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

Contents

List of Tables

[Remove this section if it isn't needed —SS]

List of Figures

[Remove this section if it isn't needed —SS]

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
MSR	Mode Selection Ring
CCR	Cooling Control Ring
TMR	Tractive Motor Ring
VDR	Vehicle Dynamics Ring
"Ring"	A module responsible for issuing control commands monitoring system feedback (ie. "control ring")

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.

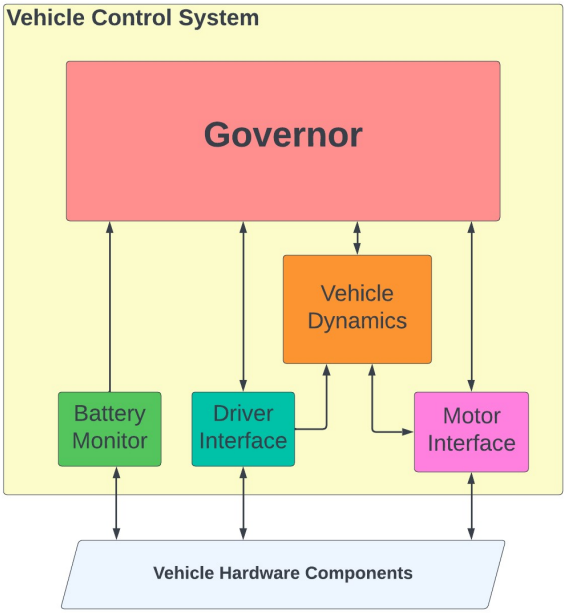

Figure 1: System overview

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 Team	Assist in hardware-related validation, eg. expected hardware operation, safe operation bounds, integration with embedded system
Dharak Verma	System-level validation coordinator
Abhishek Magdum	Vehicle Mode Selection & Validation
Jason Surendran	Battery Monitoring
Laura Yang	Driver Inputs
Derek Paylor	Motor Control & Vehicle Dynamics
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?

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

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.

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.

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.

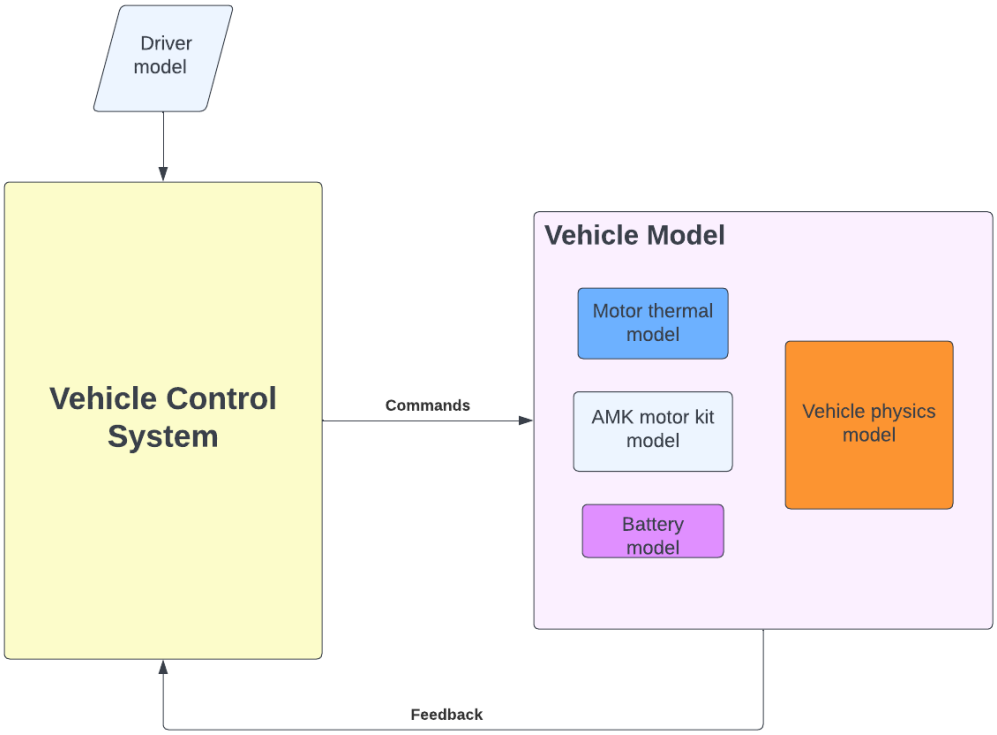

Figure 2: Simulation Environment Overview

System-level verification involves placing our highest-level module (the entire control system, containing all subsystems) in a simulation environment, along with the following simulation models:

- **Driver Model** - supplies timeseries signals as input to the control system. These signals simulate driver input (pedals, steering sensors) for a given operation scenario. This model will supply these pre-defined signals using Signal generator blocks, or From Workspace/CSV blocks.

- **Vehicle 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, temperatures, motor/battery state). This model will be constructed using hardware component documentation (from suppliers and internal McMaster FSAE documentation), Simscape library blocks, and through application of basic first-principles physics.

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

Unit-level verification for individual subsystems / modules (eg. Vehicle Dynamics Ring) also involves a testing environment, albeit simpler than the system-level simulation above. Module outputs Simulink provides many options for viewing and/or processing

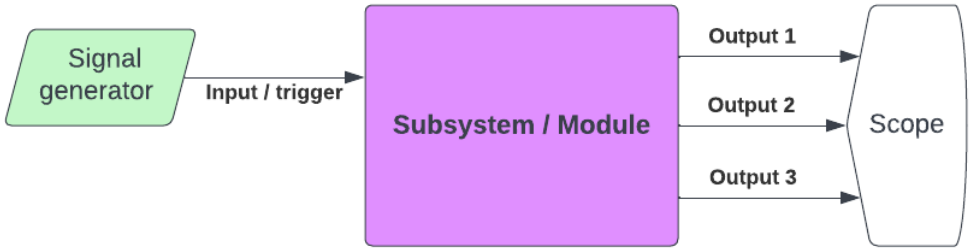

Figure 3: Unit Testing Example Setup

- **Module Inputs** can be simulated in several ways, most notably (1) with a Signal generator (shown in Figure 2) which produces user-specified waveforms, or (2) using From Workspace/CSV blocks, which supply timeseries data that has been pre-defined in the MATLAB workspace or in a CSV file. Encapsulating input data in method (2) helps ensure consistent test scenario across iterations of implementation.
- **Module Outputs** can be visualized via Scope (shown in Figure 2) or Display blocks, and/or can be saved for processing using a To Workspace block. The latter method allows for automated comparison to pre-defined expected values, and again helps ensure consistent test scenarios across iterations.

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, and 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. These environments may be semi-automated using methods described above, such as setting up the test environment to automatically import pre-defined test input data, and export output data for comparison to expected values.

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

The below tests for functional requirements loosely follow the SRS. After further iterating on our control system architecture, and continuing to work closely with MAC Formula Electric to gain a deeper understanding the embedded system in which our control system will be integrated, we chose to remove certain control rings. Therefore, any missing functional requirements from the SRS are intentionally ignored.

4.1 Tests for Functional Requirements

The functional requirements for our system, as covered in the SRS, are split up based on each sub-system of the vehicle we are to control, and thus split into unique control rings in Simulink. The subsections below are split up in the same way, where each subsection covers a single control ring.

4.1.1 Accumulator Management Ring

The subsections below cover the requirements of the Accumulator Management Ring (AMR) such that the inputs to said ring will issue a shut down command to other relevant rings. Compared to the SRS, the new requirement set for the AMR is a heavily stripped down version from previous. The AMR no longer controls or requests contactors to change state and simply receives the current state of the contactors and issues critical system state commands accordingly.

Accumulator Critical Failure

1. FR-CRIT-FAIL-AMR1

   Control: Automatic

   Initial State: Control system is initialized and ready to process incoming requests.

   Input: PreCharge contactor is open, HV Positive and HV Negative contactors are closed.

   Output: AMR error state due to invalid contactor states.

   Test Case Derivation: The PreCharge contactor must be closed before the HV Positive and Negative contactors can close.

   How test will be performed: Simulate contactor status inputs to control system

2. FR-CRIT-FAIL-AMR2

   Control: Automatic

   Initial State: Control system is initialized and ready to process incoming requests. The BMS is periodically sending updates to the control system via CAN.

   Input: No input messages or blank input messages from BMS regarding state.

   Output: AMR error state due to loss of BMS communications.

Test Case Derivation: Control system must maintain communication with BMS, otherwise critical safety scenarios may not be caught in time.

How test will be performed: Disable the simulated BMS status inputs to control system.

4.1.2 Vehicle Dynamics Ring

The subsections below cover requirements that have to be met for the Vehicle Dynamics Ring (VDR). Due to a slight change in architecture, some of the VDR components may differ from the SRS document. The VDR must have multiple safety contingencies in place, to keep the vehicle and driver safe. This includes making sure that any brake application overrides throttle placement. Also, whenever a value received from the accelerator pedal is outside a known range, a fault should be thrown.

Vehicle Dynamics Warnings & Faults

1. FR-WARN-VDR1

   Control: Automatic

   Initial State: Brake Pedal has a force being applied to it.

   Input: Accelerator Pedal is depressed.

   Output: No torque request should be given to the AMK Motor Kit controller.

   Test Case Derivation: Whenever brakes are applied, any throttle application should be ignored. This is a safety concern, where slowing down should always be prioritized over acceleration.

   How test will be performed: Monitor torque requests being sent to motors while the brake and accelerator are both depressed

2. FR-CRIT-FAIL-VDR1

   Control: Automatic

   Initial State: Throttle position is in a known state/value.

Input: The throttle position potentiometer gives the control system a value that is outside an expected range.

Output: A fault should be thrown, and no acceleration torque requests should be sent to the motors.

Test Case Derivation: Any unknown acceleration values should throw a fault, since we wouldn't want any computation errors that could send the car accelerating in an uncontrolled manner. This could mean that there is an issue with the potentiometer reading acceleration values, which is a huge safety concern.

How test will be performed: Inputs will be sent to the Matlab state that is filtering incoming potentiometer values from the accelerator pedal. We will send a value outside of the known range, to test if a hard critical fail fault is thrown, stopping the control system by entering a safe state.

4.1.3 Mode Selection Ring

The subsections below cover the requirements of the Mode Selection Ring (MSR) such that the inputs to said ring will issue commands to request a state change for any/all other relevant rings, depending on inputs

Vehicle Startup

1. FR-VEH-START-MSR1

   Control: Automatic

   Initial State: Control system is uninitialized and ECU just woke up.

   Input: No inputs.

   Output: Waiting on initialization.

   Test Case Derivation: The control system should not spit out random outputs or error outputs while waiting on initialization.

   How test will be performed: Start simulation and provide no inputs to system.

2. FR-VEH-START-MSR2

Control: Automatic

Initial State: Control system is initialized and ready to receive inputs.

Input: Brake pedal, ready to drive button, and tractive system active signals.

Output: The MSR must send the relevant output to the VDR to begin sending motor torque requests if and only if the inputs are all true and send in the correct order of; (1) tractive system active; (2) brake pedal is depressed and held; (3) the ready to drive button is pressed. Otherwise, the system must not tell the VDR to begin motor torque requests.

Test Case Derivation: Conform to rule FSAE EV.10.4.3 ["Ready to Drive"].

How test will be performed: Start simulation and provide inputs in the incorrect order. Can also fluctuate the percentage that the brakes are pressed down while going through startup sequence.

4.2 Tests for Nonfunctional Requirements

4.2.1 Usability

As MFE will be converting our Simulink models to C code, in order to flash it onto their respective ECU, the usability of our control system, from a non-function requirements standpoint, will be revolving around the code conversion process.

1. NFR-CONVERSION-USABILITY

   Type: Manual

   Initial State: Relevant control ring opened and loaded in Simulink.

   Input/Condition: Control system has no errors, both syntactically or functionally.

   Output/Result: Auto-generated C code that replicates the control system functionality 1:1.

   How test will be performed: Utilize Simulink auto-generation feature by hitting "build" and "generate" buttons on Simulink ribbon.

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

4.2.2 Performance

1. Computation Time Analysis

   Type: Automatic

   Initial State: Control system is running normally, taking in all inputs and sending out the corresponding outputs.

   Input/Condition: Setting start and end time variables, to then find the difference between the two when the whole control logic has gone through one loop.

   Output/Result: The output will give us a time value in milliseconds. Based on our judgement, the time value should be below a set value. This is important since we'd like to for example, have the vehicle stop accelerating after the user takes their foot off the accelerator.

   How test will be performed: We will utilize the time.h library to measure time at the beginning and end of the control system logic. Taking the difference of the two will be a simple subs traction of the two values.

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

4.2.3 Maintainability and Support

This section will primarily focus on how easy it is to iterate and update the control system once it is passed down to the members of MFE.

1. NFR-MODEL-SUPP

   Type: Manual

   Initial State: Relevant control ring opened and loaded in Simulink.

   Input/Condition: Modification of any logic within opened control ring.

   Output/Result: After minor addition or modification of control ring logic, the control system as a whole should continue functioning as expected. Even if a breaking change is introduced, the control system a whole should not stop functioning but instead output the correct error based on provided inputs to the ring.

   How test will be performed: Control logic on a single ring will be modified and system will be run through all functional tests.

4.2.4 Security

Security, regarding our control system, will be purely focused on the software and maintainability aspect once it is passed down to MFE. We will not be accounting for security related to the hardware on which the control system will run, as that is outside the scope of this project.

1. NFR-MAINT-SECURITY

   Type: Automatic

   Initial State: Control system and all relevant files are uploaded to the central Git repository.

   Input/Condition: Control system or any relevant file is modified and user attempts to push change directly to main branch of Git repository.

   Output/Result: The push to main fails and the user is required to create a branch, push their changes to the branch, and create a pull or merge request into main.

How test will be performed: By cloning main branch of Git repository, making modifications to cloned files, and attempting to push directly to main.

4.3 Traceability Between Test Cases and Requirements

Requirement	Test Case
AMR1	FR-CRIT-FAIL-AMR1
AMR2	FR-CRIT-FAIL-AMR2
VDR1	FR-WARN-VDR1
VDR1	FR-CRIT-FAIL-VDR1
MSR1	FR-VEH-START-MSR1
MSR2	FR-VEH-START-MSR2

5 Unit Test Description

As our MIS detailed design document has been completed, this section will be more fleshed out.

5.1 Unit Testing Scope

Our unit testing scope is specific to the Simulink control systems we actually develop (Governor, Vehicle Dynamics, Battery Monitor, Driver Interface, and Motor Interface), and by extension the vehicle model (motor thermal model, AMK motor kit model, battery model, and vehicle physics model) which will be used to test the control system. We are not verifying the default/imported Simulink blocks that our control system consists of. In other words for the Simulink default/imported blocks we assume they work as intended and are reliability outputting their respective expected values.

5.2 Tests for Functional Requirements

Using the automated feedback cycles that Simulink uses, we will be able to test the following functional requirements. As mentioned earlier this will be done by creating a vehicle model with its relevant components (motor thermal model, AMK motor kit model, battery model, and vehicle physics model) and using it alongside the driver model to interact with our vehicle

control system and seeing if the inputs/outputs are as expected. There will not be any unconventional unit tests conducted.

5.2.1 Accumulator Management Ring

Will be expanded upon once MIS document is completed

1. test-id1

   Type:
   Initial State:
   Input:
   Output:
   Test Case Derivation:
   How test will be performed:

2. test-id2

   Type:
   Initial State:
   Input:
   Output:
   Test Case Derivation:
   How test will be performed:

3. ...

5.2.2 Vehicle Dynamics Ring

Will be expanded upon once MIS document is completed

1. test-id1

   Type:

Initial State:

Input:

Output:

Test Case Derivation: How test will be performed:

2. test-id2

   Type:

   Initial State:

   Input:

   Output:

   Test Case Derivation:

   How test will be performed:

3. ...

5.2.3 Mode Selection Ring

Will be expanded upon once MIS document is completed

1. test-id1

   Type:

   Initial State:

   Input:

   Output:

   Test Case Derivation:

   How test will be performed:

2. test-id2

   Type:

Initial State:

Input:

Output:

Test Case Derivation:

How test will be performed:

3. ...

5.3 Tests for Nonfunctional Requirements

Will be expanded upon once MIS document is completed

5.4 Traceability Between Test Cases and Modules

Each controls subsystem is clearly defined, and since out test modules (i.e Vehicle Dynamics Model) interacts with multiple subsystems, there is an extremely low probability that something does not get tested. Our controls subsystems are heavily dependant on feedback with each other, and communicate with our Governor, thus all subsystems along with their corresponding test module must be in a working state. Most test cases holistically test multiple subsystems and thus we will ultimately have full coverage. Unless all aspects of the test vehicle model (i.e motor thermal model, amk motor kit model etc) is actually completed and are interacting with each other, our test cases for the vehicle control system will not be able to run in the first place.

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.