HOCHSCHULE ESSLINGEN
Nah an Mensch und Technik.

An Integrated MBSE and Multibody Dynamics Approach for the Development of a High-Fidelity Vehicle Digital Twin

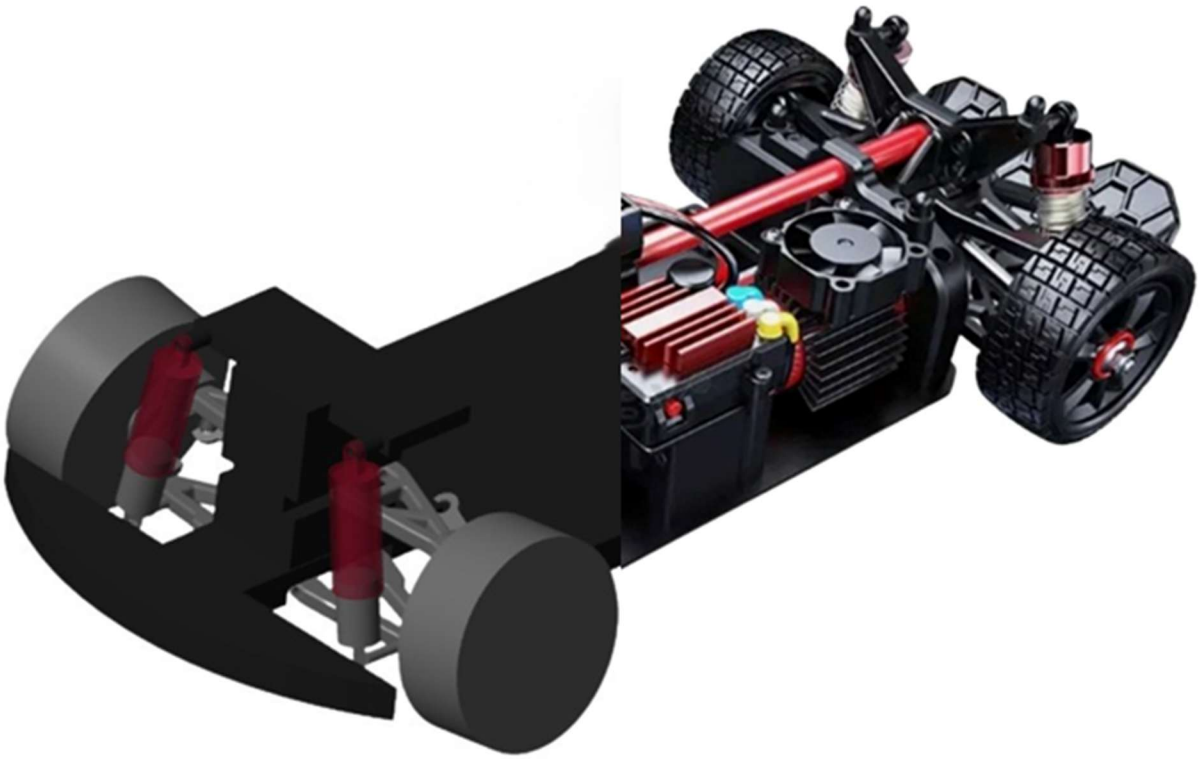

Figure 1: Digital Vehicle Twin

Neil Ponsonnard - 780556
Shikhar Takke - 780552
Raj Srivastava - 780538
Keshav Sundar - 772077

Supervisor: Prof. Ralf Schuler
Hochschule Esslingen

Table of contents

I. Introduction: The Convergence of Digital Twinning and MBSE in Automotive Validation

Context and Motivation

The automotive industry is undergoing a profound transformation, driven by the rapid development of automated driving systems (ADS) and advanced driver-assistance systems (ADAS). The complexity of these systems, which involve a tight coupling of perception, planning, and control algorithms with vehicle dynamics, presents unprecedented challenges for validation and verification. Traditional development cycles, heavily reliant on physical prototypes and track testing, are becoming prohibitively expensive and time-consuming. In response, the industry is shifting towards a "shift-left" paradigm, emphasizing virtual testing in high-fidelity simulation environments to identify and resolve issues early in the design process. This project, undertaken for the Center for Automated Driving and Service Technology (CAST), directly addresses this need by developing a comprehensive simulation model of a scaled vehicle, intended to serve as a foundational tool for testing automated driving functions.
A critical enabler for this shift is the use of miniaturized traffic environments. The establishment of a 1:14 scale environment at the CAST lab provides a cost-effective, safe, and agile platform for validating complex algorithms. Such platforms allow for rapid iteration and the execution of thousands of test scenarios that would be impractical or dangerous with full-scale vehicles. However, the value of this physical testbed is magnified exponentially when paired with an equally high-fidelity virtual counterpart—a digital twin. This project's primary objective was to create such a digital twin for the HYPER GO LR14 PRODRIFT 1:14 scale vehicle, focusing on its geometry and mechanical properties to create a benchmark model for future simulation-based validation efforts.

Defining the Modern Digital Twin

The concept of the digital twin has evolved far beyond a simple three-dimensional CAD model. A modern digital twin is a dynamic, multi-physics virtual representation of a physical asset, system, or process that is persistently synchronized with its real-world counterpart throughout its lifecycle. In the automotive context, a vehicle's digital twin integrates real-time data and advanced simulation techniques to replicate performance, behavior, and physical conditions under a vast array of operating scenarios. This enables a transformative approach to engineering, allowing for virtual prototyping, predictive maintenance, and continuous performance optimization.
Recent research and industry applications, particularly those recognized by organizations like the Society of Automotive Engineers (SAE), highlight the digital twin's role in accelerating development. By creating a virtual model that accurately imitates a physical vehicle, engineers can conduct extensive virtual experimentation, reducing the reliance on costly physical prototypes and test tracks. This project embraces this advanced definition by aiming not just to model the vehicle's geometry, but to create a dynamic, physics-based simulation model capable of predicting the vehicle's response to various maneuvers, thereby laying the groundwork for a true digital twin that can be used for comprehensive ADAS validation. The development of such high-fidelity digital twins using tools like Simscape Multibody is a well-documented and effective approach in both academia and industry for applications ranging from robotics to complex vehicle systems.

The Role of Model-Based Systems Engineering (MBSE)

As vehicle systems grow in complexity, managing the intricate web of requirements, functions, logical components, and physical hardware becomes a monumental task. Traditional document-centric engineering approaches, which rely on static text documents and diagrams, are often fraught with ambiguity, inconsistency, and a lack of traceability. Model-Based Systems Engineering (MBSE) represents a paradigm shift to address these challenges, formalizing the practice of systems engineering through the use of interconnected, domain-specific models as the primary means of information exchange.
MBSE provides a "single source of truth" that ensures all stakeholders, from systems architects to domain-specific engineers, are working from a consistent and up-to-date system definition. Tools like MathWorks System Composer are designed to facilitate this process, enabling engineers to intuitively sketch hierarchical system architectures using a formal component, port, and connector modeling approach. By capturing the system's architecture in a formal model, teams can perform early-stage analysis, manage requirements traceability, and automatically generate documentation, significantly improving communication and reducing integration errors. This project's implementation of an MBSE workflow using System Composer is therefore not an ancillary task but a core component of its modern engineering approach, ensuring the digital twin is well-structured, maintainable, and extensible.

Thesis Statement

The true efficacy of a digital twin for complex systems, such as an autonomous vehicle, is realized not through isolated high-fidelity physical simulation or abstract architectural modeling, but through their synergistic integration. The development of a detailed physics model in Simscape Multibody provides the "how"—the accurate prediction of dynamic behavior. The creation of an architectural model in System Composer provides the "what"—the logical structure, interfaces, and parameters of the system. This project demonstrates a novel and powerful workflow that establishes a "digital thread" between these two domains. By seamlessly linking the Simscape Multibody dynamics model to the System Composer architectural framework, a unified environment is created for holistic design, integrated analysis, and centralized parameter management. This approach effectively bridges the gap between high-level MBSE concepts and detailed Model-Based Design (MBD) implementation, creating a robust, executable, and fully traceable digital twin that serves as a superior platform for the validation of automated driving technologies. The lasting value of this work lies not only in the final simulation model but in the establishment of this integrated and reusable workflow, which provides a clear and scalable methodology for future digital engineering efforts at CAST and beyond.

II. Constructing the High-Fidelity Multibody Vehicle Model in Simscape

2.1. Data Acquisition: From Physical Asset to Digital Components

The creation of a high-fidelity digital twin begins with the accurate acquisition of data from the physical asset. The project's initial strategy was to employ 3D scanning, a common and efficient method for reverse engineering and capturing the complex geometries of existing components. This technique is often used to quickly generate mesh data for CAD reconstruction or for direct use in simulation environments. However, the team encountered a significant and illustrative technical hurdle rooted in the physical scale of the target vehicle. The small size and intricate

details of the 1:14 scale suspension and steering components proved challenging for the 3D scanning equipment, resulting in data of insufficient quality and resolution for creating an accurate simulation model. This experience highlights a critical consideration in multibody simulation: the quality of the input geometry is paramount, and for small, complex parts, direct scanning may not always be the optimal solution.

In response to this challenge, the project team demonstrated engineering agility by pivoting to a more fundamental, albeit labor-intensive, methodology: manual CAD modeling. This involved the complete disassembly of the vehicle's key mechanical systems. Each individual component—including control arms, steering knuckles, tie rods, and chassis mounting points—was meticulously measured using precision instruments. These measurements were then used to create detailed, feature-based solid models in CAD software. While this approach requires more upfront effort, it yields a significant advantage for simulation. Unlike a "dumb" mesh from a 3D scanner, a feature-based CAD model contains inherent design intent. Critical geometric features such as planes, axes, and cylindrical surfaces that define pivot points and motion constraints are explicitly defined. This "simulation-aware" geometry is far more robust for import into a multibody dynamics environment, as it provides a clean and unambiguous basis for defining joints and constraints, ultimately leading to a more accurate and stable physical model. The table below compares the two data acquisition methodologies in the context of this project.

Methodology	Description	Advantages	Disadvantages	Project Applicability
3D Scanning	Uses optical or laser scanners to capture the surface geometry of a physical object, generating a point cloud or polygon mesh.	- Rapid data acquisition for complex shapes. Captures as-built geometry, including manufacturing imperfections.	- Difficulty capturing small, shiny, or intricate features. The resulting mesh may require significant cleanup. Does not inherently define functional features like axes or planes.	**Rejected:** The small scale of the RC vehicle parts led to poor scan quality and inaccurate geometric representation, making it unsuitable for high-fidelity simulation.
Manual CAD Modeling	Involves disassembling the physical asset, measuring each component with precision tools, and recreating it as a feature-based solid model in CAD software.	- High degree of control over geometric accuracy.Explicit definition of functional features (planes, axes, holes) crucial for simulation.Results in clean, "simulation-aware" models.	- Labor-intensive and time-consuming. Relies on the accuracy of manual measurements. Models an idealized version of the part, omitting manufacturing variations.	**Adopted:** Despite the higher effort, this method provided the necessary geometric fidelity and feature definition required to build a robust and accurate Simscape Multibody model of the vehicle's mechanical systems.

Figure 2: Table of the data acquisition methodologies

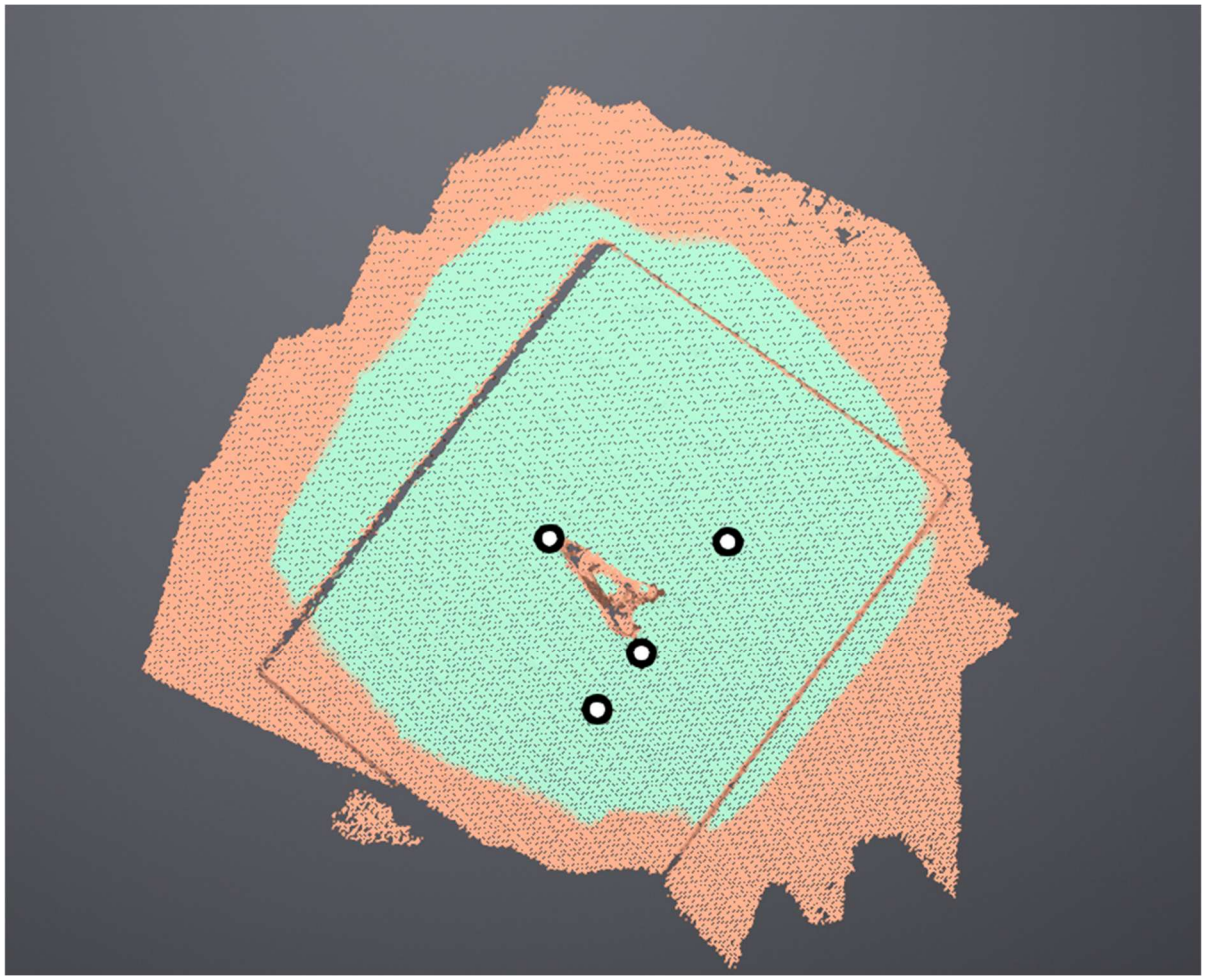

Figure 3: Scan of the lower wishbone (top view)

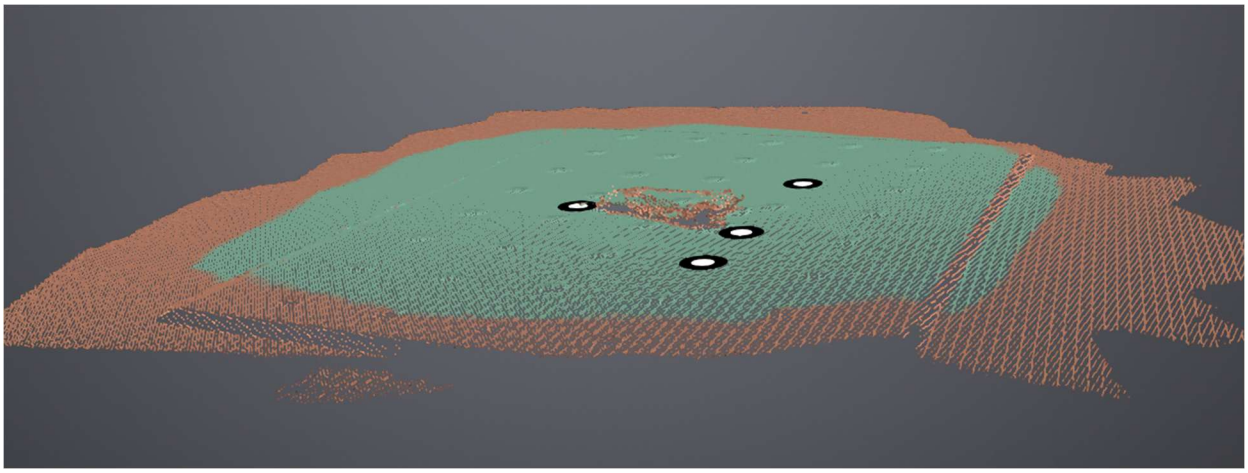

Figure 4: Scan of the lower wishbone (side view)

2.2. CAD Import and Multibody Assembly Workflow

With a complete and accurate CAD assembly of the vehicle's suspension and steering systems, the next step was to translate this geometric and inertial data into the Simscape Multibody simulation environment. MathWorks provides a streamlined workflow for this process, designed to automatically convert CAD assemblies into functional multibody models, preserving the hierarchical structure, mass properties, and kinematic relationships defined by the designer.
The process begins with exporting the CAD assembly into a format that Simscape can interpret. This is typically achieved using the Simscape Multibody Link, a plug-in for major CAD platforms like SolidWorks, Inventor, and Creo, or through dedicated functions like smexportonshape for cloud-based CAD. This export step generates two key sets of files: a master XML file that describes the multibody system (including part names, mass properties, joint types, and their locations) and a collection of geometry files (often in STEP or STL format) that are used for 3D visualization within the simulation.
Once the export is complete, the MATLAB function smimport is used to parse the XML description file and automatically construct the corresponding Simscape Multibody model. This powerful function performs several critical conversions:

- **Parts to Bodies:** Each part or subassembly in the CAD model is converted into a Simscape subsystem containing a Solid block. This block encapsulates the physical properties of the body, including its mass, moments of inertia, and a reference to the associated geometry file for visualization.
- **Mates to Joints:** The mates, constraints, and assembly relationships defined in the CAD environment are intelligently interpreted and converted into the appropriate Simscape Joint blocks (e.g., a "concentric" mate becomes a Revolute Joint, a "coincident" mate between planar faces might become part of a Prismatic Joint).
- **Hierarchy Preservation:** The hierarchical structure of the CAD assembly is maintained, with CAD subassemblies becoming Simulink subsystems, resulting in a clean, organized, and intuitive block diagram that mirrors the original design.

A crucial aspect of this automated import process is the immediate parameterization of the model. The smimport function generates a MATLAB script (a .m file) that defines all numerical values for the model—such as masses, inertias, and initial joint positions—as MATLAB variables stored in a structured array. The blocks in the generated Simscape model then reference these variables instead of hard-coded values. This foundational step is essential for creating a flexible simulation model, as it allows for easy modification, scripting of parameter sweeps, and, as will be discussed later, integration with a higher-level architectural model in System Composer.

2.3. Defining Kinematic and Dynamic Constraints

At its core, Multibody Dynamics Simulation (MBDS) is a computational method for analyzing the motion of interconnected bodies. A mechanical system is decomposed into a collection of rigid or flexible bodies, and their relative motion is governed by a set of joints and constraints that define the available degrees of freedom. The primary advantage of environments like Simscape Multibody is that they abstract the complex underlying mathematics, typically a system of differential algebraic equations allowing engineers to build and analyze complex mechanisms graphically, without needing to derive the equations of motion manually.
Following the automated import of the CAD data, we refined the generated model to accurately capture the vehicle's dynamic behavior. This involved verifying and configuring the key elements of the multibody system:

- **Bodies:** The Solid blocks, representing components like the chassis, upper and lower control arms, steering knuckles, and wheel hubs, were verified to ensure their mass and inertia properties were correctly calculated from the CAD geometry and material definitions.
- **Joints:** The automatically generated Joint blocks were carefully inspected to confirm they accurately represented the real-world kinematics. For the vehicle's double-wishbone suspension, this involved a combination of Revolute joints for the control arm pivots and Spherical or Universal joints to connect the arms to the steering knuckle, allowing for the complex spatial motion required for both suspension travel and steering input. The steering rack was modeled with a Prismatic joint to represent its translational motion, which is then transferred to the wheels via the tie rods.
- **Forces and Torques:** To move beyond pure kinematics and into dynamics, force elements were added. The coilover shock absorbers were modeled using the Spring-Damper Force block, which applies a linear force based on the relative displacement and velocity between its two connection points. Motion inputs, representing the driver's steering commands, were applied to the prismatic joint of the steering rack to actuate the system and simulate turning maneuvers.
- **Contact Modeling:** A complete vehicle dynamics model must account for the interaction between the tires and the road surface, as this is where all control forces are generated. While the primary focus of this project was the mechanical systems, the framework allows for the future inclusion of sophisticated tire models. Simscape provides tools like the Contact Forces Library, which can model the forces generated during contact between different geometries, such as the sphere-to-plane interaction representing a simplified tire on a flat road. This capability is essential for simulating realistic vehicle responses during acceleration, braking, and cornering maneuvers.

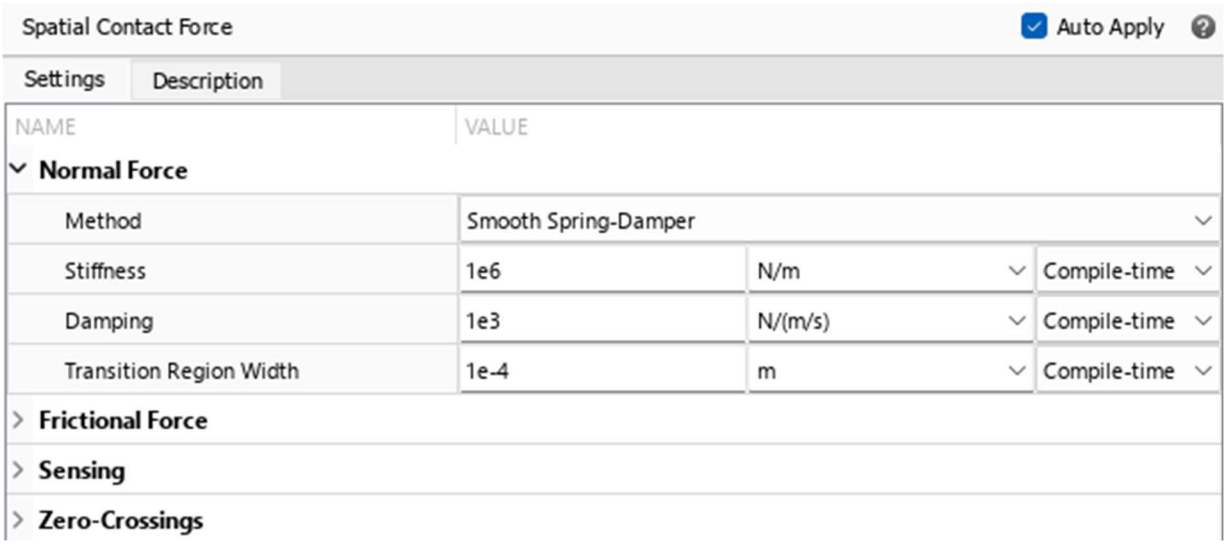

Figure 5: Scan of the lower wishbone (side view)

The assembled Simscape model, visualized in the Mechanics Explorer, provides an animated, three-dimensional representation of the vehicle's motion, allowing for intuitive understanding and verification of the system's kinematic and dynamic behavior.

2.4. Establishing the Simulation Environment: World Frame and Configuration Blocks

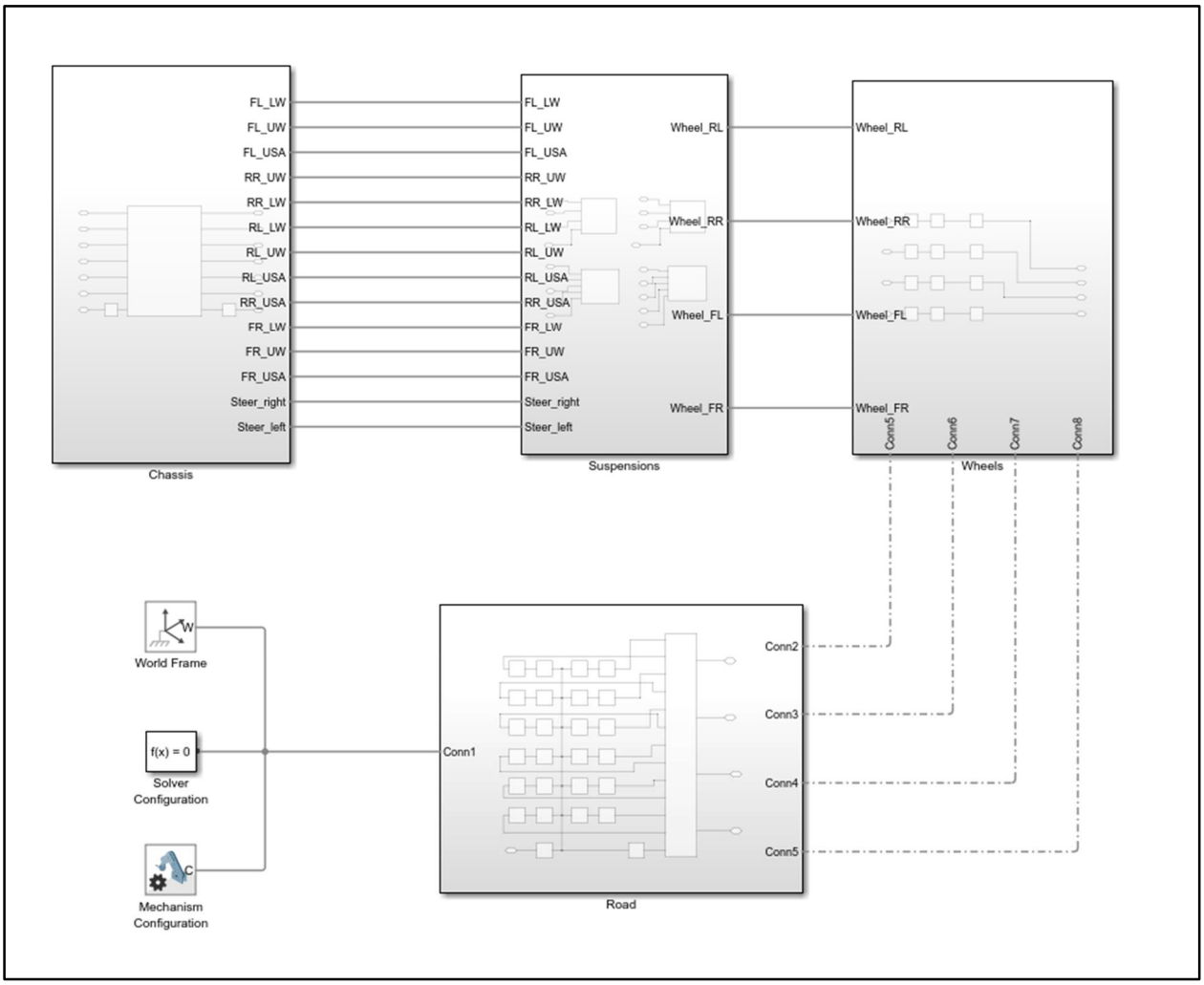

Figure 6: Top level of the multibody model

Before simulating the dynamics of individual subsystems, it is essential to establish the foundational environment in which the model operates. In Simscape Multibody, this is achieved through three critical blocks present at the top level of any model as shown in Figure 6, the World Frame, the Solver Configuration, and the Mechanism Configuration block. These blocks do not represent physical components but define the global reference, the numerical solver behavior, and the physical laws governing the entire system.

- **World Frame:** The World Frame block is the anchor of the entire multibody system. It represents the global, inertial reference frame, a fixed, non-accelerating coordinate system against which the motion of all other bodies is ultimately measured. Every component in the model, whether directly or through a chain of other components, is connected to this frame. For the vehicle model, the Chassis and the Road subsystems are connected to the World Frame, establishing their position and orientation in the simulation universe. Without this block, the model would be "floating" in an undefined space, and its motion would be indeterminate.
- **Solver Configuration:** Every distinct physical network in Simscape requires a Solver

Configuration block. This block is crucial as it provides the settings for the numerical solver that will compute the system's behavior over time. Multibody dynamics are described by a complex set of differential-algebraic equations (DAEs). The Solver Configuration block allows the user to select the appropriate algorithm (e.g., ode23t, ode15s), set tolerances, and choose between variable-step or fixed-step solvers to balance simulation speed and accuracy. A proper solver configuration is vital for achieving a stable and accurate simulation, especially for stiff systems like those involving tire models or hard stops. For a complex, integrated model with multiple physical networks, having a single, correctly configured solver is essential for performance.

- **Mechanism Configuration:** The Mechanism Configuration block is specific to the multibody domain and defines the global mechanical environment for the connected mechanism. Its most critical function is to specify the direction and magnitude of gravity. By setting the gravity vector (e.g. [0 -9.80665 0] m/s^2), a uniform gravitational field is applied to all bodies in the model that have mass, enabling the realistic simulation of effects like suspension sag and load transfer during maneuvers. This block also contains advanced settings for linearization and for handling complex joint state transitions, which are important for ensuring robust simulation under all conditions.

Together, these three blocks form the bedrock of the simulation. They establish a stable reference frame, define the numerical methods for solving the equations of motion, and apply fundamental physical laws like gravity, ensuring that the subsequent simulation of the vehicle's complex subsystems is both physically meaningful and numerically sound.

2.5. Deconstructing the Model: An Example of the Front Left Suspension Subsystem

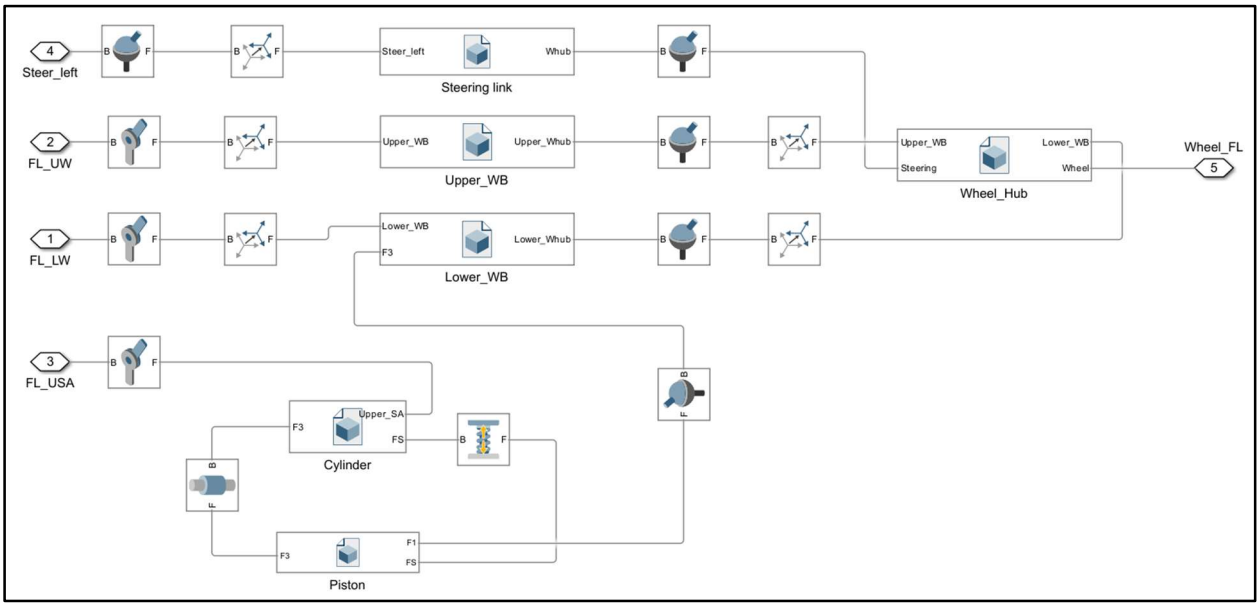

Figure 7: Subsystem of front left suspension in the multibody model

To illustrate the practical application of the modeling principles discussed, it is instructive to deconstruct a key subsystem, here the front left suspension shown in Figure 7. This assembly, a common double-wishbone configuration, is an excellent example of how fundamental Simscape Multibody blocks are combined to represent a complex mechanical system.

The provided diagram shows the interconnected blocks that form this subsystem. The core of the model consists of several kinematic chains representing the physical linkages:

- **Body Representation (Solid Blocks):** Each physical component is represented by a Solid block, which encapsulates its geometry, mass, and inertia properties derived from the CAD models. In the diagram, blocks such as Upper_WB (Upper Wishbone), Lower_WB (Lower Wishbone),Steering link, and Wheel hub are all instances of Solid blocks.

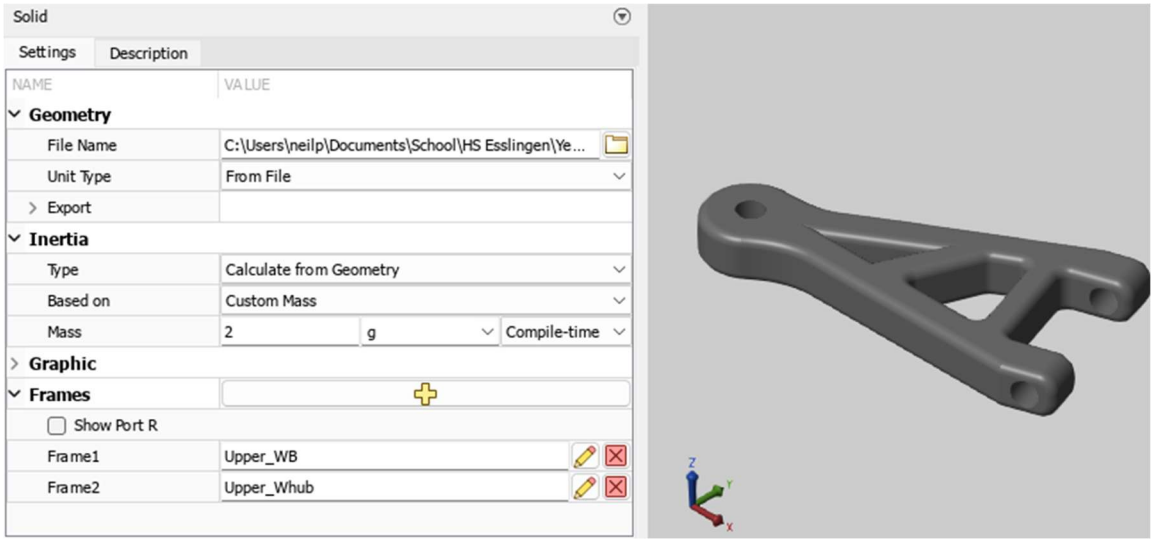

Figure 8: Upper wishbone of the front left suspension model

- **Kinematic Connections (Joint Blocks):** The relative motion between these bodies is governed by Joint blocks. The pivots connecting the inboard side of the upper and lower wishbones to the chassis (represented by inputs FL_UW and FL_LW) are modeled using Revolute Joint blocks, which permit rotation about a single axis. The outboard connections of the wishbones and the steering link to the wheel hub are modeled with Spherical Joint blocks, which act as ball joints, allowing three rotational degrees of freedom. This combination of joints accurately replicates the spatial motion required for both vertical suspension travel and steering articulation.
- **Precise Assembly (Rigid Transform Blocks):** A critical element for ensuring geometric accuracy is the Rigid Transform block. While a Solid block has a central reference frame, the actual connection points for joints (the suspension's "hardpoints") are located at specific coordinates on the physical part. The Rigid Transform blocks define these precise 3D offsets, specifying both the position and orientation of a joint's connection frame relative to the body's central frame. This ensures the virtual assembly's kinematics perfectly match the physical design.

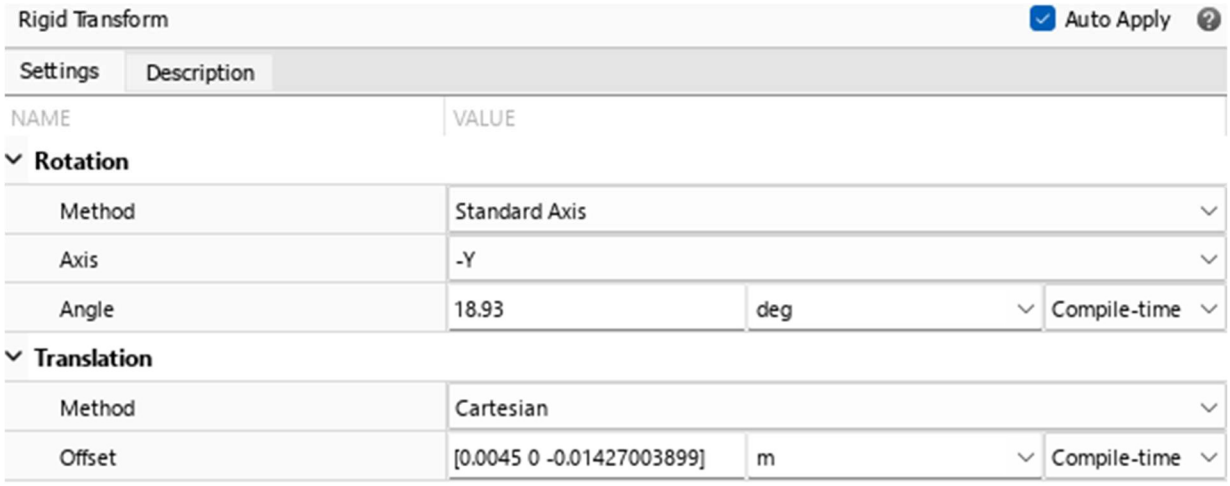

Figure 9:Mounting point of the front left upper wishbone on the chassis

- **Dynamic Elements (Spring-Damper Force):** The coilover shock absorber is modeled using a Spring-Damper Force block. This block connects the piston and cylinder, which connects to the lower wishbone and chassis respectively. It applies a force based on the relative displacement of the spring component, spring stiffness and damping coefficient between its two connection points, thereby simulating the suspension's ability to absorb bumps and control chassis motion.

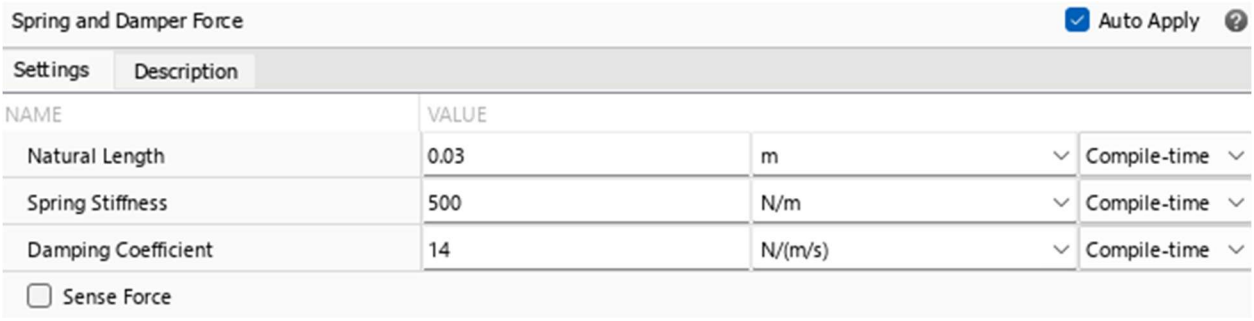

Figure 10: Front left spring and damper block

By combining these fundamental blocks, a high-fidelity, physics-based representation of the suspension mechanism is created. This modular subsystem serves as a reusable and analyzable building block within the complete digital twin.

III. Architecting the Vehicle System with System Composer

3.1. Applying the Model-Based Systems Engineering (MBSE) Paradigm

The development of the high-fidelity physics model in Simscape addresses the detailed implementation of the vehicle's mechanical systems. However, to manage the complexity of the overall vehicle including its future electrical systems, sensors, and control software a higher-level, architectural perspective is required. This is the domain of Model-Based Systems Engineering (MBSE), a methodology that uses formal models as the primary artifacts throughout the system lifecycle, replacing traditional, disparate, and often inconsistent text-based documents. This model-centric approach provides a "single source of truth," fostering clear communication among multidisciplinary teams, enabling early analysis and trade studies, and establishing robust traceability from requirements to design and verification.
MathWorks System Composer is a dedicated MBSE tool that is natively integrated within the MATLAB and Simulink environment. This tight integration is a significant advantage, as it creates a seamless pathway from high-level system architecture to detailed, executable models in Simulink and Simscape. This capability directly addresses a common challenge in systems engineering where architectural models, often created in standalone SysML tools, become disconnected from the actual implementation models, leading to a divergence between design intent and final behavior. By employing System Composer, the project team adopted a modern workflow that keeps the system architecture "live" and synchronized with the underlying physics, a practice increasingly advocated in industry case studies for complex systems in aerospace and automotive sectors.

3.2. Developing the Hierarchical Component Architecture (The "Bill of Models")

The foundational activity in System Composer is the creation of a logical architecture that describes the system's structure. The primary canvas for this activity is the "composition view," which is conceptually analogous to a Bill of Materials (BOM) or, more accurately for this context, a "Bill of Models". This view provides a hierarchical decomposition of the system into its constituent parts, defining both the components themselves and the interfaces that connect them.
The team constructed the vehicle's architecture following a logical, hierarchical approach:

- **Components:** Each logical or physical subsystem of the vehicle was represented as a Component block. At the highest level, this included components such as Chassis, Front suspension assembly, Rear suspension assembly, and Steering system.
- **Decomposition:** To manage complexity, high-level components were broken down into their constituent sub-components. For instance, the front suspension assembly component was decomposed to contain left wheel assembly and right wheel assembly components. This hierarchical structure allows engineers to view the system at various levels of abstraction, from a top-level overview down to detailed subsystems.
- **Ports and Connectors:** The interactions between components were formalized using Ports and Connectors. Ports define the interfaces of the doorways through which information, energy, or physical connections pass while connectors represent the pathways that link these ports. This creates an explicit and unambiguous representation of the system's topology and data flow, a significant improvement over informal block diagrams.

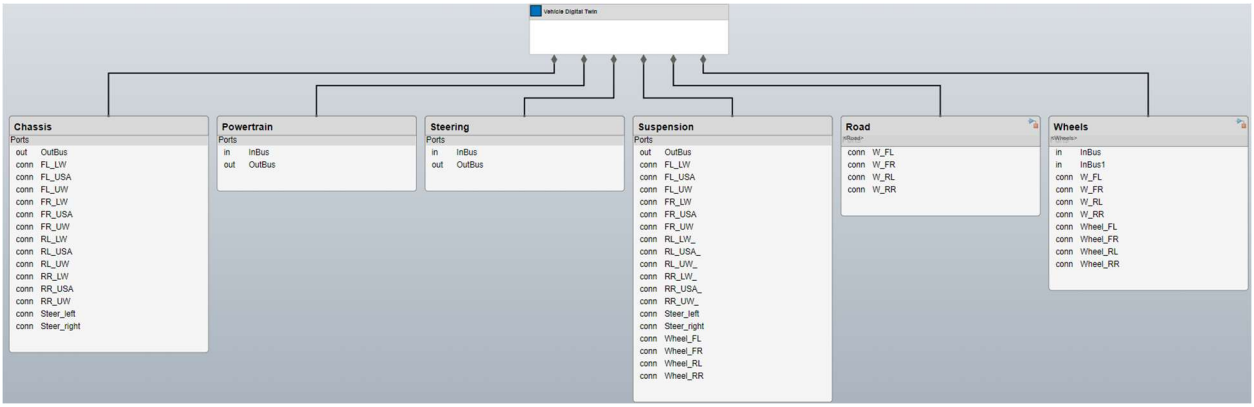

Figure 11: Component Hierarchy

A key capability leveraged in this process was the parameterization of the architecture itself. Using features like Stereotypes and the Parameter Editor, domain-specific properties can be attached directly to the architectural components. For example, a stereotype named MechanicalComponent could be created with properties like Mass (in kg), Material, and Cost (in EUR). By applying this stereotype to components in the architecture, the model becomes a rich repository of system data that can be programmatically queried and analyzed, forming the basis for sophisticated trade studies and automated report generation.

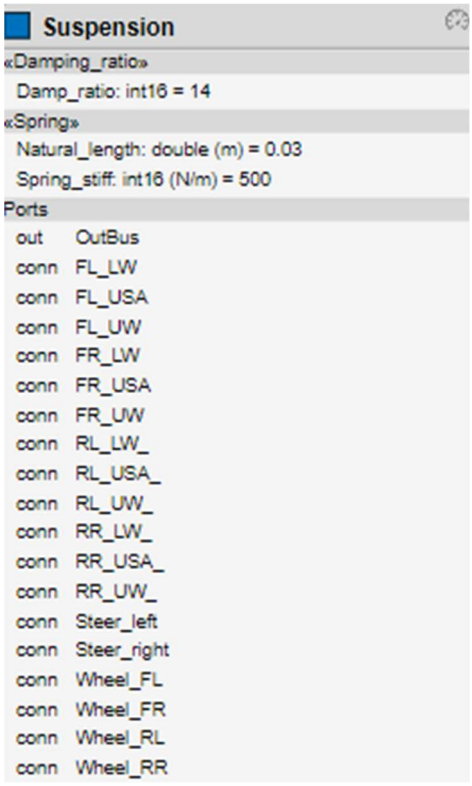

Figure 12: Front left spring and damper block

3.3. Mapping System Relationships with the Allocation Matrix

In a comprehensive MBSE workflow, a system is often described through multiple architectural viewpoints. For example, a functional architecture describes *what* the system does, a logical architecture describes *how* it is organized into functional blocks, and a physical architecture describes the actual hardware implementation. Allocation is the formal process of creating directed, traceable links between the elements of these different architectural models.
For this project, the team created an "Allocation Matrix" to formally document the connections and relationships between the different physical parts of the vehicle. This was accomplished using the Allocation Editor in System Composer, a powerful tool that provides a matrix-based interface for defining and visualizing these relationships. In this matrix, the components of a source model are listed along the rows, and the components of a target model are listed along the columns. An allocation is created simply by marking the intersection between a source and a target element.
This matrix serves as a powerful tool for system analysis and documentation. In a complex assembly, the dependencies between parts are not always obvious. The allocation matrix makes these relationships explicit. For example, it can be used to formally capture which specific components of the steering system are physically mounted to the chassis, or which elements of the suspension are kinematically linked to the wheel assembly. This formalization of "tribal knowledge"—the implicit understanding that experienced engineers have about system dependencies—is invaluable. It provides a clear, queryable map for performing impact analysis. An engineer can ask, "If I modify the design of the steering knuckle, what other components in the system are directly affected?" The allocation matrix provides an immediate and unambiguous answer, making the system easier to understand, maintain, and evolve, which directly supports the project's goal of creating clear documentation for future teams.

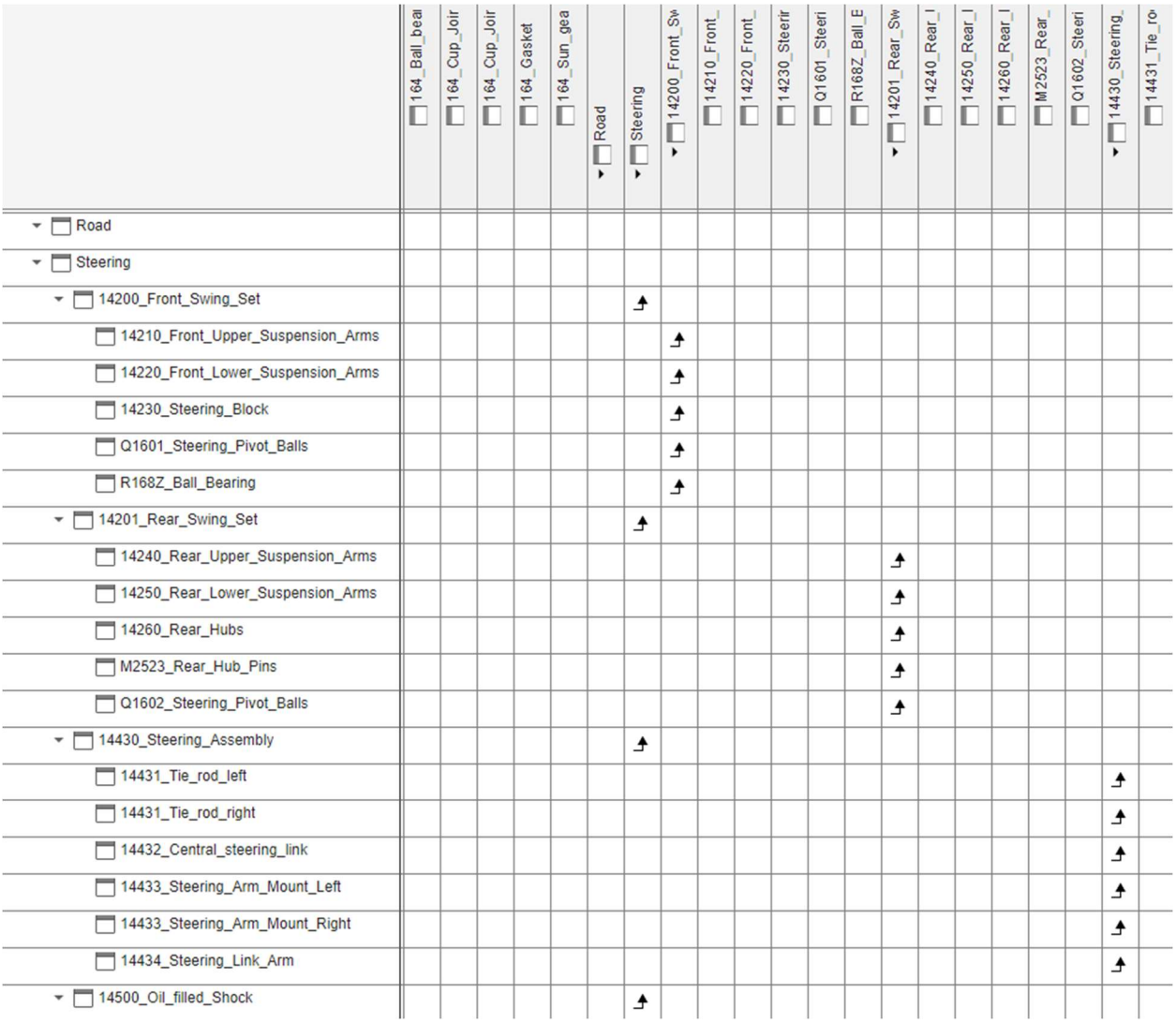

Figure 13: Allocation Matrix

3.4. Programming Allocation Using MATLAB

In Model-Based Systems Engineering (MBSE), the allocation matrix serves as a critical tool for mapping elements between different architectural layers—typically linking functional components to their physical or behavioral counterparts. MATLAB, through System Composer, allows the creation of these allocation mappings programmatically, enabling seamless integration between architectural models and simulation platforms such as Simulink and Simscape.

This programmatic approach enhances repeatability, automation, and traceability, especially useful in iterative development environments like ADAS validation, where system components are frequently refined.

Using **MATLAB scripting and the System Composer API**, we programmatically created the allocation matrix to define **connections between assemblies and subassemblies**, as well as **relationships between different vehicle assemblies**. This was implemented using a **bottom-up approach**, where we first allocated individual subassemblies to their corresponding physical

components in the simulation model. After establishing these base-level mappings, we extended the allocation to link subassemblies with higher-level assemblies in the architecture model. This approach, executed entirely through code, ensured precision, repeatability, and a clear hierarchical trace from physical parts to logical architecture, thereby reinforcing modularity and traceability within the MBSE workflow.

```
clc
clear all
% open_system('BOM_Test')
allocSet = systemcomposer.allocation.createAllocationSet('ComponentAllocation', 'BOM_Test_Func','BOM_Test');
systemcomposer.allocation.editor
```

Creation of Functional and Logical Architecture

```
functionalArch = systemcomposer.loadModel('BOM_Test_Func');
logicalArch = systemcomposer.loadModel('BOM_Test');
```

Allocation for Chassis subcomponents

```
FrontBumper = functionalArch.lookup('Path','BOM_Test_Func/Chassis/14100_Front_Bumper_Assembly');
chassis = logicalArch.lookup('Path','BOM_Test/Chassis');
defaultScenario = allocSet.getScenario('Scenario 1');
defaultScenario.allocate(FrontBumper,chassis);
RearBumper = functionalArch.lookup('Path','BOM_Test_Func/Chassis/14110_Rear_Bumper_Assembly');
defaultScenario.allocate(RearBumper, chassis)
Frame = functionalArch.lookup('Path','BOM_Test_Func/Chassis/Frame');
defaultScenario.allocate(Frame, chassis)
```

Allocation for Suspension subcomponents

```
suspension = logicalArch.lookup('Path','BOM_Test/Suspension');
suspension_func = functionalArch.lookup('Path','BOM_Test_Func/Suspension');
```

Allocation for Suspension subcomponents

```
suspension = logicalArch.lookup('Path','BOM_Test/Suspension');
suspension_func = functionalArch.lookup('Path','BOM_Test_Func/Suspension');
suspension_child =  suspension_func.Architecture.Components();
for i = 1:length(suspension_child)
    defaultScenario.allocate(suspension_child(i),suspension)
end
```

Allocation for Wheel Hub subcomponents

```
wheelHub = logicalArch.lookup('Path','BOM_Test/Wheel_Hubs');
wheelHub_func = functionalArch.lookup('Path','BOM_Test_Func/Wheel_Hubs');
wheelHub_child = wheelHub_func.Architecture.Components();
for i = 1:length(wheelHub_child)
    defaultScenario.allocate(wheelHub_child(i),wheelHub)
end
```

Allocation for Wheel subcomponents

```
wheels = logicalArch.lookup('Path','BOM_Test/Wheels');
wheels_func = functionalArch.lookup('Path','BOM_Test_Func/Wheels');
wheels_child = wheels_func.Architecture.Components();
for i = 1:length(wheels_child)
    defaultScenario.allocate(wheels_child(i),wheels)
end
```

Figure 14: Allocation of subcomponents

Allocation for Steering subcomponents

```
% Servo Assembly
steering = logicalArch.lookup('Path','BOM_Test/Steering');
servo = logicalArch.lookup('Path','BOM_Test/Steering/14700_Servo_Assembly');
servo_func = functionalArch.lookup('Path','BOM_Test_Func/Steering/14700_Servo_Assembly');
servo_child = servo_func.Architecture.Components();
for i = 1:length(servo_child)
    defaultScenario.allocate(servo_child(i),servo)
end
defaultScenario.allocate(servo_func,steering);

% Oil Filled Shock Assembly
oilShock = logicalArch.lookup('Path','BOM_Test/Steering/14500_Oil_filled_Shock');
oilShock_func = functionalArch.lookup('Path','BOM_Test_Func/Steering/14500_Oil_filled_Shock');
oilShock_child = oilShock_func.Architecture.Components();
for i = 1:length(oilShock_child)
    defaultScenario.allocate(oilShock_child(i),oilShock)
end
defaultScenario.allocate(oilShock_func,steering);

% Steering Assembly
steer_asmbly = logicalArch.lookup('Path','BOM_Test/Steering/14430_Steering_Assembly');
steer_asmbly_func = functionalArch.lookup('Path','BOM_Test_Func/Steering/14430_Steering_Assembly');
steer_asmbly_child = steer_asmbly_func.Architecture.Components();
for i = 1:length(steer_asmbly_child)
    defaultScenario.allocate(steer_asmbly_child(i),steer_asmbly)
end
defaultScenario.allocate(steer_asmbly_func,steering);

% Rear Swing Set
rearSwingSet = logicalArch.lookup('Path','BOM_Test/Steering/14201_Rear_Swing_Set');
rearSwingSet_func = functionalArch.lookup('Path','BOM_Test_Func/Steering/14201_Rear_Swing_Set');
rearSwingSet_child = rearSwingSet_func.Architecture.Components();
for i = 1:length(rearSwingSet_child)
    defaultScenario.allocate(rearSwingSet_child(i),rearSwingSet)
end
defaultScenario.allocate(rearSwingSet_func,steering);

% front Swing Set
frontSwingSet = logicalArch.lookup('Path','BOM_Test/Steering/14200_Front_Swing_Set');
frontSwingSet_func = functionalArch.lookup('Path','BOM_Test_Func/Steering/14200_Front_Swing_Set');
frontSwingSet_child = frontSwingSet_func.Architecture.Components();
for i = 1:length(frontSwingSet_child)
    defaultScenario.allocate(frontSwingSet_child(i),frontSwingSet)
end
defaultScenario.allocate(frontSwingSet_func,steering);
```

Figure 15: Allocation of steering subcomponents

Connection: Wheel to Wheel Hubs

```
defaultScenario.allocate(wheels_func,wheelHub);
```

Connection: Wheel to Road

```
road= logicalArch.lookup('Path','BOM_Test/Road');
defaultScenario.allocate(wheels_func,road)
```

Connection: Wheel Hubs to Suspension

```
defaultScenario.allocate(wheelHub_func,suspension);
```

Connection: Suspension to Chassis

```
defaultScenario.allocate(suspension_func,chassis);
```

Connection: Powertrain to Wheels + Chassis

```
powertrain_func = functionalArch.lookup('Path','BOM_Test_Func/Powertrain');
% defaultScenario.allocate(powertrain_func, wheels);
defaultScenario.allocate(powertrain_func, chassis);
```

Connection: Steering to Chassis

```
steering_func = functionalArch.lookup('Path','BOM_Test_Func/Steering');
defaultScenario.allocate(steering_func, chassis);
```

Connection: World Frame

```
worldframe_func = functionalArch.lookup('Path','BOM_Test_Func/World Frame');
connection = [chassis, suspension, road];
for i = 1:length(connection)
    defaultScenario.allocate(worldframe_func, connection(i))
end
```

Figure 16: Connection between subcomponents

3.5 Limitations of Programming the Allocation Matrix

- **No automatic generation from model connections**: Allocation matrices in MATLAB do not automatically reflect the connections or relationships defined in Simulink models. The mapping must be created and updated manually, increasing the risk of inconsistencies.
- **Manual constraint validation**: Rules like one-to-one mapping, exclusivity, or capacity limits must be explicitly coded and checked, as MATLAB doesn't support constraint enforcement by default.
- **No built-in traceability**: Allocation matrices lack native support for tracking or linking design elements, functions, or system requirements.

- **No built-in traceability**: Allocation matrices lack native support for tracking or linking design elements, functions, or system requirements.

IV. The Digital Thread: Integrating Simscape and System Composer Models

4.1. Methodology for Linking Architecture to Physics

The pivotal innovation of this project lies in the creation of a "digital thread" that seamlessly connects the abstract architectural model in System Composer with the high-fidelity physical model in Simscape. This integration transforms the architecture from a static diagram into a live, executable framework, ensuring that the system-level representation remains perpetually synchronized with the component-level physics. The methodology for achieving this link is a structured workflow that leverages the native integration between the two MathWorks tools.
The implementation process follows a series of precise steps to define a component's behavior using a Simscape model:

1. **Define Physical Ports and Interfaces:** The process begins at the architecture level in System Composer. Instead of using standard data ports, a component intended to have physical behavior (e.g., a motor or a suspension assembly) is outfitted with Physical ports. These special ports are designed to represent Simscape physical modeling connector ports. To govern the connections made to these ports, Physical Interfaces are defined in the Interface Editor. These interfaces act as strongly-typed contracts, bundling one or more physical elements, each of which is assigned to a specific Simscape physical domain (e.g., foundation.electrical.electrical for an electrical connection, or foundation.mechanical.rotational.rotational for a mechanical shaft). This ensures that only physically compatible connections can be made in the architecture, preventing modeling errors at the system level.
2. **Create and Link a Simulink Behavior:** Once the architectural component and its physical interfaces are defined, a behavior is linked to it. For direct integration with Simscape, the component behavior is created as a Simulink Subsystem. This action generates a new, empty subsystem that is intrinsically linked to the parent component in System Composer. The physical ports defined on the architectural component automatically appear inside this subsystem as corresponding physical connection ports.
3. **Implement the Simscape Model:** With the interface established, the final step is to implement the detailed physics inside the linked subsystem. The engineer can now open this subsystem and build the multibody dynamics model using standard Simscape blocks. The physical ports inherited from the architecture serve as the connection points to the rest of the system. For example, the rotational mechanical port of a SteeringSystem component in the architecture would connect directly to the input shaft of the Rack and Pinion mechanism modeled within its linked Simscape behavior. This workflow creates a direct and robust bridge between the two modeling domains.

4.2. Centralised Parameter Management and Propagation

The key advantage of this integrated architecture is the establishment of a centralised framework for parameter management. The System Composer model elevates itself from a simple block diagram to become the "single source of truth" for the entire system's defining parameters. This

enables a highly efficient top-down design workflow, where system-level decisions can be made and their effects on detailed physical behaviour can be simulated immediately, without the need for manual data transfer or model modification across different tools.
This is achieved using the Parameter Editor in System Composer, which facilitates a top-down authoring process :

- **Parameter Definition:** Key physical parameters are defined and assigned to their corresponding components within the System Composer architecture. For instance, the FrontSuspensionAssembly component would have parameters such as springStiffness and dampingCoefficient.
- **Parameter Propagation:** These architectural parameters are then linked to the MATLAB workspace variables that drive the underlying Simscape model. When a parameter's value is changed in the System Composer interface—for example, increasing the springStiffness from 45 N/m to 55 N/m—that change is automatically propagated through the MATLAB workspace to the Stiffness parameter of the corresponding Spring-Damper Force block in the Simscape model.
- **Instance-Specific Values:** This mechanism also supports instance-specific parameterisation. If the architecture contains multiple instances of the same component (e.g., four instances of a WheelAssembly component), each instance can be assigned a unique value for its parameters (e.g., different tire pressures for each wheel).

This centralised control paradigm is a powerful enabler for conducting trade studies and design optimisation. An engineer can write a simple MATLAB script that iterates through a range of values for a parameter defined in System Composer, running a full vehicle dynamics simulation for each iteration and plotting the impact on key performance indicators like cornering stability or ride comfort. This automates a process that would otherwise be manual, time-consuming, and prone to error, thereby accelerating the design cycle and leading to more optimised system performance. The following table provides concrete examples of this parameter mapping.

Parameter Name (in System Composer)	Architectural Component	Simscape Block Affected	Unit	Description
springStiffness	SuspensionAssembly	Spring-Damper Force	N/m	The linear stiffness coefficient of the coilover spring.
dampingCoefficient	SuspensionAssembly	Spring-Damper Force	N/(m/s)	The linear damping coefficient of the shock absorber.
wheelMass	WheelAssembly	Solid	kg	The total mass of the wheel and tire assembly.
wheelInertia	WheelAssembly	Solid	kg*m^2	The moment of inertia of the wheel assembly about its axis of rotation.
steeringRatio	SteeringSystem	Simple Gear	dimensionless	The gear ratio of the steering rack and pinion mechanism.

Figure 17: Table of mapping of parameters

4.3. Co-simulation and Integrated Validation

The fully linked digital twin is more than just a static data model; it is an executable, multi-domain co-simulation environment. When a simulation is initiated from the top-level System Composer model, the Simulink engine orchestrates the execution. It manages the flow of signals and data between architectural components while invoking the Simscape solver to compute the time evolution of the physical states within the linked behavioural models. This creates a holistic simulation where the architectural logic and the detailed physics are solved concurrently.
This integrated framework was used to perform the critical task of model validation. The project team designed and executed a series of virtual test manoeuvres to assess the fidelity and accuracy of the digital twin against expected vehicle dynamics principles. These scenarios included:

- **Steady-State Cornering:** A constant steering input was applied to the SteeringSystem component by changing the position of the steering rods on the chassis (see Figure 17). The simulation results for vehicle path, body roll, and lateral acceleration were then analysed to verify that the model behaves realistically during turning manoeuvres.

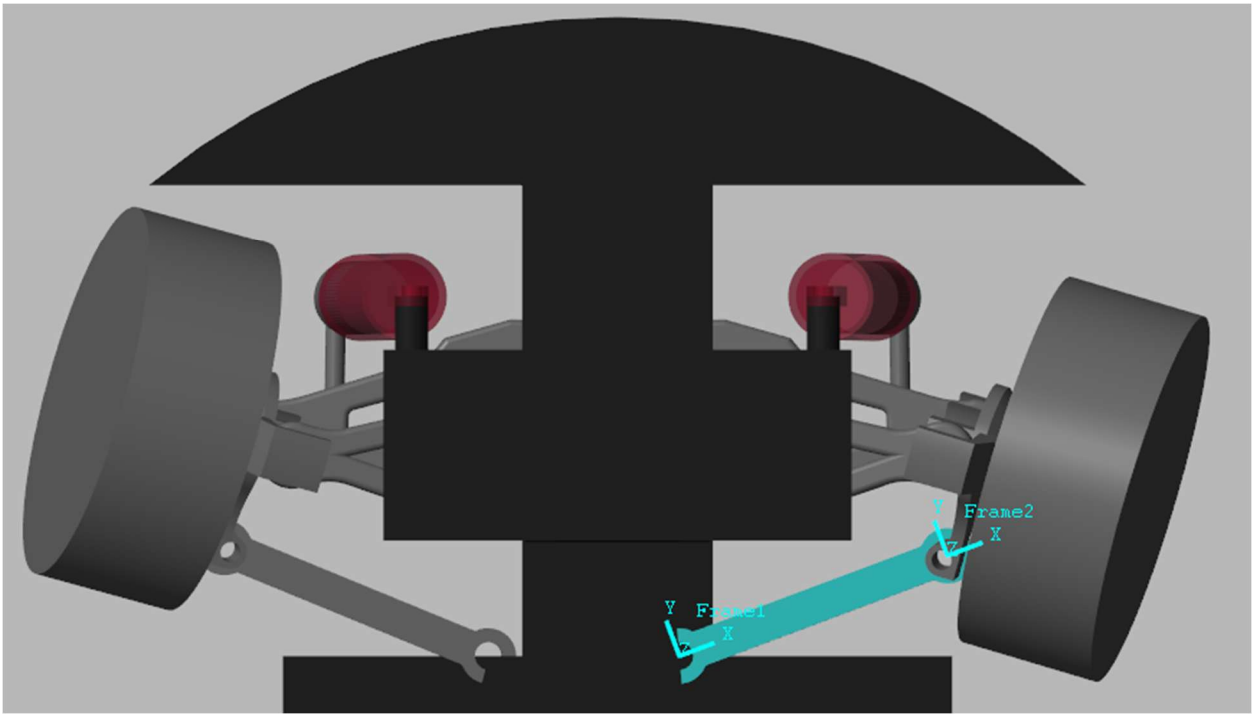

Figure 18: Steering input

- **Driving Over a Bump:** The road model was modified to include a vertical disturbance. The simulation captured the response of the suspension system, including wheel travel and chassis displacement, validating the behaviour of the spring and damper elements.

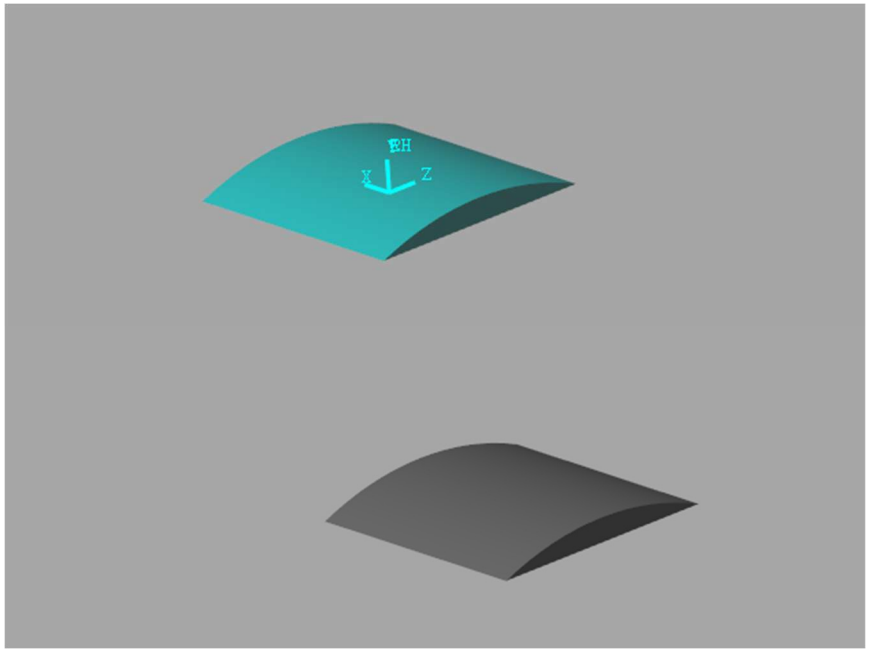

Figure 19: Speed Bump Model

- **Acceleration Manoeuver:** A rotational force was applied to the revolute joints at the wheels to simulate acceleration, allowing for the analysis of pitch dynamics and load transfer between the front and rear axles.

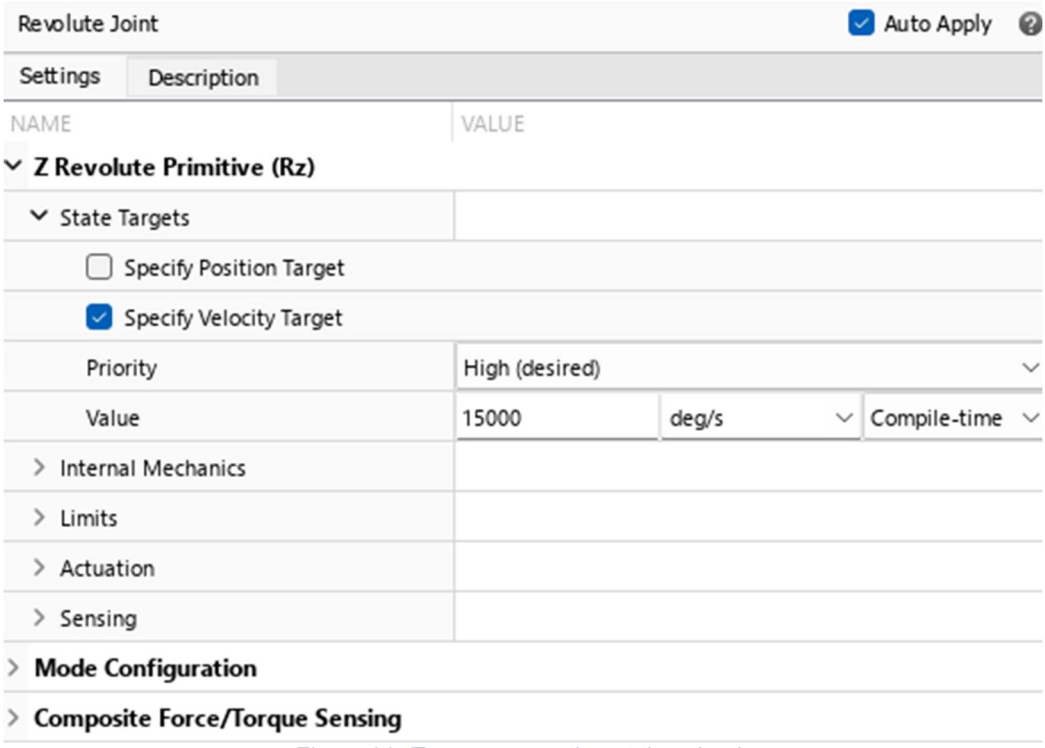

Figure 20: Torque generation at the wheels

The successful execution of these tests and the plausible results obtained (as shown in the project presentation) served a dual purpose. Firstly, they provided confidence in the fidelity of the

underlying Simscape Multibody model, confirming that the CAD-based components and kinematic constraints accurately represent the physical vehicle. Secondly, they proved the robustness and utility of the integrated System Composer and Simscape framework as a viable platform for conducting complex, system-level virtual testing. This validated model now serves as a reliable baseline for future development and analysis.

V. Analysis and Future Scope

5.1. Evaluation of the Integrated Digital Twin Framework

The successful completion of this project resulted in the development of a sophisticated digital twin that effectively meets its primary objectives. A precise, physics-based model of the 1:14 scale vehicle was created, a formal MBSE workflow was implemented, and a robust, extensible foundation for the future validation of automated driving functions was established. The evaluation of the integrated framework reveals several key benefits that align with the best practices of modern digital engineering.

The most significant achievement is the creation of a strong "digital thread" that ensures consistency and traceability between the architectural and physical domains. By linking System Composer components directly to their Simscape behaviours, the framework eliminates the possibility of the two models diverging over time—a common and costly source of error in complex, multi-tool development environments. This tight integration yields several tangible advantages:

- **Enhanced Model Robustness and Consistency:** The architecture serves as a master blueprint that governs the interfaces and parameters of the physical models. Any change at the architectural level is automatically reflected in the implementation, ensuring the system remains consistent by design.
- **Superior Documentation and Knowledge Transfer:** The System Composer model is not a static diagram; it is an interactive, executable form of documentation. It provides an unambiguous, hierarchical view of the system's structure and dependencies. This "living document" is far more effective for knowledge transfer than traditional reports or diagrams, directly addressing the project's goal of creating tutorials and resources for successor teams.
- **A Reusable and Extensible Platform:** The modular design, enforced by both the component-based architecture in System Composer and the use of subsystems in Simscape, creates a highly extensible platform. New components can be added, existing ones can be replaced with higher-fidelity versions, and different system configurations can be explored with minimal rework. This modularity confirms the model's value as a "benchmark" for future projects.
- **Streamlined Analysis and Trade Studies:** As demonstrated, the centralised, top-down parameter management workflow drastically simplifies the process of exploring the design space. The ability to programmatically sweep parameters at the architectural level and automatically simulate the system-level consequences is a powerful tool for optimisation and data-driven decision-making.

5.2. Challenges and Constraints

While the integrated approach of MBSE and Simscape Multibody proved effective for creating the vehicle's digital twin, several limitations were identified that influenced the development process and the depth of validation.

The most significant limitation encountered was the lack of robust, direct integration for executing Simscape models within the System Composer environment. To enable execution driven by the system architecture, the team found it necessary to manually copy entire Simscape models into the System Composer architectures. This workaround, while functional, introduced additional complexity and manual effort into the workflow, detracting from a fully seamless integration.

Another considerable challenge was the laborious nature of performing automatic allocations between different architectural levels using MATLAB code. The process required extensive manual coding for each individual allocation. The project team did not identify any scope or methods to automate this allocation process, making it a time-consuming and intricate task that could be prone to human error.

Lastly, the validation of the Simscape models, while guided by practical experience and observations of the vehicle's driving behaviour, was limited by the availability of concrete, data-driven validation efforts. Although some physical measurements were undertaken, the project did not have the capacity to conduct a comprehensive suite of data-based validations. This means that while the model's behaviour aligns with expectations based on engineering judgment, a more extensive empirical dataset would be beneficial for a higher degree of confidence in its predictive accuracy.

5.3. Pathways for Future Development

This project has successfully laid the foundational layer of a comprehensive vehicle digital twin. The established framework is designed for expansion, and several clear pathways exist for future development to build upon this work and fully realise its potential as an ADAS validation platform.

- **Powertrain Integration:** The most immediate and logical next step is the development and integration of the vehicle's electric powertrain model, an optional goal of the original project scope. This would involve creating new architectural components in System Composer, such as Battery, MotorController, and ElectricMotor. These components would then be linked to behavioural models developed using specialised toolboxes like Simscape Electrical and Simscape Driveline. This would allow for the simulation of full-system performance, including acceleration, energy consumption, and the interaction between the powertrain and chassis dynamics.
- **Sensor and Environment Modelling:** To be a truly effective tool for validating autonomous driving functions, the digital twin must be able to perceive a virtual world. This requires the integration of sensor models into the vehicle architecture. Using tools like the Automated Driving Toolbox, virtual representations of cameras, LiDAR, and radar can be added to the model. These virtual sensors would "see" a simulated 3D environment, which can be created and managed using interfaces to gaming engines like Unreal Engine. The output from these sensors would then serve as the input to the ADS control algorithms.
- **Control Strategy Development and X-in-the-Loop (XIL) Testing:** With a complete plant model (vehicle, powertrain, sensors) and a virtual environment, the digital twin becomes an ideal platform for Model-in-the-Loop (MIL) testing. Control algorithms for functions like lane-keeping assist or automated emergency braking can be developed in Simulink and tested against the high-fidelity vehicle model. The workflow can then be extended to X-in-the-Loop (XIL) validation. By using Embedded Coder, the control algorithms can be automatically converted to C code for Software-in-the-Loop (SIL) testing. Subsequently, this code can be deployed to a target hardware controller and tested against the real-time simulation of the vehicle model in a Hardware-in-the-Loop (HIL) setup, providing a safe, efficient, and

comprehensive method for validating the embedded system before it is ever tested on the physical RC car.

VI. Conclusion

This project successfully demonstrated an advanced, integrated workflow for the creation of a high-fidelity vehicle digital twin by synergistically combining the capabilities of Simscape Multibody and System Composer. By overcoming initial data acquisition challenges through meticulous CAD modelling, a robust physics-based model of the vehicle's mechanical systems was established. This detailed implementation was then architected within a formal Model-Based Systems Engineering (MBSE) framework, which provided a hierarchical structure, formalised interfaces, and a clear map of system dependencies through an allocation matrix.
The central achievement of this work is the establishment of a seamless digital thread between the architectural and physical modelling domains. This integration transforms the system architecture into an executable specification, enabling a powerful top-down design paradigm where system parameters are managed centrally and propagated automatically to the underlying physics simulation. The resulting co-simulation environment was proven to be a robust platform for conducting virtual validation tests, confirming both the fidelity of the model and the viability of the integrated workflow.

The digital twin developed in this project serves as a valuable and extensible asset for the Centre for Automated Driving and Service Technology (CAST). It not only provides a validated benchmark model for immediate use but also establishes a scalable and repeatable methodology for future digital engineering endeavours. The clear pathways for integrating powertrain, sensor, and control system models position this work as a critical first step towards creating a comprehensive X-in-the-Loop validation ecosystem for advanced automated driving research. Ultimately, this project validates the thesis that the convergence of MBSE and multi-domain physical simulation is not merely an academic exercise but a practical and powerful approach to mastering the complexity of modern automotive systems.

References

[1] A. Kodagoda, S. M. Elsayed and B. Sithamparanathan, “A Model Based System Engineering Methodology for an Autonomous Driving System Design,” *ResearchGate*, [Online]. Available: https://www.researchgate.net/publication/330290137

[2] “Review of Digital Twin in the Automotive Industry on Products, Processes and Systems,” *Scilight International Journal of Applied Mechanics and Materials*, [Online]. Available: https://www.sciltp.com/journals/ijamm/article/view/971

[3] M. J. Rhodes and C. Leung, “Applying Digital Engineering Digital Twin to Support Ground Vehicle Validation,” *SAE Technical Paper 2024-01-4084*, 2024. [Online]. Available: https://www.sae.org/publications/technical-papers/content/2024-01-4084/

[4] MathWorks, “Model-Based Systems Engineering (MBSE),” *MATLAB & Simulink*, [Online]. Available: https://www.mathworks.com/solutions/model-based-systems-engineering.html

[5] MathWorks, “System Composer Documentation,” *MATLAB & Simulink*, [Online]. Available: https://www.mathworks.com/help/systemcomposer/index.html

[6] MathWorks, “Bridging Model-Based Systems Engineering and Model-Based Design,” *Video Presentation*, [Online]. Available: https://www.mathworks.com/videos/bridging-model-based-systems-engineering-and-model-based-design-1634884070332.html

[7] MathWorks, “What Is Simscape Multibody?” *MATLAB & Simulink*, [Online]. Available: https://www.mathworks.com/videos/simscape-multibody-overview-117986.html

[8] MathWorks, “Importing CAD Assemblies into Simscape Multibody,” *MATLAB & Simulink Racing Lounge*, [Online]. Available: https://www.mathworks.com/videos/matlab-and-simulink-racing-lounge-importing-cad-assemblies-into-simscape-multibody-1491402752220.html

[9] MathWorks, “Model Import – Simscape Multibody,” *MATLAB & Simulink*, [Online]. Available: https://www.mathworks.com/help/sm/cad-import.html

[10] MathWorks, “Author Parameters in System Composer Using Parameter Editor,” *MATLAB & Simulink*, [Online]. Available: https://www.mathworks.com/help/systemcomposer/ug/top-down-parameters-workflow.html

[11] MathWorks, “Implement Component Behavior Using Simscape,” *System Composer – MATLAB & Simulink*, [Online]. Available: https://www.mathworks.com/help/systemcomposer/ug/describe-component-behavior-using-simscape.html

[12] A. Bertolotto, “Multibody Dynamic Simulation with Simscape: Methods and Examples,” *Webthesis*, Politecnico di Torino, 2021. [Online]. Available: https://webthesis.biblio.polito.it/17488/1/tesi.pdf

[13] MathWorks, “Formula Student Vehicle Modeling Using Simscape Multibody,” *MATLAB & Simulink*, [Online]. Available: https://www.mathworks.com/videos/formula-student-vehicle-modeling-using-simscape-multibody-1683608443602.html

[14] MathWorks, “Virtual Vehicle: Transforming Vehicle Engineering Through Simulation,” *MATLAB & Simulink*, [Online]. Available: https://www.mathworks.com/videos/virtual-vehicle-transforming-vehicle-engineering-through-simulation-1736192133093.html

[15] MathWorks, “Getting Started with System Composer,” *MATLAB & Simulink*, [Online]. Available: https://www.mathworks.com/help/systemcomposer/getting-started-with-system-composer.html