TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Robin Teaste 222852

FS Team Tallinna isejuhtiva vormeli simulaatori arendamine

Bakalaureusetöö

Juhendaja: Gert Kanter

PhD

Tallinn 2025

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Robin Teaste

11.06.2025

Annotatsioon

Tudeng vormel on üle maailmne võistlussari kus meeskondade ülesanne on ehitada kõige kiirem masin vastavalt võistlusarja reeglitele. FS Team Tallinn on TalTechi ja Tallinna Tehnikakõrgkooli ühine meeskond.

Simulaator on tähtis abitööriist isejuhtiva süsteemi arendamise jaoks, võimaldades testida uut koodi riskivabalt ning kordades kiiremini kui päriselus. Simulaator võimaldab testida kontrolli, trajektoori planeerimist ja SLAM-i.

Simulaatori loomise käigus tuli luua vormeli füüsikaline mudel ja ka palju lisa funktsionaalsusi isejuhtiva süsteemi testimise abistamiseks.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 24 leheküljel, 5 peatükki, 15 joonist.

Abstract

Developing a Self-Driving Formula Car Simulator for FS Team Tallinn

Student formula is a worldwide competition where the teams compete to build the fastest formula-style car according to competition rules. FS Team Tallinn is a joint team between TalTech and Tallinn Tehnikakõrgkool.

Simulator is an important tool for developing the self driving system, it allows testing new code riskfree and is many times faster than testing in real life. Simulator allows the testing of control, path planning and SLAM.

During the development of the simulator a physics model of the vehicle had to be made and a lot of extra functionality was implemented to make the simulator more useful.

The thesis is in Estonian and contains 24 pages of text, 5 chapters, 15 figures.

Lühendite ja mõistete sõnastik

FS	Tudeng vormel (*Formula Student*)
GSS	Maakiirusandur (*Ground Speed Sensor*
IMU	Inertsiaalandur (*Inertial Measurement Unit*)
LIDAR	Valguse tuvastamine ja ulatuse määramine (*Light Detection And Ranging*)
RK4	Fourth order Runge-Kutta method
SAE	Autoinseneride Selts(*Society of Automotive Engineers*)
SLAM	Samaaegne lokaliseerimine ja kaardistamine (*Simultaneous Localization And Mapping*)
UE4	Unreal Engine 4

Sisukord

Jooniste loetelu

1 Sissejuhatus

Tudeng vormel sari sai alguse aastal 1981 Ameerika Ühendriikides kui see loodi autoinseneride seltsi poolt (SAE). Üle aastate muutus võistlussari järjest populaarsemaks ning sarjas osalevate meeskondade arv kasvas. Aastal 2010 lisandusid võistlussarja 3 korraldaja riiki: Jaapan, Saksamaa ja Austria. Tänapäevaks on lisandunud veel riike kus korraldatakse tudengvormeli võistluseid ning paljud neist kasutavad Saksamaa reegleid.

FS Team Tallinn sai alguse aastal aastal 2006 ning loodi Tallinna Tehnikaülikooli ja Tallinna Tehnikakõrgkooli ühise meeskonnana. Aastal 2013 loodi meeskonna esimene elektrimootoriga vormel. Aastal 2018 hakkas meeskond arendama isejuhtivat süsteemi.

Isejuhtiva meeskond arendab isejuhtiva vormeli süsteemi kasutades roboti operatsioonisüsteemi ROS noetic[1]. Isejuhtiv süsteem koosneb mitmest komponendist kuid peamised neist on: Nägemine, SLAM, trajektoori planeerimine ja kontroll. Nägemine kasutab LIDARit, et tuvastada koonuseid vormeli läheduses. SLAM kaardistab nähtud koonuseid ning arvutab vormeli asukohta. Path planning leiab trajektoori koonuste vahel mida mööda vormel peaks sõitma. Kontroll juhib vormelit, et vormel sõidaks trajektoori järgi.

Isejuhtiva süsteemi arendamise lihtustamiseks vajab meeskond simulaatorit. Simulaator võimaldab testida kas erinevaid isejuhtiva süsteemi komponente eraldi või kõiki koos. Simulaator kiirendaks arendamise protsessi massiivselt kuna kõik algelised testid ning paljud parameetrite optimiseerimised saab teha otse simulaatoris.

Simulaator peab olema võimalikult realistlik. Selle jaoks on vaja realistlikku füüsikalist mudelit kui ka realistlikku sensoorikat. Isejuhtiva vormeli sensoorika hulka kuuluvad: intersiaalandur (IMU), maakiirusandur (GSS), LIDAR, ratta enkoodrid ja roolinurk. Plaanis on ka kasutada päris vormeli juhtsüsteemi simulaatoris.

2 Analüüs

FS Team Tallinna isejuhtiva alammeeskond kasutab simulaatorit isejuhtiva süsteemi komponentide arendamiseks ja valideerimiseks. Simulaator teeb arendusprotsessi kiiremaks ning aitab teha isejuhtivat süsteemi robustsemaks, leides paljud vead üles ennem päris elus testimist.

Isejuhtiv süsteem koosneb mitmest komponendist (joonis 1). Simulaator peab esialgselt toetama järgnevate komponentide testimist.

- Control
- Trajectory planning
- Estimation

Perception on jäetud esialgsete seast välja kuna LIDARi simulatsioon realistlikult on väga keerukas protsess. Kuna LIDARit mõjutavad väga paljud muutujad siis meeskond kasutab nägemise arendamiseks kas päris elus testimist või ROS bage.

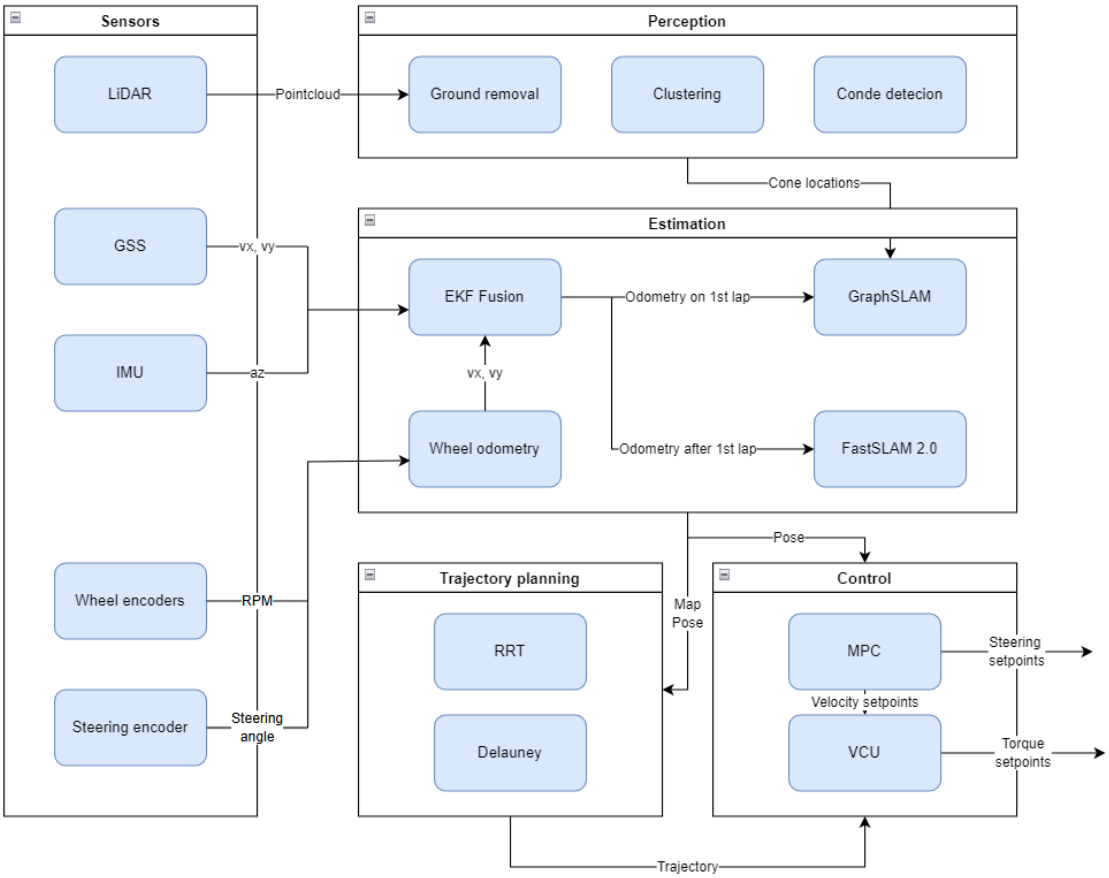
Simulaatorit saab kas luua ise algusest või ehitada olemasoleva projekti peale. Selleks tuleb uurida teisi simulaatoreid ning muid meetodeid millega saaks teha hea simulaatori.

2.1 Simulaatori arendamise algus

2.1.1 Teised tudeng vormeli simulaatorid

FS Team Tallinnal on eelmisest aastast alles simulaator mis on töötab Linuxi operatsioonsüsteemi peal. Vana simulaator on loodud gazebo keskkonda ning kasutab jalgratta mudelit. See toetab ainult kontrolli testimist ning sellega on mitmeid probleeme:

- Kiiruse sisend ei ole kiirus meerites sekundis
- Vormeli mudeli nägemiseks on vaja lisa käske
- Trajektoori sisse laadimine on liigselt keerukas protsess



Joonis 1. Isejuhtiva süsteemi skeem

Kuna vana simulaator on disainitud maast üles kontrolli testimiseks võib muutuda teiste vajalike süsteemide lisamine keerukaks.

Üle maailma on olemas mitmeid avalikult kättesaadavaid simulaatoreid. Neist üks populaarseim on fssim [2]. Fssim on arendatud AMZ meeskonna poolt kes on maailmas üks tipp meeskondadest tudeng vormeli sarjas. Simulaator kasutab samuti gazebo keskkonda, kuid füüsika on lahendatud kasutades lihtsustatud vormeli mudelit ning euleri meetodit integreerimiseks. Fssim kasutab LIDARi asemel nägemiskoonust, mis tagastab kõik koonuse sees paiknevad koonused. Simulaatoris on ka mitmeid muid funktsionaalsusi, nagu raja ringide lugemine, automaatne testimine ja logide salvestamine. Fssim on siiski vana, projekt sai alguse aastal 2018 ning loodi algselt ubuntu 16.04 ja ROS kinetic jaoks.

Veel üks simulaator on Formula-Student-Driverless-Simulator (FSDS[3]). Simulaator on ehitatud üles Unreal Engine 4 (UE4[4]) mootori peal ning kasutab füüsika jaoks AirSim[5]. Simulaatorit on ka kasutatud isejuhtivate vormelite võistluseks aastal 2020. FSDS on edasiarendus projektist FSTDriverless[6]. FSTDriverlessi projekti on ka kasutatud imitatsiooniõppega trajektoori planeerimise jaoks, kasutades simulaatoris treenitud kaamera pilti.

2.1.2 Muud mootorid

CARLA[7] on isejuhtivate autode arendamiseks loodud avatud lähtekoodiga simulaator. CARLA põhineb Unreal Engine'il ning pakub realistlikku keskkonda koos dünaamilise liiklusega isesõitvate autode testimiseks. CARLA kasutab ka realistlike andurite ja LIDARite simulatsioone ning toetab kogu isejuhtiva auto süsteemi testimist. Kuna CARLA fookus on isejuhtiva süsteemi arendamine liikluses sõitmiseks siis võivad olla paljud keskkonna funktsioonid üleliigsed vormeli jaoks.

Unreal Engine 4 on populaarne mängumootor mis ise ei toeta ROS-i, kuid sellele on loodud palju pluginaid mis lisavad ROS toe, üks populaarseim neist on ROSIntegration[8]. UE4 programmeerimiskeel on C++, kuid see toetab ka Unreal Blueprint süsteemi, mis võimaldab kirjutada koodiblokke ning hiljem ühendada kõik kokku vooskeemidega. UE4 toetab ka substep süsteemi, ehk füüsika sagedus on sõltumatu kaadrisagedusest, see tagab, et füüsika püsib ühtlasena ka aeglasematel arvutitel.

Unity[9] on ka populaarne mängumootor millele on samuti loodud pluginaid mis lisavad ROS toe, üks neist on Unity-Robotics-Hub[10]. Unity programmeerimiskeel on C#. Unity toetab samuti substep süsteemi. Unity on sarnane UE4-le kuid see pole nii levinud isejuhtivate autode alal.

Simulaatori loomiseks valisin Unreal Engine 4. Otsuse põhjusteks on, et simulaatori saab luua täpselt meeskonna vajadustele ning vormeli dünaamika saab luua nii realistliku kui võimalik kasutades meeskonna siseseid vormeli dünaamika andmeid. Lisaks aitas UE4 valimisele kaasa eelnev kogemus C++ keelega.

2.2 Füüsikalise mudeli valik

Vormeli füüsika modelleerimiseks on palju erinevaid valikuid. Kõige enimlevinud mudelid on jalgratta mudelid. Jalgratta mudelite kõige suuremaks eeliseks on nende lihtsus. Jalgratta mudelid jagunevad omakorda edasi kinemaatiliseks ja dünaamiliseks. Kinemaatiline jalgratta mudel lihtsustab auto neli ratast kaheks, läbi selle on võimalik kiiresti leida auto liikumissuund ja pöördenurk vastavalt esiratta nurgale. Kinemaatiline mudel püsib aga lähedal reaalsusele ainult madalatel kiirustel, kuna see jätab arvestamata rehvi jõudude limiidid. Dünaamiline jalgratta mudel vähendab neid probleeme, arvestades rehvi jõudude piire, kuid on seetõttu keerukam.

Füüsika modelleerimiseks on ka võimalik kasutada jõudude põhist lähenemist. Simuleerides igat ratast on võimalik leida, mis jõudu need rakendavad vormelile ning integreerides neid jõude, saab realistlikult simuleerida vormeli liikumist. Füüsikaline lähenemine hõlmab kogu vedrustuse simuleerimist ning võimaldab ka simuleerida kaalu nihkumist kiirenduste tõttu.

Simulaatoris vormeli modelleerimiseks valisin jõudude põhise lähenemise, kuna meetod võimaldab arvestada iga väiksema jõuga. Selle jaoks on plaanis kasutada mitmeid erinevaid allikaid, neist peamine "The Physics of Racing"[11], mis kirjeldab võidusõidu auto füüsikat ning potentsiaalseid simuleerimis tehnikaid. Integreerimiseks plaanin kasutada Runge-Kutta 4 meetodit mille abiks tuleb artikkel "How to Solve ODEs in MATLAB Using Runge-Kutta"[12]. Runge-Kutta 4 on keerukam kui Euleri meetod kuid on palju stabiilsem.

3 Simulaator

Simulaator on üles ehitatud *Unreal Engine 4* mängumootoril. Füüsikasüsteem kasutab Unreal Engine'i substepping võimalust, mille abil jagatakse iga kaader väiksemateks füüsika sammudeks. Füüsika väiksemateks sammudeks jagamine aitab kaasa simulatsiooni stabiilsusele ning vähendab ressursi nõudlikkust. Füüsika sihtsageduseks on seatud 200 Hz. 200 Hz on hea kesktee täpsuse ja ressursi nõudlikkuse vahel.

Simulaatorisse on integreeritud ka vormeli juhtsüsteem, mille töösagedus on samuti 200 Hz. Juhtsüsteem kasutab siseselt fikseeritud ajasammu $\Delta t = 0.005\,\mathrm{s}$. Kuna simulaatori füüsika sagedus kattub juhtsüsteemi nõutud sagedusega, saab juhtsüsteemi kutsuda välja igal füüsikasammul, mis lihtsustab juhtsüsteemi ja füüsika sünkroniseerimist.

3.1 Füüsikaline mudel

Vormeli füüsikaline mudel peab tagama võimalikult realistliku käitumise ning samal ajal säilitama simulatsiooni stabiilsuse. Füüsika sammuks valiti 200 Hz, et see kattuks juhtsüsteemi töösagedusega. Teiseks peab füüsika sagedus olema kõrgem kui vormeli vedrustuse sagedus, et vedrustuse simulatsioon püsiks stabiilsena. Juhul kui füüsika sagedus on liiga väike võib tekkida situatsioon kus jõud ei koondu ning vormel võib liikuda erraatiliselt.

Simulatsiooni stabiilsuse ja täpsuse tagamiseks kasutatakse Runge–Kutta 4. järku integreerimismeetodit (RK4), mis on stabiilsem kui Euleri meetod, kuid ei tee simulaatorit liiga ressursinõudlikuks. Kogu vormeli füüsikaline mudel on implementeeritud C++ keeles Unreal Engine'i keskkonnas.

Üheks simulaatori nõudeks on võimekus tuvastada, kui vormel sõidab vastu koonust. Selle saavutamiseks käsitletakse vormelit mängumaailmas lihtsustatud kujul kui kolmemõõtmelist kasti, mis kasutab Unreal Engine'i kokkupõrkesüsteemi. Lisaks salvestatakse ka kõik tabatud koonused ühte jatta, et hoida logi.

Kõik vormeli liikumiskiirused ja pöördemomendid arvutatakse isekirjutatud füüsikamudelis. Unreal Engine'i sisemisi jõudude arvutusi ei kasutata, kuna need sisaldavad ebarealistlikke jõude ning ei võimalda täielikku kontrolli füüsika üle. Hübriidne lähenemine, kus füüsika toimub isekirjutatud koodiga, kuid kokkupõrked käsitletakse mängumootori vahendusel, tagab nii füüsikalise kontrolli kui ka korrektse interaktsiooni mängumaailma objektidega.

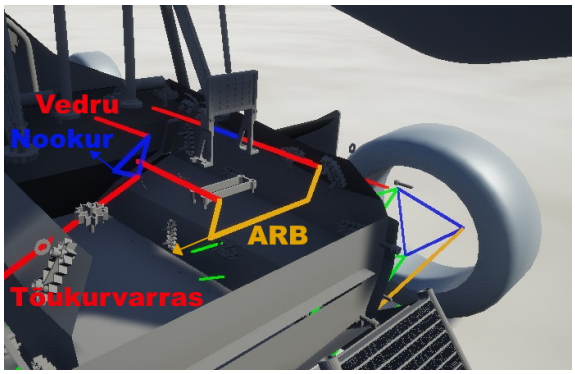
3.1.1 Vedrustus

Vedrustus on füüsikalise mudeli üks tähtsaim komponent, see määrab kuidas rattad on kontaktis maaga ning leiab jõud mis mõjuvad vormelile antud hetkel. FEST24 kasutab ees ja taga topelt õõtshoovadega, tõukurvardaga ning U-kang stabilisaatorvardaga vedrustust. Simulatsiooni jaoks kasutan vedrustuse algpositsiooni punkte ning loon nendest vedrustuse kinemaatilised seosed. Kinemaatilised seosed võimaldavad leida kogu vedrustuse kõik punktid teades ainult vedru pikkust ja roolinurka.

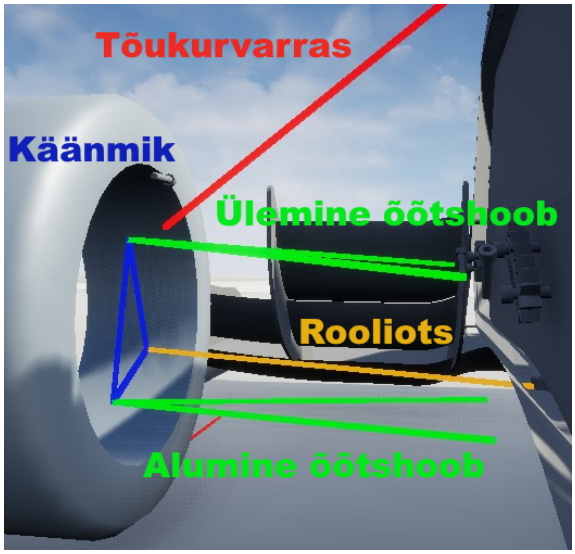
Selleks, et teada üldsegi kas vormeli rattad on kontaktis maaga tuleb vaadata kui kõrgel on vormeli rattad maapinnast. Mõõtmise teen kasutades Unreal Linetracei alustades vormeli ratta pealmisest tasapinnast ning mõõtes distantsi maapinnani mõõda vormeli Z telge, ehk alt-üles telge. Saadud pikkusest lahutan ratta diameetri ning saan kätte vahemaa ratta ja maapinna vahel, positiivne distants näitab kui palju kõrgemal on ratas maapinnast ning negatiivne, kui sügaval maa sees on ratas. Kuna vormel sõidab alati tasase maa peal siis piisab ühe punkti mõõtmisest, kuid kui on vajadus sõita ebatasase tee peal siis on võimalik mõõta distantsi üle mitme punkti kasutades sarnast meetodit Assetto Corsa 5 punkti meetodile[13].

Vastavalt vahemaale tuleb muuta vedru pikkust lühemaks või pikemaks. Peale vedru pikkuse muutmist tuleb arvutada kogu vedrustuse kinemaatika kasutades tagurpidi kinemaatikat. Vedrustuse kinemaatikas on kõik ühendused määratud geomeetriliste suhetega ning lahendatavad läbi ring, kera lõikepunktide leidmise. Tagurpidi kinemaatika algab vedru kere kinnitusest ning leiab nookuri nurga. Nookurist saab leida tõukurvarda algus punkti ja sealt edasi ülemise õõtshoova punktid, käänmiku ülemine punkt, alumise õõtshoova punktid ja lõpuks rooliotsa puntkid. Sarnase protsessiga saab ka leida ARB, ehk stabilisaatorvarda nurga (joonised 2 3).

Teades käänmiku asukohta ja nurka saab leida ratta asukoha ja nurga. Kuna ratta liikumine



Joonis 2. Vormeli tagumine vedrustuse kinemaatika



Joonis 3. Vormeli esivedrustuse kinemaatika

mõõda Z telge ja vedru pikkus ei ole omavahel ideaalse üks ühese suhtega on kasutuses iteratiivne lähenemine. Kogu mõõtmise ja kinemaatika arvutamise protsess kordub N korda kuni soovitud on täpsus saavutatud. Edaspidi peale kinemaatika arvutamist läheb vaja ratta asetust, vedru pikkust ning stabilisaatorvarda nurka algasendi suhtes.

Kui vedrustuse kinemaatika on lahendatud, on võimalik liikuda edasi füüsikaliste jõudude juurde. Vedrustuse modelleerimisel kasutatakse Hooke'i seadust koos lineaarse amortisatsioonigiga, et kirjeldada vedru tekitatud jõudu:

$$F = -kx - cv$$

Kuna kiirus v sõltub eelnevast vedru olekust, kasutatakse vedru kiiruse ja sellest tuletatud kiirenduse arvutamiseks neljandat järku Runge-Kutta (RK4) integraatorit. See võimaldab saavutada piisava täpsuse ja stabiilsuse ka olukordades, kus ajasamm Δt on ettenähtust suurem. Et vältida arvutuslikke instabiilsusi, piiritletakse amortisatsioonijõu maksimaalset väärtust. See on vajalik, kuna suured kiirused võivad põhjustada järske jõude, eriti suurtel ajasammudel.

C++ Pseudokood, vedrustuse tuletis

```
float SpringDeformation = (SuspensionFreeLength -
                           CurrentSuspensionLength);
float SuspensionVelocity = 0.0f;
float SuspensionAcceleration = 0.0f;
if (DeltaTime > 0) {
    SuspensionVelocity = (CurrentSuspensionLength -
                          WheelState.SuspensionLength) / DeltaTime;
    SuspensionAcceleration = (SuspensionVelocity -
                              WheelState.SuspensionVelocity) / DeltaTime;
}

float SpringForce = SpringDeformation * SuspensionStiffness;
float SpringDamping = SuspensionVelocity *
                      SuspensionDamping;
SpringDamping = FMath::Clamp(SpringDamping, -10000.0f, 10000.0f);

WheelDerivative.SuspensionVelocity = SuspensionVelocity;
WheelDerivative.SuspensionAcceleration = SuspensionAcceleration;
```

Vedru tekitatud jõu saab kätte liites kokku vedru nihke tekitatud jõu (SpringForce) ja amortisatsiooni jõu (SpringDamping). Edaspidi tähistatakse seda jõudu sümboliga F_z. See jõud rakendatakse vormelile igal rattal, ratta ja maapinna kontaktpunktis, vastavalt maapinna normaalsuunale.

Stabilisaatorvarda jõud arvutatakse hiljem kasutades mõlema ratta vedrustuse stabilisaatorvarraste nurkade vahet.

3.1.2 Rehvi mudel

Rehvi tekitatud jõudude simuleerimiseks kasutatakse empiirilist Pacejka rehvimudelit, mida tuntakse ka kui "Magic Formula". Simulaatoris on kasutuses FS Team Tallinna sisene modifitseeritud Pacejka rehvimudel, kuid mudeli põhivorm on säilitatud. See mudel võimaldab hinnata rehvi poolt genereeritavaid pikku- ja külgpidi jõude, kasutades sisendparameetritena slip ratio't S_r ja slip angle'it α.

$$S_r = \frac{R\omega - v_x}{\max(R\omega, v_x)}$$

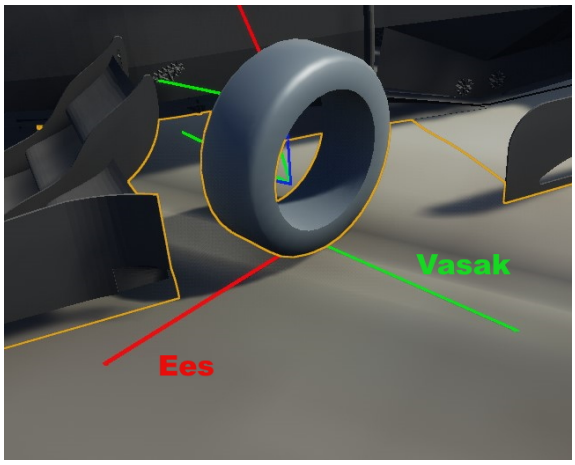
$$\alpha = \arctan\left(\frac{v_y}{|v_x|}\right) - \delta$$

Kõik kiirused arvutatakse ratta lokaalses koordinaatsüsteemis, lähtudes ratta kontaktpunktis esinevast absoluutkiirusest. Kontaktpunkti absoluutkiirus saadakse liites vormeli masskeskme transleeruv kiirus ja pöörlev kiirus kokku.

$$\vec{v} = \vec{v_{com}} + \vec{\omega} \times \vec{r}$$

Võttes ratta absoluutkiiruse komponendid ratta lokaalses kordinaatsüsteemis telgedel Ees ja Vasak, saab leida vastavalt kiirused v_x ja v_y; joonis 4. Siiski, slip ratio ja slip angle definitsioonid muutuvad ebastabiilseks madalatel kiirustel, kuna need sõltuvad pöördvõrdeliselt v_x-ist. Sellises olukorras, näiteks madalate kiiruste või peatumise läheduses, võivad isegi väikesed kiirusvektorite muutused põhjustada suuri libisemisparameetrite kõikumisi.

Selle probleemi leevendamiseks kasutatakse relaksatsioonipikkusel L_r põhinevat lähenemist. Meetod võtab inspiratsiooni praktilistest simuleerimistehnikatest [14] ning seisneb slip angle'i väärtuse ajas interpoleerimises, võttes arvesse läbitud teepikkust ja kiiruse muutust. Selle eesmärk on võimaldada rehvil füüsikaliselt usutaval viisil jõu kogunemist



Joonis 4. Vormeli esiratta lokaalsed teljed

ning jõu säilitamist staatilistes olukordades, näiteks kui vormel seisab kallaku peal. Meetod aitab ka hoida vormeli rehvijõude stabiilsena kui vormel seisab paigal.

Interpoleeritud α valem, α_{new} on geomeetriline praeguse hetke α.

$$\alpha = \alpha + (\alpha_{\text{new}} - \alpha) \cdot \text{Clamp}\left(\frac{|v_y|}{L_r} \cdot \Delta t, 0, 1\right)$$

Interpoleeritud S_r valem on sarnane kuid erineb kiiruse poolest. S_r valem kasutab ratta ja maa vahelist läbi libisemis kiirust v_s.

$$v_s = \omega \cdot r - v_x$$

$$S_r = S_r + (S_{\text{new}} - S_r) \cdot \text{Clamp}\left(\frac{|v_s|}{L_r} \cdot \Delta t, 0, 1\right)$$

S_{new} arvutan kasutades v_x alusel interpoleeritud väärtust hetke reaalse S_r ja aeglase kiiruse S_{slow} vahel. S_{slow} on sama mis läbi libisemis kiirus.

$$S_{\text{slow}} = \omega \cdot r - v_x$$

Arvutatud S_r ja α alusel määratakse ratta tekitatud jõud F_x ja F_y, kasutades Pacejka rehvimudelit.

Ratta pöörlemiskiirust uuendatakse igal simulatsiooni sammul, võttes arvesse sisendina mootori jõumomenti T_m, eelmisel sammul tekitatud rehvi longitudinaaljõudu F_x ning pidurduse jõumomenti T_b. Alguses leian ratta nurkkiirenduse $\dot{\omega}_m$ mootori ja rehvi jõu alusel. I on ratta inertsmoment pöörlemistelje ümber.

$$\dot{\omega}_m = \frac{T_m - F_x \cdot r}{I}$$

Kasutades nurkkiirendust leitakse ratta potentsiaalne kiirus.

$$\omega_{\text{fut}} = \omega_{cur} + \dot{\omega}_m \cdot \Delta t$$

Järgmisena leitakse pidurduskiirendus $\dot{\omega}_b$. Clamp piirab pidurduskiirendusel kiirendada ratast rohkem kui vaja tagades, et ratta pöörlemis suund ei vahetu pidurdamise tõttu.

$$\dot{\omega}_b = \text{Clamp}\left(\frac{T_{\text{b}}}{I},\ 0,\ \left|\frac{\omega_{\text{fut}}}{\Delta t}\right|\right) \cdot (-\text{sgn}(\omega_{\text{fut}}))$$

Uus nurkkiirus arvutatakse liites eelnevale potentsiaalsele kiirusele pidurduskiirendus.

$$\omega = \omega_{fut} + \dot{\omega}_m \cdot \Delta t$$

Ratta pöörlemiskiirus integreeritakse ajas edasi kasutades Runge-Kutta neljanda astme meetodit (RK4).

3.1.3 Aerodünaamika

Aerodünaamiliste jõudude realistlikuks modelleerimiseks kasutatakse kahte erinevat aero dünaamilist mudelit: üks sirgjoonelise liikumise jaoks ja teine kurvilisel trajektooril liikumise jaoks. Sirgjoonelise sõidu mudel põhineb kahemõõtmelisel põhjakõrguste kaardil, kus iga kombinatsiooni jaoks on määratletud aerodünaamiline tõstejõu koefitsient (Cla), takistuskoefitsient (Cda) ning aerodünaamiline esi- ja tagatelje jõudude jaotus. Kurvis sõitmise jaoks kasutatakse ühemõõtmelist tabelit, mis seob Cla, Cda, esi- ja tagatelje jõudude jaotuse ning vasak- ja parempoolse aerodünaamilise tasakaalu sõiduki pöördenurga kiirusega (yaw rate). Kahe mudeli kasutamisega saab tagada realistliku aerodünaamika jõudude jaotuse nii sirgjoonelisel kiirendamisel kui ka kurvis sõites. Kuna kurvis sõites võib nihkuda ligi 5% jõust ühelt küljelt teisele siis see on piisav jõudude muutus, et mõjutada simulatsiooni realismi.

Sirgjoonelise liikumise mudelis kasutatakse bilineaarset interpoleerimist, mis tagab sujuvad üleminekud väärtuste vahel. Kurvis sõitmise mudelis rakendatakse lineaarset interpoleerimist. Kui sõiduki pöördenurga kiirus on väiksem kui kurvis sõitmise esimene kontrollpunkt (võti), interpoleeritakse kõik aerodünaamilised suurused kahe mudeli vahel pöördenurga kiiruse alusel. Kui pöördenurga kiirus ületab esimese kontrollpunkti väärtuse, kasutatakse täielikult kurvis liikumise mudelit.

Saadud väärtustega arvutan välja aerodünaamilised jõud mis mõjuvad vormelile. Kiirus on vormeli kiiruse vektor simulaatori gloobalses kordinaatsüsteemis.

$$F_d = \frac{1}{2} \cdot \rho \cdot Cda \cdot |\vec{v}|^2$$

$$F_l = \frac{1}{2} \cdot \rho \cdot Cla \cdot |\vec{v}|^2$$

Õhutakistusjõud F_d rakendatakse vormeli masskeskmele ja on suunatud kiirusvektoriga vastassuunas. Survejõud F_l rakendatakse vormeli vertikaalsuunas alla ning jaotatakse kõigi nelja ratta kontaktpunkti vahel vastavalt aerodünaamika jõudude jaotusele.

3.1.4 Integreerimis meetod

Simulaatoris kasutatakse simulatsiooni stabiilsuse tagamiseks Runge-Kutta neljanda astme integreerimist. Koodi arusaadavuse eesmärgiga kasutan C++ struktuure vormeliga seonduvate väärtuste esitamiseks. Vormeli enda struktuuris integreerimise jaoks vajalikud parameetrid on järgnevad:

- FTransform Transform
- FVector Velocity
- FVector AngularVelocity
- FVector COMOffset
- FVector PreviousAccelerationIMU
- TArray<FWheelState> Wheels;

Antud parameetrid kirjeldavad vormeli asukohta, lineaarkiirust, nurkkiirust, masskeskme asukohta, IMU kiirendusi ja ka rataid. Rataste struktuurid hoiavad kõiki väärtusi mis on ratastele omased nagu nende hetke nurkkiirus, rakendatav jõud, jõu asukoht, libisemisnurk jne.

Listing 3.1. Runge Kutta 4 struktuur pseudokood

```
K1 = evaluate(0.0, initialState);
midState1 = initialState;
getFutureState(midState1, K1, 0.5 * deltaTime);

K2 = evaluate(0.5 * deltaTime, midState1);
midState2 = initialState;
getFutureState(midState2, K2, 0.5 * deltaTime);

K3 = evaluate(0.5 * deltaTime, midState2);
endState = initialState;
getFutureState(endState, K3, 1.0 * deltaTime);

K4 = evaluate(1.0 * deltaTime, endState);

vehicleState = (K1 + 2.0 * K2 + 2.0 * K3 + K4) * (deltaTime / 6.0);
```

Runge-kutta meetodi jaoks tuletise leidmine hõlmab vormelile mõjuvate jõudude leidmist. Protsessi käigus leitakse kõik aerodünaamika, vedrustuse ja rehvi jõud ning liidetakse need kokku.

Listing 3.2. Vormeli hetkeseisu tuletis

```
function Evaluate(deltaTime, vehicleState):
    initialize vehicleDerivative
    centerOfMass = vehicleState.position + rotation(COMOffset)

    frontHeight, rearHeight = getGroundHeights(vehicleState)
    cla, cda, frontBal, leftBal = aeroMap(frontHeight, rearHeight)

    velocitySq = magnitude(square(vehicleState.velocity))
    downForce = 0.5 * airDensity * cla * velocitySq
    dragForce = 0.5 * airDensity * cda * velocitySq

    if frontHeight and rearHeight > 0.045:
        downForce = 0

    dragVector = -normalize(vehicleState.velocity) * dragForce
    frontARB, rearARB = computeARBForces(vehicleState)

    for each wheel i in 0..3:
        steer = getSteerValue(i)
        aeroCoef = aeroDistribution(i, frontBal, leftBal)
        arbForce = getARBForce(i, frontARB, rearARB)
        wheelDownForce = downForce * aeroCoef

        force = computeWheelForce(i, steer, deltaTime, vehicleState)
        force -= (wheelDownForce + arbForce) * upDirection
        torque = cross(forcePosition - centerOfMass, force)

        accumulate force and torque into totalForce and totalTorque
        store wheelDerivative

    totalForce += gravity + dragVector
    vehicleDerivative.acceleration = totalForce / mass
    vehicleDerivative.angularAcceleration = totalTorque / inertia

    return vehicleDerivative
```

Vormeli tuleviku seisu leidmiseks liidetakse vormeli tuletise ja aja korrutis vormeli

praegusele seisule. Selle käigus saab leida vormeli uue asukoha ning liikumissuuna.

3.2 Testimis komponendid

Simulaatori eesmärk on võimaldada vormeli isejuhtiva süsteemi testimist ja arendamist. Selleks peab simulaator suutma suhelda reaalajas vormeli isejuhtiva süsteemiga, mis on üles ehitatud *Robot Operating System* (ROS) Noetic versioonile. ROS Noetic töötab ametlikult ainult Linuxi operatsioonisüsteemides, kui simulaator mis on arendatud *Unreal Engine* keskkonnas, toimib kõige paremini Windowsi platvormil.

Platvormidevahelise suhtluse võimaldamiseks kasutatakse *ROSIntegration* pluginit [8], mis võimaldab Unreal Engine'il vahetada andmeid ROS-sõlmedega. Selle lahenduse puhul käivitatakse ROS poolel *rosbridge_server*, mis loob standardiseeritud WebSocket-põhise liidese. Selle kaudu saavad simulaator ja ROS-põhine juhtsüsteem andmeid vahetada, võimaldades nii juhtkäskude edastamist kui ka sensoriandmete tagasisidet.

3.2.1 ROSi ümber kaardistamine

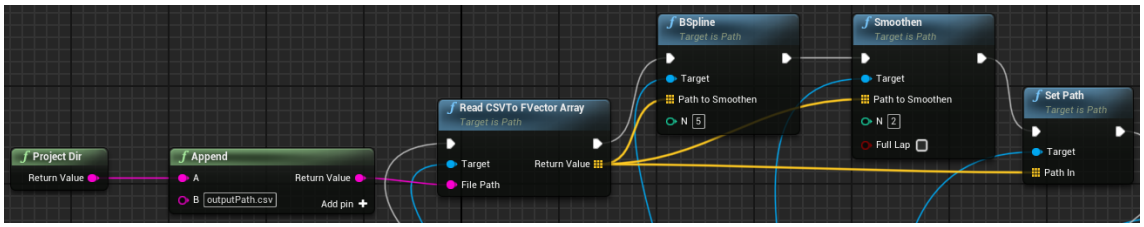
Kuna simulaator ei toeta kõiki ROS-is kasutatavaid sõnumitüüpe, on vaja täiendavat tõlkekihti, mis võimaldab erinevate süsteemide vahel andmeside säilitamist. Selleks on loodud vahelüli *Sim-remapper*, mille ülesandeks on ROS-i sõnumitüüpide ümberkaardistamine simulaatoriga ühilduvateks andmestruktuurideks.

Lisaks ühilduvuse tagamisele võimaldab *Sim-remapper* vähendada andmemahtu. Tõlkides ümber mahukamad andmetüübid numbritejadaks, *Float32MultiArray-iks*, saab edastada andmeid palju efektiivsemalt.

3.2.2 Trajektoor

Control testimiseks on vajalik ettemääratud trajektoor, millel järgi proovib vormel sõita. Kuigi isejuhtiv süsteem suudab trajektoori genereerida reaalajas vastavalt raja märgistele, ei ole selle algoritmiline töökindlus ja täpsus alati piisav katsetingimuste tagamiseks. Seetõttu on simulaatorisse integreeritud funktsionaalsus, mis võimaldab edastada olemasolevat trajektoori ROS-süsteemile.

Trajektoor esitatakse kui järjestikuste positsioonivektorite jada. Trajektoori sujuvuse paran-



Joonis 5. Vormeli trajektoori eelseadistamine Unreal blueprint süsteemis

damiseks kasutatakse B-spline interpolatsiooni, millega saab trajektoori teha sujuvamaks ning ka punktide poolest tihedamaks. Punktide arvu vähendamiseks, säilitades samal ajal trajektoori üldise kuju, rakendatakse Douglas–Peuckeri algoritmi.

Trajektoori klass kasutab Unreal blueprint süsteemi ning seekaudu võimaldab kergelt laadida sisse erinevaid trajektoore ning muuta neid vastavalt vajadustele; joonis 5.

3.2.3 Lihtsustatud LIDAR

Vormel kasutab koonuste tuvastamiseks LIDAR-it. Kuna LIDAR on tehnoloogiliselt väga keerukas süsteem, mille realistlik simuleerimine nõuaks olulist arvutusressurssi ja arendusaega, on käesolevas töös rakendatud lihtsustatud LIDAR-mudelit.

Antud mudel töötab kasutades vormeli asukohta ning teades kõigi koonuste asukohta simulaatori maailmas. Simuleeritud LIDAR tuvastab kõik koonused, mis asuvad määratletud nägemiskoonuse sees. Koonuste asukohad muudetakse polaarkordinaatideks. Nägemiskoonuse alguspunkt on paigutatud vormeli tagatelje keskpunkti, kuna see vastab nullpunktile, mida kasutab SLAM-algoritm koonuste asukohtade sisendina. (joonis 6). Tulevikus on plaanis ka lisada funktsionaalsus mis takistab koonuste nägemist koonuste tagant, tehes simulatsiooni realistlikumaks.

Kuna lihtsustatud LIDAR võib osutuda liiga ideaalseks on implementeeritud ka lihtsam müra funktsioon mis muudab mõõdetud koonuse asukohta kindla vahemiku piires. Müra teeb simulatsiooni realistlikumaks ning võimaldab testida SLAMi päris elule sarnastemal tingimustel.

Joonis 6. Koonused vormeli lihtsustatud LIDARi nägemisväljas

3.2.4 Automaat testimine

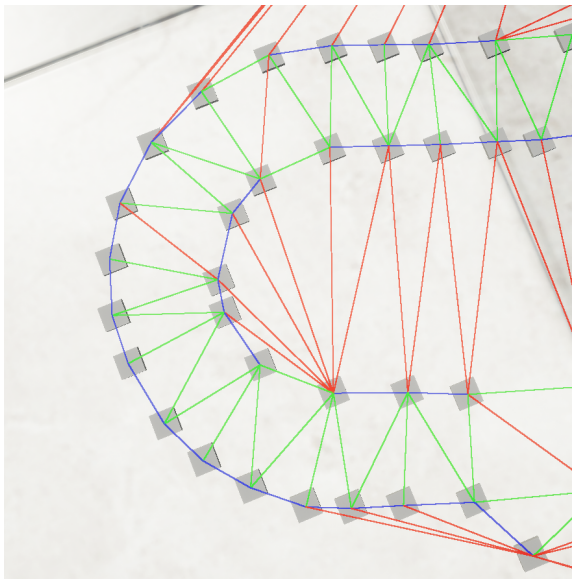
Simulaatorisse on integreeritud mitmed lisafunktsionaalsused, mis võimaldavad automatiseeritud testimist ilma inimese sekkumiseta.

Üheks oluliseks funktsionaalsuseks on vormeli asukoha lähtestamine. Simulaator võimaldab vormeli simulatsioonisisest taaskäivitamist, mille käivitamiseks tuleb saata ROS-sõnum uue vormeli asukohaga. Vormel paigutatakse saadud asukohta ning starditakse algseisundis. Selline lahendus võimaldab efektiivset testimist erinevate juhtimisparameetrite ja alamsüsteemide konfiguratsioonidega, ilma vajaduseta inimese sekkumiseks.

Lisaks on simulaatorisse lisatud võimalus finišijoone tuvastamiseks. Kasutaja saab paigutada maailma mitu finišiväravat, olenematta mis värvast sõidab vormel läbi loendatakse ringide arvu üles ning saadetakse välja ROS sõnum. Sõnum sisaldab informatsiooni ringi numbri ja ringi aja kohta, võimaldades tulemuste automaatset logimist ja analüüsi.

3.2.5 Radade loomine

Trajektoori planeerimise ja controlli testimise jaoks on vaja testida mitmetel erinevatel radadel. Selleks on simulaatorisse sisse ehitatud radade loomise plugin. Pluginal on kaks reziimi, koonuste ning trajektoori reziim. Koonuste reziimis saab paigutada või eemaldada koonuseid. Trajektoori reziimis saab lisada trajektoori punkte või eemaldada neid. Nii koonuseid kui trajektoori saab eksportida pluginast csv formaati.



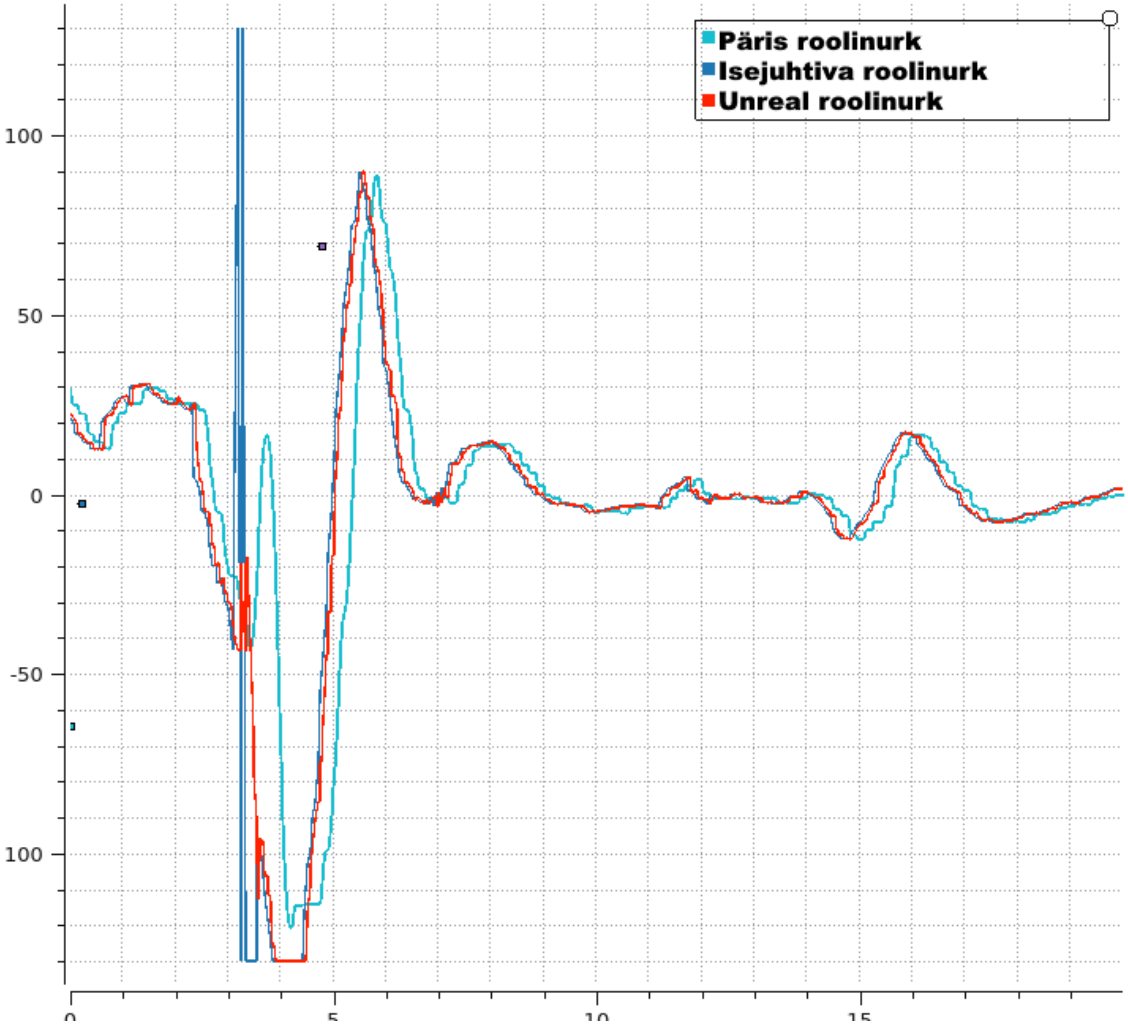
Joonis 7. Raja loomine

Kuna rajad peavad järgima reegleid koonuste paigutuse poolest, kasutab raja loomise plugin Delaunay triangulatsiooni. Kuna Delaunay triangulatsiooni algoritmid on väga üldkasutatavad otsustasin kasutada olemasolevat lahendust, et lisada Delaunay Triangulatsioon simulaatorisse (allikas[15]). Delaunay triangulatsiooniga saab visualiseerida plugin koonuste vahelised distantsid värvidega. Sinine on alla 3 meetri, roheline on 3-5 meetrit ja punane on üle 5 meetri. Reeglite kohaselt ei tohi rada piirlevad koonused olla üle 5 meetri üksteisest ning raja laius ei tohi olla kitsam kui 3 meetrit (joonis 7).

3.2.6 Rooli mootori simulatsioon

Isejuhtiv süsteem saadab välja soovitud roolinurga vormeli roolimootorile, ning roolimootor keerab rooli. Kuna jõud ning kiirus millega roolimootor keerab sõltub kogu vedrustuse kinemaatikast ning rehvi ja maa hõõrdejõust siis on keeruline simuleerida kogu protsessi realistlikult. Seega simulaatoris on roolimootor lihtsustatud lineaarsel kiirusel pööramisele. Päris elu vormeli testidelt põhinevatel andmetel on roolimootor isejuhtiva süsteemi käskudest umbes 300 millisekundit maas, kuid see pole implementeeritud simulaatorisse.

Simulaatoris töötab roolimootor lineaarse interpoleerimisega, iga ajasammu tagant liigub roolimootori nurk sihile lähemale piiratud kiirusega. Kiiruse piiranguks on hetkel seatud umbes 400 kraadi sekundist. See seab rooli keeramise aja otse asendist täis pöördesse umbes 300 millisekundit. Lahendus teeb järsud isejuhtiva süsteemi käsud sujuvamaks (joonis 8).



Joonis 8. Roolinurga võrdlus simulaatori ning ROS bagi vahel

3.2.7 Juhtsüsteem ja IMU

Vormel kasutab päriselus juhtsüsteemi, et juhtida mootoreid ning kuulata isejuhtivat süsteemi. Juhtsüsteem on loodud Simulink tarkvaras. Kuna Simulink toetab projekti eksportimist C/C++ kujul ei ole juhtsüsteemi integreerimine simulaatorise probleemne. Kogu juhtsüsteemi projekt on integreeritud simulaatori lähtekoodi UE4 mooduli kujul. Kuna Simulinki kood ei vastanud alguses UE4 koodi standarditele tuli osades .cpp failides muuta importide järjekorda. Juhtsüsteem kasutab fikseeritud ajasammu ning eeldab jooksmis sagedust 200 Hz, kuna simulaatori füüsika jookseb samuti 200 Hz kutsutakse juhtsüsteemi iga füüsika samm ilma lisa loogikata.

IMU on vormeli kiirendusandur ning päris vormelil asub see masskeskme lähedal. Simulaatoris on IMU asukoht lihtsustatud vormeli masskeskme asukohaks. Kiirenduste arvutamiseks kasutan eelneva ja käesoleva füüsika sammu vormeli hetkekiiruste vahet ning jagan need ajasammuga.

3.2.8 Unreal Blueprint

Unreal blueprint süsteem on UE4 vooskeemi stiilis programmeerimiskeel. Kõik klassid mis lisavad funktsionaalsusi testimiseks on kirjutatud C++ keeles. Kasutades vooskeeme on võimalik ühendada kõik klassid kokku ning saavutada modulaarne vormeli ROS liidese ülesehitus. Vooskeemid võimaldavad kergelt lisada või lahti ühendada erinevaid komponente ilma C++ koodi muutmata. Vooskeemid võimaldavad kasutajatel muuta simulatsiooni seadeid ilma C++ teadmisteta.

4 Valideerimine

Simulaatori arendamine algas 2024 aasta Oktoobri kuus ning esimene töötav füüsikaline mudel saadi tööle 2024 aasta Detsembri alguses. Ennem kui simulaatorit sai kasutada kontrolli testimiseks tuli valideerida simulaator ning roolimootori simulatsioon. Selleks kasutati 2024 hooaja isejuhtiva sõidu salvestisi. Valideerimiseks mängiti salvestise sisendeid simulaatoris ning võrreldi kuidas vormel sõidab simulaatoris ning kuidas vormel sõitis salvestises. Vormel sõitis simulaatoris õige kiirusega ning läbis pöörded väga sarnase nurkkiirusega, kuid probleemseks osutusid kohad kus salvestises andis isejuhtiv süsteem anomaalseid roolinurga sisendeid. Kuna simulaatoris on roolimootor simuleeritud ainult läbi roolinurga lineaarse interpoleerimise hakkas vormel simulaatoris tegema väga järske jõnkse kui salvestises tegi vormel väikse jõnksu. Siiski oli simulaator piisavalt korrektne enamus ajast, et sellega sai hakata arendama uut kontrolli.

2025 aasta Mai kuuks on enamus simulaatori füüsika komponentidest valmis. Võrreldes esimese töötava mudeliga on nüüdseks lisandunud simulaatorisse aerodünaamika jõud, vedrustuse kinemaatiline simulatsioon, realistlikum rehvimudel, ning korrigeeritud vormeli masskeskme paiknemist ja inertsmomenti. Simulaator on teoorias paljun realistlikum kuid selle valideerimiseks puuduvad andmed. Kuna isejuhtiv vormel sõidab ilma juhita erineb see juhiga vormelist paljude tegurite poolest, ning seetõttu ei saa kasutada juhiga sõidu salvestisi valideerimiseks. Eelnevatel aastatel ei ole isejuhtiv vormel sõitnud nii kiiresti kui simulaatoris sõidetakse.

Simulaatori valideerimise plaan hõlmab simulaatori ja päris elu vahelist sõitmise võrdlemist erinevates sõidu püsiseisundites. Valideerimis andmete jaoks on plaanis kasutada kiirenduse, kaheksasõidu ning raja sõidu salvestisi. Kiirendusega saab valideerida isejuhtiva vormeli kiirendust ning massinihkumist eest-taha. Kaheksasõiduga saab valideerida isejuhtiva vormeli külgmise kiirenduse. Raja sõiduga saab valideerida kuidas roolimootor käitub ning vormeli dünaamikat.

Siiski on võimalik osaliselt valideerida mitmeid komponente, kasutades kehtivaid füüsika valemeid. Kiirenduse ajal nihkub vormeli mass taha ning see kajastub vedru jõududes. Kasutades geomeetrilise ning elastse kaalu ülekandumise valemeid saab arvutada kui palju nihkub kaal kiirenduse all. Kui simulaatoris panna vormel kiirenduse alla ning võrrelda rataste F_z-e valemite põhjal ennustatuga siis on erinevus väike, seega saab väita, et vedrustuse simulatsioon käitub korrektselt sirgjoonelise kiirenduse all.

Meeskonna tagasiside simulaatori vastu on olnud positiivne. Uus simulaator on kiirendanud kogu isejuhtiva süsteemi arendamist ja testimist, kõige rohkem on see mõjutanud kontrolli arendamist. Meeskonna liikmetelt saadud tagasisidest saab välja tuua uue simulaatori suurimad plussid:

- Jälgitav 3D mudel
- Võimalik luua ise radu
- Saab muuta trajektoori punktide tihedust
- Simulaator logib mahasõidetud koonuseid
- Realistlikum füüsika
- Samad sisendid mis päris vormelil

5 Kokkuvõte

Töö käigus sai arendatud uus simulaator FS Team Tallinna jaoks. Simulaatoris saab testida kas kogu isejuhtivat süsteemi korraga või erinevaid komponente ükshaaval. Simulaatoris on füüsikaline mudel loodud kasutades 2024 vormeli parameetreid, kuid projekti ülesehitus võimaldab kergelt luua uue mudel kasutades 2025 vormeli parameetreid. Simulaatoris on mitmed funktsionaalsusi mis teevad testimise mugavamaks. Raja loomise plugin võimaldab luua uusi radu või modifitseerida olemasolevaid. Sama pluginaga saab ka luua trajektoore millega saab testida kontrolli, jättes vahele trajektoori planeerimise. Simulaator kasutab LIDARi asemel lihtsamat nägemiskoonust, millega saab valideerida SLAMi.

Täna kasutatakse simulaatorit aktiivselt kontrolli ja trajektoori planeerimise testimiseks ning SLAMi valideerimiseks. Simulaatorit plaanitakse kasutada ja arendada edasi see suvi ning järgmistel aastatel.

Kasutatud kirjandus

[1] Open Source Robotics Foundation. *Robot Operating System (ROS)*. http://wiki.ros.org/noetic. Accessed: 2025-05-12. 2020.

[2] Juraj Kabzan *et al. AMZ Driverless: The Full Autonomous Racing System*. 2019. eprint: arXiv:1905.05150.

[3] *Formula Student Driverless Simulator*. https://github.com/FS-Driverless/Formula-Student-Driverless-Simulator. Accessed: 2025-05-12. 2020.

[4] Epic Games. *Unreal Engine*. Versioon 4.27. 2021. URL: https://www.unrealengine.com.

[5] Shital Shah *et al.* „AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles“. Teoses: *Field and Service Robotics*. 2017. eprint: arXiv:1705.05065. URL: https://arxiv.org/abs/1705.05065.

[6] Dean Zadok *et al.* „Explorations and Lessons Learned in Building an Autonomous Formula SAE Car from Simulations“. *arXiv preprint arXiv:1905.05940* (2019).

[7] Alexey Dosovitskiy *et al.* „CARLA: An Open Urban Driving Simulator“. Teoses: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, lk. 1–16.

[8] *ROSIntegration*. https://github.com/code-iai/ROSIntegration. Accessed: 2025-05-12. 2025.

[9] Unity Technologies. *Unity*. 2025. URL: https://unity.com/.

[10] *Unity-Robotics-Hub*. https://github.com/Unity-Technologies/Unity-Robotics-Hub. Accessed: 2025-05-12. 2025.

[11] Brian Beckman. *The Physics of Racing*. http://autoxer.skiblack.com/phys_racing/contents.htm. Accessed: 2025-05-12. 1991.

[12] Zack Fizell. *How to Solve ODEs in MATLAB using Runge-Kutta*. https://towardsdatascience.com/how-to-solve-odes-in-matlab-without-built-in-functions-9eaa01c9fe10/. Accessed: 2025-05-12. 2022.

[13] *ACC: the 5 point tyre model*. https://trinacriasimracing.wordpress.com/acc-the-5-point-tyre-model/. Accessed: 2025-05-12. 2025.

[14] . | *Transient Longitudinal Force*. https://www.youtube.com/watch?v=p30gCgsveGI. YouTube video, accessed 2025-05-12. 2021.

[15] *Unreal Delaunay Triangulation*. https://forums.unrealengine.com/t/math-in-delaunay-triangulation-algorithm-too-many-triangles-appearing/677429/. Accessed: 2025-05-12. 2022.

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

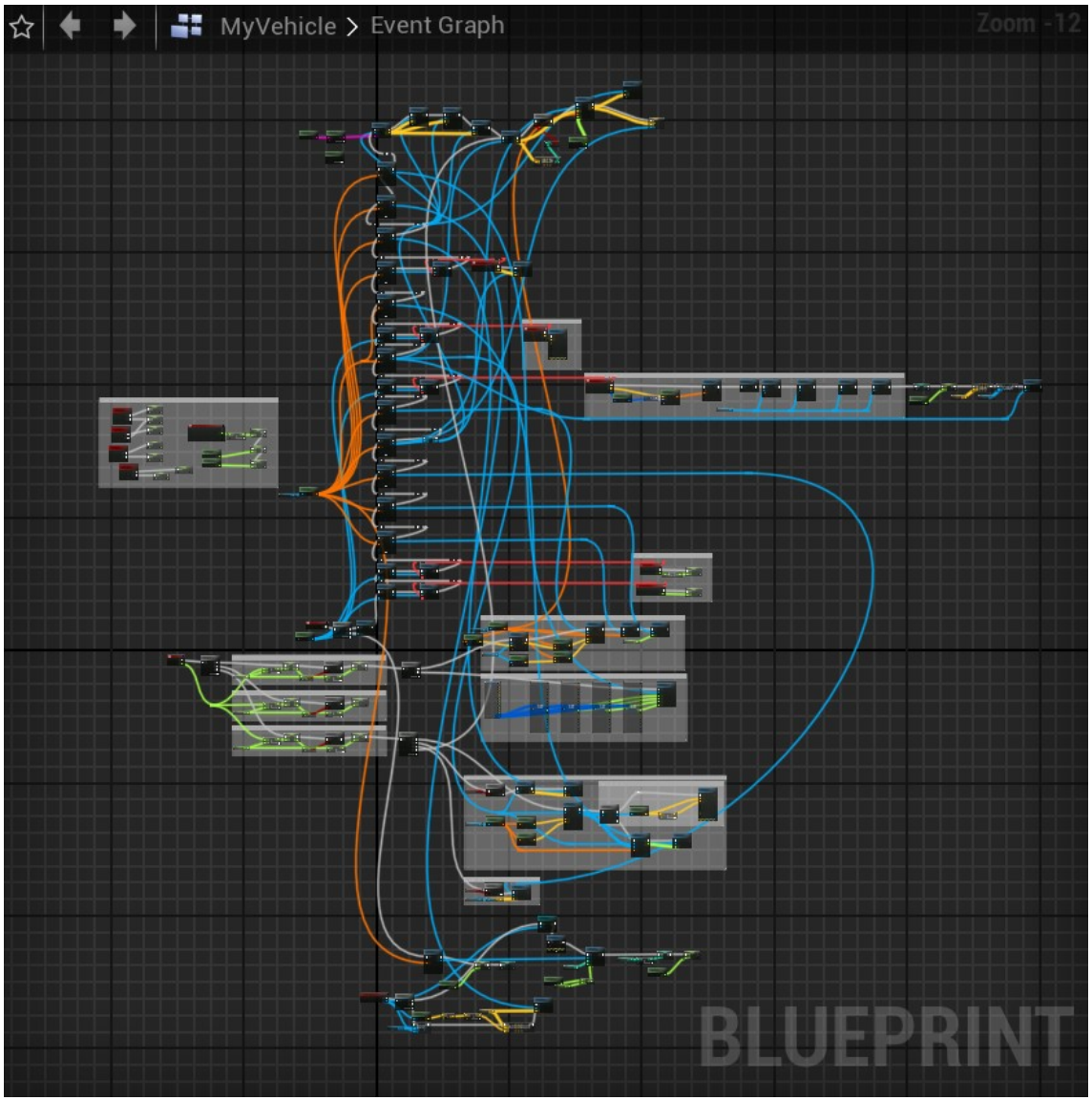
Mina, Robin Teaste

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "[FS Team Tallinna isejuhtiva vormeli simulaatori arendamine]", mille juhendaja on Gert Kanter
    1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
    1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

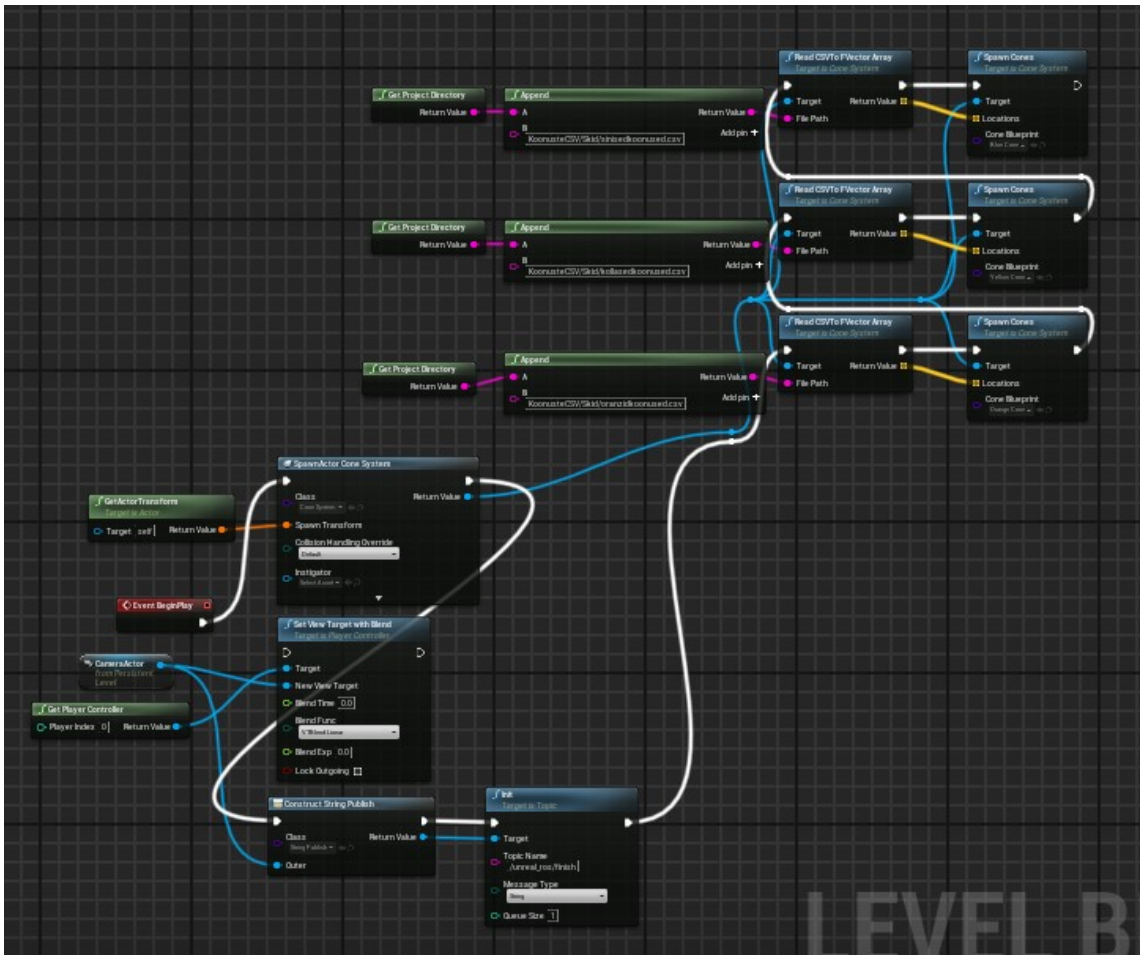
11.06.2025

[1] Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

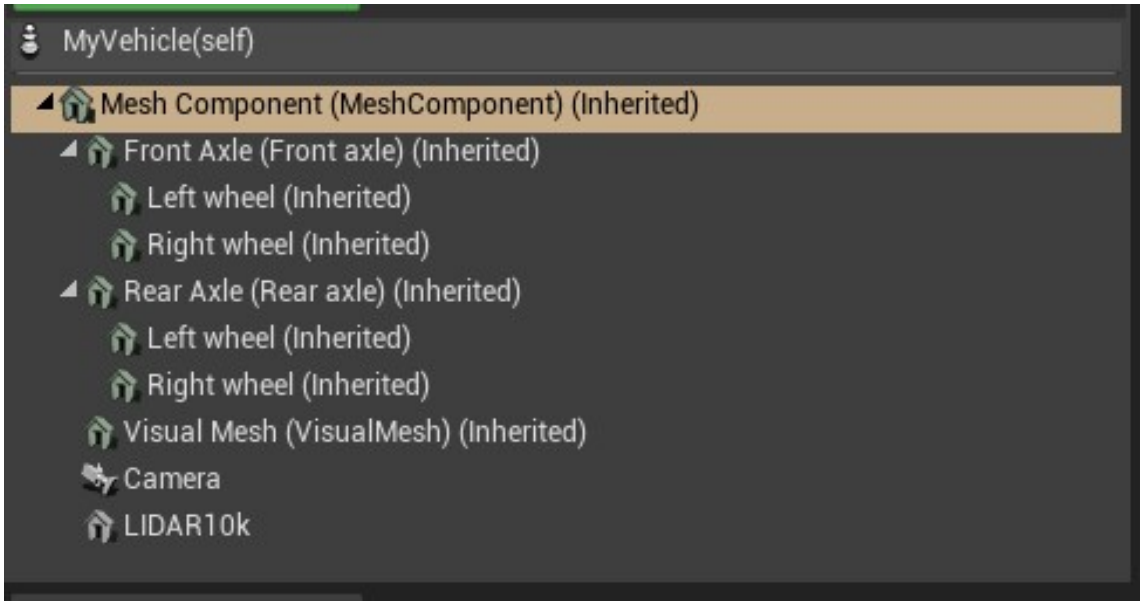
Lisa 2 – UE4 vormeli ülesehitus



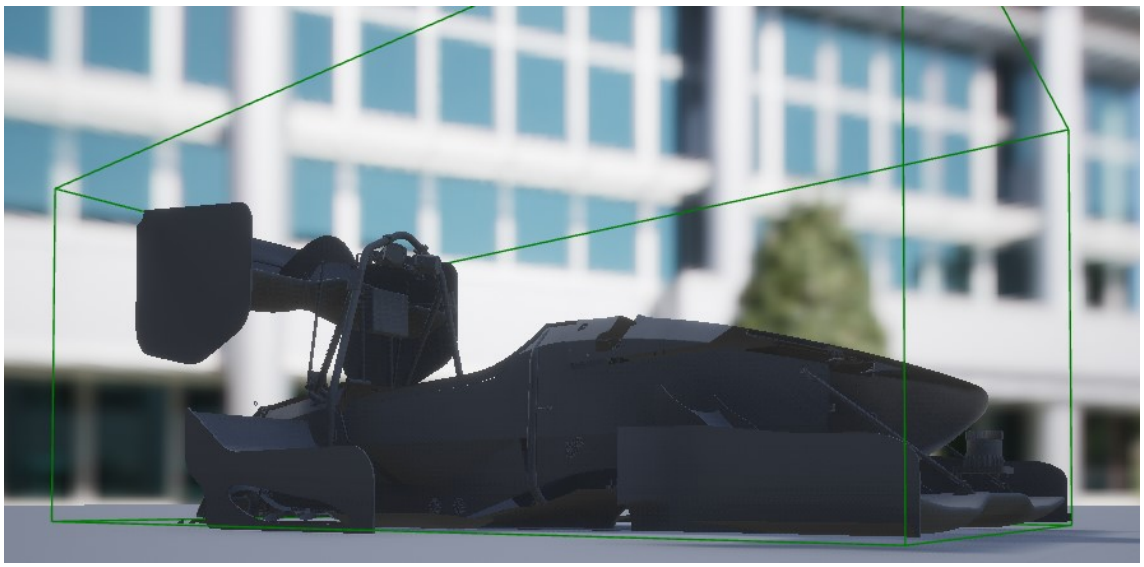
Joonis 9. Simulaatori vormeli UE4 blueprint vooskeem



Joonis 10. Simulaatori koonuste sisselaadimine

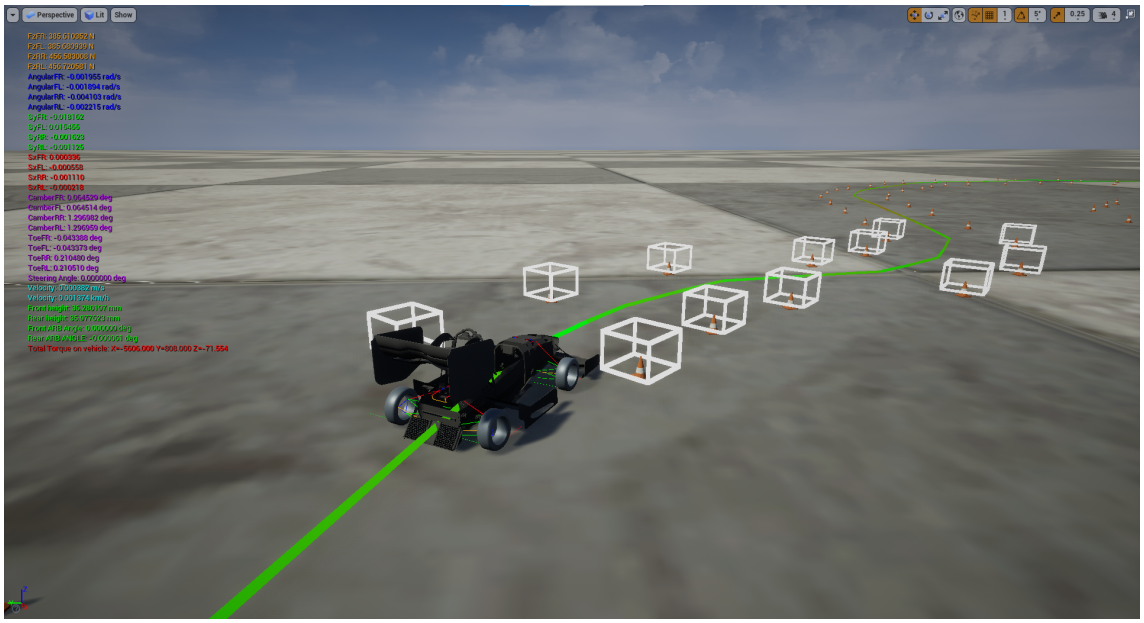


Joonis 11. Vormeli komponentide puu



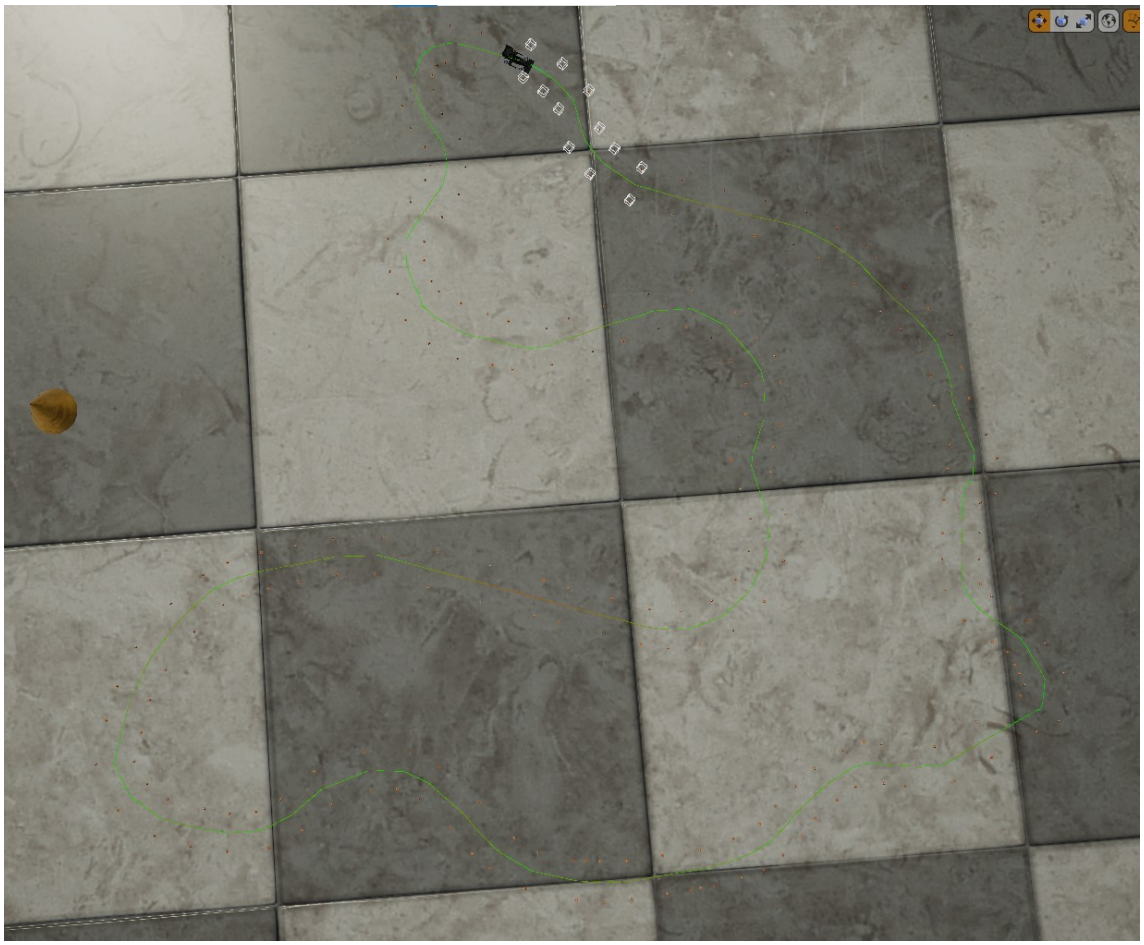
Joonis 12. Vormeli kokkupõrke kast

Lisa 3 – Vormel simulaatoris



Joonis 13. Simulaatoris suur ekraani vaade

Joonis 14. Vormeli sisemised parameetrid kiiresti sõidul



Joonis 15. Simulaatoris reaalajas kalkuleeritud trajektoor