ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA DOPRAVNÍ

Bc. Matěj Hanousek

FYZIKÁLNÍ MODEL POHONNÉHO ÚSTROJÍ PRO VOZIDLOVÝ SIMULÁTOR

Diplomová práce

2019

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta dopravní
děkan
Konviktská 20, 110 00 Praha 1

K616.....**Ústav dopravních prostředků**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení studenta (včetně titulů):

Bc. Matěj Hanousek

Kód studijního programu a studijní obor studenta:

N 3710 – IS – Inteligentní dopravní systémy

Název tématu (česky): **Fyzikální model pohonného ústrojí pro vozidlový simulátor**

Název tématu (anglicky): Detailed powertrain physical model for interactive vehicle simulator

Zásady pro vypracování

Při zpracování diplomové práce se řiďte osnovou uvedenou v následujících bodech:

- Průzkum programovacího jazyka Modelica a FMI (Functional Mock-up Interface) pro co-simulaci, výhody, limity, využitelnost
- Průzkum a vyhodnocení dostupných nástrojů pro podrobné modelování pohonných ústrojí s možností exportování modelu jako FMU
- Průzkum moderních fyzikálních enginů a jejich implementace v herních enginech (PhysX, Bullet, ODE...)
- Implementace modelu pohonného ústrojí do herního enginu
- Porovnání výsledků implementovaného modelu se srovnávacími daty

Rozsah grafických prací: dle pokynů vedoucích DP

Rozsah průvodní zprávy: minimálně 55 stran textu (včetně obrázků, grafů a tabulek, které jsou součástí průvodní zprávy)

Seznam odborné literatury:

1. Fritzson,Peter - Principles of Object Oriented Modeling and Simulation with Modelica 3.3

2. Stone, Richard K. Ball, Jeffrey - Automotive Engineering Fundamentals

3. Andersson, Christian - Methods and Tools for Co-Simulationof Dynamic Systems

Vedoucí diplomové práce: **Ing. Dmitry Rozhdestvenskiy**
doc. Ing. Petr Bouchner, Ph.D.

Datum zadání diplomové práce: **28. července 2018**
(datum prvního zadání této práce, které musí být nejpozději 10 měsíců před datem prvního předpokládaného odevzdání této práce vyplývajícího ze standardní doby studia)

Datum odevzdání diplomové práce: **28. května 2019**
a) datum prvního předpokládaného odevzdání práce vyplývající ze standardní doby studia a z doporučeného časového plánu studia
b) v případě odkladu odevzdání práce následující datum odevzdání práce vyplývající z doporučeného časového plánu studia

L. S.

doc. Ing. Petr Bouchner, Ph.D.
vedoucí
Ústavu dopravních prostředků

doc. Ing. Pavel Hrubeš, Ph.D.
děkan fakulty

Potvrzuji převzetí zadání diplomové práce.

Hanousek

Bc. Matěj Hanousek
jméno a podpis studenta

V Praze dne.....28. července 2018

Poděkování

Na tomto místě bych rád poděkoval všem, kteří mi poskytli podklady pro vypracování této práce. Zvláště pak děkuji panu inženýru Rozhdestvenskiy a docentu Bouchnerovi za odborné vedení a konzultování diplomové práce a za rady, které mi poskytovali po celou dobu mého studia. V neposlední řadě je mou milou povinností poděkovat svým rodičům a blízkým za morální a materiální podporu, které se mi dostávalo po celou dobu studia.

Prohlášení

Předkládám tímto k posouzení a obhajobě diplomovou práci, zpracovanou na závěr magisterského studia na ČVUT v Praze Fakultě dopravní.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorských a o změně některých zákonů (autorský zákon).

V Praze dne 20.května 2019 Bc. Matěj Hanousek

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta dopravní

FYZIKÁLNÍ MODEL POHONNÉ JEDNOTKY PRO VOZIDLOVÝ SIMULÁTOR

Diplomová práce

Květen 2019

Matěj Hanousek

ABSTRAKT

Předmětem diplomové práce „Fyzikální model pohonné jednotky pro vozidlový simulátor“ je prozkoumat programovací jazyk Modelica, zjistit jeho výhody, nevýhody, limity a využitelnost. Následně prozkoumat dostupné nástroje pro podrobné modelování pohonných ústrojí s možností exportování modelu jako FMU. Prozkoumat moderní fyzikální enginy a jejich implementace v herních enginech (PhysX, Bullet, ODE…). Zvolit vhodné nástroje a prostředí na základě předchozího a vytvořit a implementovat model pohonného ústrojí do grafického enginu. Nakonec porovnat výsledky implementovaného modelu s validovanými jízdními daty.

ABSTRACT

The subject of diploma work „Detailed powertrain physical model for interactive vehicle simulator“ is research of programming language Modelica, it’s advantages, disadvantages, limits and usage. Research of available tools for detailed modeling of powertrain with possibility to export the model as FMU. Research of modern physical engines and it’s implementations in game engines (PhysX, Bullet, ODE…). Choose of suitable tools and environment and creation and implementation of the powertrain model into the game engine. Then comparison of the results of the implemented model with validated driving data.

KLÍČOVÁ SLOVA

grafický engine, fyzikální engine, Modelica, functional mock-up interface, functional mock-up unit, dopravní simulátor, simulace, co-simulace, Unreal engine, Unity, Ignite, jízdní cyklus

KEYWORDS

graphical engine, game engine, physical engine, traffic simulator, simulation, co-simulation, drive cycle

Obsah

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE 3
Fakulta dopravní 3
ABSTRACT 3
Obsah 5
1. SEZNAM POUŽITÝCH ZKRATEK: 8
2 ÚVOD 9
3 Dopravní simulátory 10
3.1 Typy vozidlových simulátorů 11
3.3 Experimenty na simulátorech 12
4 Programovací jazyk Modelica 13
4.1 Functional mock-up interface 13
4.1.1 Model Exchange, Co-Simulace 14
4.1.2 Výhody 14
4.1.3 Nevýhody 15
4.2 Využitelnost 15
4.3 Příklady využití FMI 15
5 Nástroje pro modelování v Modelice 18
5.1 Dymola 18
5.2 OpenModelica 19
5.3 MapleSim 20
5.4 Ignite 21
6 Fyzikální a grafické enginy 24
6.1 Aplikační rozhraní 24
6.1.1 DirectX 24
6.1.2 OpenGL 24
6.1.3 Rozdíly, výhody, nevýhody 25
6.2 PhysX 25
6.2.1 Využití 25

6.2.1.1 Unreal engine 26
6.2.1.2 Unity 27
6.3 Bullet 28
6.3.1 Využití 29
6.4 ODE 29
6.3.1 Využití 30
6.4 Porovnání 30
7 Implementace 31
7.1 Model pohonného ústrojí 31
7.1.1 Volba vstupů a výstupů 31
7.1.2 Problém s proměnnou cycle_driver_state 32
7.2 Struktura modelu 33
7.2.1 Motor 34
7.2.2 Převodovka 35
7.3 FMU 35
7.3.1 Co-simulace a související problémy 36
7.3.2 Optimalizace parametrů ovládání Driver komponentu 38
7.4 Implementace do UE4 39
7.4.1 FMU simulátor 39
7.4.2 Vytvoření modelu vozidla v UE4 42
7.4.3 UE4 komponent pro FMU a jeho funkce 44
7.4.4 Inicializace a deinicializace FMU 46
7.4.5 Simulace v paralelním vláknu 48
7.4.6 Celkový flow diagram 49
7.4.7 Získávání okamžité úhlové rychlosti kol 50
7.4.8 Zápis dat do externího souboru 50
8 Výsledky implementací 53
8.1 Testovací metodologie 53
8.1.1 Jízdní cykly 54

8.1.2 Parametry 55
8.2 Výsledky porovnání dat 56
8.2.1 NEDC 56
9 Závěr 61
10 Použité zdroje 63
10.1 Citované zdroje 63
10.2 Zdroje k obrázkům 64
11 Appendix 66
12 Seznam obrázků 66
13 Seznam příloh 69

1. SEZNAM POUŽITÝCH ZKRATEK:

HMI	Human – Machine interface
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
ABS	Anti Blockier Systém
ESP	Electronic stability program
LAS	Lane Assistance Systém
ME	Model Exchange
CS	Co-Simulation
FPS	Frames per Second
API	Aplication Programming Interface
OpenGL	Open Graphics Library
ODE	Open Dynamics Engine
UE4	Unreal Engine 4
DOE	Design of Experiments

2 ÚVOD

Cílem této práce je do jednoho z grafických enginů navrhnout a implementovat fyzikální model pohonného ústrojí pro přesnější a detailnější simulaci chování vozidla, než jakou fyzikální enginy poskytují samy o sobě. Výsledek bude možné využít v dopravním simulátoru pro výzkum rozhraní člověk – stroj, vývoj a testování jednotlivých asistenčních systémů a výzkum vlivu automatizace na chování řidiče.

Dopravní simulátory jsou nezbytnou součástí vývoje v téměř všech odvětvích dopravy. Jsou základem pro výcvik pilotů, mohou být i součástí výcviku strojvedoucích nebo i pomocným nástrojem pro výuku v autoškole. Proto je třeba, aby se co nejvíce blížily realitě. To už v posledním desetiletí nepředstavuje příliš velký problém. Simulátory mohou téměř identicky kopírovat skutečná vozidla, letadla nebo i vlaky a lodě. Největší posun ovšem představoval vývoj grafický a s tím související i vývoj fyzikálního chování vozidel ve virtuálním prostředí. Se stále se zlepšujícím hardwarovým vybavením je možno klást na počítače mnohem větší nároky a tím se limitně přibližovat chování skutečného světa.

V této práci je tématem fyzikální stránka simulátorů. Vytvoříme detailní fyzikální model pohonného ústrojí a představíme si nejvýznamnější grafické enginy. Porovnáme je a popíšeme si vlastnosti fyzikálních enginy.

Model pohonného ústrojí bude navržen v programovacím jazyce Modelica, který je využíván napříč mnoha softwary ve firmách po celém světě. Na jeho základě bude poté vytvořena tzv. Functional Mock-up Unit (FMU), která se implementuje do vozidla v grafickém enginu.

Výsledkem budou data z experimentu, která se porovnají s výsledky simulace v softwaru validovaném na reálných jízdních datech. Na základě tohoto porovnání se vytvoří závěry o tom, jak se výsledky liší, z jakých důvodů a jestli představují skutečné chování pohonného ústrojí vozidla.

Samotnou implementaci bude poté možno využít na jednom z vozidlových simulátorů na fakultě pro přiblížení chování virtuálních vozidel realitě a tím i zpřesnění experimentů. Hlavní výhodou této práce bude možnost do simulátoru jednoduše implementovat jakýkoli fyzikální model pohonného ústrojí. Od spalovacích motorů přes moderní hybridní vozidla, elektromobily až po pohony využívající palivové články.

3 Dopravní simulátory

Důležité je si určitým způsobem vymezit pojem simulátor jako virtuální svět napodobující realitu vizuálně a fyzikálně ve virtuálním prostředí a simulátor jako plnohodnotné začlenění virtuálního prostředí do reálného světa. V našem případě to znamená využití skutečného vozidla a jeho připojení k ovládacím prvkům hry, viz **obr. 1**. Stejně tak jako promítání na více pláten rozmístěných okolo vozidla a případně i zavedení pohyblivé plošiny pro simulaci sil působících na řidiče při řízení.

Obrázek 1 - Simulátor osobního automobilu – Dopravní VaV Centrum **[I]**

V oblasti dopravy je nepřeberné množství simulátorů, které začaly vznikat a vyvíjet se už od minulého století. Přes letadlové, drážní, lodní až k automobilovým. Přičemž v náročnosti napodobení reality jsou nejsložitější simulátory letadlové kvůli svým 6 stupňům volnosti a vysokému vlivu počasí, teplot, hustoty a tlaku vzduchu na fyzikální chování letadla. Naproti tomu v tomto ohledu nejjednodušší jsou simulátory drážní, protože drážní prostředky mají pouze dva stupně volnosti. Automobilové (4 stupně volnosti) simulátory jsou tedy středně náročné pro co nejdůvěryhodnější přiblížení reálnému světu. Zároveň jsou ale nejnáročnějšími z hlediska interakce s prostředím a dalšími účastníky dopravního provozu. Na silnicích se pohybuje vysoké množství vozidel různých druhů stejně jako lidí a vytvořit matematický model představující skutečné chování všech prvků dopravního systému na základě jejich interakcí a cílů se tím stává velmi složitou úlohou.

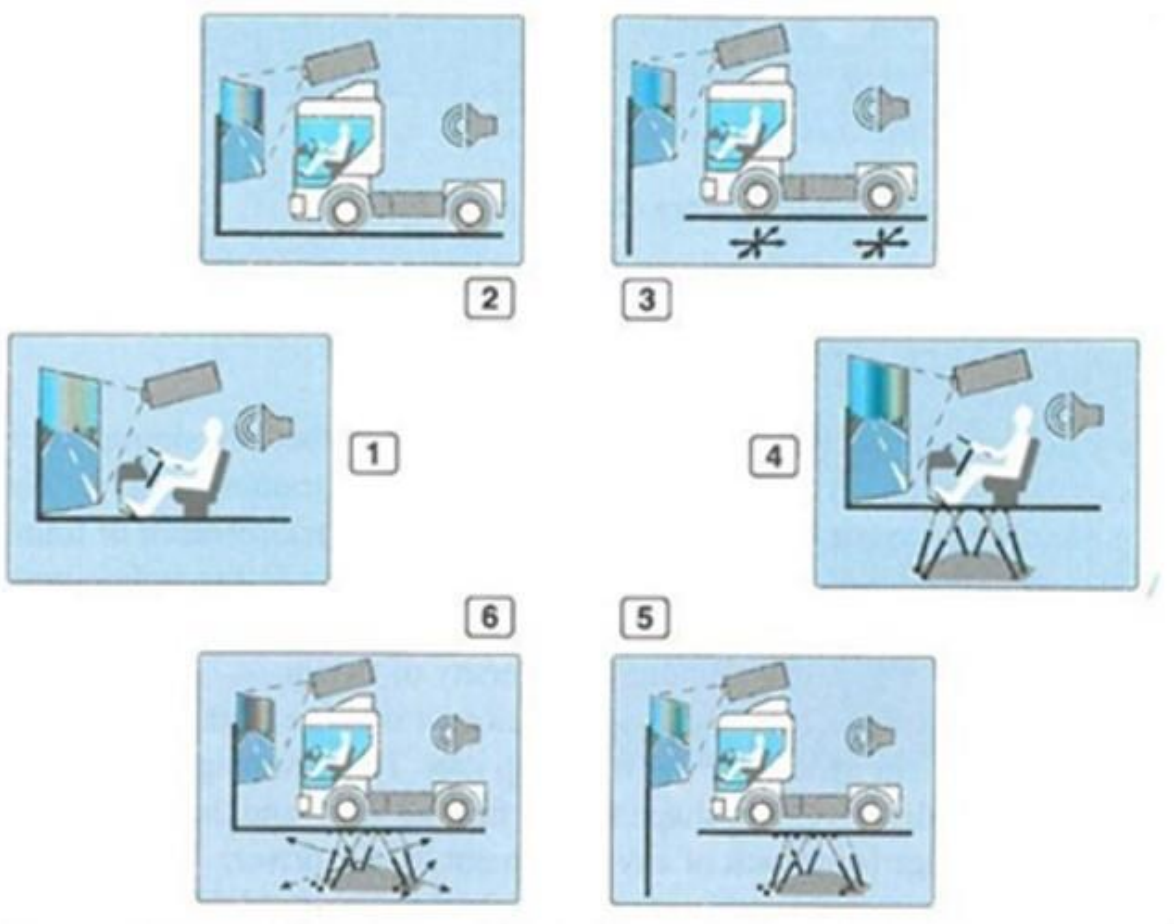
3.1 Typy vozidlových simulátorů

Vozidlové simulátory se mohou z hlediska jejich částí a složitosti dělit na lehké a plnohodnotné. U lehkých simulátorů není využit celý skeleton vozidla, ale pouze jeho základní části nezbytné pro správný průběh experimentů. Mezi tyto části patří sedadlo, volant, displej kokpitu vozidla, případně řadící páka atd., poté je to nezbytná promítací plocha, projektor, nebo více projektorů, reproduktory a zbývající nutná hardwarová a softwarová výbava.

U plnohodnotných simulátorů už řidič nastupuje do skutečného vozidla přestavěného do simulátoru, tedy do celého, nebo alespoň do většiny skeletu vozidla. U takového simulátoru mohou promítací plochy i přímo nahrazovat okna v kabině vozidla, viz **obr. 2**.

Obrázek 2 - Promítací plochy simulátoru **[II]**

U obou druhů simulátorů mohou být instalovány pohybové plošiny, které značně zvýší „důvěryhodnost“ řízení. Typy simulátorů ve shrnutí jsou na **obr. 3**, kde čísla 1 a 4 představují lehké simulátory bez plošiny a s plošinou a čísla 2,3 a 5,6 představují plnohodnotné simulátory bez plošiny a s plošinou.



Obrázek 3 - Typy simulátorů **[III]**

3.3 Experimenty na simulátorech

Zmiňovaná snaha přiblížení virtuálního prostředí reálnému není třeba jen kvůli školícím účelům např. pro piloty, strojvedoucí, nebo žáky autoškoly. Je třeba i kvůli výzkumu pro vývoj nových technologií na reálných vozidlech. V první řadě to jsou prvky Human Machine Interface (HMI), tzn. prvky, které jsou součástí komunikace mezi řidičem a vozidlem (volant, plyn, řadící páka, kontrolky displeje atd.). V dnešní době se ale stává mnohem důležitějším vývoj asistenčních systémů (ABS, ESP, LAS atd.) a možnost validace algoritmů chování autonomních a polo-autonomních vozidel. Základem výzkumů jsou data nasbíraná specializovanými institucemi z experimentů na vozidlových simulátorech.

Čím autentičtější bude pro testovaný subjekt prostředí, ve kterém během experimentů operuje, tím přesnější budou naměřená data, a i jejich pozdější interpretace. Na základě těchto interpretací mohou potom vývojová oddělení automobilek postavit svůj výzkum a pozdější systém, technologii, nebo ergonomii vozidla.

4 Programovací jazyk Modelica

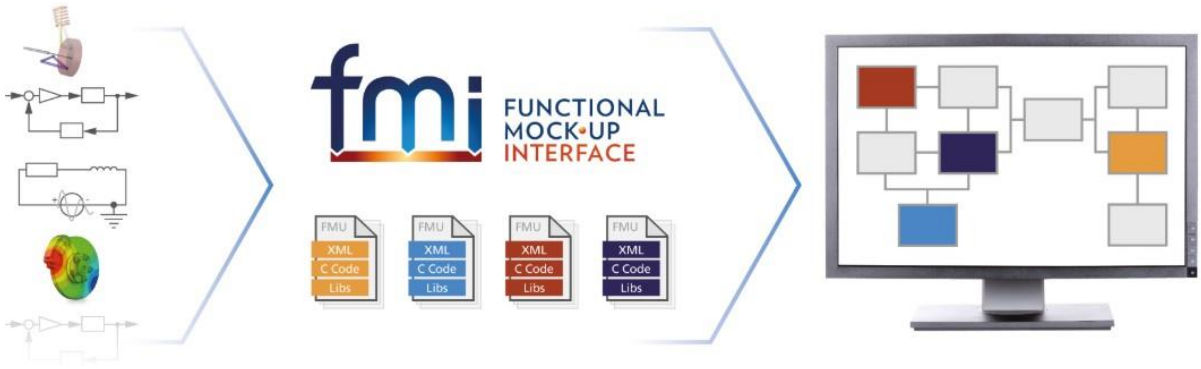
Modelica je produktem neziskové organizace The Modelica Association, jejíž tým se skládá z odborníků z Evropy, Spojených států, Kanady a Asie **[1]**. Od roku 1996 nabízí tento software jako open source, a i proto je v odvětvích, kde se využívají modelování a simulace, Modelica hojně využívána.

Je to objektově orientovaný jazyk, který využívá rovnice k účinnému modelování komplexních fyzikálních systémů např. mechanických, elektronických, hydraulických, termálních, řídicích atd. **[2]**.

Pro tuto práci jsou důležité všechny vyjmenované druhy systémů, jelikož všechny by měli být subsystémy pro vymodelované pohonné ústrojí. Modelica umožňuje sestavit model pohonného ústrojí vozidla, které se poté jako „black box“ dá využít i v softwarech třetích stran. Zmíněná černá skříňka je v přesnějším názvosloví popsána v následujícím odstavci.

4.1 Functional mock-up interface

FMI, jak už název kapitoly napovídá je zkratkou pro Functional mock-up interface. FMI je standardem pro podporu „model exchange“ a „co-simulací“ dynamických modelů. Tento standard využívá soubory formátu *.xml, ve kterých jsou popsány proměnné, parametry, vstupy, výstupy a další nezbytné informace pro daný model. A zároveň s *.xml využívá i zkompilovaný kód v programovacím jazyce C **[3]**. Ilustrace z oficiálních stránek na **obr. 4**.



Obrázek 4 - Ilustrace struktury FMI **[IV]**

Jednotka, která je na základě tohoto standardu vytvořena se nazývá FMU, tedy Functional mock-up Unit. Což je soubor ve formátu *.fmu. Tento formát lze otevřít např. pomocí softwaru 7zip. Kde uživatel nalezne zmíněné *.xml, poté všechny potřebné *.dll soubory pro model (binaries) a také pomocné soubory (resources), které jsou využívány v modelu, viz **obr. 5**.

Obrázek 5 - Struktura FMU **[V]**

FMI standard má dnes dvě verze. První, FMI 1.0, byla publikována v roce 2010 a druhá, FMI 2.0, byla publikována v roce 2014. Mezi oběma verzemi není moc velký rozdíl, nicméně FMI 2.0 má přidanou podporu pro směrové derivace a upřesňuje, některé části z předchozí verze **[4]**. Zároveň FMI 2.0, jakožto novější verze, bude v budoucnu i více využívána, proto je lepší pro účely novějších aplikací zvolit druhou verzi. Bohužel ne všechny simulační softwary umožňují práci s oběma verzemi, nicméně to se bude do budoucna také zlepšovat **[17]**.

4.1.1 Model Exchange, Co-Simulace

Již bylo zmíněno že FMI podporuje „model Exchange“, dále ME, a „co-simulaci“, dále CS. Co ale tyto pojmy znamenají?

FMU, které bylo vytvořeno jako ME, definuje systém na základě diferenciálních rovnic. Jestliže chceme simulovat takové FMU, musí software, ve kterém tak chceme učinit, připojit FMU k tzv. solveru. Solver určuje jednotlivé stavy simulace, velikost integračního kroku a to, jak vypočítat následující stav.

FMU, exportované jako CS, má tento solver zabudovaný v sobě. Simulace tedy poté probíhá tak, že simulační software nastaví hodnoty vstupů FMU pro daný čas, poté řekne FMU, aby se posunulo o integrační krok, a poté přečte hodnoty na výstupech FMU. Tento proces se stále opakuje, dokud simulace nedosáhne požadovaného času, výsledku, nebo se v ní nestane nějaká neočekávaná chyba.

4.1.2 Výhody

Největší výhodou FMI je jednoznačně možnost využití napříč mnoha softwary, bez nutnosti měnit původní model. Pokud software obsahuje možnost importování FMU, dá se s ním v něm dále pracovat. To znamená, že pokud např. spolupracují dvě různé strany využívající každá jiný software, nemusí to být takový problém. Vytvoří se model nějakého systému s využitím programovacího jazyka Modelica a exportuje se jako FMU. Obě strany poté mohou tento model využívat ve svých nástrojích.

Stejně tak každý nástroj, nebo software, má jiné možnosti. Software, který je ideální pro jeden projekt, nemusí být dobrou volbou pro projekt s jiným cílem. Díky FMI se tento problém sníží na minimum, protože se vždy dá nástroj pro práci s FMU změnit na jiný, který se více hodí na danou práci.

4.1.3 Nevýhody

Velkým omezením FMI je ovšem to, že ne všechny softwary a nástroje pro simulaci FMI podporují obě jeho verze. Pokud by se tedy FMU vyexportovalo z jednoho softwaru jako FMI 2.0 a druhý software by ještě neměl podporu pro tuto verzi, nelze ho do něj naimportovat a dále s ním pracovat. Dalším problémem může být i to, že některé softwary nabízejí pouze možnost FMU importovat, nikoli exportovat. Může se tedy stát, že tým bude potřebovat vytvořit model v softwaru, který se pro daný úkon nejvíce hodil, ale nebudu ho moci použít jen proto, že FMI není podporováno z hlediska všech jeho funkcí.

Je proto důležité, aby se v budoucnu softwarové trh vyvinul tak, aby všechny softwary a nástroje, které se rozhodnou pro podporu FMI, podporovali tento standard naplno a nejen částečně. Bohužel z reálného hlediska, není příliš pravděpodobné zakomponování všech funkcí podpory FMI najednou.

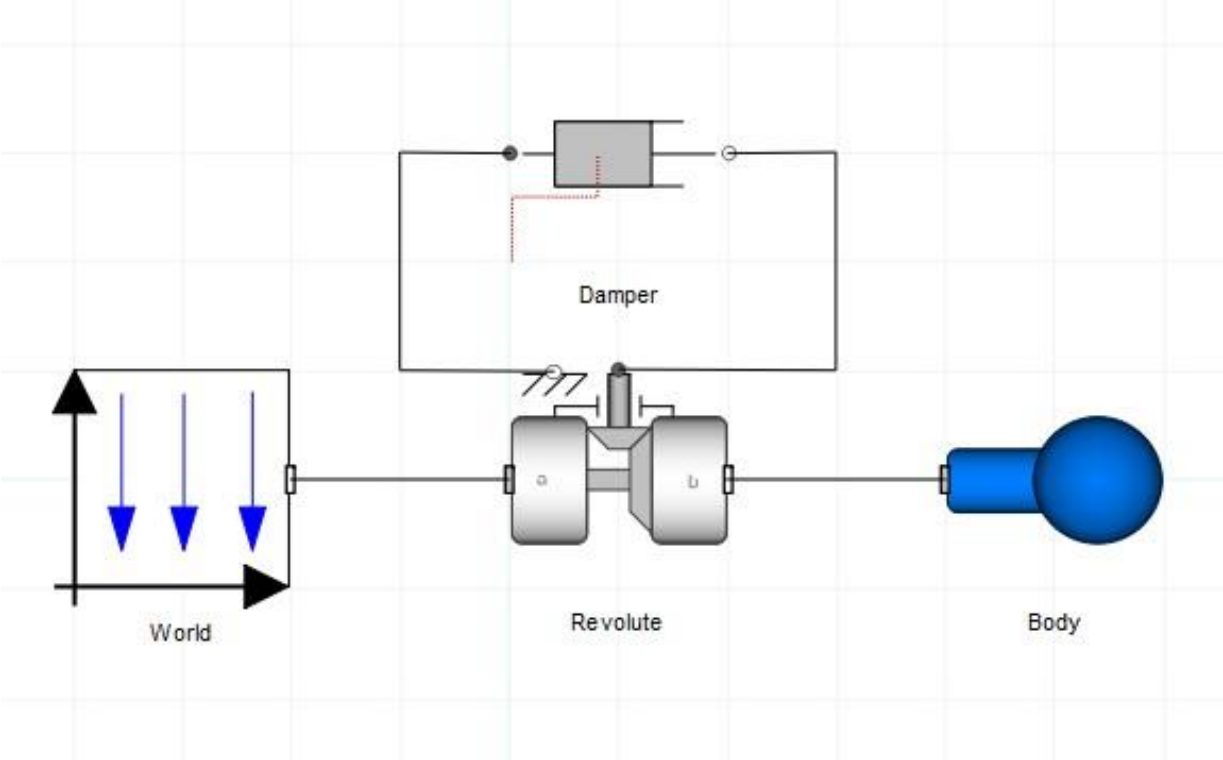
4.2 Využitelnost

Využitelnost FMI závisí na dostupných knihovnách pro import a simulaci FMU. Společnost Modelon takové knihovny nabízí. Jsou psány v jazyce „C“. To znamená, že nejvýhodnější je rozhodnutí využít FMU v jazyce „C“, „C++“. Nicméně všechny jazyky, které mají možnost importování externích „C“ knihoven, je možno pro simulaci FMU využít. Příkladem může být „C#“, ovšem není to zde tak jednoduché jako v případě C++.

4.3 Příklady využití FMI

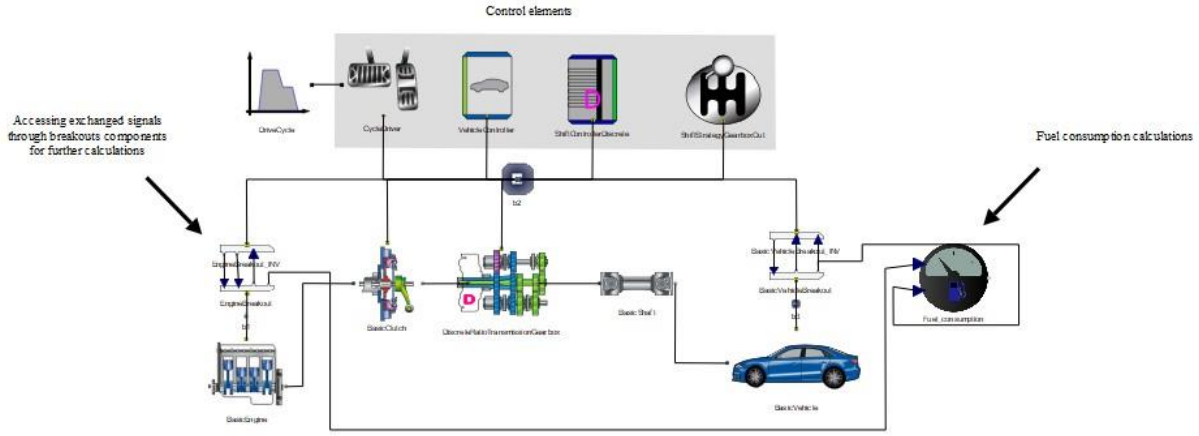
Jedním z odvětví využití FMI je právě dopravní průmysl, který napříč využívá všechny typy systémů, které modelování v Modelice nabízí. Jako první se u automobilů vybaví mechanické systémy, tedy např. přenos točivých momentů. Poté jsou to systémy termální – tepelné a energetické přenosy napříč komponenty v celém systému. Dále to mohou být systémy hydraulické, v případě detailnějšího modelu i elektrické, představující baterii, nebo i výkon spotřebovaný pro běh systémů automobilu jako jsou např. světla. V neposlední řadě jsou to systémy řídicí, bez kterých by se většina systémů, které existují, neobešla.

Příklad jednoduchého fyzikálního kyvadla je vyobrazen na **obr. 6**.



Obrázek 6 - Příklad modelu fyzikálního kyvadla **[VI]**

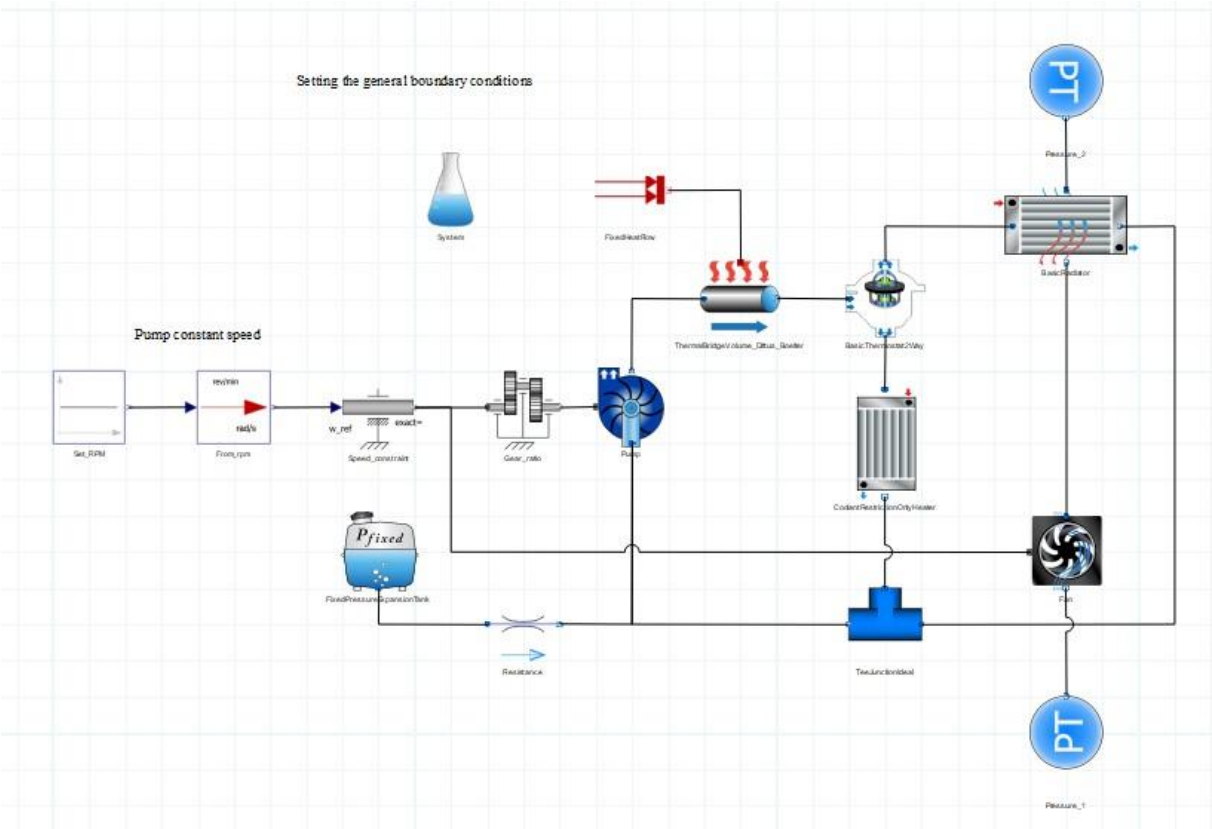
Příklad systému osobního automobilu vypadá v jednom ze softwarů rozebíraných v následující kapitole jako na **obr. 7**.



Obrázek 7 - Příklad modelu s využitím jazyka Modelica **[VI]**

Takovému modelu mohou být přiřazeny vstupy a výstupy a může být exportován podle standardu FMI. Následně je možno ho využít i v jiných softwarech, jak bylo popsáno v předchozích kapitolách.

Jak už bylo řečeno, FMI se dá využít i jako systémy termální, např. chladící. Takový systém se dá pochopitelně spojit i s modelem na **obr. 7**. Samostatný chladící systém můžeme vidět na **obr. 8**.



Obrázek 8 - Příklad modelu systému chlazení **[VI]**

Kromě zmíněných složitějších modelů, lze samozřejmě vytvořit i základní systémy jako je např. jednoduché kyvadlo. Jeho model je vyobrazen na **obr. 8**.

S příklady modelů, které je možné vytvořit v jazyce Modelica a využít je dle standardu FMI, by se dalo by se dalo pokračovat dále přes systémy v letectví – fyzika letadla, řídicí systémy, výrobě – roboty a znovu jejich řídicí systémy až po např. systémy v jednotlivých domácnostech, do kterých se dnes dostává stále více technologií pod hromadnějším označením „smart“. V našem případě nás budou zajímat modely osobních automobilů a systémy s nimi spojené.

5 Nástroje pro modelování v Modelice

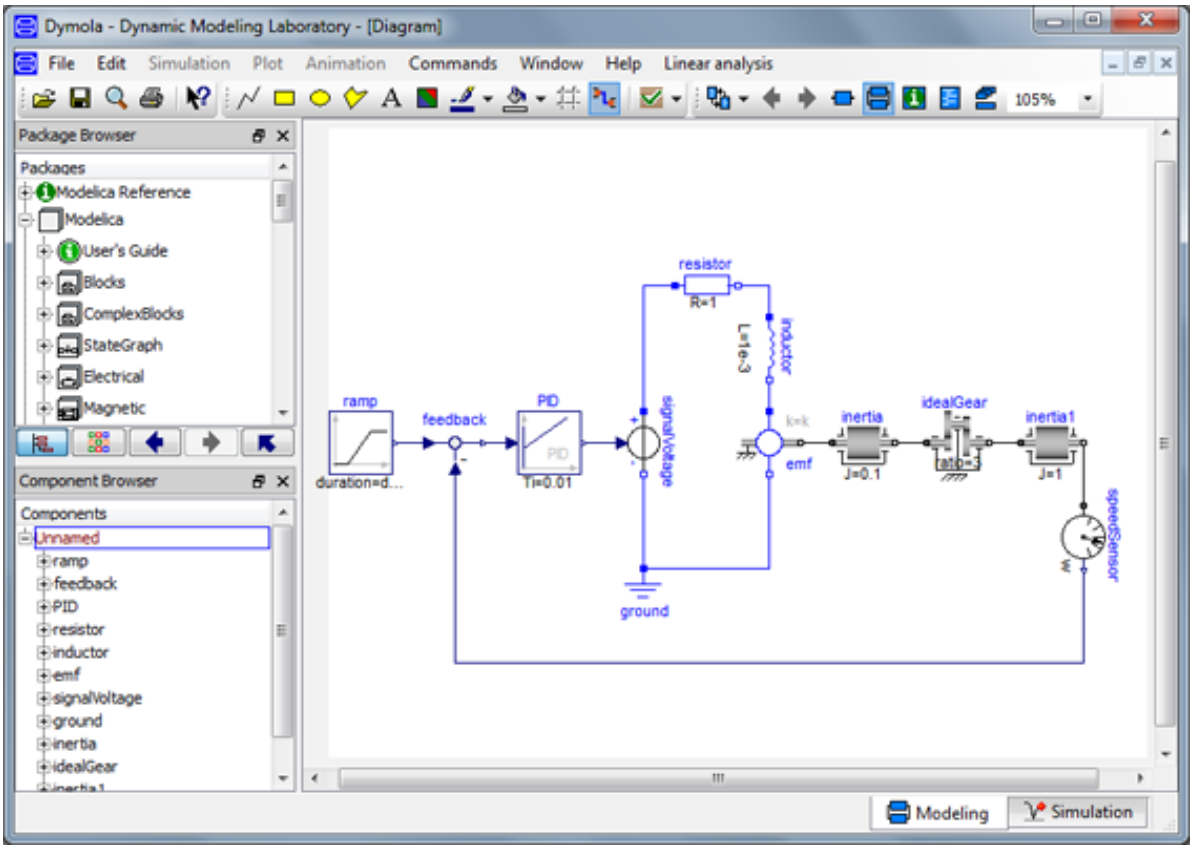
Nástrojů a softwarů využívajících pro své účely základ jazyka Modelica je mnoho. Každý má své specifické zaměření, a proto je důležité při plánování projektu definovat cíle, kterých je třeba těmito nástroji dosáhnout a podle toho zvolit, který bude použit. V této kapitole bude rozebráno několik takových nástrojů a popsány důvody volby jednoho nebo více nástrojů pro tuto práci.

5.1 Dymola

Dymola je software vyvíjený firmou Dassault Systems spadající do skupiny systémů Catia Engineering. Původně byl navržený pro jiný jazyk než Modelica, a to se stejným názvem jako je název samotného softwaru, Dymola. S postupem času a vývojem se ale software nakonec v roce 2002 dopracoval k tomu, že podporuje pouze jazyk Modelica **[5]**.

Umožňuje využití všech dostupných knihoven se základem v Modelica Standard Library od The Modelica Association až po dostupné knihovny od jiných společností a v neposlední řadě si v Dymole můžete vytvořit i knihovny svoje a jednoduše je uložit připravené pro implementaci do jiných softwarů.

Patří mezi nejpoužívanější ve svém oboru, díky jednoduchosti a přímočarosti uživatelského prostředí. Nicméně jednoduchost má vždy i svá proti a může se stát, že z jednoduchého uživatelského prostředí, které má práci uživateli usnadnit se stane prostředí, které dělá přesný opak. Ukázku prostředí softwaru Dymola můžeme vidět na **obr. 9**.



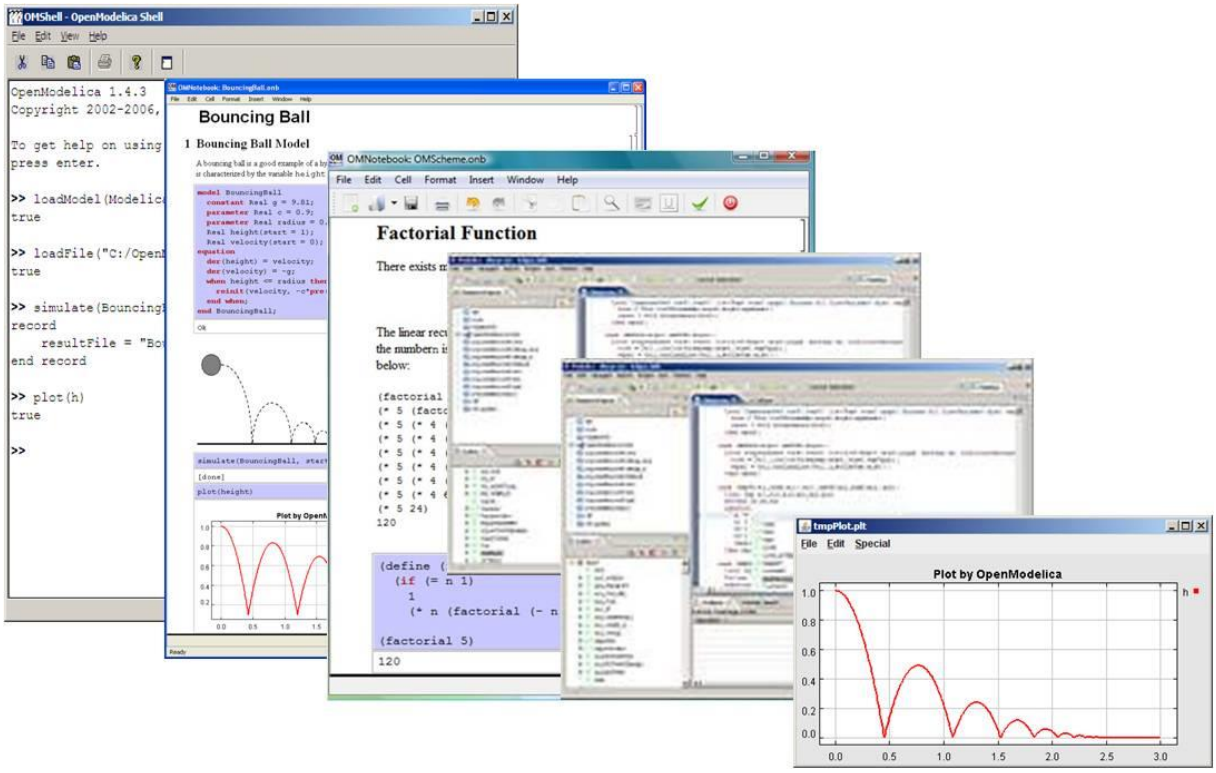
Obrázek 9 - Ukázka prostředí softwaru Dymola **[VII]**

5.2 OpenModelica

OpenModelica je znovu open-source software, jak už název napovídá. A stejně tak název napovídá o tom, že pro modelování a simulace využívá jazyk Modelica. Podobně jako Modelica Association, která vyvinula Modelicu jako programovací jazyk, je OpenModelica podporována neziskovou organizací Open Source Modelica Consortium (OSMC) **[6]**.

Cílem bylo vytvořit software dostupný nejen pro firmy, které tento typ nástrojů potřebují ke své práci, ale i např. pro studenty technických oborů, kteří si nemohou dovolit platit za každý nástroj, který by ke své práci potřebovali využít. Stejně tak otevírá OpenModelica možnost výuky předmětů na jejím základě, což může do budoucna rapidně zvýšit kvalitu a atraktivitu technických oborů na trhu práce.

Kromě velkého množství nástrojů spojených s modelováním a simulacemi nabízí OpenModelica i online tutoriály, kterých je pro Modelicu velmi málo. Je proto jednou z nejpřívětivějších voleb pro používání. Ukázka prostředí z oficiálních stránek na **obr. 10**.

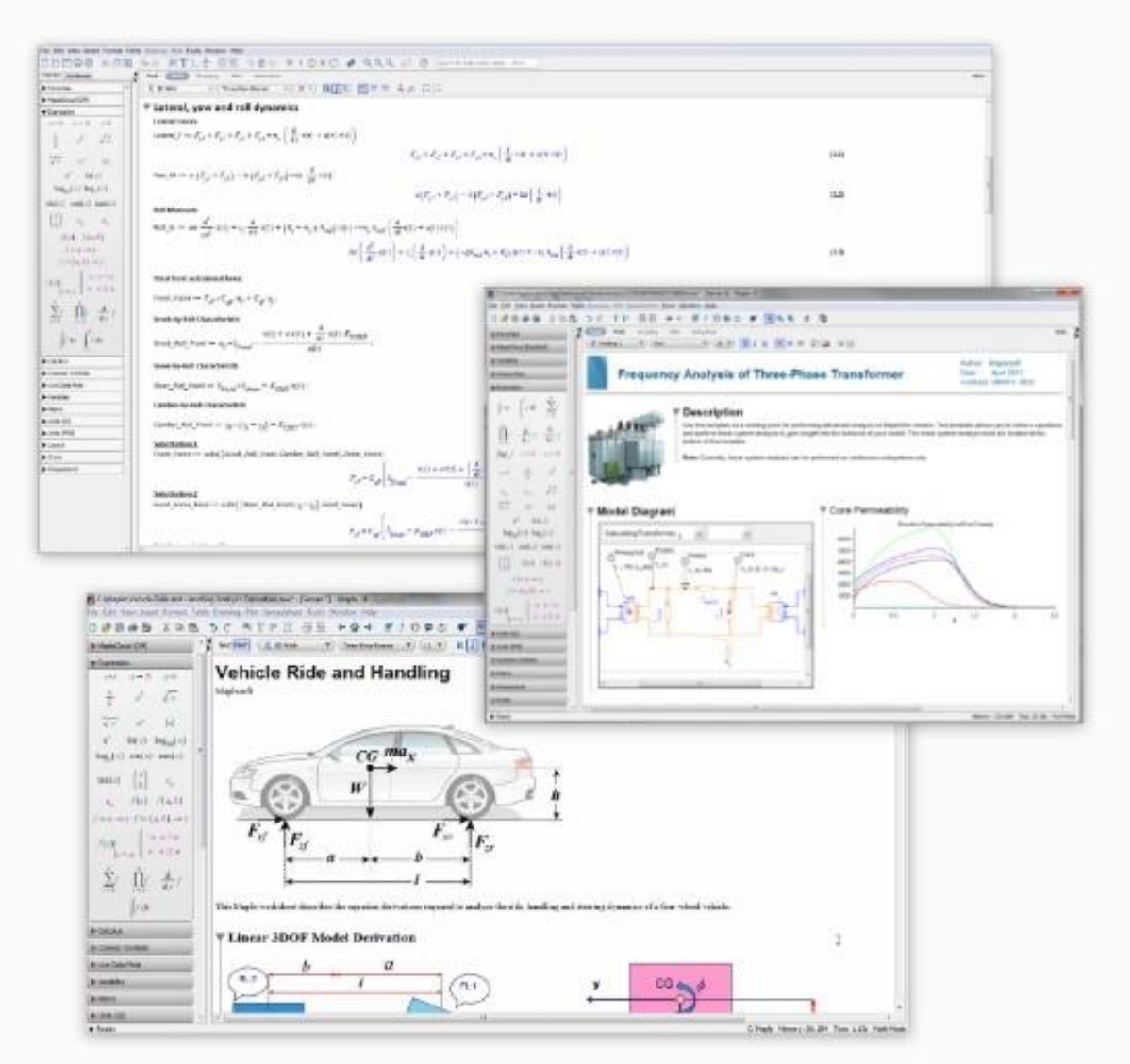


Obrázek 10 - Ukázka prostředí OpenModelica **[VIII]**

5.3 MapleSim

Dalším z prostředí pro využití k Modelica simulacím je MapleSim od společnosti MapleSoft. Společně s prostředím je možno využít přes 700 vytvořených komponentů **[7]**, což může velmi usnadnit práci a ušetřit uživatele od tvorby nových knihoven a komponentů.

Tento nástroj je poněkud obsáhlejší a sofistikovanější než předchozí zmíněné a z toho vyplývá i to, že není zdarma. Pro výhody jako předvytvořené komponenty a rychlejší práce a simulace v prostředí, musí určitou částku platit i studenti, a proto je tento software pro naše účely jen těžko využitelný. Jedna z ilustrací nástrojů MapleSoft je na **obr. 11**.



Obrázek 11 - Ukázka nástrojů společnosti MapleSoft **[IX]**

5.4 Ignite

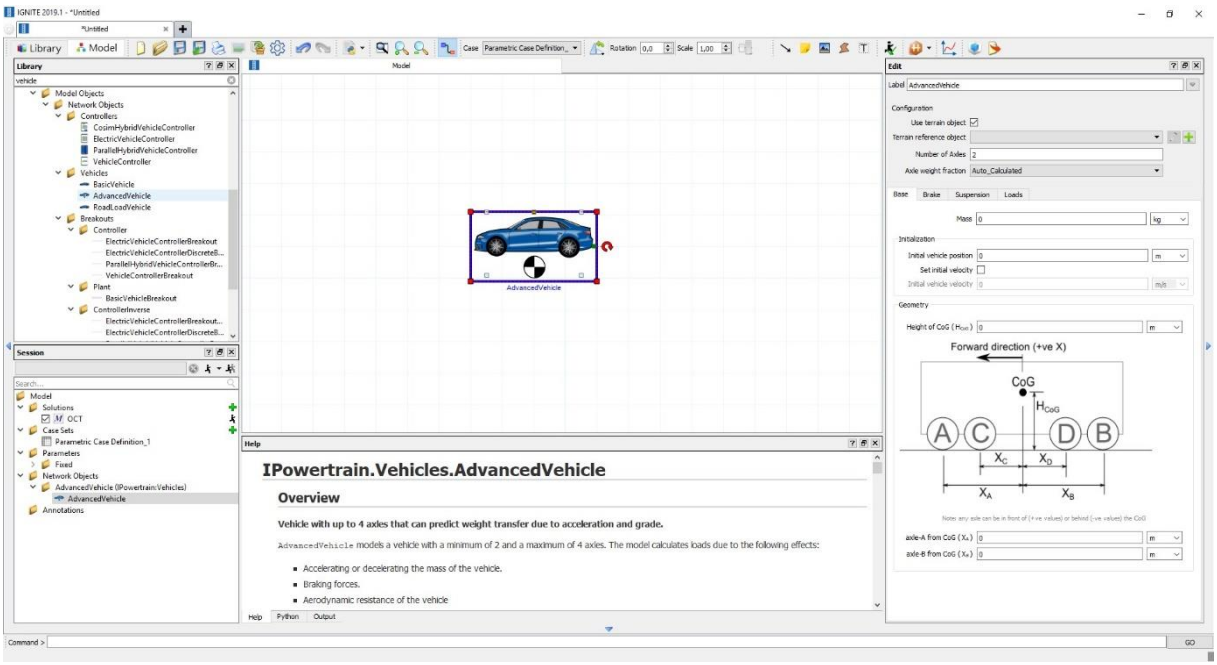
Dalším z placených nástrojů pro modelování v Modelice je Ignite od britské společnosti Ricardo. Tento software umožňuje využití všech dostupných modelica knihoven a zároveň nabízí několik knihoven, které svými vlastnostmi velmi dobře splňují požadavky pro tuto práci.

Základem Ignitu je knihovna IStandard, na které, dá se říci, staví i ostatní knihovny. IPowertrain – komponenty vozidla, jako převodovka, spalovací a elektrické motory, pneumatiky a jejich tření s vozovkou atd. Dále je využít knihovnu IThermofluid nabízející komponenty pro tvorbu hydraulických, nebo chladících systémů. Dalo by se říci, že knihovna IThermofluid je konkurencí pro společnosti Modelon, která nabízí knihovny podobné.

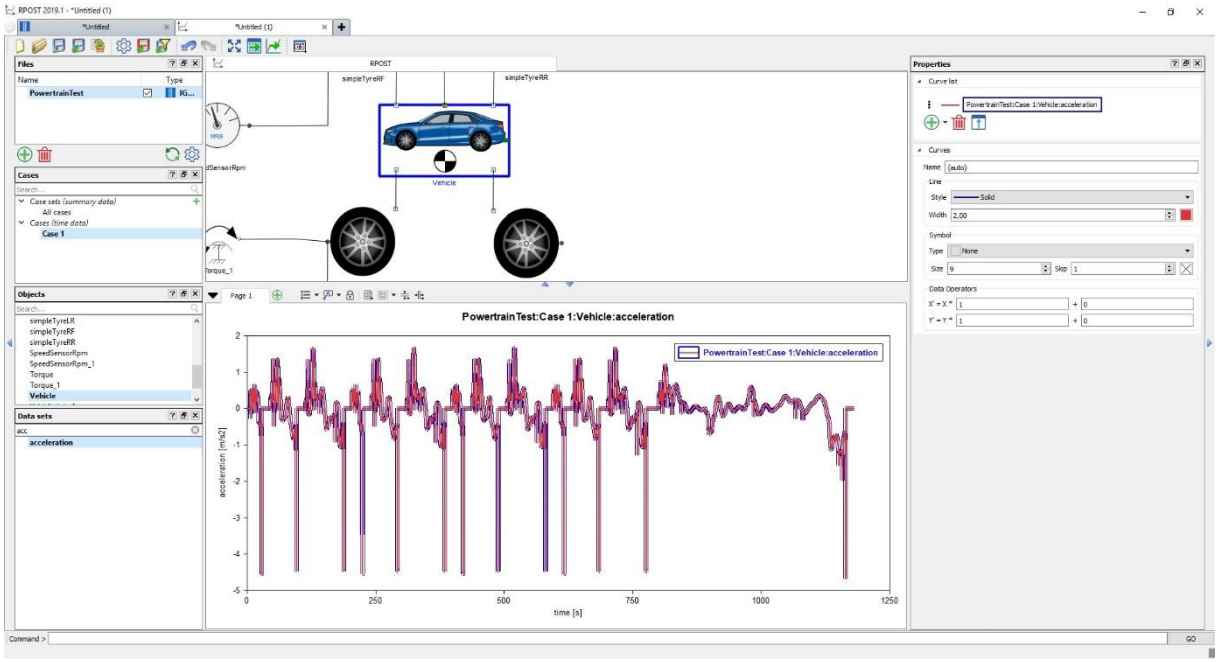
V neposlední řadě nabízí Ignite knihovnu IMoved, která umožňuje vizualizaci jízdy a pohybu vozidel včetně odpružení a dalších fyzikálních vlastností.

Kromě zmíněných vlastností nabízí Ignite exportování FMU 1.0 v 32bitové i 64bitové verzi. Importovat a simulovat dokáže FMU 1.0 64bit. Tedy splňuje základní požadavky pro tuto práci.

Fakulta dopravní má na tento software licenci a je proto nejlepší volbou pro využití, protože kromě exportování potřebného FMU nabízí i zmíněné knihovny využitelné pro model pohonného ústrojí vozidla, které by bylo ve většině ostatních případů třeba vytvořit samostatně, což by bylo, díky své obsáhlosti, tématem pro několik dalších prací. Ukázku prostředí softwaru Ignite a nástroje pro zobrazování dat R-Post můžeme vidět na **obr. 12** a **13**.



Obrázek 12 - Ukázka prostředí softwaru Ignite **[VI]**



Obrázek 13 - Ukázka prostředí nástroje R-Post od společnosti Ricardo **[X]**

6 Fyzikální a grafické enginy

Grafické a fyzikální enginy jsou v dnešní době nepostradatelnými součástmi téměř všech vyvíjených virtuálních prostředí, a to včetně softwarů v dopravních simulátorech, případně i jiných odvětví. Jejich účelem je usnadnění a částečné zautomatizování nezbytných funkcí pro tvorbu virtuálního světa, scénářů v simulátoru a podobně. Enginy jsou navrženy tak, aby za uživatele/vývojáře dělaly, co nejvíce věcí sami bez potřeby navrhovat a programovat součásti jako je např. základní fyzikální chování objektů – to zastávají fyzikální enginy, nebo chování světla atd. V některých grafických enginech dokonce ani není za potřebí umět programovací jazyk, protože ten je generován na základě modelu v uživatelském prostředí, většinou tvořeného z bloků s určitými funkcemi a konektory mezi nimi.

Toto je důvodem, proč je v dnešní době virtuální realita v tak pokročilém stadiu. Vývojáři se nemusí zaobírat malichernostmi, které zaberou velkou část jejich času a mohou se soustředit na inovace a nové přístupy.

6.1 Aplikační rozhraní

Nepostradatelným základem grafických enginů je práce s 2D a 3D objekty. Jejich vyobrazením v prostoru, přidělováním textur na plochy atd. Na základě těchto aplikací dostává hardware počítače instrukce, jak se má chovat, a to nejen v oblasti grafické, ale i zvukové. Hlavním cílem těchto aplikací je využít hardware tak, aby poskytoval, co největší výkon. Znamená to tedy, že se snaží maximalizovat účinnost procesů od hardwaru, přes software až k tomu, co vidí uživatel, např. FPS – frames per second.

6.1.1 DirectX

DirectX je balíček tzv. API – Aplication programming interface, díky nim mají programátoři jednodušší přístup k fundamentálním funkcím dané aplikace, na kterých je postavena. Byl vydán v roce 1995 společností Microsoft a dnes nenajdeme počítač s operačním systémem Windows, ve kterém by tento balíček nebyl **[8]**.

Označení „X“ zde nebylo od počátku, původně existovaly 3 verze těchto balíčků, a to byly Direct3D, DirectDraw a DirectMusic. V pozdější fázi vývoje byly ale všechny tyto balíčky shrnuty do jednoho skrytého pod názvem DirectX.

6.1.2 OpenGL

OpenGL je zkratkou pro Open Graphics Library. Z názvu tedy vypovídá, že tento konkrétní balíček zastává pouze funkce grafické, tedy obdobné jako původní Direct3D a DirectDraw. Je o 3 roky starší z hlediska data vydání a od té doby se stal jedním z nejpoužívanějších API napříč mnoha počítačovými platformami **[9]**.

6.1.3 Rozdíly, výhody, nevýhody

Nejvýznamnějším rozdílem mezi dvěma zmíněnými API balíčky je možnost využití na různých platformách. Zatímco OpenGL může pracovat např. s Linuxem, DirectX pracuje zejména na .NET platformách. Zdroje se v dalších faktech poměrně rozcházejí, a to např. v porovnání rychlostí obou balíčků, nebo i v uživatelských možnostech. Za zmínku ovšem stojí to, že pro DirectX je z hlediska programování dostupná mnohem obsáhlejší dokumentace než pro OpenGL.

6.2 PhysX

Jedním z dnes nejrozšířenějších fyzikálních enginů je PhysX, který je poskytován společností NVIDIA. Je využíván v mnoha grafických enginech, a proto ho najdeme ve více než 150 současných hrách. Engine je využíván více než 10000 herními vývojáři **[10]**. PhysX umožňuje interakce objektů v herním prostředí v reálném čase. Bez fyzikálního enginu je to možné také, ale s exponenciálně narůstajícím množstvím částic (komponentů) v prostředí by se rapidně zvyšoval nárok na výkon hardwaru. PhysX optimalizuje vztah mezi výkonem hardwaru a výslednou grafickou stránkou věci, což maximalizuje požitek pro uživatele, nebo hráče.

Porovnání efektů bez PhysX a s ním je vidět na **obr. 14**.

Obrázek 14 - Efekt výbuchu ve hře PlanetSide 2 s a bez PhysX **[XI]**

Je vidět, že PhysX viditelně zvyšuje kvalitu efektů díky interakcí objektů ve hře v reálném čase. V simulátoru to znamená, že se zvýší kvalita virtuálního prostředí a přesnost experimentů.

6.2.1 Využití

Bylo řečeno, že PhysX je využíván ve více než 150 hrách. Znamená to, že je využíván napříč několika herními enginy. Mezi ty nejvýznamnější a nejznámější patří Unreal Engine 3 a 4, Unity3D nebo Stingray. V Lednu roku 2019 udělala NVidia významný krok a začala

poskytovat PhysX jako open source, což znamená, že se v nejbližší době zajisté začnou ve velkém objevovat další nástroje, simulace i hry využívající tento fyzikální engine.

6.2.1.1 Unreal engine

Unreal Engine 4 je dnes jedním z nejpoužívanějších grafikých enginů vůbec. Jeho dostupnost a poměrně přívětivé uživatelské prostředí jej dělá velmi žádaným. Poukazuje na to i fakt, že je poskytován jako free software, ovšem s podmínkou, že určité procento z případné tržby z vytvořené hry jde na účet tvůrců herního enginu. Je proto zajímavým konkurentem pro Unity, o kterém se mluví v další kapitole.

Unreal Engine 4 podporuje tvorbu na 11 platformách: Windows PC, PlayStation 4, Xbox One, Mac OS X, iOS, Android, AR, VR, Linux, SteamOS, a HTML5. Nezbytnou součástí vývoje v Unreal Enginu je Unreal Editor, prostřednictvím kterého vývojáři tvoří virtuální prostředí. Tento editor funguje na Windows, OS X a Linuxu **[11]**. Programovací jazyk, který Unreal engine používá je C++, což skýtá velké možnosti ze strany vývojáře, ovšem ke kvalitnímu výsledku jsou potřeba obsáhlejší znalosti než pouze základní znalost C++.

Engine poskytuje tzv. šablony základních mechanik pro hry různých typů od RPG přes akční střílečky až po závodní hry. Je proto možné využít šablonu vozidla pro úpravy a nástavbu, která je cílem této práce. Tyto šablony jsou vyvinuty přímo vývojáři UE4 a jsou dostupné zdarma ve většině případů i s tutoriálem. Nicméně dokumentace týkající se Unreal Enginu je značně omezená a komunita si na nedostatek materiálů opakovaně stěžuje.

V našem případě nesmíme opomenout hry týkající se vozidel a jejich chování. Mezi jednu z nejzajímavějších využívající UE4 patří GTR 3, ukázku můžeme vidět na **obr. 16**.

Obrázek 15 - Ukázka ze hry GTR 3 **[XII]**

6.2.1.2 Unity

Konkurentem Unreal Enginu je Unity, ovšem jen z části, protože se více zaměřuje na platformy jiného typu než Unreal Engine, který klade důraz spíše na Windows a Konzole. Unity je nejznámější díky svým využitím na platformách jako je Android, nebo iOS.

Výčet platforem podporovaných Unity3D je následující: iOS, Android, Windows PC, Universal Windows Platform, Mac, Linux, WebGL, PlayStation 4, Xbox One, Nintendo 3DS, Oculus Rift, Google Cardboard, Steam, PlayStation VR, Gear VR, Windows Mixed Reality, Daydream, Andoird TV, tvOS, Nintendo Switch, Facebook Gameroom, Apple ARKit, Google ARCore, Vuforia. Počet vyjmenovaných platforem je 24 a roste, což je více než dvakrát větší počet oproti UE4. Tato výhoda činí Unity dostupnějším pro mnohem více uživatelů a může ulehčit práci na složitějších projektech.

Unity využívá programovací jazyk C#, který je z hlediska porozumění přívětivější než C++. Na druhou stranu může se stát, že u něj narazíte na překážky, které by byly v C++ jednoduše řešitelné, nicméně to by se dalo říci napříč všemi programovacími jazyky.

Unity nabízí velkou škálu šablon a předvytvořených her ve svém obchodě, ale ne vždy jsou dostupné zdarma, či s kvalitním popisem nebo dokumentací. Stejně jako Unreal, nabízí Unity několik šablon jako součást tutoriálů. Typově se ovšem liší tím, že UE4 poskytuje téměř všechny typy fundamentálních mechanik pro různé typy her, zatímco Unity poskytuje více konkrétní hry. Fundamentální mechaniky těchto her mohou být velmi těžko identifikovatelné

na první pohled. Zároveň v Unity budete těžko hledat celistvou šablonu jednoduchého vozidla s kvalitním popisem.

Z hlediska simulace chování vozidla a dalšího je jedním z nejnovějších zástupců závodních her vytvořených v Unity např. Nascar Heat 3, k vidění na **obr. 17**.

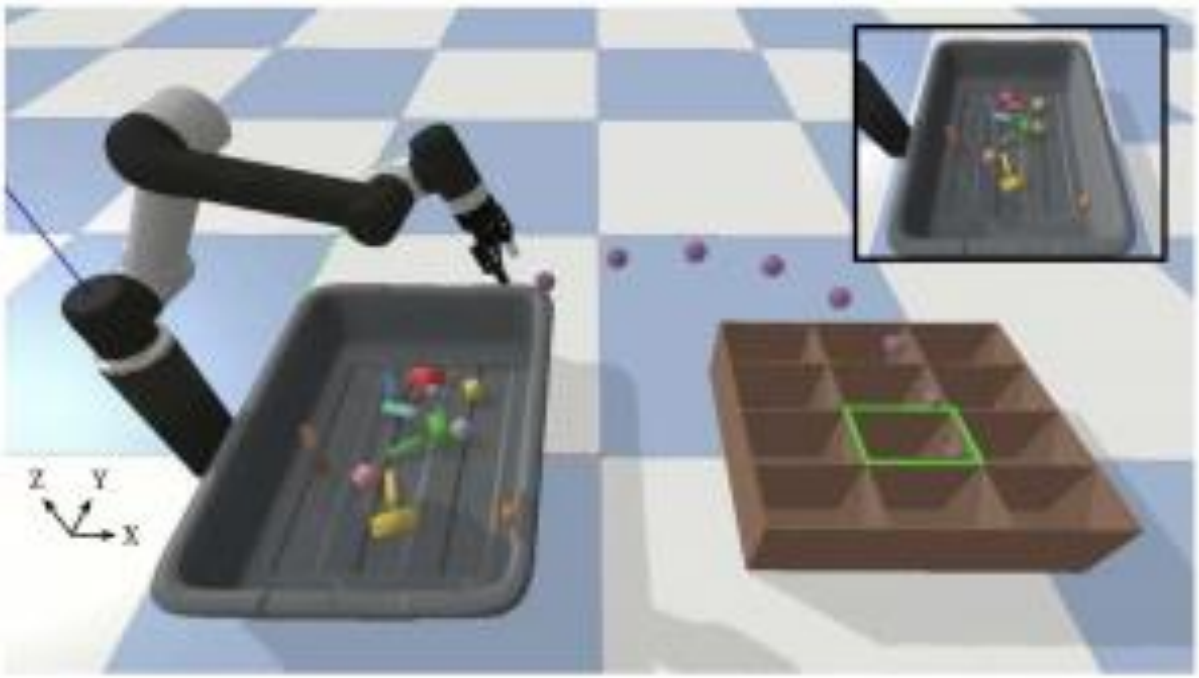
Obrázek 16 - Ukázka ze hry Nascar Heat 3 **[XIII]**

Při porovnání prvního pohledu obou zmíněných závodních her, které jsou přibližně ze stejného období, je vidět, že UE4 je mnohem kvalitnější v oblasti práce se světelnými efekty, ale i vykreslováním textur atd. Nicméně to souvisí i s vyššími systémovými požadavky. I tento fakt dokazuje to, že UE4 se zaměřuje na výkonnostně silnější platformy, zatímco Unity svou jednoduchostí cílí na menší platformy jako jsou mobilní telefony.

6.3 Bullet

Ne tolik rozšířeným fyzikálním enginem je Bullet Real-Time Physics. Jeho největší předností je dostupnost. Je to open-source software, čímž se pro developery otevírá téměř neomezené množství možností, jak fyzikální engine implementovat do své práce. Stejně jako PhysX umožňuje interakce objektů v reálném čase. Tedy stará se o kolize a dynamiku tuhého tělesa. Vzhledem k tomu, že je to open-source a nemá žádnou přímou spojitost s hardwarem nějaké firmy jako např. právě PhysX a NVidia, není dnes k nalezení jeho implementace ve větších grafických enginech.

Další jeho nespornou výhodou je, že kromě Windowsu, Linuxu a macOS, podporuje i tzv. „single board computers“. Konkrétně např. Raspberry Pi nebo Tinker Board **[12]**. Příklad simulace s pomocí Bullet fyziky můžeme vidět na **obr. 18**.



Obrázek 17 - Využití Bullet **[XIV]**

Stejně jako se využívá simulační software pro výzkum a průzkum chování vozidel a jeho součástí. Dá se Bullet využít jako součást pro simulace např. robotů do výroby. Na základě těchto simulací lze poté naprogramovat a optimalizovat chování robotů, čímž se může značně zjednodušit proces vývoje.

6.3.1 Využití

Bullet budeme těžko hledat ve velkých grafických enginech jako je např. Unreal Engine, nicméně není problém ho nalézt v několika open source grafických enginech. Příkladem je Cocos2d-x **[13]**. Tento engine je napsán v programovacím jazyce C++ a využívá se jak k tvorbě her, tak aplikací a grafických rozhraní napříč platformami. Oproti PhysX a jeho 3D, je Cocos vyvinutý spíše pro 2D grafiku a využívá OpenGL. Na komunitních stránkách enginu je k nalezení nespočet her vyvinutých profesionály i herními nadšenci.

6.4 ODE

Stejně jako Bullet je Open Dynamics Engine – ODE open source softwarem pro simulaci dynamiky tuhého tělesa. Využívá se v programovacích jazycích C a C++ a celý jeho zdrojový kód je dostupný na BitBucketu **[14]**. ODE se hodí na simulaci vozidel a je proto velmi dobrou

volbou pro vozidlové simulátory, nicméně možnosti spojení s kvalitním herním enginem už jsou omezené. Proto jsou některé simulátory na fakultě Dopravní vytvořeny na čistém OpenGL. To znamená, že funkce, které při vývoji jsou k dispozici např. v Unreal Editoru – jednoduchý pohyb objekty, nastavení světla atd., nebyly k dispozici a vše bylo třeba vytvářet pouze na základě zdrojového kódu C++. Tedy jednoduchá manipulace s prostředím v grafickém rozhraní a přímá grafická zpětná vazba zde nebyly.

6.3.1 Využití

ODE je využíváno v mnoha simulacích, nicméně pro nás je nejvýznamnější jeho využití ve vozidlových simulátorech. Příklad simulátoru vytvořeného na základě ODE a OpenGL můžeme vidět na **obr. 19**. **[16]**

Obrázek 18 - Ukázka ze simulátoru s ODE a OpenGL **[XV]**

6.4 Porovnání

Výsledkem práce má být implementace pohonného ústrojí v Modelice do jednoho z grafických enginů. Na základě diskuze v předchozích kapitolách byl pro implementaci zvolen Unreal Engine 4. Nejen kvůli potenciálně kvalitnějšímu grafickému prostředí, ale i kvůli vhodnějšímu programovacímu jazyku pro práci s FMU a jeho možnostmi.

7 Implementace

Na základě kapitol 5 a 6 byl pro implementaci zvolen Unreal Engine 4 a jako nástroj pro vytvoření modelu pohonného ústrojí a FMU byl zvolen Ignite od společnosti Ricardo.

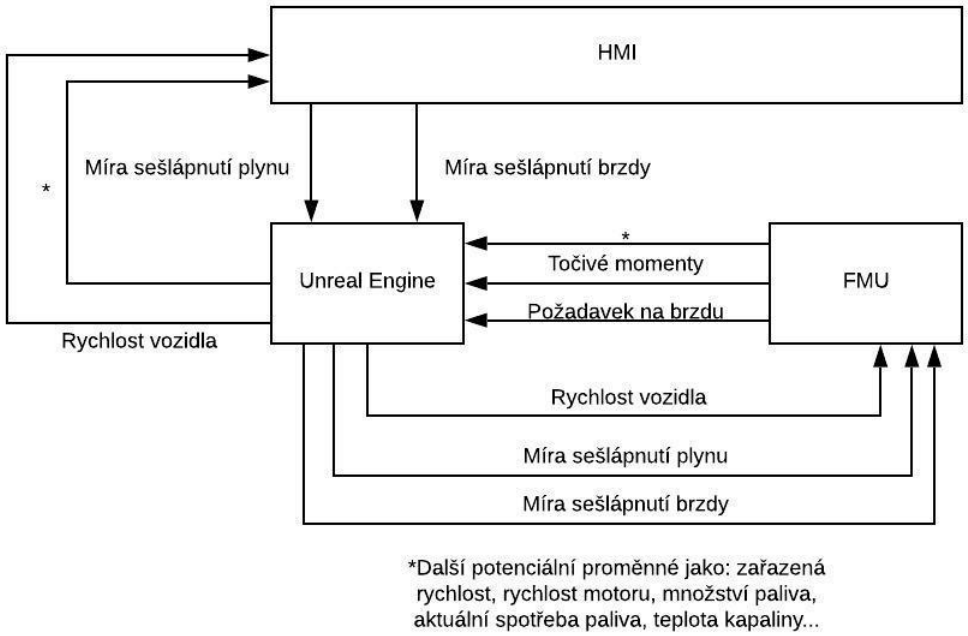
7.1 Model pohonného ústrojí

Před vytvořením modelu pohonného ústrojí bylo třeba navrhnout, jak bude fungovat s ohledem na to, co je možné implementovat v UE4. Práci tedy nebylo možné dělat v přesném pořadí v jakém jsou body v této kapitole.

Knihovna IPowertrain, kterou využívá Ignite umožňuje využití komponentů od motoru, převodovky, přes řídící systémy, řidiče až po vozidlo. V našem případě má být vozidlo vytvořeno v herním enginu, bylo tedy třeba vytvořit vstupy a výstupy tak, aby vozidlo v herním enginu dodávalo nezbytné proměnné modelu a zároveň dostávalo potřebné proměnné pro správné chování vozidla. Přesnější napojení a tok proměnných bude vysvětleno v následující kapitole.

7.1.1 Volba vstupů a výstupů

Na základě zjištěných informací v UE4 bylo možné navrhnout vstupy a výstupy pro náš „blok“ pohonného ústrojí. Výchozím bodem byl náčrt fungování simulátoru, tedy interakce mezi HMI simulátoru, Unreal Enginem a FMU. Tento náčrt je na **obr. 19**.



Obrázek 19 - Diagram simulátoru s UE4 a FMU **[XVI]**

FMU v Ignitu potřebuje pro správnou simulaci dostávat informace o:

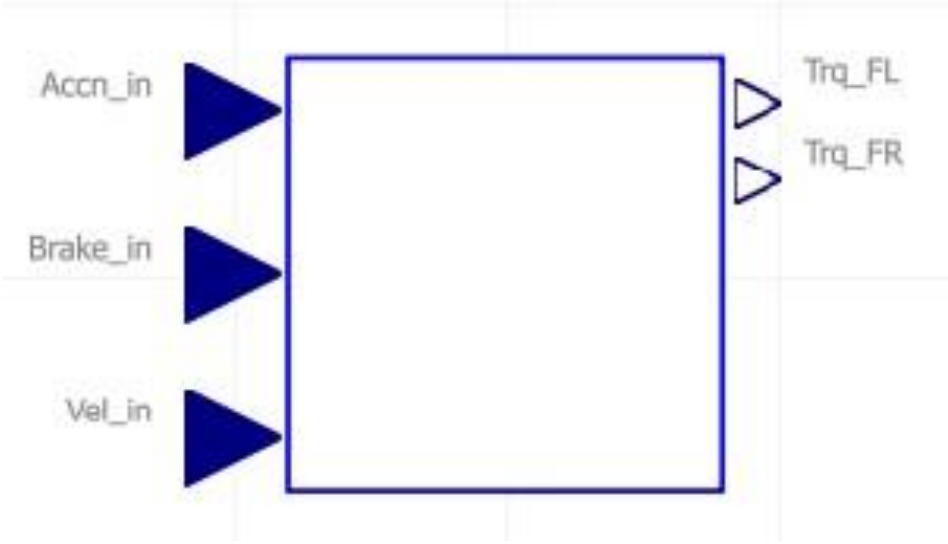
1. Míře sešlápnutí plynového pedálu
2. Míře sešlápnutí brzdového pedálu
3. Rychlosti vozidla

Vstupy 1 a 2 jsou evidentní. Ale proč potřebuje model znát rychlost vozidla? Odpovědí jsou výstupy modelu, kterými jsou točivé momenty na jednotlivá kola, pro naše vozidlo to jsou následující (bylo zvoleno vozidlo s náhonem na přední kola):

1. Točivý moment na levé přední kolo
2. Točivý moment na pravé přední kolo

Pro správný výpočet točivého momentu v simulaci je kromě točivého momentu dodávaného od převodovky také rychlost jednotlivých kol. Tato rychlost je vypočtena na základě celkové rychlosti vozidla jako: $v_k = v_v * t_r$, kde v_k je úhlová rychlost kola v rad/s, v_v je celková rychlost vozidla v m/s a t_r je radius kola v m. V kapitole 7.4.7 je popsáno proč není možné využít přímou úhlovou rychlost jednotlivých kol a je nutné ji vypočítávat z rychlosti celkové.

Schéma bloku se vstupy a výstupy je na **obr. 20**.



Obrázek 20 - Vstupy a výstupy modelu **[VI]**

7.1.2 Problém s proměnnou cycle_driver_state

Knihovna IPowertrain pracuje s proměnnou cycle_driver_state, která představuje stav vozidla. Tyto stavy jsou 3: PARKED, DRIVE, BRAKE. Stav PARKED, znamená, že vozidlo stojí, DRIVE, že vozidlo akceleruje, nebo jede „rovnoměrným přímočarým pohybem“, BRAKE znamená, že vozidlo brzdí a tím pádem zpomaluje. Tato proměnná je potřeba pro správnou

funkci spojenou s řídícím komponentem „CycleDriver“, ten zajišťuje, že vozidlo sleduje předepsanou rychlost pomocí ovládání míry sešlápnutí plynového a brzdového pedálu.

V našem systému je ale řidičem primárně člověk za klávesnicí, případně člověk sedící v simulátoru, držící volant a ovládající pedály. Model by tedy komponent „CycleDriver“ neměl obsahovat. Nicméně celkový systém modelu proměnnou cycle_driver_state využívá a potřebuje. Bylo tedy nutné vytvořit vlastní komponent v Modelice, který to zajistí namísto komponentu „CycleDriver“.

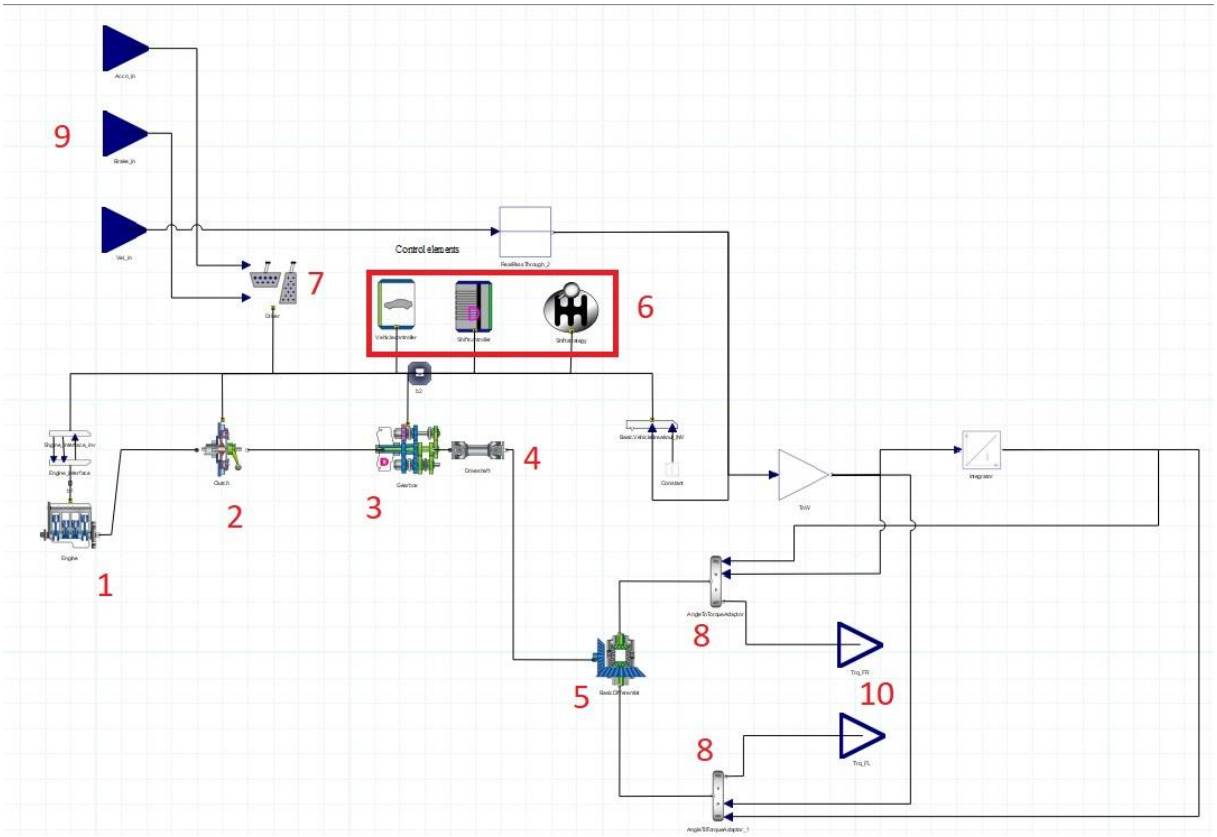
Vstupem do tohoto komponentu je míra sešlápnutí plynu a brzdy (input_acceleration, input_brake). Výstupem jsou potom požadavek na akceleraci a brzdění (driver_accn_demand, driver_brake_demand) a cycle_driver_state. Tento komponent bude poté zapojen do modelu pohonného ústrojí, k vidění na **obr. 21** pod číslem 7.

7.2 Struktura modelu

Bylo řečeno, že vstupy jsou proměnné Accn_in, Brake_in, Vel_in a výstupy Trq_FL, Trq_FR (**obr. 20**), jejich význam je popsán v kapitole 7.1.1. Na základě tohoto základu byl postaven model, který se skládá z těchto komponentů:

1. Motor
2. Spojka
3. Převodovka
4. Hřídel
5. Diferenciál
6. Řídící komponenty – VehicleController, ShiftController, ShiftStrategy
7. Custom driver komponent
8. Komponenty pro výpočet točivého momentu
9. Vstupy
10. Výstupy

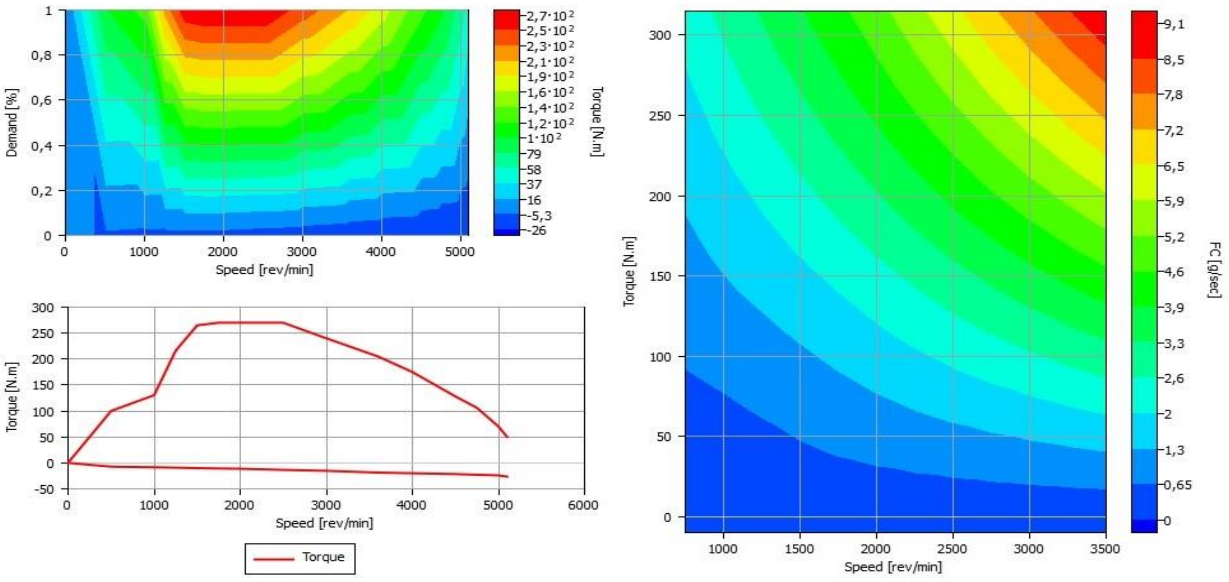
Náhled na model je na **obr. 21**. Pro svou rozsáhlost není možné shrnout model do jednoho obrázku, na kterém by bylo možné vidět vše důležité v modelu. Pro získání více informací je třeba model prozkoumat přímo v Ignitu, případně částečně je to možné i prostřednictvím *.xml, které je součástí FMU.



Obrázek 21 - Model pohonného ústrojí **[VI]**

7.2.1 Motor

Spalovací motor v našem modelu funguje na základě map, které jsou definovány v externích souborech *.m. Prostřednictvím těchto map může motor poskytovat nejen točivý moment, ale také spotřebu paliva a emise. Charakteristiku motoru a jeho spotřebu vidíme na **obr. 22**.



Obrázek 22 - Charakteristika a spotřeba spalovací motoru **[VI]**

7.2.2 Převodovka

Další důležitou částí modelu je strategie řazení. Ta je primárně definována v komponentu „ShiftStrategy“, který podobně jako motor využívá externí soubor *.m. Na jeho základě jsou definovány mapy, které definují, kdy se nadřazuje a podřazuje v závislosti na otáčkách za minuty na výstupu z převodovky, požadavku na plyn a zařazené rychlosti. Tyto mapy jsou vidět na **obr. 23**.

Down-shift speed [rev/min]				
Gear	Demand [%]			
	0.0	0.2	0.9	1.0
0.0	-1000.0	-1000.0	-1000.0	-1000.0
1.0	0.0	0.0	50.0	50.0
2.0	200.0	200.0	500.0	500.0
3.0	750.0	750.0	1200.0	1200.0
4.0	1500.0	1500.0	1800.0	1800.0
5.0	1850.0	1850.0	2000.0	2000.0
6.0	2000.0	2000.0	2100.0	2100.0

Up-shift speed [rev/min]				
Gear	Demand [%]			
	0.0	0.2	0.8	1.0
0.0	0.0	0.0	0.0	0.0
1.0	600.0	600.0	850.0	850.0
2.0	1000.0	1000.0	1500.0	1500.0
3.0	1600.0	1600.0	1900.0	1900.0
4.0	2200.0	2200.0	2500.0	2500.0
5.0	2400.0	2400.0	2650.0	2650.0
6.0	10000.0	10000.0	10000.0	10000.0

Obrázek 23 - Strategie řazení v komponentu ShiftStrategy **[VI]**

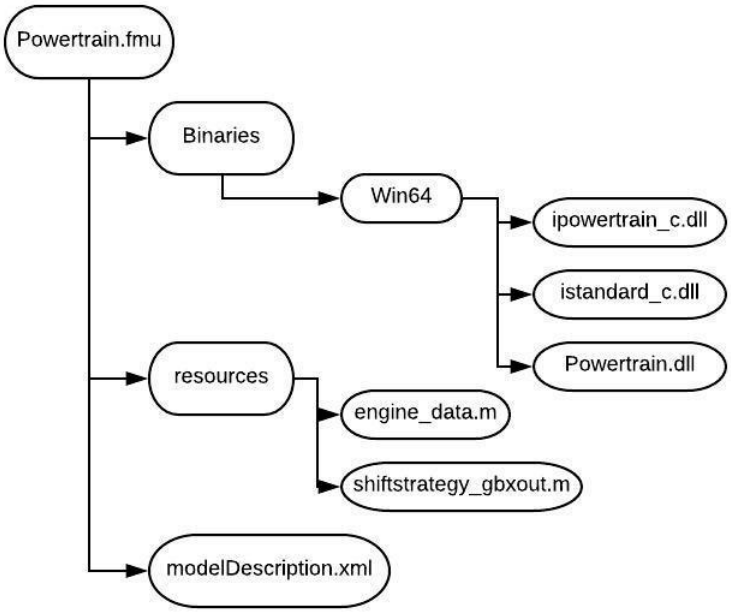
7.3 FMU

Model z předchozí kapitoly by sám o sobě v Ignitu nefungoval. Jinak řečeno nelze ho simulovat přímo. Obdobný model vytvořený pouze pro fungování v Ignitu je na **obr. 6**.

Pro využití námi vytvořeného modelu je třeba ho exportovat do FMU. Čímž se vytvoří dříve popsaný zip s nezbytnými informacemi o modelu. Je důležité vědět na jaké platformě budeme pracovat a podle toho vyexportovat FMU. Unreal Engine primárně pracuje na platformě Win64, je proto třeba FMU vyexportovat pro stejnou platformu. Typ FMU musí být zvolen pro Co-Simulaci, protože simulace FMU bude probíhat společně se simulací v UE4.

Jakmile se FMU exportoval, vytvořil se soubor „Powertrain.fmu“, který je základem pro naší následnou práci. Jeho struktura je taková, jak bylo popsáno v kapitole 4.1 a na **obr. 5**. Ale vzhledem k tomu, že model využívá i jiné knihovny než standardní Modelica knihovnu, přibydou nám ve struktuře další dva *.dll soubory. Odpovídající knihovně IStandard a IPowertrain od společnosti Ricardo. Kromě těchto dvou *.dll je zde i základní *.dll s názvem našeho FMU.

Důležitou věcí v našem jsou „.m“ soubory, které jsou využívány, jak je popsáno v kapitolách 7.2.1 a 7.2.2. Tyto soubory jsou v exportovaném FMU ve složce resources. Schéma struktury vyexportovaného FMU můžeme vidět na **obr. 24**.



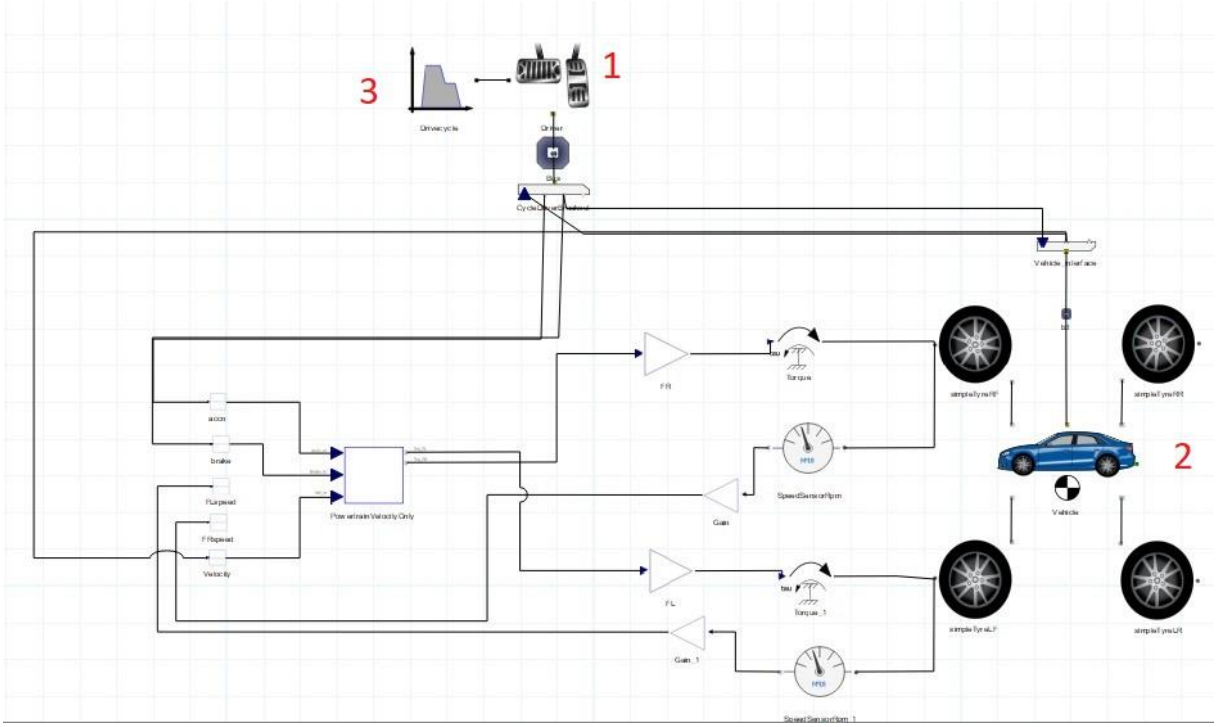
Obrázek 24 - Struktura vyexportovaného FMU **[XVI]**

7.3.1 Co-simulace a související problémy

Vyexportované FMU je možné nejdříve otestovat přímo v Ignitu, který umožňuje co-simulaci. Bylo tedy nutné vytvořit model s naším FMU, které bude napojeno na systém obdobný tomu, který bude vytvořen v herním enginu. Takový systém znamená:

1. Řidiče
2. Vozidlo
3. Jízdní cyklus

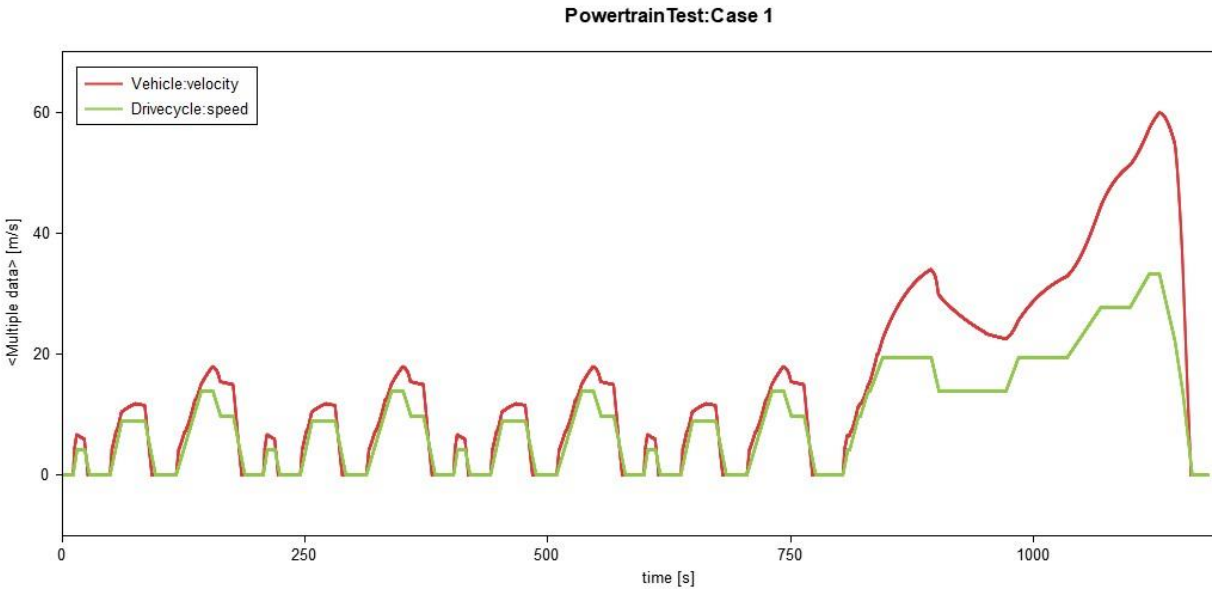
Vnější model bude poskytovat vstupy pro FMU, tedy sešlápnutí plynu, sešlápnutí brzdy a rychlost vozidla. A zároveň bude od FMU dostávat proměnné točivého momentu na obě přední kola. Takový model napojený na FMU je na **obr. 25**.



Obrázek 25 - Testovací model pro co-simulaci FMU **[VI]**

Pro správné výsledky je třeba zvolit adekvátní délku komunikačního pro FMU. V našem případě byla zvolena hodnota 0,1.

Simulace s tímto nastavením proběhne úspěšně, ovšem cosimulace způsobila, že parametry, které byly na komponentu řidiče původně nastavené, teď nejsou nastavené správně. Vozidlo správně nesleduje předepsaný NEDC cyklus a poměrně hodně se odchyluje od požadovaných rychlostí, **obr. 26**.

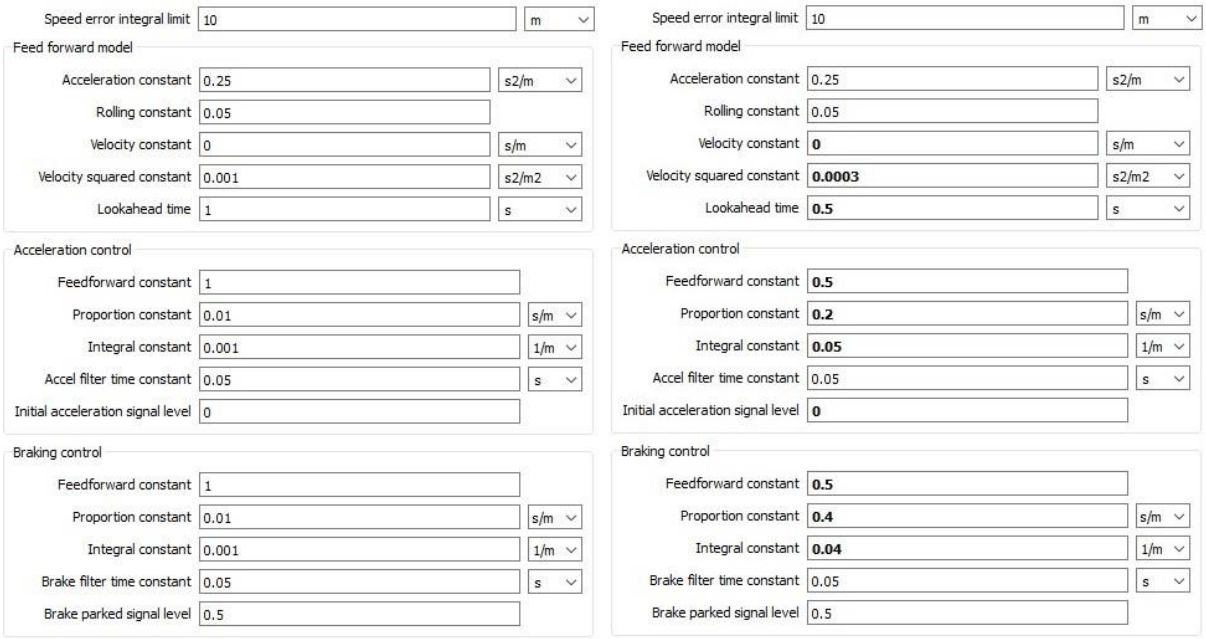


Obrázek 26 - Rychlost vozidla v co-simulaci se špatnými parametry řidiče **[X]**

Je proto třeba pozměnit parametry řidiče tak, aby se rychlost vozidla co nejvíce shodovala s rychlostí jízdního cyklu.

7.3.2 Optimalizace parametrů ovládání Driver komponentu

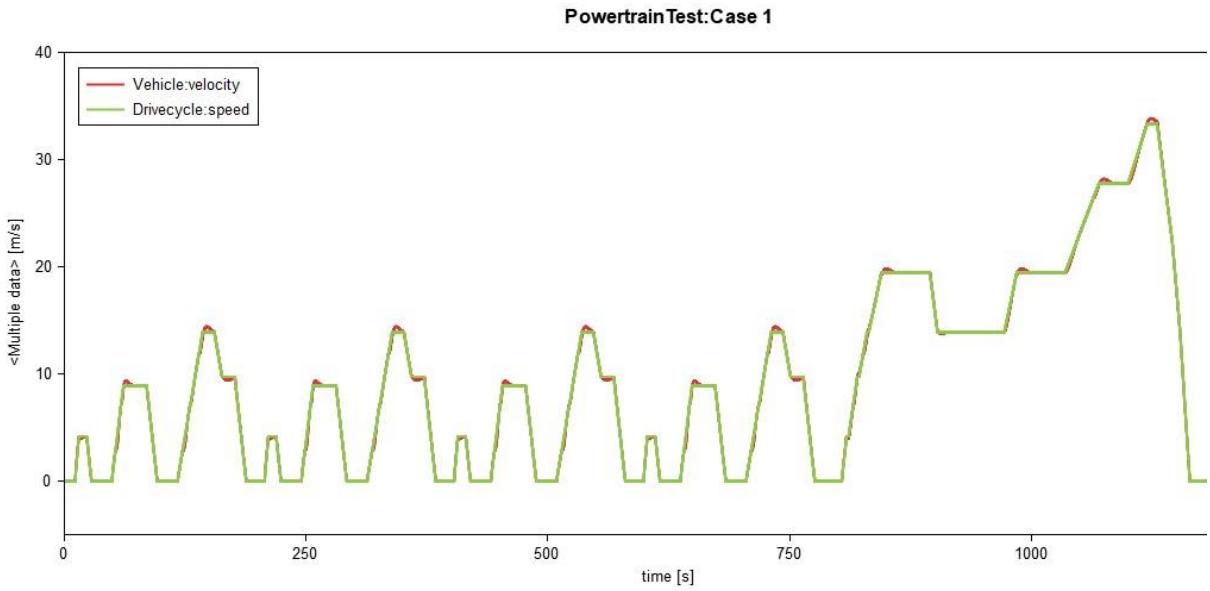
V předchozí kapitole byl popsán problém s parametry řidiče, které je třeba optimalizovat. Jelikož komponent řidiče pracuje na základě PI kontroleru, byla by taková optimalizace ručně velmi náročná, protože ji lze víceméně dělat jen metodou pokus omyl. Je proto příhodné využít nějaký optimalizační software. V našem optimalizačním softwaru byl zvolen cíl, aby rozdíl obou rychlostí na **obr. 26** byl minimální. Parametry, které podle toho byly optimalizovány jsou na **obr. 27** vlevo.



Obrázek 27 - Původní a nové parametry řidiče **[VI]**

Pro optimalizaci byla zvolena metoda SHERPA s 50 simulacemi. Smyslem této metody je přiřadit model $M(\vec{p})$ k sadě pozorovaných dat, kde $\vec{p}$ definuje parametry modelu. Metoda poté určí vektor hodnot parametrů $\overrightarrow{p_0}$, pro které je zvolená statistika přiřazení minimální. Výsledkem byly nové parametry, které jsou v komponentu řidiče vidět na **obr. 27** vpravo.

Díky novým optimalizovaným parametrům na komponentu řidiče vypadají výsledky testovací simulace mnohem lépe než před tím. Rychlost vozidla už kopíruje požadovanou rychlost jízdního cyklu (**obr. 28**). Lze tedy konstatovat, že náš model a FMU jsou navrženy a vymodelovány správně, a lze pokračovat k dalším krokům.



Obrázek 28 - Rychlost vozidla v co-simulaci s novými parametry řidiče **[X]**

7.4 Implementace do UE4

FMU model je připravený a je možné ho použít pro naše účely. Jako herní engine, do kterého bude FMU implementováno byl zvolen Unreal Engine 4, který pracuje na základě C++. Bude proto nutné najít knihovny, které umožní importování a simulaci FMU v C++. Jednu takovou knihovnu nabízí společnost Modelon a je volně dostupná **[15]**.

7.4.1 FMU simulátor

S knihovnami s potřebnými funkcemi je možno napsat potřebný kód pro simulaci FMU, který se později implementuje do herního enginu. Na začátek je ale nutné vytvořit a otestovat jeho základní verzi, před tím, než se přistoupí k samotné implementaci.

K testování bylo původně zvoleno FMU se dvěma vstupy navíc:

1. Úhlová rychlost pravého předního kola
2. Úhlová rychlost levého předního kola

Protože v tu chvíli nebyl znám problém o získávání úhlových rychlostí kol z Unreal enginu popsaný v kapitole 7.4.7. K původním 3 vstupům jsou tu tedy další 2 navíc.

Na ústavu K616 byl již vyvinut program se základními prvky, které se daly využít pro naše účely, nicméně program bylo třeba změnit a aktualizovat tak, aby pracoval s našimi modely. Jelikož původní program pracoval pouze se 2 vstupy a 1 výstupem, bylo třeba vytvořit kód, který se postará o všechny proměnné, které potřebujeme v našem případě využít.

Tyto proměnné jsou potom uživatelem manuálně zapsány do souboru powertrain.ini, ze kterého si program vezme jejich názvy v FMU a přiřadí je k proměnným v C++. Tento soubor je textový a v našem testovacím případě vypadá jako na **obr. 29**.

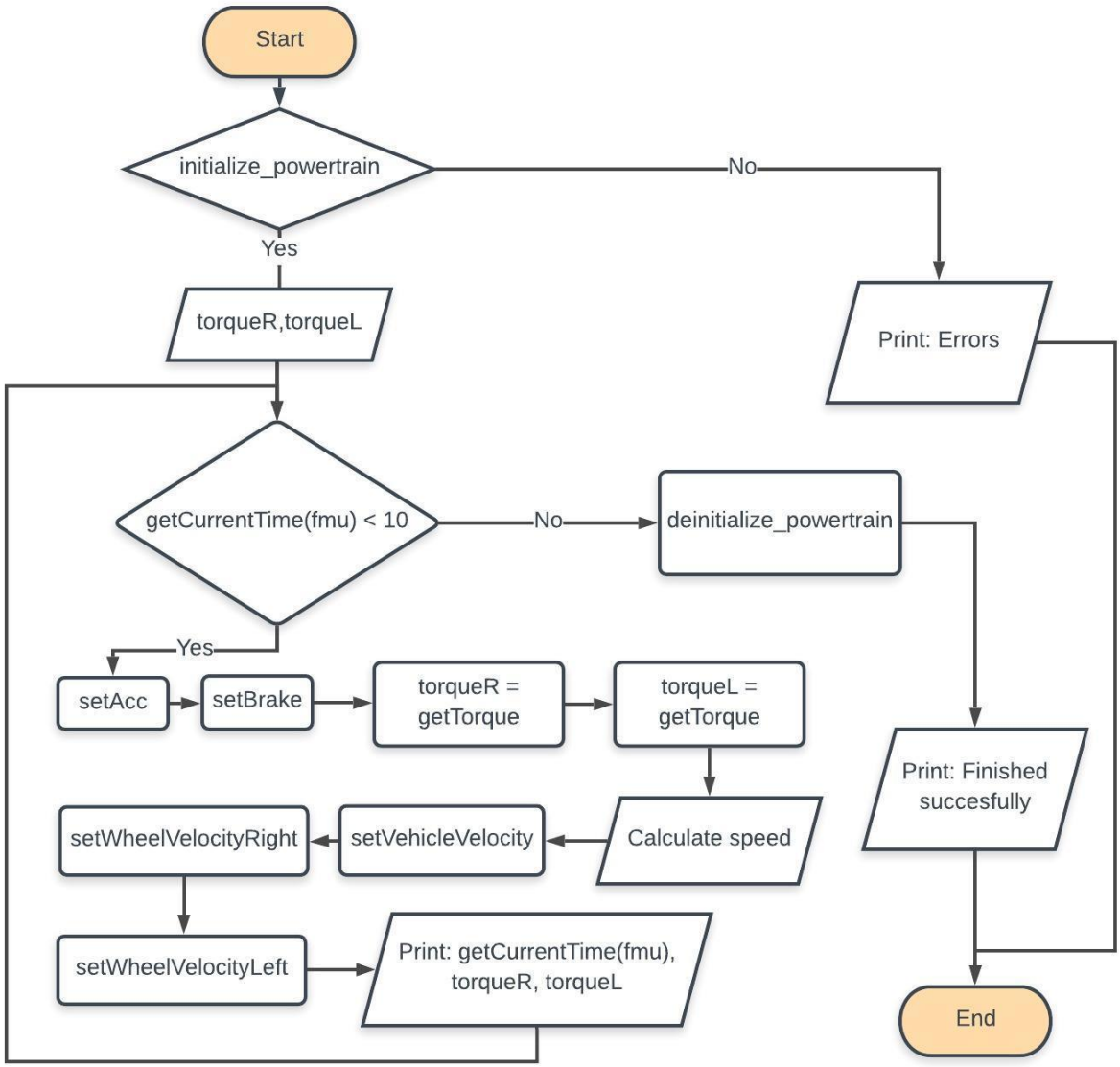
```
[Main]
fmuPath=p_model/PowertrainNEDC.fmu
tmpPath=p_model/tmp
instanceName=PowertrainNEDC
step=0.1
multipleWheels=true

[Variables]
accName=Accn_in
brakeName=Brake_in
velocityName=Vel_in
wheelVelocityName[0]=FRSpeed
wheelVelocityName[1]=FLSpeed
wheelVelocityName[2]=RRSpeed
wheelVelocityName[3]=RLSpeed
torqueName[0] = Trq_FR
torqueName[1] = Trq_FL
torqueName[2] = Trq_RR
torqueName[3] = Trq_RL
```

Obrázek 29 - testovací nastavení powertrain.ini, screenshot

Program potřebuje znát cestu k samotnému fmu (fmuPath) a potřebuje znát cestu ke složce, ve které se vytvoří dočasné soubory (tmpPath). Je třeba nastavit stejný „instanceName“, který byl nastaven při exportování FMU v Ignitu. Proměnná step představuje komunikační krok FMU a promněnná multipleWheels definuje, jestli má program pohánět pouze jedno kolo, nebo více (přesnější specifikace náhonu se mění přímo v kódu). Pod [Variables] se skrývají názvy vstupů a výstupů, které se musí shodovat s těmi, které byly před exportováním FMU v Ignitu. Podle specifikací náhonu kol nemusí být využity všechny proměnné, ale tento případ by se dal teoreticky využít i na náhon na všechny 4 kola.

Jednoduchý flow diagram základní funkce FMU simulátoru je na **obr. 30**.



Obrázek 30 - Flow diagram testovacího simulátoru **[XVI]**

Program musí nejprve inicializovat FMU (initialize_powertrain), což obnáší převzetí nastavení ze souboru nastavení powertrain.ini, nastavení instanceName, poté nastavení cest k FMU, nastavení komunikačního kroku, načtení samotného FMU a poté nastavení jmen jednotlivých proměnných. Pokud se načtení FMU nepodaří, objeví se chybové hlášky a program se přeruší. Pro využití žádaných točivých momentů je třeba definovat proměnné torqueR a torqueL. Poté následuje smyčka, ve které je FMU simulováno. Pro testovací účely stačí 10 sekund simulačního času, což představuje podmínka getCurrentTime (fmu) < 10. Tedy dokud bude simulační čas v FMU menší než 10, simulace bude pokračovat.

V každém simulačním kroku je poté třeba nastavit všechny vstupy, tzn. Acc_in, Brake_in, Vel_in, WheelFR, WheelFL. Tyto proměnné se nastavují funkcemi s prefixem „set“ a jsou pojmenovány na **obr. 29**. V každém kroku potom simulátor získává výsledné výstupy, které

jsou vypočteny na základě zadaných vstupů. Tedy v našem případě Trq_FR a Trq_FL. Funkce získávající výstupy najdeme s prefixem „get“.

V našem simulátoru bylo třeba vstupující proměnné nastavit buď konstantní, nebo je vypočítat příslušnými rovnicemi. V tomto případě bylo pro sledování správného fungování FMU nutné vytvořit rovnice pro výpočet rychlosti vozidla, které se skrývají pod blokem „Calculate speed“. Výsledek je potom využit pro vstupy Vel_in, Wheel_FR a Wheel_FL.

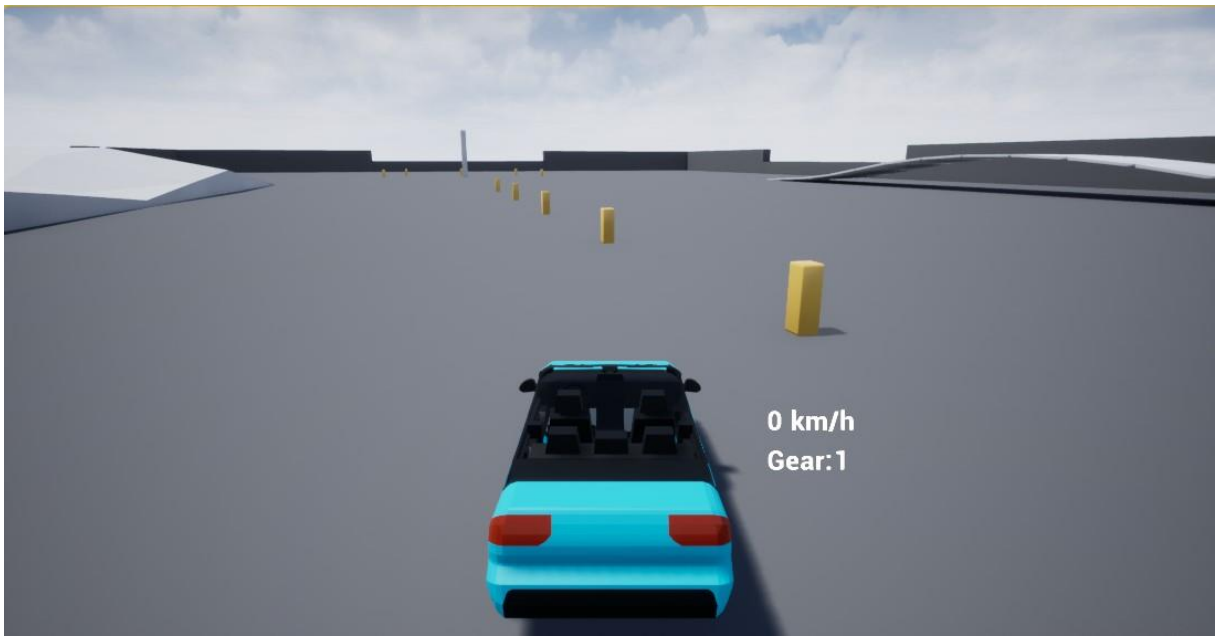
Simulační smyčka běží dokud čas FMU nedosáhne 10 sekund. Poté je FMU deinicializováno (deinitialize_powertrain), čímž se zastaví simulace a vymaže se objekt fmu. Na konci simulace se potom za hodnotami výstupů a času objeví hláška „Finished succesfully“.

Problémem u tohoto postupu je, že fmi knihovna si do tmp složky zkopíruje externí .m soubory, bez kterých fmu nemůže fungovat, ale přidá k nim náhodně generovaný prefix. Je proto potom nutné tyto soubory s vygenerován prefixem zkopírovat do pracovní složky, kde má program běžet, aby fungoval správně.

Při tomto postupu byla simulace FMU úspěšná s výsledky dle očekávání a bylo možné postoupit k další části.

7.4.2 Vytvoření modelu vozidla v UE4

Unreal Engine 4 nabízí velkou škálu tzv. blueprintů, tedy předvytvořených šablon jako základ budoucího virtuálního prostředí. Jedním z takových blueprintů je i vozidlo, které se ovládá pomocí klávesnice a jeho fyzikální model přepočítává požadavek v podobě zmáčknutých kláves na pohyb vozidla, jeho snímek při běhu v UE4 je na **obr. 31**.

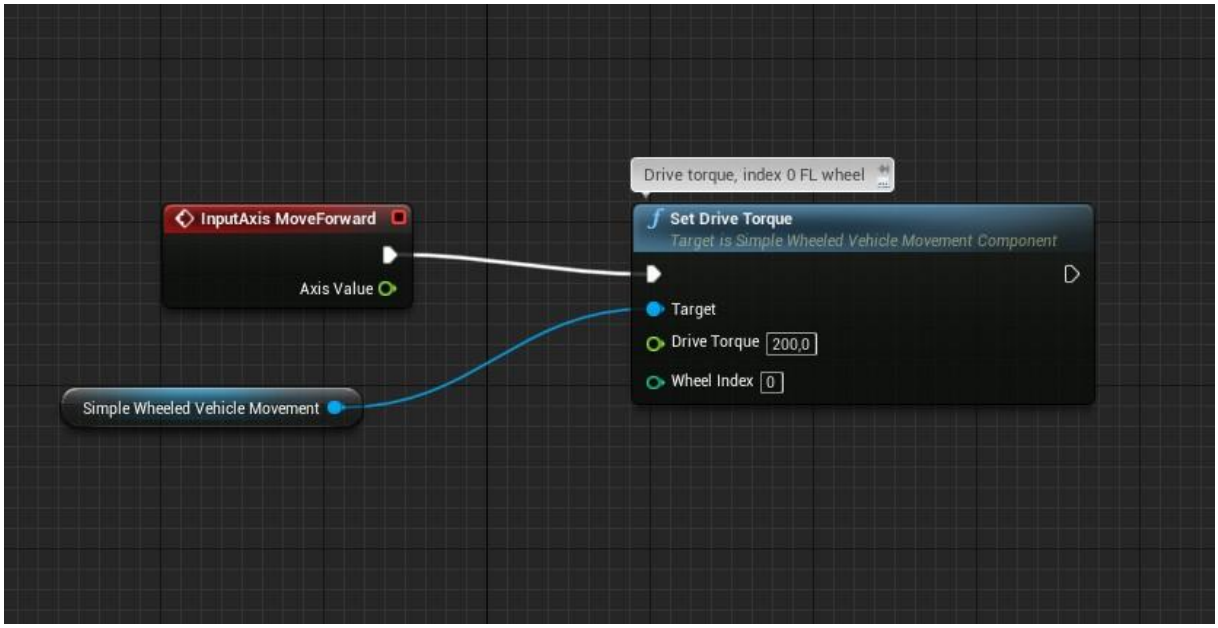


Obrázek 31 - Ukázka z Blueprintu vozidla v UE4 **[XVII]**

Tento blueprint ovšem nevyhovoval našim požadavkům, protože daný požadavek z klávesy (např. šipka dopředu) se využíval ve funkci „setThrottleInput“ a následně byl točivý moment přepočítán a převeden na kole uvnitř předvytvořených komponentů a funkcí. Pro naše účely je třeba mít přímý přístup k točivým momentům a také je třeba odstranit jakoukoli fyziku týkající se pohonného ústrojí, protože my budeme implementovat ústrojí vlastní.

Blueprint využíval komponent „WheeledVehicleMovement“. UE4 nabízí i možnost jednoduššího komponentu, na kterém se dá dobře vytvářet nástavba a tím komponentem je „SimpleWheeledVehicleMovement“. Tento komponent umožňuje aplikovat točivý moment na jednotlivá kola přímo bez jakýchkoli mezivýpočtů, a to je přesně funkce, která je pro náš účel třeba.

Model bylo tedy třeba přestavět tak, aby využíval tento jednodušší komponent. V principu poté vozidlo fungovalo na základě ukázky z event grafu na **obr. 36**. V tomto případě je při zmáčknutí odpovídající klávesy pro pohyb vpřed aplikován točivý moment v hodnotě 200 N.m na dané kolo. Zároveň bylo potřeba nějakým způsobem získat celkovou rychlost vozidla v UE4. K tomu posloužila funkce „Get Forward Speed“, která získává dopřednou rychlost objektu v cm/s. V kódu je tedy pro práci se správnými jednotkami implementován přepočet na m/s.



Obrázek 32 - UE4 ukázka z event grafu vozidla - aplikace točivého momentu **[XVII]**

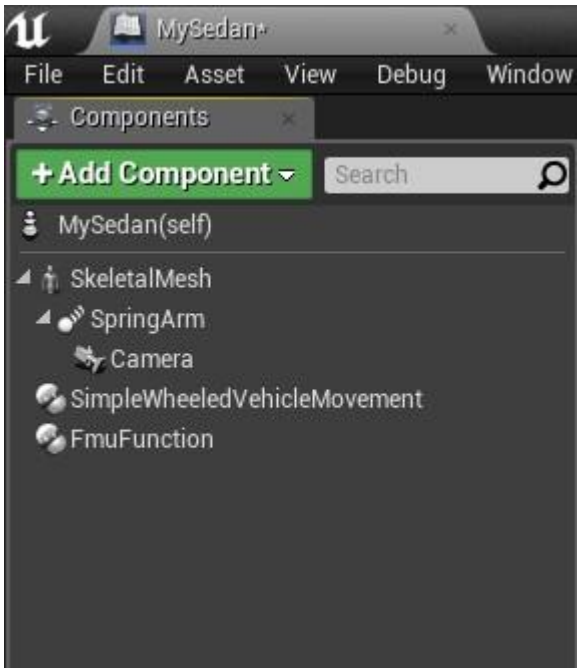
Po vytvoření správného typu pohonu kol pro náš účel bylo vše připraveno pro vytvoření funkce/funkcí, které implementují např. mezi blok „InputAxis MoveForward“ a „Set Drive Torque“ na **obr. 32** samotné FMU, tedy pohonné ústrojí vozidla.

Proto bylo třeba v „actoru“ UE4, tedy v našem vozidle, vytvořit nový C++ komponent, který umožňuje implementování čistého C++ kódu do vozidla. Funkce vytvořené v tomto komponentu se poté dají v event grafu (**obr. 32**) využít podobným způsobem jako např „Set Drive Torque“.

V kapitole 7.4 je popsána již vytvořená knihovna pro práci s FMU v C++ včetně odkazu na ní. Tuto knihovnu je třeba do našeho nového C++ komponentu implementovat a tím využít v UE4.

7.4.3 UE4 komponent pro FMU a jeho funkce

C++ komponent byl nazván „FmuFunction“ a na **obr. 33** ho můžete vidět na stejné úrovni jako „SimpleWheeledVehicleMovement“.



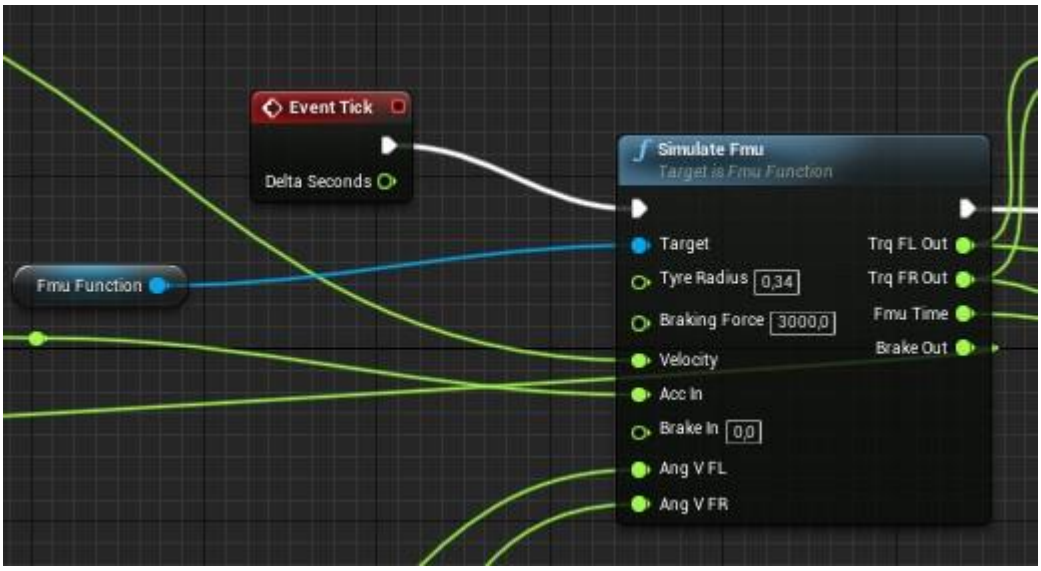
Obrázek 33 - Komponenty vozidla v UE4 včetně FmuFunction **[XVII]**

Základem bylo v tomto komponentu vytvořit na základě testovacího kódu v kapitole 7.4.1 takovou funkci, která nahradí „while loop“ na **obr. 30** tak, aby správně probíhala co-simulace obou modelů, tedy v UE4 a v FMU.

Pro správné fungování FMU byly nakonec vytvořeny funkce 4, první funkcí je:

- SimulateFmu

Tato funkce má, jak z názvu napovídá, na starost simulaci samotného FMU. Je vyvolávána každý frame, neboli snímek. Pokud je tedy FPS (Frames Per Second) na hodnotě 35, je tato funkce vyvolána celkem 35x za sekundu. Její ukázku můžeme vidět na **obr. 34**.

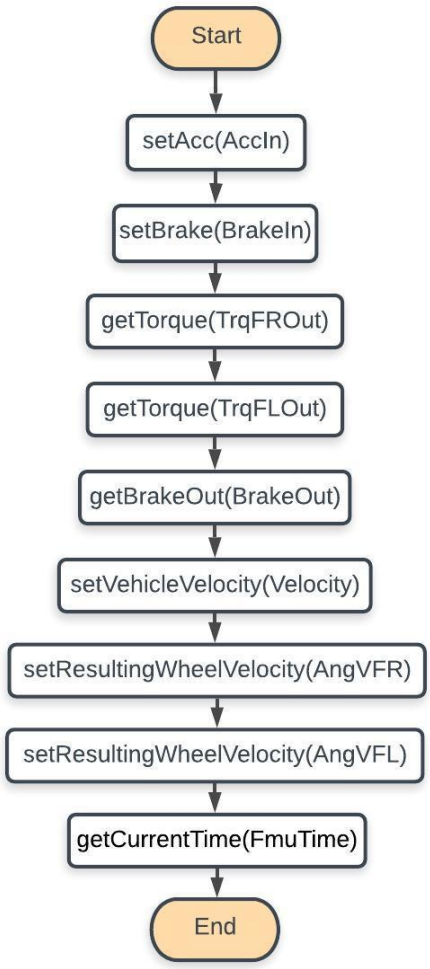


Obrázek 34 - Ukázka funkce SimulateFMU v UE4 editoru **[XVII]**

Ukázka obsahuje, kromě nezbytných vstupů, které poté směřují do FMU, také 2 parametry, které je umožněno nastavit přímo z UE4 editoru, a to: Tyre Radius a Braking Force. Tyre Radius je třeba nastavit takový, jaký je rádius kol vozidla v UE4 a Braking Force určuje sílu, která působí na brzdy při maximálním požadavku na brzdový pedál. K základním výstupům Trq FL Out a Trq FR out, což jsou přímé výstupy FMU, byl přidán další, a to BrakeOut, který bude užitečný v kapitole 8.

Výstupem funkce SimulateFMU je navíc FmuTime, který vypisuje čas uvnitř FMU modelu. Tento výstup byl základem pro postup popsaný v kapitole 7.4.5.

Flow diagram popisující princip funkce SimulateFMU je na **obr. 35**.



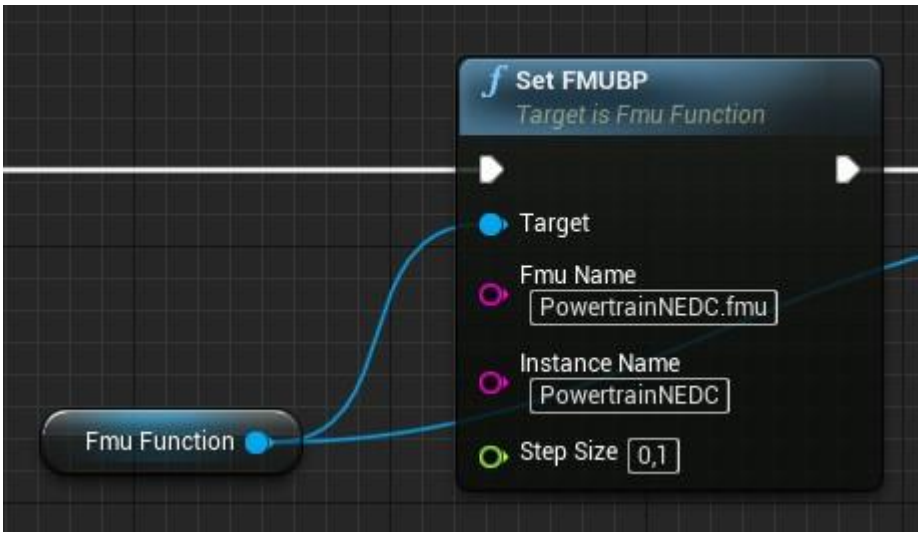
Obrázek 35 - Flow diagram funkce SimulateFMU **[XVI]**

7.4.4 Inicializace a deinicializace FMU

Jelikož musí být funkce SimulateFMU vyvolávána každý snímek, nebo v určitém intervalu, nelze do ní přidat incializování a deinicializování FMU, protože tyto dva procesy mohou proběhnout jen jednou: inicializace na začátku simulace, deinicializace na konci simulace. Bylo proto třeba vytvořit další dvě funkce, a to:

- SetFMUBP
- DestroyFMU

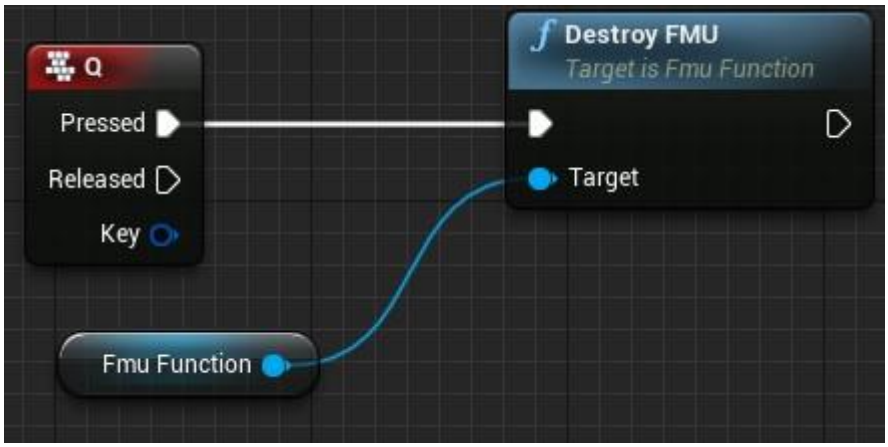
Funkce SetFMUBP má na starost incializaci. Jejími parametry pro event graf jsou FmuName(název FMU), InstanceName(nejlépe stejný název jako FmuName) a StepSize. Ostatní parametry, jako cesta k FMU, cesta k tmp složce a názvy jednotlivých proměnných v FMU, se prozatím natavují přímo v C++ kódu. Ukázku Funkce v UE4 editoru můžeme vidět na **obr. 36**.



Obrázek 36 - Ukázka funkce SetFMUBP v UE4 editoru **[XVII]**

Tato funkce je vyvolána po stisknutí klávesy F. Dokud není stisknuta klávesa F, vozidlo se nebude pohybovat. Existuje i možnost vyvolat tuto funkci při spuštění hry bez interakce s uživatelem, nicméně pro testovací účely byla tato varianta výhodnější a není ji třeba měnit. Detailní flow diagram této funkce by byl příliš rozsáhlý pro jeden obrázek, bude se s ní tedy nadále zacházet jako s celkem. Pro detailní popis je třeba projít samotný C++ kód.

Po ukončení simulace, nebo při ukončování hry je třeba FMU deinicializovat, tedy „vyčistit“ objekt, ve kterém jsou všechny informace o FMU uloženy. K tomuto účelu slouží funkce DestroyFMU, které je vyvolána na stisknutí klávesy Q. Další variantou je vyvolat tuto funkci na stisknutí klávesy, která ukončuje hru. Tato volba už je na uživateli. Ukázku funkce můžeme vidět na **obr. 37**. Co tato funkce dělá je popsáno na začátku odstavce. Později je rozšířena o funkci další, a to bude popsáno v kapitole 7.4.8.



Obrázek 37 - Ukázka funkce DestroyFMU v UE4 editoru **[XVII]**

7.4.5 Simulace v paralelním vláknu

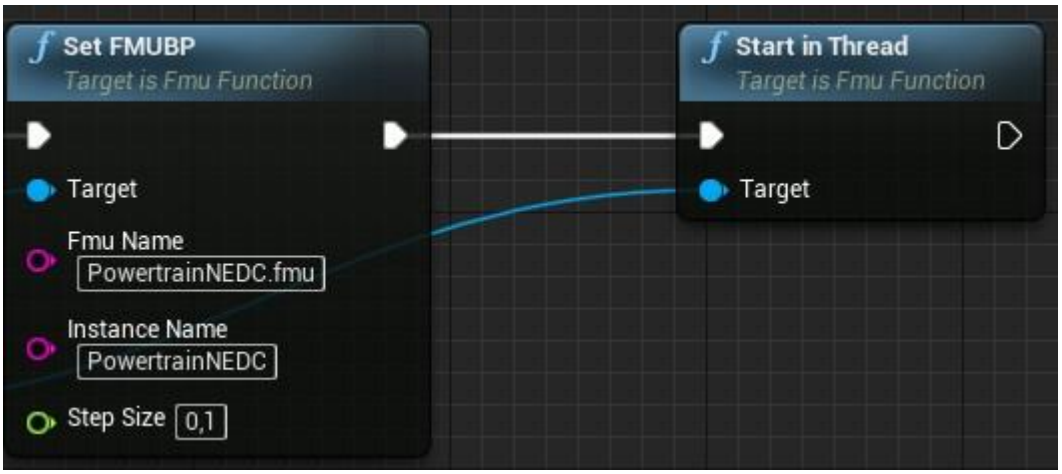
Při co-simulaci je třeba správně nastavit komunikační krok, který byl již popsán v předchozích kapitolách. Problémem je, že FPS ve hře nemají konstantní hodnotu a podle hardwarové náročnosti ve hře se mohou snižovat, či zvyšovat. To znamená, že když se nastaví step_size, která by odpovídala např. 30 snímkům za sekundu (1/30) a skutečné FPS ve hře by bylo např. 40 snímků za sekundu, čas ve hře a čas v FMU by se rozcházely. Čas v FMU by se konstantně oddaloval od času v UE4. Protože jedna sekunda v UE4 by znamenala při zmíněných hodnotách 1,34 sekundy v FMU. Simulace při takovémto chování není stabilní, a kromě špatných výsledků může také velmi často padat.

Bylo tedy třeba dosáhnout toho, aby čas v UE4 a FMU ubíhal stejně. Řešením tohoto problému je spuštění simulace v paralelním vláknu, které poběží v reálném čase nezávisle na UE4. Proto byla vytvořena funkce:

- StartInthread

Tato funkce poté na základě reálného času posouvá čas v FMU o definovaný komunikační krok. Samotná simulační funkce SimulateFMU pouze nastavuje aktuální hodnoty vstupů a získává aktuální hodnoty výstupů.

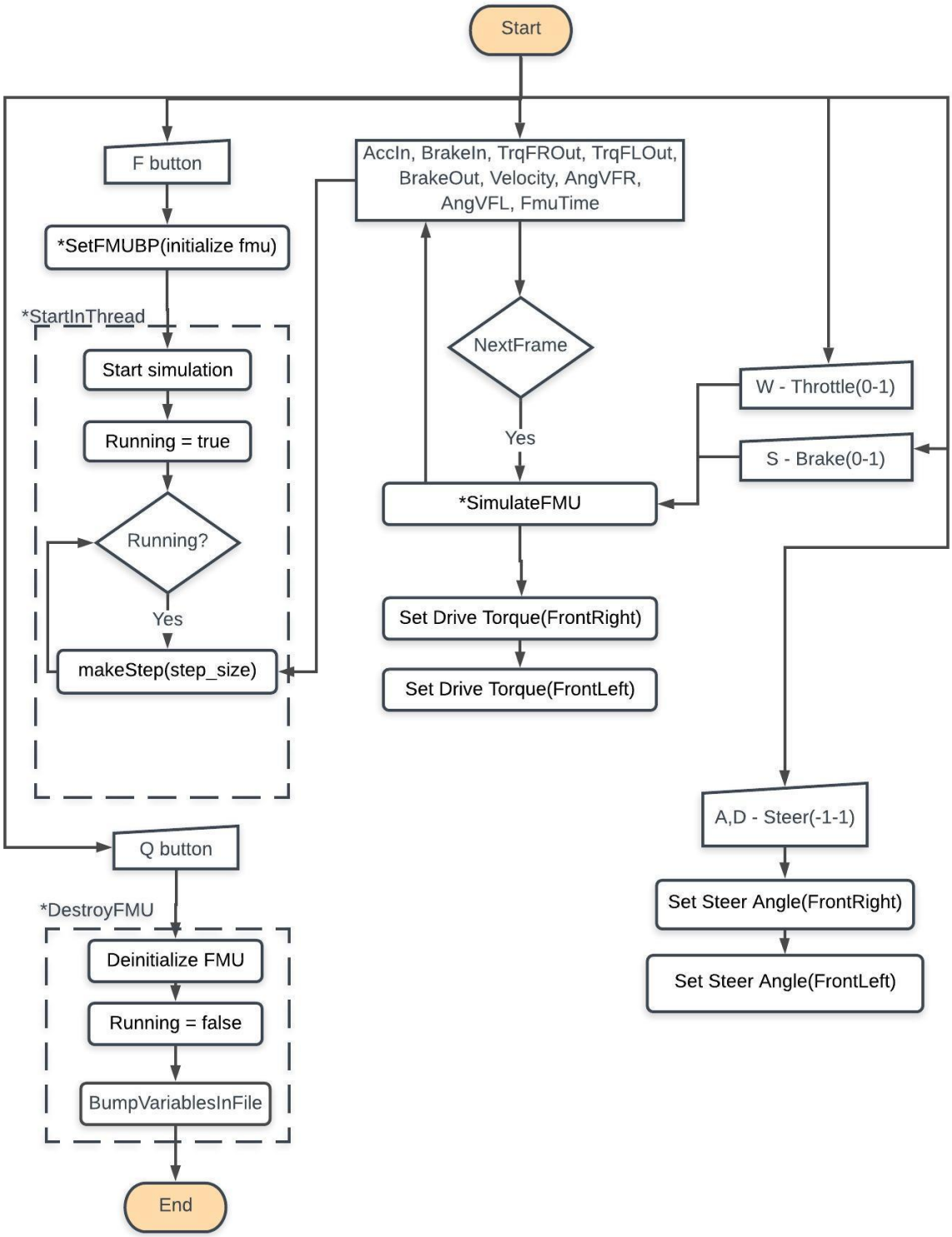
Funkce StartInThread zahajuje simulaci, je proto vyvolána hned po inicializaci FMU. Funkce DestroyFMU poté běh této funkce zastaví. Ukázka funkce společně s inicializační funkcí je na **obr. 38**.



Obrázek 38 - Ukázka funkce StartInThread v UE4 editoru **[XVII]**

7.4.6 Celkový flow diagram

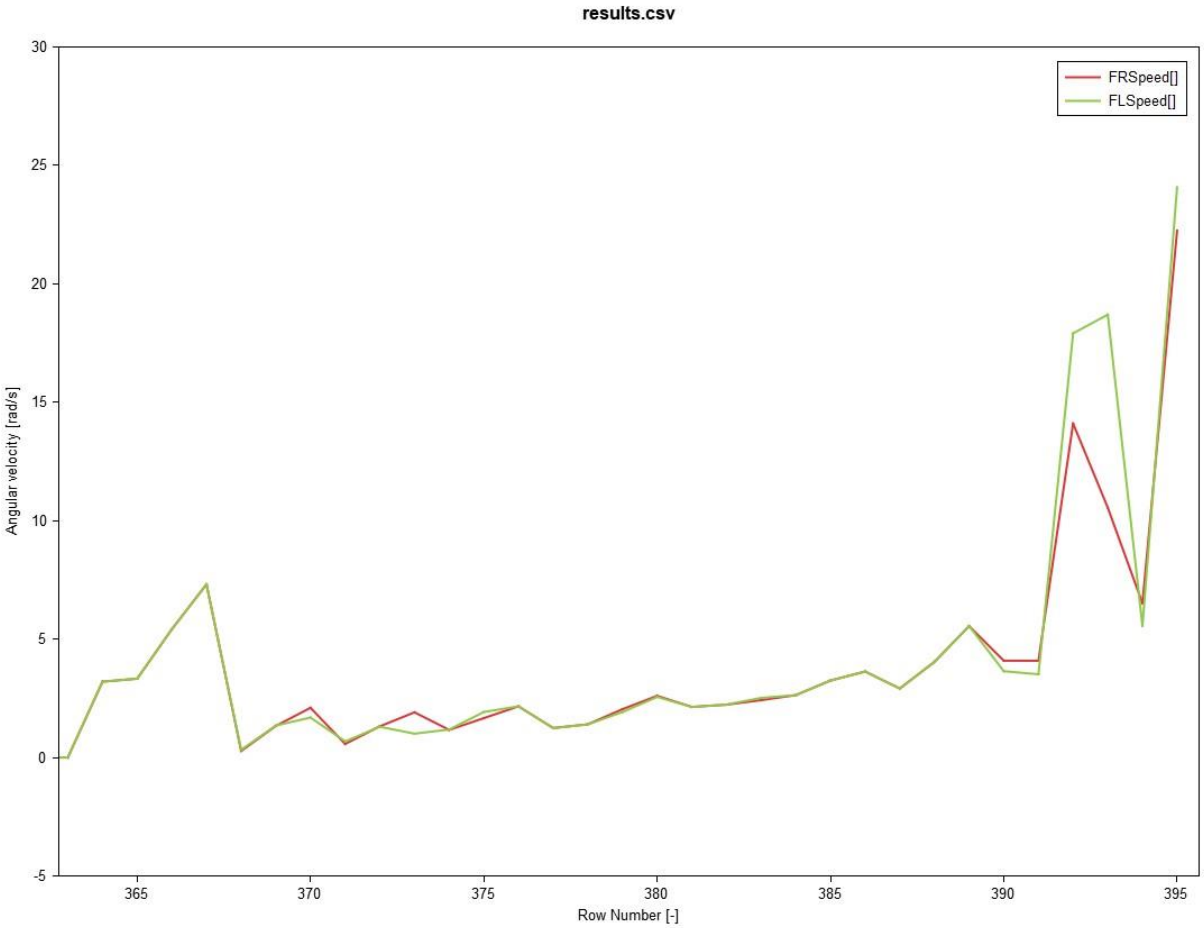
Flow diagram celého modelu v UE4 včetně FMU je na **obr. 39**.



Obrázek 39 - Flow diagram co-simulace UE4 a FMU **[XVI]**

7.4.7 Získávání okamžité úhlové rychlosti kol

Původní záměr bylo získávání úhlových rychlostí kol přímo z objektů kol v UE4, nicméně objevil se problém, který znemožňoval smysluplnou simulaci. Vstupy rychlostí příliš kmitaly (**obr. 40**), což způsobovalo, že model v FMU, který ke správnému fungování potřebuje plynulejší křivky těchto proměnných, se choval nepředvídatelně.



Obrázek 40 - Křivky vstupů úhlových rychlostí na předních kolech v rad/s **[X]**

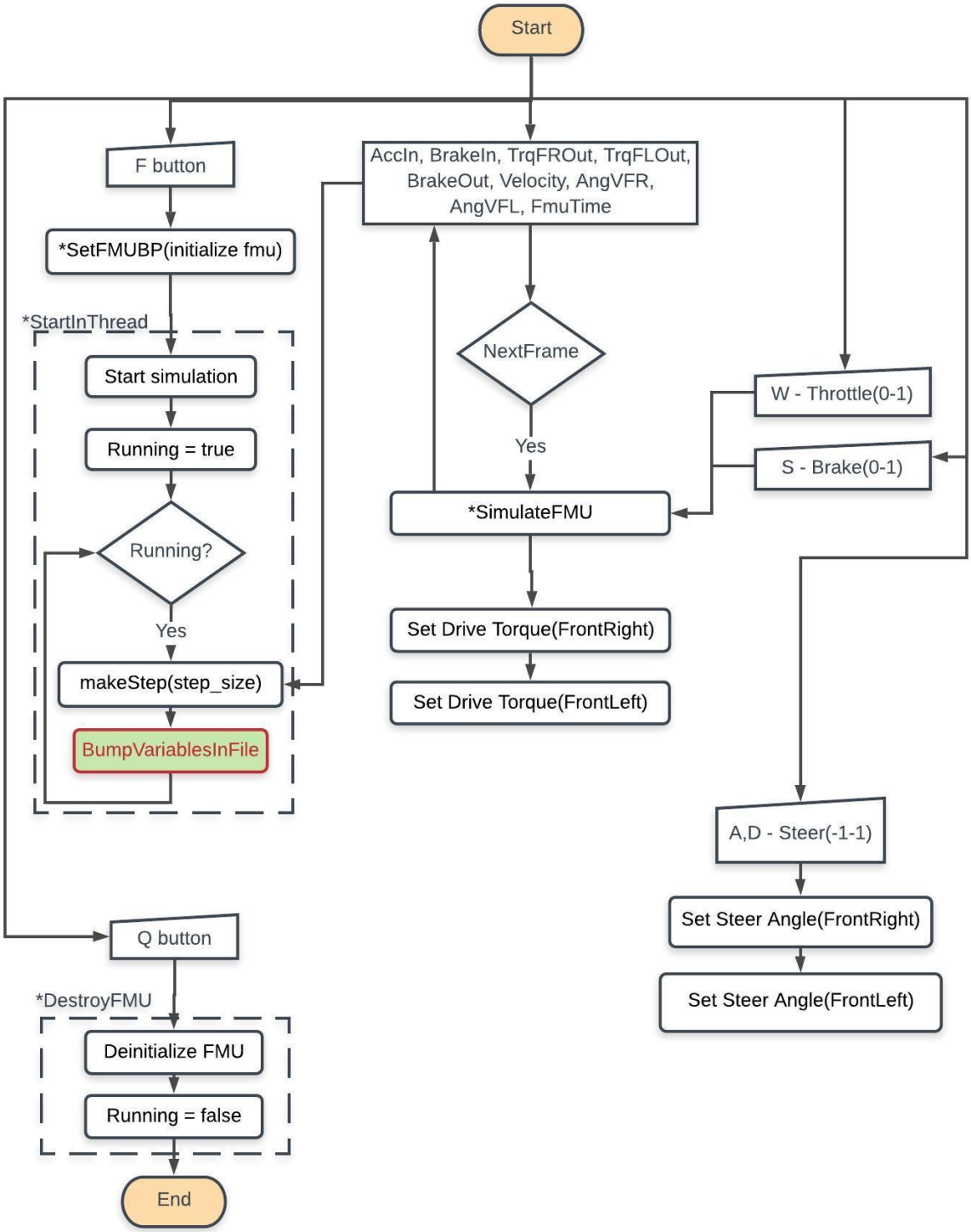
Byl tedy zvolen alternativní postup získávání úhlových rychlostí kol, a to výpočtem z celkové rychlosti vozidla, který je popsán v kapitole 7.1.1. Dalším postupem by mohla být interpolace křivek na **obr. 40**, ale tento postup nebyl v rámci této práce testován.

Po zavedení vstupů rychlostí kol na základě celkové rychlosti vozidla, se simulace začala chovat tak, jak by měla. Detailní popis výsledků je součástí kapitoly 8.

7.4.8 Zápis dat do externího souboru

Pro pozdější analýzu dat vzešlých ze simulace, je třeba data z FMU někam zapsat. Je to třeba udělat potom, co je simulace ukončena. Je proto příhodné zařadit funkci pro zápis dat k funkci DestroyFMU. Zápis dat poté proběhne po stisknutí klávesy Q.

Toto řešení je dostačující pro naše testovací účely této práce, nicméně při větším množství dat v rámci několika desítek minut až hodin, by mohl zápis selhat v důsledku nedostatečné paměti RAM. Pro takové účely by bylo třeba zápis provádět v určitých intervalech definovaných buď časově, nebo např. podle počtu snímků. Příklad implementace takového zápisu ve flow diagramu je na **obr. 41**.



Obrázek 41 - Flow diagram s potenciální změnou v zápisu dat **[XVI]**

Zapsaná data jsou ve formátu „csv“, samotný soubor se pak jmenuje „results.csv“ a je uložen ve složce tmp, jejíž cesta je definována funkcí SetFMUBP.

Data obsahují jak hodnoty definovaných parametrů, tak průběžné hodnoty proměnných v čase.

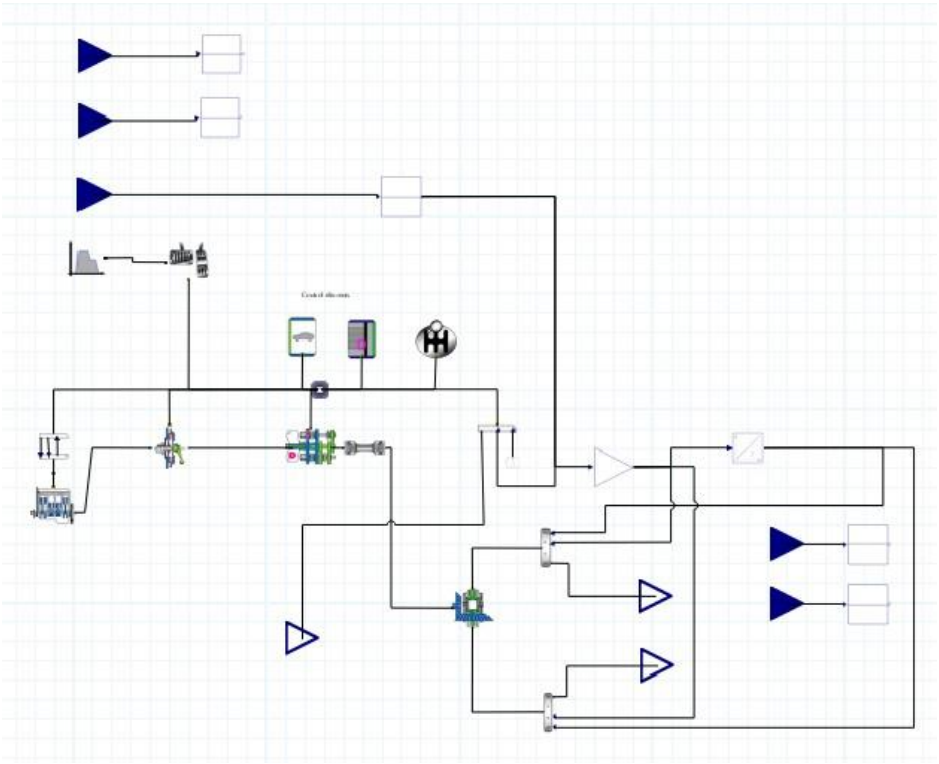
8 Výsledky implementací

Po úspěšném vytvoření fungujícího modelu FMU v UE4 je třeba výsledky určitým způsobem ověřit. Porovnat je s jinými daty a zjistit jak a v čem se liší. Jako data pro porovnání byla zvolena simulace v softwaru Ignite.

8.1 Testovací metodologie

Pro porovnání dat je třeba definovat jakým způsobem mají simulace probíhat, aby měly obě simulace přibližně stejný cíl. Existuje mnoho možností, jak naše modely testovat. Jednou z možností je testování výkonu např. při zrychlení z 0 na 100. Ovšem nejběžnějším a nejrozšířenějším způsobem jsou jízdní cykly, které se využívají i při reálném testování vozidel.

Přidání jízdního cyklu do naší co-simulace znamená, že v tomto případě není potřeba žádný vstup od řidiče v UE4 (Plyn, brzda), protože tyto vstupy budou definovány uvnitř FMU na základě jízdního cyklu. Model FMU bylo třeba tedy trochu upravit. Původní vstupy a výstupy zůstávají pro přehlednost stejné, ale nejsou v tomto případě využívány. Jediný využívaný vstup je celková rychlost vozidla, výstupy jsou točivé momenty na přední kola a míra sešlápnutí brzdového pedálu (brzdný mechanismus je sestaven vně FMU, je proto třeba použít tento výstup). Rozdíl oproti pohonnému ústrojí bez vestavěného jízdního cyklu na **obr. 21**, můžeme vidět na **obr. 42**.

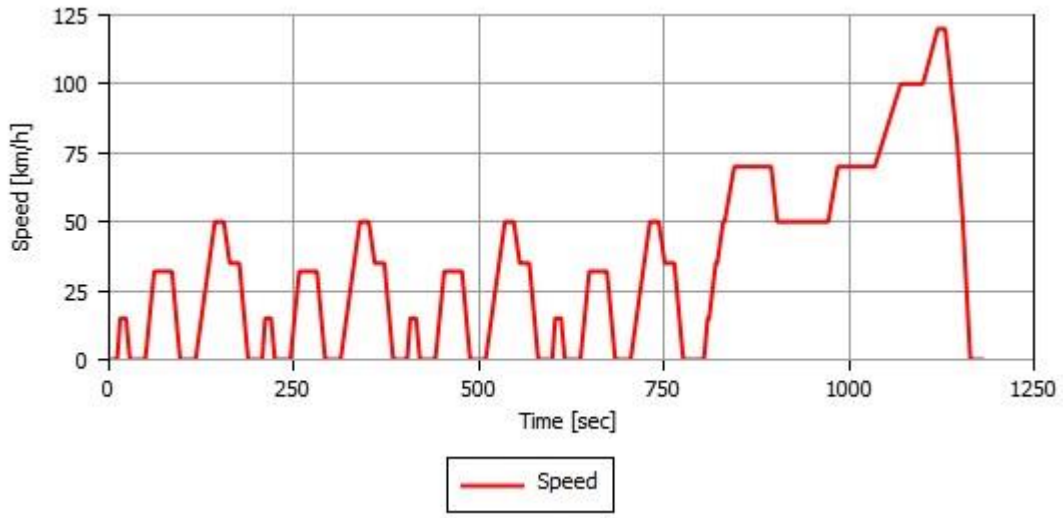


Obrázek 42 - Model pohonného ústrojí s jízdním cyklem uvnitř **[VI]**

8.1.1 Jízdní cykly

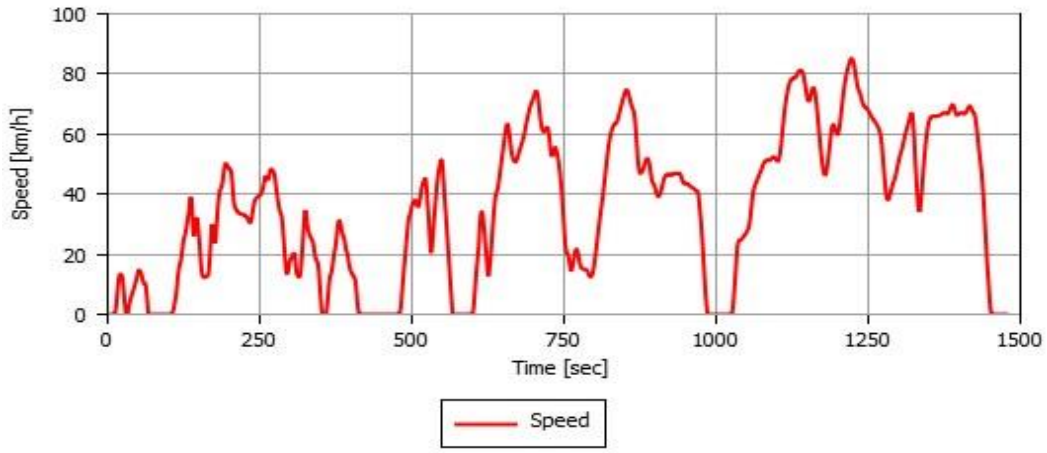
Jízdní cyklus je křivka rychlosti v čase, která definuje, jakou rychlostí se při testování má vozidlo pohybovat. Po dokončení je poté možné sledovat jakým způsobem se vozidlo chovalo a zjistit potenciální problémy daného modelu.

Takových cyklů je mnoho, ovšem nejznámější je NEDC – New European Drive Cycle, který vozidlo otestuje až při rychlosti 120 km/h. Vidět ho můžeme na **obr. 43**. Nicméně v dnešní době už se nepoužívá a byl nahrazen cyklem popsaným níže.



Obrázek 43 - Jízdní cyklus NEDC **[X]**

Další existující cykly mohou simulovat chování vozidel přímo na určitých komunikacích, např. na dálnici nebo i ve městě. V poslední době se začal využívat i cyklus WLTC – Worldwide harmonized Light vehicles Test Procedure, který je dnes standardem pro měření spotřeby paliva a emisí. Byl vyvinut k přiblížení testovacího cyklu realitě, protože cyklus NEDC je reálnému pohybu vozidla poměrně vzdálen. Na **obr. 44** je tedy vidět, že rychlost v tomto cyklu není téměř nikdy konstantní a může se skokově měnit.



Obrázek 44 - Jízdní cyklus WLTC třídy 2 **[X]**

Ovšem i přes určitá vylepšení oproti NEDC je cyklus stále poměrně vzdálen realitě. Zrychlení v něm jsou příliš pomalá pro účinné otestování výkonných, ale i některých průměrných osobních automobilů. Nejúčinnější variantou by měl být dynamický jízdní cyklus, který není předem daný, ale vytváří se na základě interakce s ostatními vozidly ve virtuální realitě.

8.1.2 Parametry

Pro správné porovnání výsledků musí mít obě simulace totožné parametry. Problémem bylo, že UE4 využívá jiný způsob výpočtů daných odporů, a proto nabízí i odlišné parametry. Model bylo proto třeba optimalizovat tak, aby se odpory v Ignitu a UE4 lišily co nejméně. Toho bylo dosaženo novým nastavením parametrů v UE4 na základě zrychlení a maximální rychlosti vozidla při konstantním točivém momentu na kolech. Parametry, které v našich modelech hrají nejvýznamnější roli jsou:

- Hmotnost vozidla = 1350 kg
- Koeficient odporu valení = 0.01
- Poloměr kola = 0.34 m
- Přední plocha vozidla = 2.2176 m^2
- Maximální brzdná síla = 3000 N
- Koeficient aerodynamického odporu = 0.3

Nicméně parametrů je mnohem více a jejich vypisování by bylo nepřehledné. Pro zjištění přesnějších informací o parametrech a jejich hodnotách je třeba prozkoumat samotné modely.

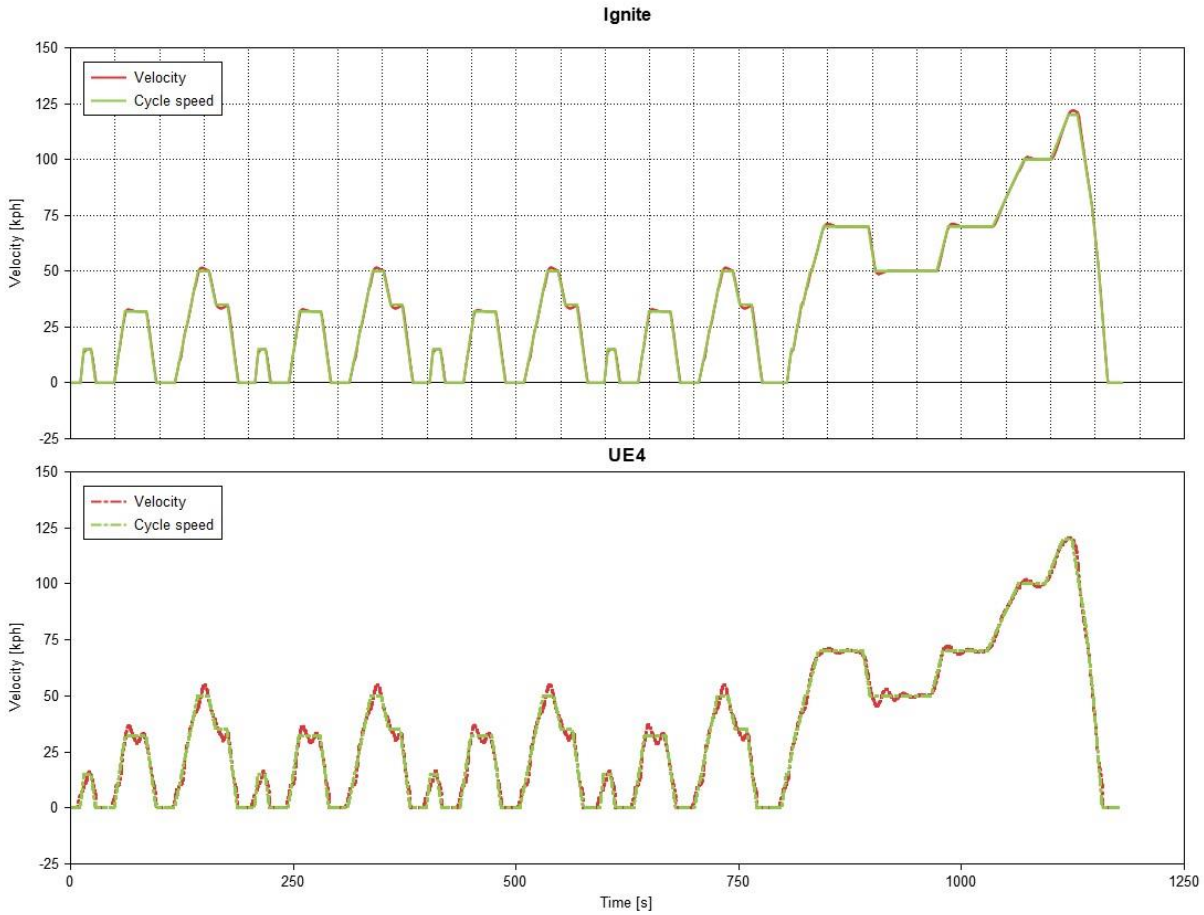
8.2 Výsledky porovnání dat

Pro účely porovnání dat byl primárně vybrán cyklus NEDC. Dalšími případnými možnostmi jsou WLTC a zrychlení z 0 na 100 km/h. Ukázka ze simulace při probíhajícím cyklu NEDC je na **obr. 45**.

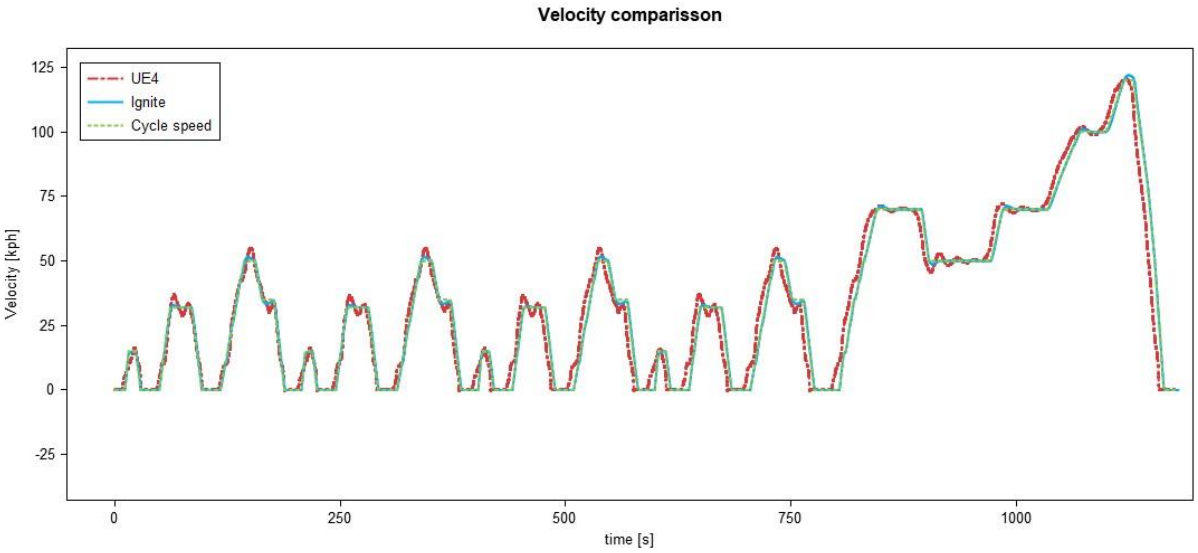
Obrázek 45 - Screenshot z UE4 při probíhajícím NEDC cyklu **[XVII]**

8.2.1 NEDC

Pro vizualizaci výsledných dat byl využit software R-Post od společnosti Ricardo. Na **obr. 46** a **obr. 47** vidíme rozdíl v rychlostech vozidla v Ignitu a vozidla v UE4.



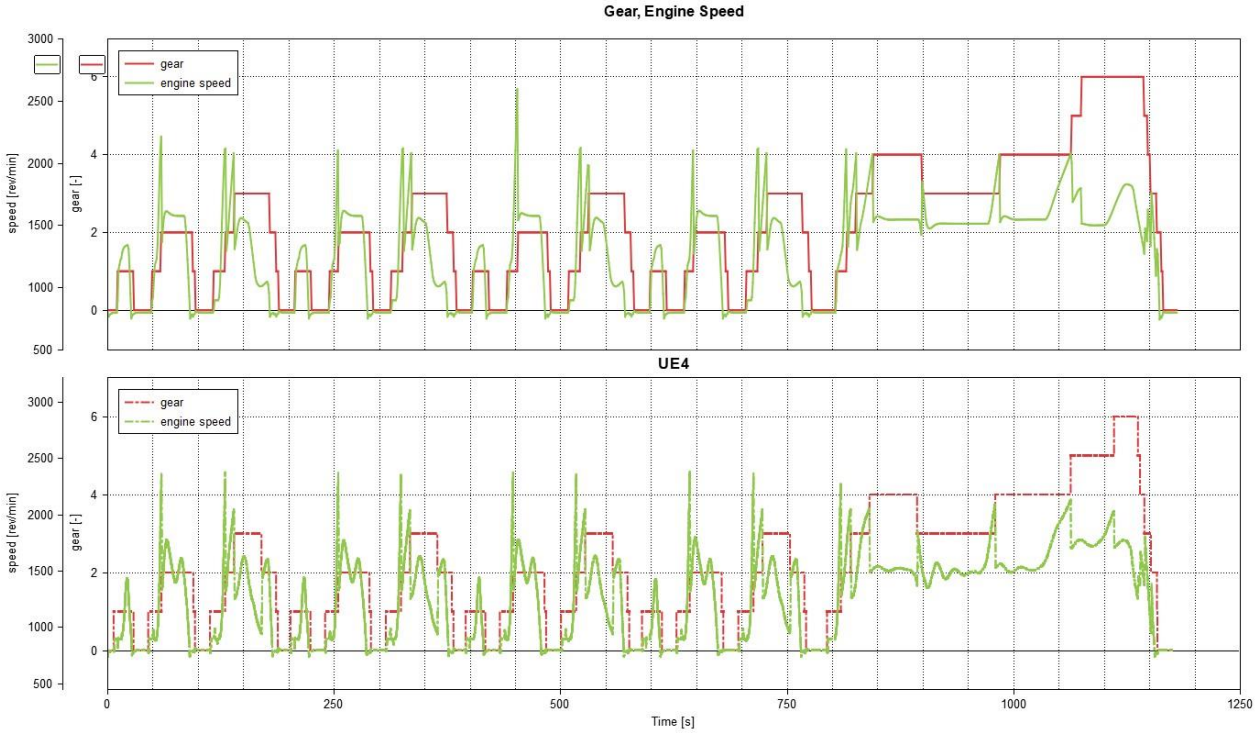
Obrázek 46 - Porovnání rychlostí **[X]**



Obrázek 47 - Porovnání rychlostí v jednom grafu **[X]**

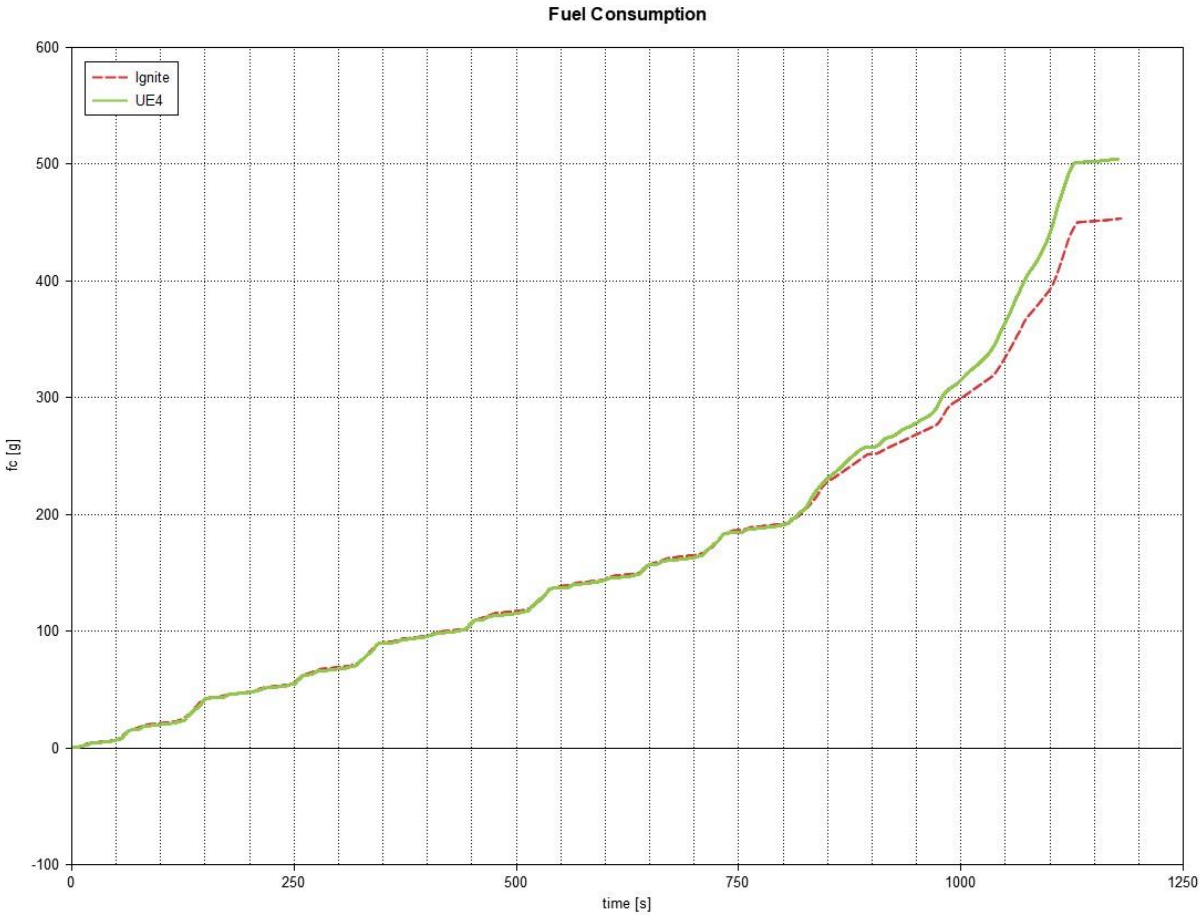
Vidíme, že výsledky z obou simulací jsou téměř totožné, nicméně v UE4 stále rychlost vozidla příliš kmitá kolem požadované rychlosti, což se může projevit na spotřebě paliva. Je to důsledek nedokonalé optimalizace parametrů nebo chyb v co-simulaci. Parametry by bylo

možné optimalizovat, ale jelikož by využití optimalizačního softwaru v Unreal Engine s FMU bylo příliš komplikované, pokud vůbec možné, musíme počítat se dříve optimalizovanými parametry z Ignitu. Nedokonalosti ve sledování předepsané rychlosti se už ale tolik neprojeví na rychlostech motoru, **obr. 48**.



Obrázek 48 - Porovnání rychlostí motoru a zařazené rychlosti **[X]**

V rychlostech motoru a zařazených stupních není vidět téměř žádný rozdíl. Obě simulace se chovají stejně, což bylo naším cílem. Největší porovnávací hodnotu má v jízdním cyklu ovšem spotřeba paliva, kterou vidíme na **obr. 49**.



Obrázek 49 - Porovnání spotřeby paliva **[X]**

Zatímco v první fázi cyklu nebyl rozdíl příliš veliký (na konci první fáze byl rozdíl cca 2g paliva, což odpovídá cca 1 % z celkové spotřeby v daný čas), v druhé fázi, která už probíhá ve vyšších rychlostech, začal rozdíl stoupat mnohem rychleji a na konci cyklu tento rozdíl vystoupal na 51 g paliva, což odpovídá 10 % z celkové spotřeby za celý cyklus.

Důvodem tohoto rozdílu ve spotřebě je v první řadě zmíněné nedokonalé sledování rychlosti jízdního cyklu. V druhé řadě je to rozdílný výpočet dynamiky vozidla v Ignitu a UE4. Parametry obou fyzikálních modelů byly sice nastaveny tak, aby dávaly stejné výsledky, ale těchto parametrů bylo docíleno na základě konstantního točivého momentu a zrychlení. V cyklu NEDC, ale točivý moment konstantní není a jeho závislost na jednotlivých parametrech není vždy lineární. Tyto nelineárnosti v dynamice vozidla tvoří druhou část rozdílu ve spotřebě. A posledním důvodem, proč se spotřeba mírně liší je co-simulace, která se v závislosti na komunikačním kroku může místy chovat jinak, než simulace jednoduchá a vytvářet menší výsledkové rozdíly.

Z porovnaných výsledků tedy vidíme, že vozidlo vozidlo v UE4 koreluje s daty z Ignitu, až na nedokonalosti popsané v předchozím odstavci. Vozidlo by potřebovalo lépe optimalizovat, což by ale pravděpodobně bylo otázkou vytvoření nového softwaru pro tyto účely.

Optimalizaci lze provádět i metodou pokus/omyl, ale v našem případě jsou parametry nastavené poměrně přesně a tato metoda by výsledky většinou jen zhoršovala.

9 Závěr

Představili jsme si důležitost dopravních simulátorů a jejich jednotlivé typy. Stejně jako přímý význam experimentů, které na těchto simulátorech probíhají.

Cílem práce bylo vytvoření fyzikálního modelu pohonného ústrojí v programovacím jazyce Modelica a implementace do grafického simulátoru nové generace. Proto jsme si popsali jeho význam, funkce a využití, stejně jako jsme si představili standard Functional Mock-up Interface, díky kterému bylo tuto práci možné realizovat.

Modelica je využívána napříč mnoha softwary. Bylo proto nutné prozkoumat některé dostupné softwary a nástroje tak, abychom na základě získaných informací mohli zvolit nejlepší variantu pro následnou práci. Zejména z důvodu knihoven vytvořených pro simulace vozidel a jejich subsystémů byl zvolen software Ignite od společnosti Ricardo.

Fyzikální model byl implementován do grafického enginu Unreal Engine 4. Důvodem pro zvolení této varianty byl poměr vysokého grafického potenciálu s celkem přívětivým uživatelským prostředím a také programovací jazyk C++.

V softwaru Ignite byl vytvořen fyzikální model pohonného ústrojí se spalovacím motorem, spojkou, převodovkou, jednotlivými řídicími komponenty a virtuálním řidičem, který pro účel testování sleduje jízdní cyklus. Tento model se dá v budoucnu jednoduše předělat pro účely simulace hybridních pohonů, elektrických pohonů, či pohonů využívajících palivové články. Protože pohonné ústrojí mělo být exportováno do Functional Mock-up Unit (FMU) a samotné vozidlo a jeho fyzikální chování mělo být v Unreal Enginu, bylo třeba mezi těmito dvěma bloky najít vhodné vstupy a výstupy, skrz které budou komunikovat v následné co-simulaci. Na základě zvolených vstupů a výstupů byl model pohonného ústrojí přizpůsoben tak, aby tuto co-simulaci umožňoval.

Nejprve bylo FMU pohonného ústrojí otestováno v co-simulaci uvnitř Ignitu, kde bylo zjištěno, že je třeba optimalizovat parametry řízení rychlosti vozidla, protože se okamžitá rychlost vozidla příliš lišila od rychlosti předepsané jízdním cyklem. Po optimalizaci těchto parametrů, za pomoci parametrické studie DOE v optimalizačním softwaru, bylo možné využít a upravit testovací program pro simulaci FMU. Úspěšné implementování a otestování FMU v tomto programu znamenalo možnost přistoupit k implementaci FMU do Unreal Enginu.

Pro model vozidla byla využita šablona, kterou Unreal Engine nabízí. Vozidlo bylo třeba upravit tak, aby mohlo využívat naše pohonné ústrojí. Ve finální fázi model v Unreal Enginu poskytuje FMU hodnoty sešlápnutí plynu a brzdy a celkovou rychlost vozidla a zároveň od FMU přijímá točivé momenty na obě poháněná přední kola s potenciální možností rozšíření až na pohon 4X4. Mezi tyto vstupy a výstupy bylo implementováno pohonné ústrojí

prostřednictvím knihoven pro FMU od společnosti Modelon a programovacího jazyka C++. Výsledkem jsou funkce využitelné uvnitř Unreal Enginu, které poskytují vše nezbytné pro simulaci FMU uvnitř grafického enginu.

Pro porovnání výsledků byl využit formát dat „csv“, který umožnil porovnání dat přímo v zobrazovacím nástroji určeném pro Ignite v R-Postu. Z porovnání dat z experimentu v UE4 a dat z validovaného modelu v Ignitu je zřejmé, že vozidlo v UE4 s naším pohonným ústrojím svým chováním koreluje s chováním vozidla v Ignitu s mírnou odchylkou, jež je důsledkem rozdílných přístupů k výpočtu a simulaci dynamiky vozidla v Unreal Enginu a Ignitu, nedokonalho sledování rychlosti jízdního cyklu v Unreal Enginu a odchylek způsobených co-simulací mezi Unreal Enginem a FMU reprezentujícím pohonné ústrojí. Další rozvoj a studie problematiky co-simulace a parametrů modelu může tuto odchylku ještě snížit. Výsledky tedy i přes mírnou odchylku představují reálná jízdní data, což znamená, že tento přístup je možné využít v simulátorech a budoucích experimentech.

V návaznosti na tuto práci je možné implementovat vytvořenou práci přímo do konkrétního simulátoru. Vytvořit podobnou implementaci pro další grafický engine jako je např. Unity. Navrhnout a realizovat sérii experimentů, která bude využívat rozdílná pohonná ústrojí vytvořená jako FMU a vyhodnotit jejich výsledky. Na jejich základě poté navrhnout další potenciální úpravy modelu a metodiky pro testování.

10 Použité zdroje

10.1 Citované zdroje

[1] *The Modelica Association* [online]. [cit. 30.3.2019]. Dostupné na: https://www.modelica.org/

[2] *Modelica Language* [online]. [cit. 30.3.2019]. Dostupné na: https://www.modelica.org/modelicalanguage

[3] *Functional Mock-up Interface* [online]. [cit. 30.3.2019]. Dostupné na: https://fmi-standard.org/

[4] HENNINGSSON, Maria. *What is FMI?* [online]. 10.10.2016, [cit. 2.4.2019]. Dostupné na: https://www.modelon.com/what-is-fmi/

[5] ELMQVIST, Hilding. *Modelica Evolution - From My Perspective* [online]. 3.10.2014, [cit. 5.4.2019]. ISBN 978-91-7519-380-9. Dostupné na: http://www.ep.liu.se/ecp/article.asp?issue=96&article=1

[6] *Welcome to OpenModelica* [online]. [cit. 2.4.2019]. Dostupné na: https://openmodelica.org/

[7] *MapleSim Features* [online]. [cit. 2.4.2019]. Dostupné na: https://www.maplesoft.com/products/maplesim/features/

[8] Computer Hope, *What is DirectX?* [online]. 15.9.2017, [cit. 2.4.2019]. Dostupné na: https://www.computerhope.com/jargon/d/directx.htm

[9] *OpenGL Overview* [online]. [cit. 2.4.2019]. Dostupné na: https://www.opengl.org/about/

[10] *PhysX FAQ|NVIDIA* [online]. [cit. 5.4.2019]. Dostupné na: https://www.nvidia.com/object/physx_faq.html

[11] *Unreal Engine | Frequently asked questions* [online]. [cit. 5.4.2019]. Dostupné na: https://www.unrealengine.com/en-US/faq

[12] JEGX, *GeeXLab 0.25.0 released for all platforms – Bullet Physics support added* [online]. 7.6.2018, [cit. 5.4.2019]. Dostupné na:

https://www.geeks3d.com/hacklab/20180607/geexlab-0-25-0-released-for-all-platforms-bullet-physics-support-added/

[13] *GitHub – cocos2d/cocs2d-x* [online]. [cit. 5.4.2019]. Dostupné na: https://github.com/cocos2d/cocos2d-x

[14] *odedevs/ode - Bitbucket* [online]. [cit. 5.4.2019]. Dostupné na: https://bitbucket.org/odedevs/ode/

[15] *FMI Library: part of JModelica.org* [online]. 6.11.2019, [cit. 5.4.2019]. Dostupné na: https://jmodelica.org/fmil/FMILibrary-2.0.3-htmldoc/index.html

10.2 Zdroje k obrázkům

[I] *VR 67: Simulátor velkých vozidel* [online]. [cit. 30.3.2019]. Dostupné na: https://www.cdvplus.cz/vr-67-simulator-velkych-vozidel

[II] Dekra. *Vyzkoušeli jsme nový simulátor tahačů a autobusů* [online]. 11.5.2016, [cit. 30.3.2019]. Dostupné na: https://www.mmspektrum.com/navsteva/vyzkouseli-jsme-novy-simulator-tahacu-a-autobusu.html

[III] Bouchner, P., & Novotný, S. *D|S|R|G*. Praha: ČVUT Fakulta dopravní.

[IV] *Functional Mock-up Interface* [online]. [cit. 30.3.2019]. Dostupné na: https://fmi-standard.org/

[V] vlastní screenshot 7zip

[VI] Screenshot Ignite, Ricardo

[VII] *Academia - Claytex* [online]. [cit. 2.4.2019]. Dostupné na: https://www.claytex.com/sectors/academia/

[VIII] *Welcome to OpenModelica* [online]. [cit. 2.4.2019]. Dostupné na: https://openmodelica.org/

[IX] *Calculation Management Environment* [online]. [cit. 2.4.2019]. Dostupné na: https://www.maplesoft.com/products/maple/professional/Calculation-Management.aspx

[X] Screenshot R-Post, Ricardo

[XI] NVIDIA. Video: *PlanetSide 2 PhysX* Trailer [online]. 21.3.2013, [cit. 5.4.2019]. Dostupné na: https://www.youtube.com/watch?time_continue=73&v=n5qhaEghJ74

[XII] LEARY, Michael. *More Gameplay Images of GTR 3 Released* [online]. 5.2.2017, [cit. 5.4.2019]. Dostupné na: https://www.gtplanet.net/more-gameplay-images-of-gtr-3-released/

[XIII] Sports Gamers Online. Video: *NASCAR Heat 3 Gameplay Exclusive First Look* [online]. 23.8.2018, [cit. 5.4.2019]. Dostupné na: https://www.youtube.com/watch?v=k9FqM_jDP4g

[XIV] *Bullet Real-Time Physics Simulation* [online]. 30.3.2019, [cit. 5.4.2019]. Dostupné na: https://pybullet.org/wordpress/

[XV] screenshot ze simulátoru Fakulty Dopravní, Horská

[XVI] *Online Diagram Software & Visual Solution* [online]. Dostupné na: www.lucidchart.com

[XVII] Screenshot z Unreal Engine 4

11 Appendix

[16] ROZHDESTVENSKIY, Dmitry. FULEM, Josef. *Simulation of electric and hybrid vehicles in a vehicle simulator based on a detailed physical model, for the purpose of hmi evaluation* [online]. Prosinec 2017, [cit. 5.4.2019]. DOI: 10.14311/APP.2017.12.0094. Dostupné na: http://www.ep.liu.se/ecp/article.asp?issue=96&article=1

[17] *Tools|Functional Mock-up Interface* [online]. [cit. 23.5.2019]. Dostupné na: https://fmi-standard.org/tools/

12 Seznam obrázků

Obrázek 1 - Simulátor osobního automobilu – Dopravní VaV Centrum **[I]**

Obrázek 2 - Promítací plochy simulátoru **[II]**

Obrázek 3 - Typy simulátorů **[III]**

Obrázek 4 - Ilustrace struktury FMI **[IV]**

Obrázek 5 - Struktura FMU **[V]**

Obrázek 6 - Příklad modelu fyzikálního kyvadla **[VI]**

Obrázek 7 - Příklad modelu s využitím jazyka Modelica **[VI]**

Obrázek 8 - Příklad modelu systému chlazení **[VI]**

Obrázek 9 - Ukázka prostředí softwaru Dymola **[VII]**

Obrázek 10 - Ukázka prostředí OpenModelica **[VIII]**

Obrázek 11 - Ukázka nástrojů společnosti MapleSoft **[IX]**

Obrázek 12 - Ukázka prostředí softwaru Ignite **[VI]**

Obrázek 13 - Ukázka prostředí nástroje R-Post od společnosti Ricardo **[X]**

Obrázek 14 - Efekt výbuchu ve hře PlanetSide 2 s a bez PhysX **[XI]**

Obrázek 15 - Ukázka ze hry GTR 3 **[XII]**

Obrázek 16 - Ukázka ze hry Nascar Heat 3 **[XIII]**

Obrázek 17 - Využití Bullet **[XIV]**

Obrázek 18 - Ukázka ze simulátoru s ODE a OpenGL **[XV]**

Obrázek 19 - Diagram simulátoru s UE4 a FMU **[XVI]**

Obrázek 20 - Vstupy a výstupy modelu **[VI]**

Obrázek 21 - Model pohonného ústrojí **[VI]**

Obrázek 22 - Charakteristika a spotřeba spalovací motoru **[VI]**

Obrázek 23 - Strategie řazení v komponentu ShiftStrategy **[VI]**

Obrázek 24 - Struktura vyexportovaného FMU **[XVI]**

Obrázek 25 - Testovací model pro co-simulaci FMU **[VI]**

Obrázek 26 - Rychlost vozidla v co-simulaci se špatnými parametry řidiče **[X]**

Obrázek 27 - Původní a nové parametry řidiče **[VI]**

Obrázek 28 - Rychlost vozidla v co-simulaci s novými parametry řidiče **[X]**

Obrázek 29 - testovací nastavení powertrain.ini, screenshot

Obrázek 30 - Flow diagram testovacího simulátoru **[XVI]**

Obrázek 31 - Ukázka z Blueprintu vozidla v UE4 **[XVII]**

Obrázek 32 - UE4 ukázka z event grafu vozidla - aplikace točivého momentu **[XVII]**

Obrázek 33 - Komponenty vozidla v UE4 včetně FmuFunction **[XVII]**

Obrázek 34 - Ukázka funkce SimulateFMU v UE4 editoru **[XVII]**

Obrázek 35 - Flow diagram funkce SimulateFMU **[XVI]**

Obrázek 36 - Ukázka funkce SetFMUBP v UE4 editoru **[XVII]**

Obrázek 37 - Ukázka funkce DestroyFMU v UE4 editoru **[XVII]**

Obrázek 38 - Ukázka funkce StartInThread v UE4 editoru **[XVII]**

Obrázek 39 - Flow diagram co-simulace UE4 a FMU **[XVI]**

Obrázek 40 - Křivky vstupů úhlových rychlostí na předních kolech v rad/s **[X]**

Obrázek 41 - Flow diagram s potenciální změnou v zápisu dat **[XVI]**

Obrázek 42 - Model pohonného ústrojí s jízdním cyklem uvnitř **[VI]**

Obrázek 43 - Jízdní cyklus NEDC **[X]**

Obrázek 44 - Jízdní cyklus WLTC třídy 2 **[X]**

Obrázek 45 - Screenshot z UE4 při probíhajícím NEDC cyklu **[XVII]**

Obrázek 46 - Porovnání rychlostí **[X]**

Obrázek 47 - Porovnání rychlostí v jednom grafu **[X]**

Obrázek 48 - Porovnání rychlostí motoru a zařazené rychlosti **[X]**

Obrázek 49 - Porovnání spotřeby paliva **[X]**

13 Seznam příloh

Celý UE4 projekt včetně modelů v Ignitu je k dispozici na:
https://bitbucket.org/Matt4077/vehiclefmu/src/master/

Pro správné sestavení projektu je třeba přečíst readme.txt ve složce VehicleBlueprint.

Obsah CD

Repositář projektu	Repositář se všemi potřebnými soubory k sestavení projektu v Unreal Enginu a Visual Studiu včetně všech Ignite modelů