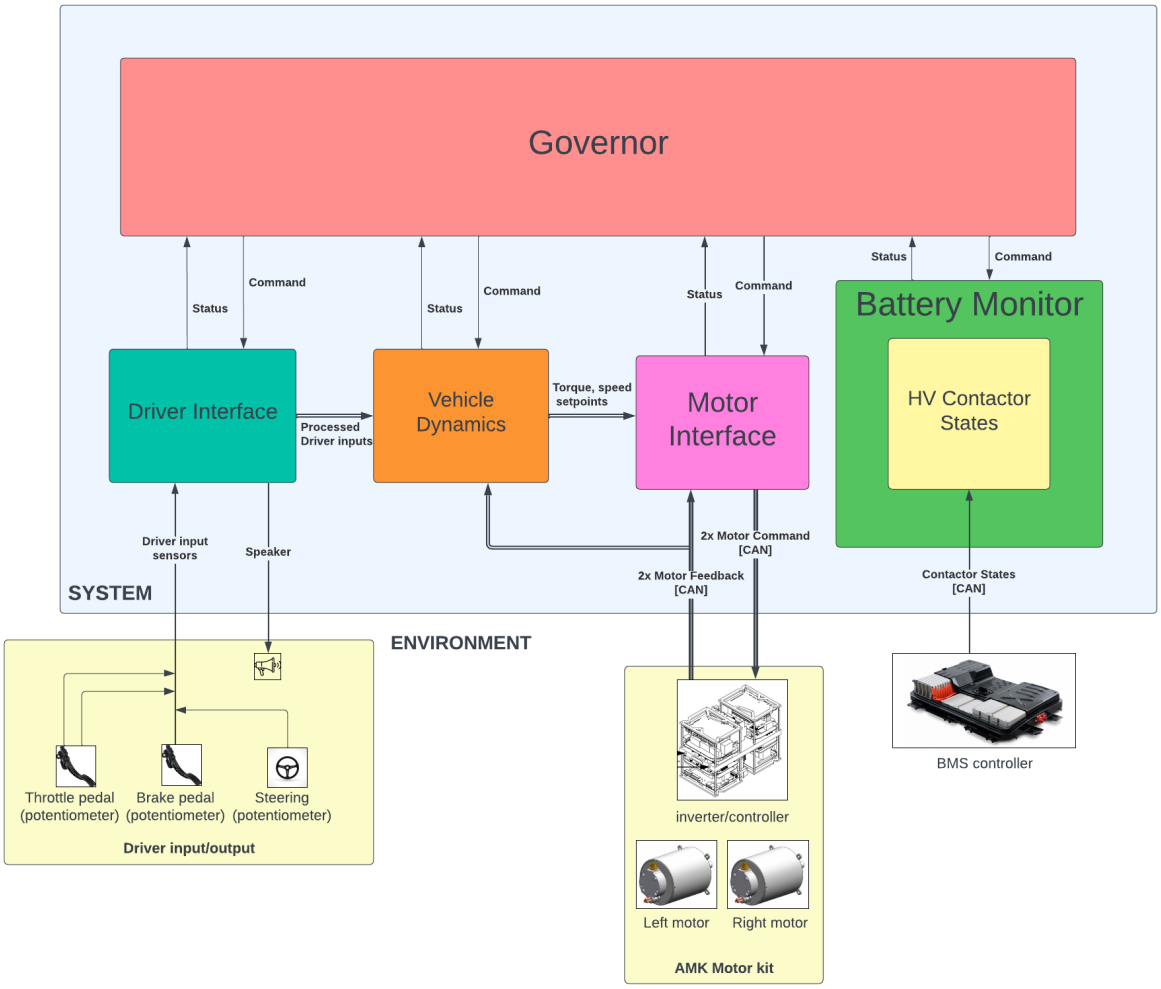

Figure 1: Control System and Environment

2.2 Objectives

- Compliance with the Formula SAE ruleset.
- Provide the necessary software for basic vehicle operation. Optimizing vehicle performance metrics (eg. peak acceleration, range) are not within scope.
- Can be compiled and ran on the target hardware (STM32).
- Ensure safe startup, shutdown and emergency procedures.

2.3 Relevant Documentation

- SRS
- FSAE Rules
- Module Guide (MG) (work in progress)
- Module Interface Specification (work in progress)

3 Plan

This section shall outline the verification and validation tools and procedures, from system design and documentation (SRS, VnV plan) to actual software implementation.

3.1 Verification and Validation Team

Team Member	Role
Mac Formula Electric	Assist in hardware-related validation, eg. expected hardware operation, safe operation bounds, integration with embedded system
Dharak Verma	Liaison with MFE
Abhishek Magdum	Code deployment
Jason Surendran	Unit & System Testing
Laura Yang	Unit & System Testing
Derek Paylor	Testing/simulation tools, test design
4TB6 Peers	Documentation feedback
4TB6 Instruction team	Deliverables and design feedback

3.2 SRS Verification Plan

The following stakeholders play a role in SRS verification:

- Review by classmates - helps ensure deliverables are in-line with project expectations
- Review by Mac Formula Electric - helps ensure our specifications are compatible with hardware considerations
- Review by capstone instruction team - feedback on deliverables will likely necessitate revisions to documentation, system design and implementation

- Review by teammates - helps ensure an individual's contribution meets standards for form and function, and is consistent with other team members' work. This composes the majority of review feedback for team members.

- Structured monthly team reviews - every 4th weekly team meeting, the team shall review and propose updates as required for the SRS, VnV Plan and system design, based on project progress and new findings.

During monthly team reviews, the following SRS checklist items shall be considered:

- Does the current project state and feedback necessitate the deletion of any requirements?

- Does the current project state and feedback necessitate addition of any requirements?

- Does the current project state and feedback necessitate modification of high level design or context diagrams?

For the duration of the project, requirements and testing updates were noted internally, to be updated in Rev 1 documentation.

3.3 Design Verification Plan

The same methods will be employed as specified in Section 3.2.

During monthly team reviews, and following feedback on major deliverables (eg. POC demonstration), the following design checklist items shall be considered:

- Does the current project state and feedback necessitate the deletion of any system modules?

- Does the current project state and feedback necessitate addition of any system modules?

- Does the current project state and feedback necessitate a change in implementation of a system module?

If yes to any of the above, document the change and justification.

As in 3.2, this was noted internally, to be reflected in Rev 1 documentation.

3.4 Verification and Validation Plan Verification Plan

The same methods will be employed as specified in Section 3.2. Naturally, a review of the project SRS should prompt a review of the corresponding VnV Plan.

During monthly team reviews, the following design checklist items shall be considered:

- Does the current project state and feedback necessitate the deletion of any system-level test cases?
- Does the current project state and feedback necessitate the addition of any system-level test cases?
- Does the current project state and feedback necessitate the deletion of any unit-level test cases?
- Does the current project state and feedback necessitate the addition of any unit-level test cases?

If yes to any of the above, document the change and justification.

The same methods will be employed as specified in Section 3.2. Naturally, a review of the project SRS should prompt a review of the corresponding VnV Plan.

3.5 Implementation Verification Plan

Verification of our system-level and unit-level implementation is somewhat unique due to the nature of model-based design in Simulink. The modules that we develop will need to have their inputs simulated, and their outputs compared to expected values. The diagrams below illustrate how this will be done.

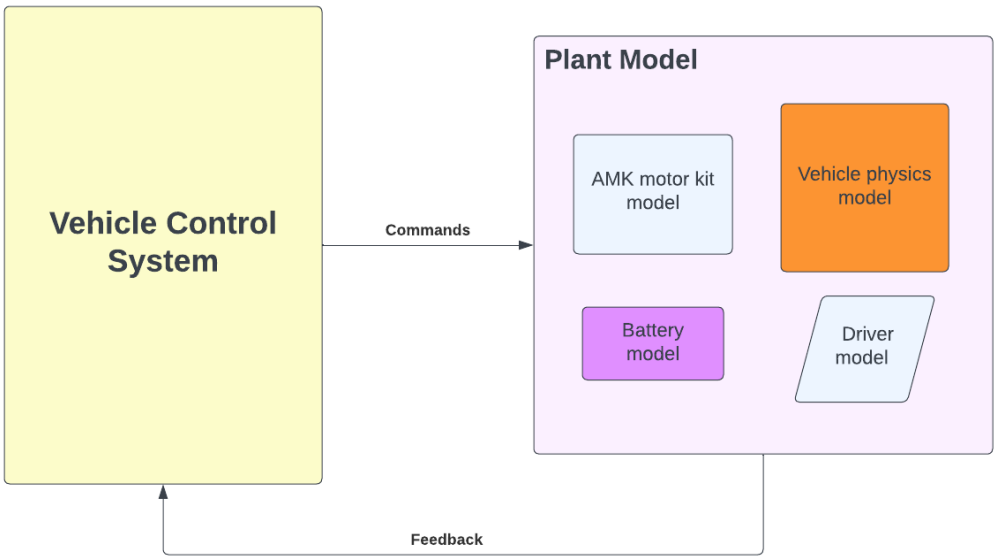

Figure 2: System Testing Environment Overview

System-level testing involves placing our highest-level module (the entire control system, containing all subsystems) in a simulation environment akin to the classic 'controller-plant' block diagram.

- The **Plant Model** takes the control system's command signals, simulates how the vehicle responds to these commands, and return feedback signals on the vehicle's state as inputs to the control system (eg. speed, acceleration, motor/battery state). This model will be constructed using hardware component documentation (eg. AMK motor documentation, McMaster FSAE documentation), Simscape library blocks, and through application of basic first-principles physics. The Driver Model will be driven by custom-defined signals (for pedals, steering, etc.), to

allow us to produce a wide variety of driver cases. There is no physics model or documentation for what a driver will do.

For specific system-level test cases, see Section 4.

Unit-level testing for individual subsystems / modules (eg. Vehicle Dynamics) also involves a testing environment, albeit simpler than the system-level simulation above.

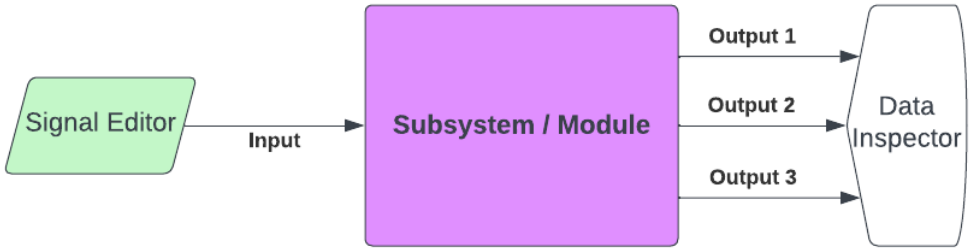

Figure 3: Unit Testing Example Setup

- **Module Inputs** can be simulated in several ways, but we employed use of Simulink's SignalEditor tool, which is a graphical interface for defininining timeseries signal data. A set of input signal data should correspond to one unit test, and is stored in a .mat file, allowing us to reuse the same testing data across versions for consistant validation.
- **Module Outputs** can be logged during simulation and analyzed in the Simulink Data Inspector, which is an oscilloscope-like interface for plotting timeseries signal data.

For specific unit-level test cases, see Section 5.

3.6 Automated Testing and Verification Tools

As mentioned above, systematic verification tools for MATLAB/Simulink workflows (model-based design) are rare, unlike tools that exist for traditional software development workflows.

Verification tools will be built by the team, in the form of Simulation environments as described in Section 3.5. Automating these simulations is not in the scope of this project.

3.7 Software Validation Plan

Externally-driven validation simply consists of the FSAE ruleset. Since the overall goal is basic FSAE vehicle operation, desired vehicle performance metrics that would usually drive validation efforts are not in-scope for this project.

4 System Test Description

4.1 Tests for Functional Requirements

These tests cover most (but not all) of the functional requirements from the SRS, at the integration testing (entire-system) level. These tests are carried out in the custom system simulation environment as described in Figure 2. More minute requirements are tested with more coverage in the functional Unit-tests in Section 5.2.

1. ST_VCS1

   Type: Manual

   Initial State: State 0 (from startup)

   Input: Battery contactors switch into 'running' state (see UT_BM1), Driver inputs allow 'driver start request' (see UT_DI1); after 2 seconds, ramp APPS1 to 50%, then back to 0%.

   Output: Governor commands MI to start motors after BM reports 'running', then sends 'ready to drive' after DI reports 'driver start request'. When APPS1 ramps, motor torque is applied and the vehicle accelerates, then coasts down when APPS1 is 0%.

   Test Case Derivation: Requirements G1, G2, G3, G4, G5, G6, DI1, DI2, DI3, DI4, DI5, DI8, BM1, BM2, BM3, MI1, MI2, MI3, MI4, MI7, VD1, VD2

   How test will be performed: Unit testing environment (Figure 3)

2. ST_VCS2

   Type: Manual

   Initial State: State 0 (from startup)

   Input: repeat ST_VCS1 to enter driving mode; then while APPS1 is on, send bError for the left motor.

   Output: MI status reports 'error', Governor issues DI command 'system error', Driver torque request is zeroed, vehicle coasts down

   Test Case Derivation: Requirements G1, G2, G7, DI1, DI2, DI3, DI8, DI10, BM1, BM2, MI1, MI2, MI3, MI8, VD1, VD2

   How test will be performed: Unit testing environment (Figure 2)

3. ST_VCS3

   Type: Manual

   Initial State: State 0 (from startup)

   Input: repeat ST_VCS1 to enter driving mode; then while APPS1 is on, set Brake pedal position = 75% for 5 seconds.

   Output: Motor torque limits (left & right) are zeroed while braking, vehicle comes to an abrupt stop.

   Test Case Derivation: Requirements G1, G2, DI1, DI2, DI3, DI8, BM1, BM2, MI1, MI2, MI3, VD1, VD2, VD7, VD8

   How test will be performed: Unit testing environment (Figure 2)

4.2 Tests for Nonfunctional Requirements

4.2.1 Look and Feel

1. NFT1

   Type: Manual

   Initial State: Control system model open in Simulink

Input/Condition: Verify subsystem blocks use green inports, red outputs, go-to tags colour-matched to the relevant sub-modules; all variable names use the scheme <origin>_<valueType>_<description>

Output/Result: All subsystem layouts and variables follow standardized format.

How test will be performed: Manual inspection.

4.2.2 Performance

1. NFT2

   Type: Manual

   Initial State: Control System ready for embedded code generation.

   Input/Condition: code is generated using Embedded Coder tool. IO needs to be mapped to CAN/ADC drivers. State machine needs to be stepped on a periodic timer. The resulting C program is then deployed to the STM32 board via STMcube IDE.

   Output/Result: The control system runs on the STM32 board, processing ADC input from a potentiometer (driver pedal) and producing corresponding motor command messages on the CAN bus (intercepted using CAN logging tools).

   How test will be performed: Manual code integration with MFE/HAL C libraries.

4.2.3 Maintainability and Support

1. NFT3

   Type: Manual

   Initial State: Control System project open in MATLAB/Simulink.

   Input/Condition: Modification of any module control logic, eg. adding a filter to MaxTorqueCapacity computation in vehicle dynamics.

   Output/Result: Re-running one of the system-level simulations should yield exact same functionality from all other modules, without the need for individual changes.

How test will be performed: Verification using the system-testing model.

2. NFT4

   Type: Manual

   Initial State: Control System project open in MATLAB/Simulink.

   Input/Condition: Modification of any plant model hardware interfacing, eg. changing the scaling factor for torque requests as they are sent over CAN, or changing the IO pin for the pedal sensor.

   Output/Result: Re-running one of the bench-tests with the MFE's LV test bench (has sensor hardware and motors) should yield unaltered operation of the control system despite the hardware parameter change.

   How test will be performed: Verification using the bench-testing model and the MFE team's LV test bench.

3. NFT5

   Type: Manual

   Initial State: Unit test environment for a given module open in MATLAB/Simulink.

   Input/Condition: Unit test case data is loaded to memory, the simulation is run in the testing environment, and signals are set to be logged.

   Output/Result: Simulation runs successfully with the provided inputs and produces module outputs as verified with Simulink Data Inspector and signal logging.

   How test will be performed: Verifying functionality of our custom unit-test environments.

4. NFT6

   Type: Manual

   Initial State: System test environment open in MATLAB/Simulink.

Input/Condition: System test case data is loaded to memory, the simulation is run in the testing environment, and signals are set to be logged.

Output/Result: Simulation runs successfully with the provided inputs and produces control system outputs as verified with Simulink Data Inspector and signal logging.

How test will be performed: Verifying functionality of our custom system-test environment.

4.2.4 Security

1. NFT7

   Type: Manual

   Initial State: A github user who is not a contributor of our project is logged in on github.

   Input/Condition: As of mid-April, User proceeds to our control system repository URL.

   Output/Result: User is not able to view, download or manipulate any aspect of the repository.

4.3 Traceability Between Test Cases and Requirements

Requirement	Test Case
G1	ST_VCS1, ST_VCS2, ST_VCS3
G2	ST_VCS1, ST_VCS2, ST_VCS3
G3	ST_VCS1
G4	ST_VCS1
G5	ST_VCS1
G6	ST_VCS1
G7	ST_VCS2

Requirement	Test Case
DI1	ST_VCS1, ST_VCS2, ST_VCS3
DI2	ST_VCS1, ST_VCS2, ST_VCS3
DI3	ST_VCS1, ST_VCS2, ST_VCS3
DI4	ST_VCS1
DI5	ST_VCS1
DI6	-
DI7	-
DI8	ST_VCS1, ST_VCS2, ST_VCS3
DI9	-
DI10	-

Requirement	Test Case
BM1	ST_VCS1, ST_VCS2, ST_VCS3
BM2	ST_VCS1, ST_VCS2
BM3	ST_VCS1, ST_VCS2

Requirement	Test Case
MI1	ST_VCS1, ST_VCS2, ST_VCS3
MI2	ST_VCS1, ST_VCS2, ST_VCS3
MI3	ST_VCS1, ST_VCS2, ST_VCS3
MI4	ST_VCS1
MI5	-
MI6	-
MI7	ST_VCS1
MI8	-

Requirement	Test Case
VD1	ST_VCS1, ST_VCS2, ST_VCS3
VD2	ST_VCS1, ST_VCS2, ST_VCS3
VD3	-
VD4	-
VD5	-
VD6	-
VD7	ST_VCS3
VD8	ST_VCS3

Requirement	Test Case
NFR1	NFT1
NFR2	NFT2
NFR3	NFT3
NFR4	NFT4
NFR5	NFT5
NFR6	NFT6
NFR7	NFT7

5 Unit Test Description

5.1 Unit Testing Scope

The scope of unit testing is coverage for all functional requirements in the SRS, which were defined based on each module from the outset.

5.2 Tests for Functional Requirements

Functional unit tests are derived directly from unit-based functional requirements from the SRS. These tests are carried out in each module's unit test environment as per Figure 3.

5.2.1 Governor

1. UT_G1

   Type: Manual

   Initial State: State 0 (from startup)

   Input: BM status = 'running', MI status = 'running', DI status = 'Driver start requested', each 1 second apart

   Output: Governor issues MI command 'motor startup' following BM status = 'running'; issues 'Ready to drive' command to DI following MI status = 'running'

   Test Case Derivation: Requirements G1, G2, G3, G4, G5, G6

   How test will be performed: Unit testing environment (Figure 3)

2. UT_G2

   Type: Manual

   Initial State: State 0 (from startup)

   Input: BM status = 'running', MI status = 'running', DI status = 'Driver start requested', DI status = 'running', MI status = 'error'; each 1 second apart

   Output: Governor issues DI command 'system error' following MI status = 'error'

   Test Case Derivation: Requirements G1, G2, G7

   How test will be performed: Unit testing environment (Figure 3)

5.2.2 Driver Interface

1. UT_DI1

   Type: Manual

   Initial State: State 0 (from startup)

   Input: Brake pedal position = 60%, Driver button = ON, 1 second apart; all other inputs 0/default

   Output: DI reports "driver start request" status after button is on

   Test Case Derivation: Requirements DI1, DI2, DI3, DI4, DI5, DI8

   How test will be performed: Unit testing environment (Figure 3)

2. UT_DI2

   Type: Manual

   Initial State: State 0 (from startup)

   Input: repeat UT_DI1 along with DI command = 'ready to drive' to enter drive mode; then Phase 1 - APPS1 = 5000, APPS2 = 2048, other inputs 0; Phase 2 - APPS2 = 5000, APPS1 = 2048, other inputs 0; Phase 3 - BPPS = 5000, other inputs 0; Phase 4 - Steering angle = 5000, other inputs 0

Output: Phase 1 - APPS1 faulted, driver torque request = 50%; Phase 1 - APPS2 faulted, driver torque request = 50%; Phase 3 - BPPS faulted, DI reports 'DI error' status; Phase 4 - Steering position faulted, DI reports 'DI error' status

Test Case Derivation: Requirements DI1, DI2, DI3, DI6, DI7, DI8, DI9

How test will be performed: Unit testing environment (Figure 3)

3. UT_DI3

   Type: Manual

   Initial State: State 0 (from startup)

   Input: repeat UT_DI1 along with DI command = 'ready to drive' to enter drive mode; then set APPS1 = 2048, then after 1 second set DI command = 'system error'

   Output: Driver torque request drops to 50% to 0% in response to the system error

   Test Case Derivation: Requirements DI1, DI2, DI3, DI5, DI8, DI10

   How test will be performed: Unit testing environment (Figure 3)

5.2.3 Battery Monitor

1. UT_BM1

   Type: Manual

   Initial State: State 0 (from startup)

   Input: all contactors start at 0, then 250ms apart: HV negative = 1, precharge = 1, HV positive = 1, precharge = 0

   Output: BM status progresses through 'IDLE', 'START', 'INIT_PRECHARGE', 'PRECHARGE', 'RUNNING'

   Test Case Derivation: Requirements BM1, BM2, BM3

   How test will be performed: Unit testing environment (Figure 3)

2. UT_BM2

   Type: Manual

   Initial State: State 0 (from startup)

   Input: precharge = 1, HV negative = 1, HV positive = 0

   Output: BM status reports 'error precharge' after 1 second

   Test Case Derivation: Requirements BM1, BM2, BM3

   How test will be performed: Unit testing environment (Figure 3)

5.2.4 Motor Interface

1. UT_MI1

   Type: Manual

   Initial State: State 0 (from startup)

   Input: MI command = 'startup', AMK startup signals all 1 (SystemReady, DcOn, QuitDcOn, InverterOn, QuitInverterOn) for both motors, *except* QuitInverterOn for the right motor is 0 for 2 seconds before flipping to 1.

   Output: Left motor status = 'running', followed by Right motor status = 'running'; overall MI status = 'running' only once both motors are running

   Test Case Derivation: Requirements MI1, MI2, MI3, MI4, MI7

   How test will be performed: Unit testing environment (Figure 3)

2. UT_MI2

   Type: Manual

   Initial State: State 0 (from startup)

   Input: startup procedure (see UT_MI1) to get both motors running; then MI command = 'shutdown', AMK shutdown signals all 0 (DcOn, QuitDcOn, InverterOn, QuitInverterOn) for both motors.

   Output: both motors proceed through 'shutdown' to 'off', along with MI status.

Test Case Derivation: Requirements MI1, MI2, MI3, MI5

How test will be performed: Unit testing environment (Figure 3)

3. UT_MI3

   Type: Manual

   Initial State: State 0 (from startup)

   Input: startup procedure (see UT_MI1) to get both motors running; then bError = 1 for the right motor.

   Output: Left motor remains in 'running' state; Right motor transitions to 'error' state; MI status transitions from 'running' to 'error'.

   Test Case Derivation: Requirements MI1, MI2, MI3, MI8

   How test will be performed: Unit testing environment (Figure 3)

5.2.5 Vehicle Dynamics

1. UT_VD1

   Type: Manual

   Initial State: N/A

   Input: Driver torque request = 100%, MaxMotorSpeed = 20000, MaxMotorTorque = 13.8, MaxMotorCurrent = 30, AMK actual velocity (left & right) = 10000; Brake pedal position ramps from [0, 25] % beginning at 3 seconds

   Output: Torque limit positive (left & right) begins at maximum (13.8), then is cut to 0 as the brakes are applied

   Test Case Derivation: Requirements VD1, VD2, VD7, VD8

   How test will be performed: Unit testing environment (Figure 3)

2. UT_VD2

   Type: Manual

   Initial State: N/A

Input: Driver torque request = 100%, MaxMotorSpeed = 20000, MaxMotorTorque = 13.8, MaxMotorCurrent = 30, AMK actual velocity (left & right) = [12000, 14000] over 4 seconds

Output: Torque limit positive (left & right) begins at maximum (13.8) in constant-torque operation, then ramps down as the motor enters constant-power operation

Test Case Derivation: Requirements VD1, VD2, VD6

How test will be performed: Unit testing environment (Figure 3)

5.3 Tests for Nonfunctional Requirements

Our system does not have unit-specific non-functional requirements.

5.4 Traceability Between Test Cases and Requirements

Requirement	Test Case
G1	UT_G1, UT_G2
G2	UT_G1, UT_G2
G3	UT_G1
G4	UT_G1
G5	UT_G1
G6	UT_G1
G7	UT_G2

Requirement	Test Case
DI1	UT_DI1, UT_DI2, UT_DI3
DI2	UT_DI1, UT_DI2, UT_DI3
DI3	UT_DI1, UT_DI2, UT_DI3
DI4	UT_DI1
DI5	UT_DI1, UT_DI3
DI6	UT_DI2
DI7	UT_DI2
DI8	UT_DI1, UT_DI2, UT_DI3
DI9	UT_DI2
DI10	UT_DI3

Requirement	Test Case
BM1	UT_BM1, UT_BM2
BM2	UT_BM1, UT_BM2
BM3	UT_BM1, UT_BM2

Requirement	Test Case
MI1	UT_MI1, UT_MI2, UT_MI3
MI2	UT_MI1, UT_MI2, UT_MI3
MI3	UT_MI1, UT_MI2, UT_MI3
MI4	UT_MI1
MI5	UT_MI2, UT_MI3
MI6	-
MI7	UT_MI1
MI8	UT_MI3

Requirement	Test Case
VD1	UT_VD1, UT_VD2
VD2	UT_VD1, UT_VD2
VD3	- *
VD4	- *
VD5	- **
VD6	UT_VD2
VD7	UT_VD1
VD8	UT_VD1

*met by default since the control system's target motor speed and torque are saturated below 0 (never go below 0)
**out of scope for project; no traction control, stability program, or torque vectoring

Appendix — Reflection

The team will collectively needs to acquire several skills to complete VnV for this specific project, namely: testing with a model-based framework, Simulation environments, dynamic modelling, auto-generated code review/analysis, EV component behaviour including electric motors, battery packs, etc.

Team Member	Knowledge/Skills to develop
Dharak	Leading code reviews, simulation environments
Abhishek	Writing unit tests, testing model-based framework
Jason	Knowledge of EV component behaviour, integration testing
Laura	Integration/system testing, testing model-based framework
Derek	Writing unit tests, code reviews

Approaches to acquire the knowledge and skills mentioned above are as follows:

Team Member	Approach to Acquire Knowledge
Dharak	To strengthen the skills I listed above, I will (1) take advantage of professors and TAs and their guidance on leading code reviews, and (2) review past projects that I've done in Simulink.
Abhishek	In order to refresh my skills on model-based frameworks and writing unit tests (1) review previous projects done in Simulink, and (2) review prior course works and projects where writing unit tests were required.
Jason	To develop the required knowledge and skills I mentioned above, I will (1) consult with the MFE team for their knowledge on the MFE vehicle, and (2) ask my classmates who have more experience than me in performing integration tests
Laura	I will (1) leverage the knowledge of my teammates and classmates on testing model-based frameworks, and (2) review documentation provided by the MFE team so that knowledge can be used in creating relevant tests for the project.
Derek	To acquire knowledge in unit testing and code reviews, I will (1) using prior work experience of my teammates to better understand unit testing and code reviews, and (2) review content from a course taken in software requirements and documentation.