Verification and Validation Report: CVT Simulator

Team #17, Baja Dynamics
Grace McKenna
Travis Wing
Cameron Dunn
Kai Arseneau

April 4, 2025

1 Revision History

Date	Version	Notes
March 10 2025	1.0	Initial Version

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test

Contents

1 Revision History **i**

2 Symbols, Abbreviations and Acronyms **ii**

3 Functional Requirements Evaluation **1**
3.1 Simulation Model 1
3.1.1 Position 1
3.1.2 Velocity 1
3.1.3 Acceleration 2
3.1.4 Clamping Forces 3
3.1.5 Shift 4
3.1.6 Engine Dynamics 6
3.2 User Interface 6

4 Nonfunctional Requirements Evaluation **7**
4.1 Accuracy 7
4.2 Usability 7
4.3 Maintainability 8
4.4 Verifiability 8
4.5 Understandability 9
4.6 Reusability 10

5 Safety Requirements Evaluation **10**
5.1 Safety Requirement 1 10
5.2 Safety Requirement 2 11

6 Comparison to Existing Implementation **11**

7 Unit Testing **11**
7.1 Back End Unit Testing 11
7.2 Front End Unit Testing 13

8 Changes Due to Testing **13**
8.1 Back End Changes 13
8.2 Front End Changes 14

9 Automated Testing **14**

10 Trace to Requirements **15**

11 Trace to Modules **16**

12 Code Coverage Metrics **17**

List of Tables

1 System Functional Requirements and Corresponding Tests . . 15
2 System Nonfunctional Requirements and Corresponding Tests 15
3 Software Modules and Corresponding Tests 16

List of Figures

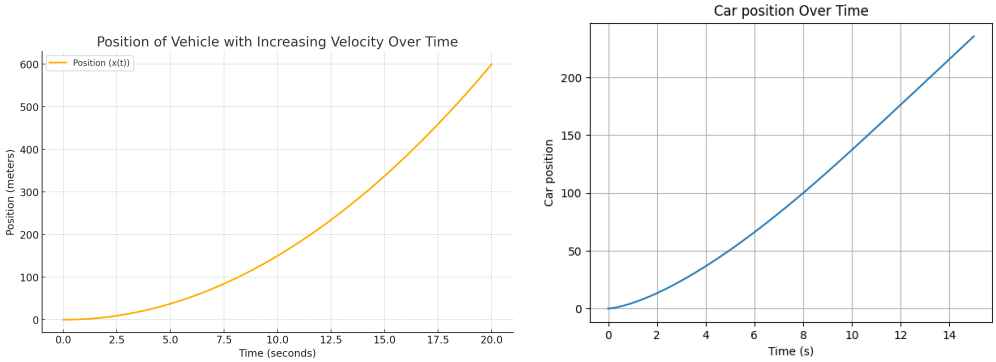
1 Back End Code Coverage 17
2 Front End Code Coverage 17

3 Functional Requirements Evaluation

3.1 Simulation Model

3.1.1 Position

1. **Position test-1**: Graphical validation

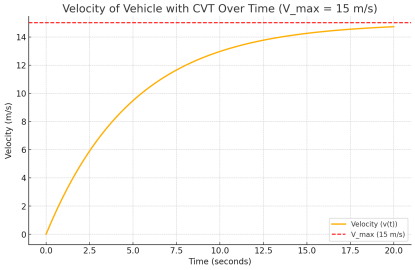


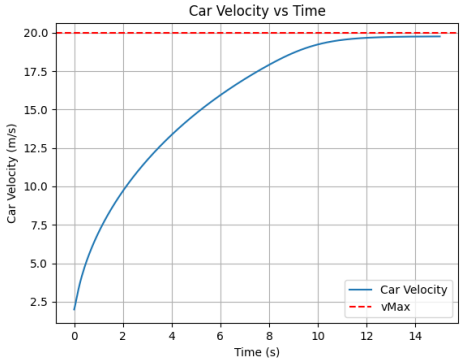
As seen above, the simulation output for the position of the car is consistent with the theoretical model. The larger initial slope can be attributed to an initial velocity, which is required due to **A:NS No belt slippage** in SRS.

2. **Position test-2**: MSE against experimental data
   See the VnV Extra.

3.1.2 Velocity

1. **Velocity test-1**: Graphical validation



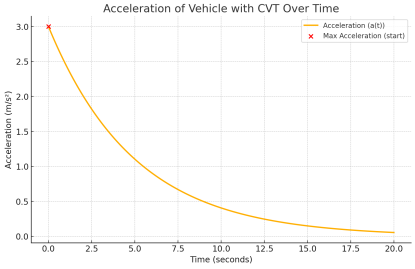


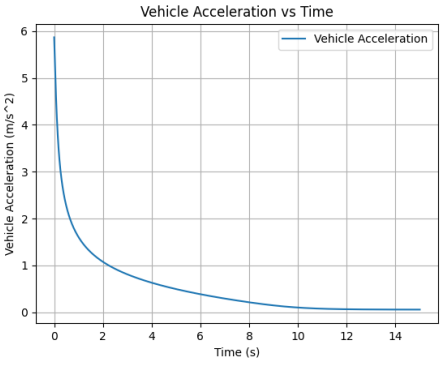
As seen above, the simulation output for the velocity of the car is consistent with the theoretical model. The $v_{\max}$ may be subject to changes according to approximations when calculating air resistance.

2. **Velocity test-2**: MSE against experimental data
   See the VnV Extra.

3.1.3 Acceleration

1. **Acceleration test-1**: Graphical validation



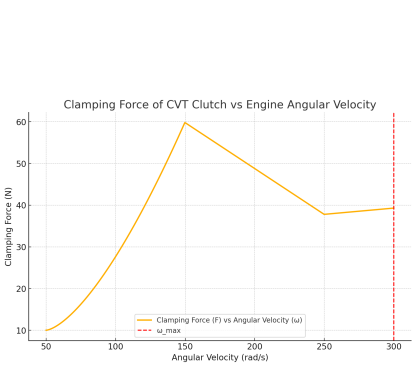


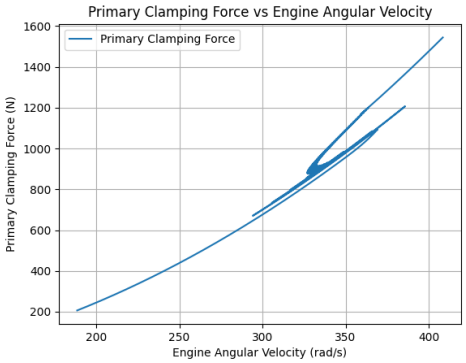
The vehicle's acceleration matches the general shape of the theoretical model. The slope of the theoretical model was simply an estimate, meant to capture the shape of the graph, and so is a bit less accurate than the other graphs. Despite this, both share a common structure and follow expected trends.

2. **Acceleration test-2**: MSE against experimental data
   See the VnV Extra.

3.1.4 Clamping Forces

1. **Clamping forces test-1**: Graphical validation





The shape of both graphs above do not match. While the expected result is an initial sharp increase, which then leads to a steady decrease, followed finally by an increase at the end, here we can see what is effectively exclusively an increase with some oscillations. This difference can be explained thanks to two factors.

First, the oscillations arise due to **A:NS No belt slippage** in SRS, which result in an underdamped system.

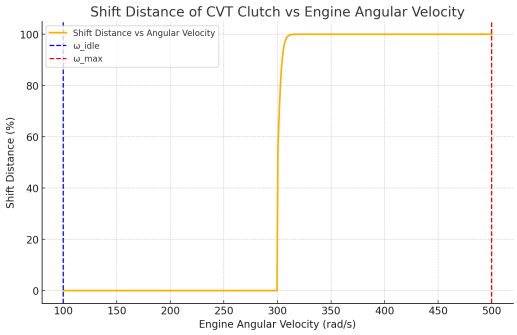
Secondly, the exclusive increase is due to a poor representation of the complex ramp geometry. While in the real world, we know the ramp begins linear, then transitions to a gentle curve that gives us the decrease in force as we shift, the simulation represents it as a linear line. This causes a more strict rising of the force at each corresponding angular velocity of the engine.

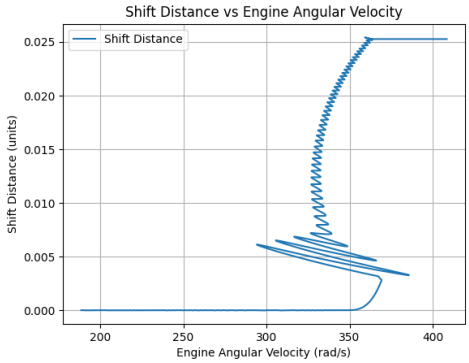
The impact this has on the system is quite miniscule. As long as the initla clamping forces which drive the shifting behaviour are correct, the majority of the system should behave as expected. Since our shifting occurs at one specific angular velocity, we need not worry about some of the extremeties being too far off. Further, the real world downsides of having an ever-increasing clamping force would be an excess of heat,

which is ignored in our simulation via **A5: Ignore Wear and Tear** and **A1: No Material Variations by Temperature** in SRS

3.1.5 Shift

1. **Shift test-1**: Graphical validation 1





We can see the graphs are quite close in shape. Some key differences include:

- the magnitude at which we shift, which is variable and can be altered based on the input parameters.
- The oscillations in the simulation are due to **A:NS No belt slippage** in SRS, which cause an underdamped system.
- The slight slope within the simulation. This is due to the poor ramp representation alongside other input parameters. Will be discussed more in Shift test-3.

2. **Shift test-2**: MSE against experimental data 1
   See the VnV Extra.

3. **Shift test-3**: Graphical validation 2

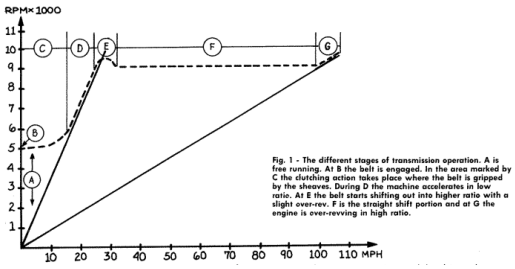
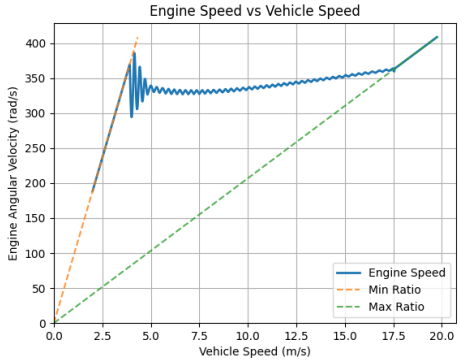


Fig. 1 - The different stages of transmission operation. A is free running. At B the belt is engaged. In the area marked by C the clutching action takes place where the belt is gripped by the sheaves. During D the machine accelerates in low ratio. At E the belt starts shifting out into higher ratio with a slight over-rev. F is the straight shift portion and at G the engine is over-revving in high ratio.



The shift curve seen here shows much about the system, as so we will go over each section within the graph.

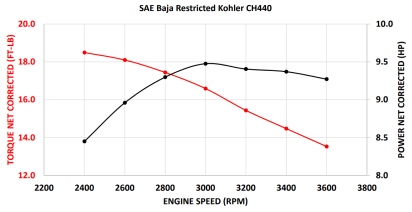
- Phase A to C: Due to **A:NS No belt slippage** in SRS, it is unreasonable to omdel these portions. Were the engine tied to the vehicle's tired without slip, at low velocities the engine would be spinning lowering than it is capable of maintaining, and would thus kill the system. The solution in our simulation is to skip this portion, beginning at an initial velocity of around 3.5m/s.
- Phase D and E: This portion is modelled quite well. Here we see the vehicle and engine gaining speed at the low-ratio gear. We also see a slight overshoot thanks to the momentum of the system.
- Phase F: This is the shifting phase, which has the bulk of the calculations in our system affecting. Overall, a similar shape is reflected in both, with a mostly horizontal line. The slight slope in the simulation arises as we begin to shift too slowly, which allows the engine to gain speed. This can be attributed to several factors, but the most prominent is the poor ramp representation. The ramp is modelled as a linear line, which causes the system to shift too slowly. This is a minor issue, as the system still shifts at the correct time, but the speed at which it shifts is too slow.
- Phase G: In the final phase, we see the system accelerate at the high-ratio gear. Both graphs match exactly in this portion.

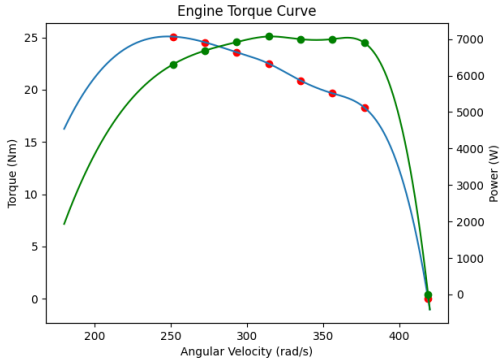
4. **Shift test-2**: MSE against experimental data 2

See the VnV Extra.

3.1.6 Engine Dynamics

1. **Engine test-1**: Graphical validation





As seen in these two graphics, all points within the theoretical figures are represented exactly in the simulation. The only addition is seen via a point at our max engine RPM of 4000, which is limited by the official Baja SAE rules. The produced torque at this point drops to 0, as a restrictor plate ensures no additional power may be produced beyond this speed.

3.2 User Interface

1. **User Interface test-1**: Input Parameters
   The input parameters are loaded with the default values. The user can adjust the parameters and the simulation will run with the new parameters. The user can also save the parameters to a file and load them back in. The user can also export the simulation data to a file.

2. **User Interface test-2**: User Interface
   When the user launches the application they are presented with a main menu. From here they can navigate to the simulation screen where they can adjust the parameters and run the simulation

3. **User Interface test-3**: Viewing Data Outputs
   The user can view the data outputs in the simulation screen. The user has an option to view the data displayed in a graph format and the

user can see the position, velocity, acceleration, and shift distance of the car over time.

4. **User Interface test-4**: Saving and Exporting Data
   From the results screen the user can save the parameters inputted to the simulation to a file. The user can also export the simulation data to a csv file for further analysis.

4 Nonfunctional Requirements Evaluation

This section will cover the evaluation of the Nonfunctional Requirements.

4.1 Accuracy

The accuracy test involves the outputs of the following Functional Test cases: **Position test-2**, **Velocity test-2**, **Acceleration test-2**, **Shift test-2** and **Shift test-4**. The following Accuracy test is to be completed in the VNV EXTRA.

1. **Accuracy test-1**: This test will take the outputted data of above listed functional tests. As each of these tests will output the MSE, each MSE should fall within 0.2 of 0. A range within 0.2 of 0 for each MSE outputted will indicate an accurate system.

4.2 Usability

The Usability/Understandability survey remains in progress at this time and results will be discussed in the Usability Report. Therefore, Usability test-1 and Usability test-2 have not been fully completed yet, however as they are in progress these tests will be completed as future work.

1. **Usability test-1**: Navigating Main Interface
   Survey question: On a scale of 1-5 with 1 being extremely difficult and 5 being extremely easy, how easy was it to navigate the main interface?

2. **Usability test-2**: Use of Most Common Features
   Survey question: For the following main features: Inputting parameters, Adjusting parameters, Viewing data outputs, Saving and export-

ing data. Rate each feature on a scale of 1-5 with 1 being extremely difficult and 5 being extremely easy

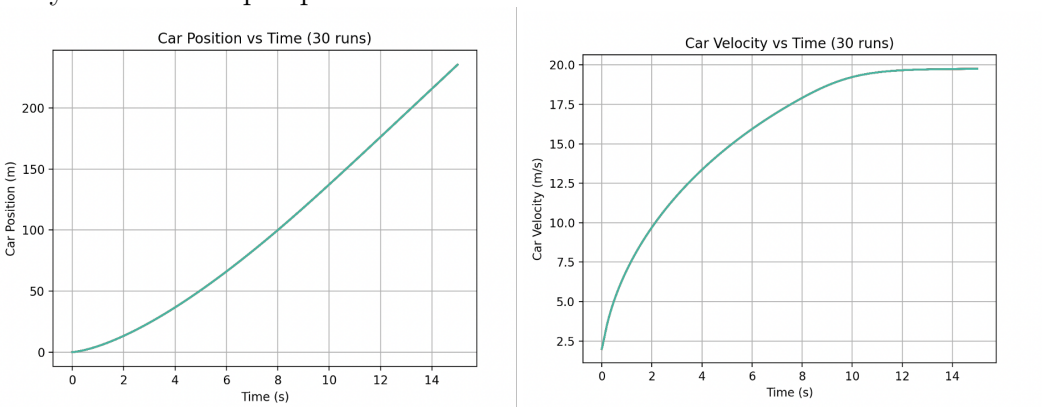
4.3 Maintainability

The maintainability tests are designed to test how maintainable the system is given likely future changes.

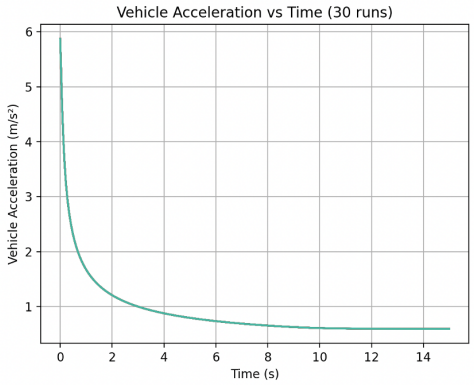
1. **Maintainability test-1**
   This test was manually completed where the amount of time and number of lines of code will be recorded when implementing the changes that correspond to the 2023 CVT configuration. Implementation of the 2023 CVT parameters takes at most 20 minutes with modifications to appropriately 26 lines of code in the `car_specs.py`. Thus, the system is easily maintainability and the number of lines modified and time taken is minimal given a total change in parameters of the CVT.

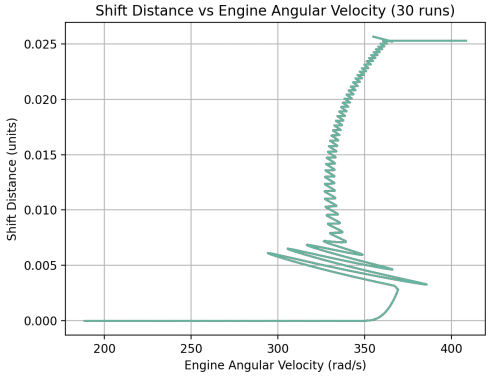
4.4 Verifiability

The tests for verifiability involve completing **Position test-1**, **Velocity test-1**, **Acceleration test-1**, **Shift test-1** and **Shift test-3**. For each of the above tests, they will be run 30 times with the same input parameters to verify that the output produced is the same each time.

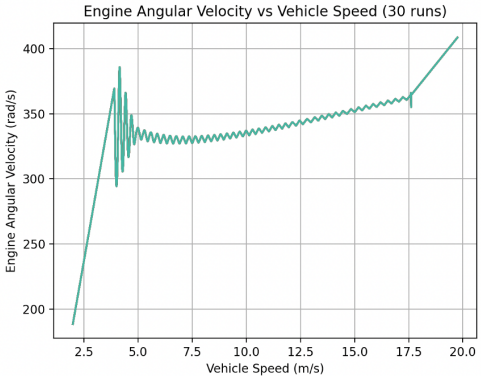


The graphs above show the Car Position vs Time(Left) and the Car Velocity vs Time(Right) after 30 simulations with the same inputs. These tests correspond to **Position test-1** and **Velocity test-1**.





The graphs above show the Car Acceleration vs Time(Left) and the Shift Distance vs Engine Angular Velocity(Right) after 30 simulations with the same inputs. These tests correspond to **Acceleration test-1** and **Shift test-1**.



The graph above show the Engine Angualr Velocity vs Vehicle Speed. These tests correspond to **Shift test-3** and **Velocity test-1**.

As seen, based on the graphs above, there is little to no variation when performing the tests: **Position test-1**, **Velocity test-1**, **Acceleration test-1**, **Shift test-1** and **Shift test-3** 30 times on the same input data. This passes the test for verifiability.

4.5 Understandability

The Usability/Understandability survey remains in progress at this time and results will be discussed in the Usability Report. Therefore, Understandability test-1 and Understandability test-2 have not been completed yet, however

as they are in progress these tests will completed as future work.

1. **Understandability test-1**
   Survey question: For the following main features: Inputting parameters, Adjusting parameters, Viewing data outputs, Saving and exporting data. Was the purpose of each function clear, on a scale of 1-5 with 1 being very unclear and 5 being extremely clear.

2. **Understandability test-2**: Understanding Simulation Outputs
   Survey question: On a scale of 1-5, with 1 being very unclear and 5 being extremely clear, how well did you understand the simulation results and output?

4.6 Reusability

The reusability tests are designed to assess how easily the system can be adapted to new or modified configurations.

1. **Reusability test-1**
   This test was manually performed by the teams, where the time taken and number of lines of code modified were recorded when implementing the 2023 CVT configuration. Implementation of the 2023 CVT parameters requires at most 20 minutes and involves modifications to approximately 26 lines of code in the `car_specs.py` file. These minimal changes demonstrate the system's high reusability, as it can be quickly adapted to new CVT configurations with little effort and modification.

5 Safety Requirements Evaluation

The safety requirements are designed to ensure that the system is safe to use and does not cause harm to the user or the environment.

5.1 Safety Requirement 1

1. **SR1 Test-1**
   This test was manually performed by the team. During the execution, when a value was inputted that caused the belt tension to exceed the maximum limit, the system promptly notified the user and stopped the

simulation. The error message was clear and easily understood, with the simulation halting within approximately 2 seconds of the input. These results confirm that the safety mechanism for belt tension is working as intended, ensuring immediate user notification and system shutdown upon detecting unsafe conditions.

5.2 Safety Requirement 2

1. **SR2 Test-1**
   This test was manually performed by the team. When the input causing an error was introduced, the system immediately threw the error and stopped the simulation. A clear error message was displayed, including relevant debugging information, enabling the user to understand what went wrong. The error handling was prompt and efficient, confirming that the system meets the safety requirement for robust error management.

6 Comparison to Existing Implementation

Not applicable for this project.

7 Unit Testing

7.1 Back End Unit Testing

The back end was fully covered by unit tests except for a several files which were not suited for unit testing as they were mainly constants or did not provide functionality that could be tested.

The test structure that was created was to essentially mirror how the codebase is organized. There is a test folder which houses all the tests, then inside that there is a simulations and utils folder which contain mirrored test files of the actual files under src/simulations or src/utils.

The tests were written using the built-in unittest module in python. The tests were run using the command coverage run -m unittest discover -s test/simulations -s test/utils. This command runs all the tests in those two

folders using the coverage library.

A coverage report was generated using the command coverage report -m.

This command generates a report that shows the percentage of code that was covered by the tests. The report also shows which lines were not covered by the tests. This can be seen in section 11 of this document.

The following is the file breakdown for the backend unit tests within test/simulations directory:
`test_belt_simulator.py` - This test suite verifies that the BeltSimulator's methods correctly compute centrifugal force, clamping-derived radial forces, net radial force, slack tension, and maximum transferable torque according to theoretical models.
`test_car_simulator.py` - This test verifies that CarSimulator calculates acceleration correctly as force divided by mass.
`test_engine_simulation.py` - This test suite verifies that EngineSimulator correctly initializes and computes torque, power, and angular acceleration using a mock torque curve.
`test_load_simulation.py` - This test suite verifies that the LoadSimulator correctly computes vehicle load forces (incline, drag, total, gearbox) and acceleration given specific parameters.
`test_primary_pulley.py` - This test suite verifies that the PrimaryPulley class correctly computes flyweight force, spring compression force, net force, and properly handles shift distance limits.
`test_secondary_pulley.py` - This test suite verifies that the SecondaryPulley correctly computes helix force, spring compression force, spring torsion torque, and net force based on theoretical models and geometric parameters.

The following is the file breakdown for the backend unit tests within test/utils directory:
`test_argument_parser.py` – This test suite verifies that `get_arguments` correctly parses command line inputs into an object for both default and custom scenarios.
`test_conversions.py` - This test suite verifies that conversion functions for angular velocity, angle measurements, circumference, and inch-to-meter conversions return correct values.
`test_simulation_result.py` - This test suite verifies that simulation result

correctly parses ODE solutions into system state objects, writes the results to a CSV file, and calls the plot method as expected.
`test_system_state.py` - This test suite verifies that the SystemState class initializes correctly and accurately converts to and from array representations.
`test_theoretical_models.py` - This test suite verifies that Theoretical Models correctly compute various physics and geometry calculations—including Hooke's law, centrifugal force, air resistance, torque, gearing, friction, capstan equation, Newton's second law, and CVT and wrap angle parameters.

7.2 Front End Unit Testing

The front end communication protocol was fully covered by unit tests besides the function that calls the python script since it is dependent on the back end Implementation.
The tests are stored in a separate project within the front end solution. They are written using the MS Unit Testing Framework and are run using the dotnet test command.
A coverage report was generated using the JetBrains dotCover tool which shows the percentage of code that was covered by the tests.

8 Changes Due to Testing

8.1 Back End Changes

Many changes were implemented at various stages of the process. Early on, we had feedback on the mathematical models that shaped the simulation. Once we began implementing some of the independent forces, we once again reviewed and tested some of the calculations and made adjustments to the equations to ensure they were within reasonable ranges.

These changes took place across the entire application on the backend, specifically referring to the mathematical implementation. The aspects that were left untouched were the shape and structure of the code, but all else was altered at least once throughout the process.

8.2 Front End Changes

On the front end, many changes were made to support unit testing. The communication protocol was separated from the main Unity project since Unity was not compatible with the MS Unit Testing Framework.
Moving the communication protocol into a separate project allowed for the simplification of the code base by isolating responsibilities and improving modularization.
There were also changes on the front end to support the requested changes from the Rev 0 demo. The input parameters were updated to be stored in a csv file and read from there.
These changes were to facilitate the implementation of uploading and downloading the parameters from the front end which was a requested feature from the Rev 0 demo.

9 Automated Testing

Both back and front end sets of unit tests were automated and added to the CI/CD pipeline. Now whenever there is a commit it must pass all the tess in order to be verified. If there is an error it will fail and state which tests failed. This is a good way to ensure that the code is always working as expected.

As well the back end coverage report is also generated in the CI/CD run so you can see how much of the code is being covered by tests at that moment.
The configuration for the CI/CD can be seen here: https://github.com/gr812b/CVT-Simulator/blob/develop/.github/workflows/ci.yaml

10 Trace to Requirements

Requirement	Test(s)
R1	Position test-2, Velocity test-2, Acceleration test-2, Shift test-4
R2	Acceleration test-1, Unit Tests-`test_car_simulator.py`
R3	Velocity test-1
R4	Position test-1
R5	Clamping forces test-1, Unit Tests-`test_primary_pulley.py`
R6	Clamping forces test-1, Unit Tests-`test_secondary_pulley.py`
R7	Shift test-1, Shift test-2, Shift test-3
R8	Shift test-1, Shift test-2, Shift test-3, `test_belt_simulator.py`
R9	Engine test-1, Unit Tests-`test_engine_simulation.py`
R10	User Interface test-1, Unit Tests-`test_argument_parser.py`
R11	User Interface test-1,
R12	User Interface test-2
R13	User Interface test-3
R14	User Interface test-4
R15	Compatibility test-1
R16	Compatibility test-1

Table 1: System Functional Requirements and Corresponding Tests

Requirement	Test(s)
NFR1	Accuracy test-1
NFR2	Useability test-1, Useability test-2
NFR3	Maintainability test-1
NFR4	Verifiability test-1
NFR5	Understandability test-1, Understandability test-2
NFR6	Reusability test-1

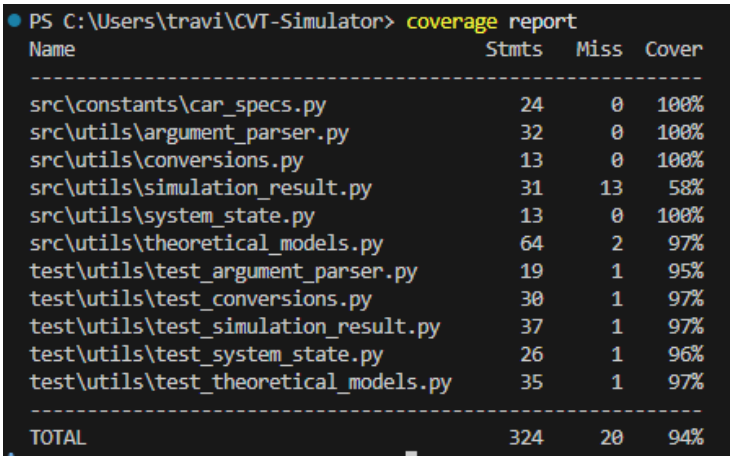
Table 2: System Nonfunctional Requirements and Corresponding Tests

11 Trace to Modules

Module	Test(s)
M1	
M2	Engine test -1, Unit Tests-`test_engine_simulation.py`
M3	Velocity test-1, Acceleration test-1, Position test-1, Unit Tests-`test_car_simulator.`
M4	Clamping forces test-1, Unit Tests-`test_primary_pulley.py`
M5	Clamping forces test-1, Unit Tests-`test_secondary_pulley.py`
M6	
M7	Tested by SciPy
M8	Shift test-1, Shift test-2, Shift test-3, `test_belt_simulator.py`
M9	Unit tests for CSV Reader
M10	
M11	
M12	
M13	
M14	User Interface test-1, 2, 3, 4
M15	
M16	Unit tests for Python Runner

Table 3: Software Modules and Corresponding Tests

12 Code Coverage Metrics



```
PS C:\Users\travi\CVT-Simulator> coverage report
Name                                      Stmts   Miss  Cover
-----------------------------------------------------------
src\constants\car_specs.py                   24      0   100%
src\utils\argument_parser.py                 32      0   100%
src\utils\conversions.py                     13      0   100%
src\utils\simulation_result.py               31     13    58%
src\utils\system_state.py                    13      0   100%
src\utils\theoretical_models.py              64      2    97%
test\utils\test_argument_parser.py           19      1    95%
test\utils\test_conversions.py               30      1    97%
test\utils\test_simulation_result.py         37      1    97%
test\utils\test_system_state.py              26      1    96%
test\utils\test_theoretical_models.py        35      1    97%
-----------------------------------------------------------
TOTAL                                       324     20    94%
```

Figure 1: Back End Code Coverage

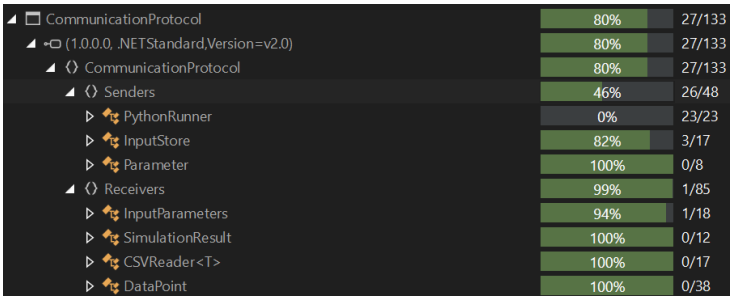


Figure 2: Front End Code Coverage

References

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Reflection.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

   Creating the unit tests for the back end was relatively simple as the code was already structured in a way that made it easy to unit test. The process of creating the test just involved going through each function and comparing the expected output to the actual output.

2. What pain points did you experience during this deliverable, and how did you resolve them?

   Getting front end unit testing to work was a challenge since Unity was not compatible with many unit testing frameworks. The solution was to separate the communication protocol into its own project which then allowed us to use the MS Unit Testing Framework. This also allowed us to simplify the code base by isolating responsibilities and improving modularization.

3. Which parts of this document stemmed from speaking to your client(s) or a proxy (e.g. your peers)? Which ones were not, and why? Sections 7.1, 7.2, 4, 3. Many sections throughout this document were heavily affected by peers on the McMaster Baja Racing team. This

was accomplished through regular meetings and feedback, as well as more formal sitdowns and presentations. In turn this has effected the following sections:

- 3 Many discussions surrounding the mathematical model and the theoretical results occured to create these tests in the first place, as well as actively inspect various graphs. Further live data collection occured throughout the team.
- 4 Worked closely with members to attempt some changes on the system that are likely in the following years to come, as well as the usability report.
- 7 Many changes applied to the system in both front end and back end were done so due to feedback received from either our supervisor, Dr. Smith or other members on the team when showed the product or questioned about mathematics, and such this was critical.

Did not receive help regarding unit testing or automated testing, as we were the domain experts on the codebase and could ensure sufficient code coverage. Many of the peers we worked closely with were not software oriented, and the challenge did not lie here.

4. In what ways was the Verification and Validation (VnV) Plan different from the activities that were actually conducted for VnV? If there were differences, what changes required the modification in the plan? Why did these changes occur? Would you be able to anticipate these changes in future projects? If there weren't any differences, how was your team able to clearly predict a feasible amount of effort and the right tasks needed to build the evidence that demonstrates the required quality? (It is expected that most teams will have had to deviate from their original VnV Plan.)

   The back end unit testing was conducted as planned in the VnV Plan while the front end unit testing required modification to our approach. On the front end, the Unity Testing Framework was significantly less user friendly and comprehensive than we expected. This required us to change our approach to unit testing on the front end and move towards our current solution. This was not anticipated in the VnV Plan as we

were not aware of the limitations of the Unity Testing Framework. In future projects, we would be able to better anticipate these changes by doing more research on the tools we plan to use. On the back end, we were able to correctly predict the amount of effort and the tasks needed based on our previous experience with unit testing. The back end unit testing followed the standard procedure and was able to easily leverage existing packages and tools to support the testing.