Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

PSZEUDO-RANDOM TESZTMANŐVER GENERÁLÁS ROBUSZTUSSÁGI TESZTEKHEZ

Mikes Marcell

Dr. Micskei Zoltán Imre
Tanszéki konzulens

Kotán Gábor
Vállalati konzulens

BUDAPEST, 2023

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Mikes Marcell**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2023.06.11

……

Mikes Marcell

Kivonat

A szervokormány sokak által ismert és használt alkatrész. Lehetővé teszi a gépjárművek irányítását és meghatározza a vezetés érzetét. A szervokormány egy biztonságkritikus rendszer, ezért a tesztelésére kiemelt figyelmet kell fordítani a fejlesztés során. Hiba viszont figyelmesség ellenére is kerülhet egy szoftverbe, ezért fontos, hogy minél alaposabban teszteljük a rendszert átadás előtt.

A rendszer tesztelésének egyik módja a robusztusságának megvizsgálása. Ezt különböző tesztmanőverek közben, hibák injektálásával tehetjük meg. Egy robusztussági tesztmanőver autó szimulátoros vagy valós vezetés méréseiből elkészítése jelentős erőforrásokat vesz igénybe, ezért ezek a tesztek csak mérsékelt mennyiségben állnak rendelkezésre. Tesztelés során így a rendszer működését mindig ugyanazokkal a meglévő tesztmanőverekkel vizsgálhatjuk csak meg. Több tesztmanőver tesztelése esetén olyan hibákat találhatnánk a rendszerben, amit a tesztmanőverek mennyisége miatt nem, vagy csak a fejlesztési folyamat során később vettünk volna észre.

A tesztelhető munkapontok és a robusztussági teszt hatékonyságának növelésére egy pszeudo-random tesztmanőver generáló algoritmust készítettem. Az algoritmus egy pszeudo-random generált zajt kormányszög és járműsebesség referenciára alakít, figyelembe véve a valós autóvezetés jellemzőit. Ennek segítségével a tesztmérnök által megadott munkapont tartományhoz tetszőleges számú, reprodukálható, valóságszerű tesztmanővert lehet készíteni rövid idő alatt. Az elkészült algoritmus működését autóipari tesztkörnyezetben vizsgáltam meg.

Az algoritmus használatával tetszőleges munkapont tartományt lehet tesztelni. A generált tesztmanőverek egymástól való eltérése miatt pedig egy adott munkapont tartományon belül sokkal több munkapontot vizsgálhatunk meg, mint a két másik tesztmanőver készítési módszerrel. A robusztussági tesztek hatékonyságát az algoritmus használatával tehát növelni tudjuk, és ezáltal gyorsabban találhatjuk meg a rendszer hibáit a fejlesztés során.

Abstract

Power steering is a well-known and widely used automotive component. It enables driving and defines the steering feel of automobiles. Power steering is a safety critical system, therefore testing is an important step during the development cycle. Software defects are made unintentionally during development, so it is important to test the system thoroughly before the start of production.

Checking the robustness of a system is a form of testing. This can be achieved by injecting faults during different test maneuvers. Making a robustness test maneuver from car simulation or car driving measurements demands significant resources, therefore these test maneuvers are available in a limited quantity only. For this reason, the system can only be tested with the same test maneuvers. Testing with more test maneuvers could mean finding more defects in the system which we would not have found because of the number of test maneuvers or which we would have only found later in the development cycle.

For increasing the testable operating points and the effectiveness of the robustness test, I made a pseudo-random test maneuver generator algorithm. The algorithm makes steering wheel angle and vehicle speed reference signals from randomly generated noise, with considering the attributes of driving a car. With the help of this algorithm, we can create an arbitrary number of reproducible, realistic test maneuvers quickly in the operating point range determined by the test engineer. I examined the operation of the completed algorithm in an automotive test environment.

With the use of the algorithm an arbitrary number of operating point ranges can be tested. Because of the diversity of the generated test maneuvers, more operating points can be tested at the specified operating point range, than with the two other test maneuver creating methods. Thus, by using the algorithm we can increase the effectiveness of the robustness tests and find defects in the system faster during development.

1 Bevezetés

1.1 Tesztelés

A fejlesztési folyamat fontos része a tesztelés, mert így győződhetünk meg a termék, szolgáltatás helyes működéséről. A szoftverfejlesztés során bonyolult rendszereknél körültekintés ellenére akaratlanul is kerülhet hiba a szoftverbe. A cél, hogy ezeket a hibákat még a fejlesztési időben beazonosítsuk és kijavítsuk.

1.1.1 Tesztelés az autóiparban

A tesztelés kiemelt fontosságú az autóiparban. A legtöbb iparágban egy szoftver hibás működésének a legrosszabb következménye az üzleti hírnév romlása, illetve pénz- és időveszteség. Az autóiparban a biztonságkritikus rendszerek esetében a helytelen működés sérülést vagy akár halált is okozhat.

A szakdolgozatomat a thyssenkrupp Components Technology Hungary Kft. (továbbiakban thyssenkrupp) vállalatnál volt lehetőségem megírni. A thyssenkrupp által fejlesztett termékek biztonságkritikus rendszerek, így kiemelt figyelmet kell fordítani a működésük helyességének biztosítására. Biztosítani kell, hogy a kormányrendszerek megfelelnek az autóipari biztonsági szabványoknak és a tesztelés elengedhetetlen része ennek az ellenőrzésnek. Ezek a szabványok biztonság szempontjából az (ISO26262) és a minőség szempontjából pedig az ASPICE (Automotive Software Process Improvement and Capability Determination).

A tesztelésnek nem csupán a biztonságról való meggyőződés a célja. Fontos ellenőrizni azt is, hogy a megrendelő elvárásainak megfelelő termék készüljön el. Kormányrendszer esetében ez például lehet kényelmi funkció, a kormány középállásba visszatérése az autó elindításakor vagy például a vezetés érzete. A nem biztonsággal kapcsolatos funkcionalitások működésére is figyelmet kell fordítani.

1.2 Robusztussági teszt

A robusztusság annak jellemzője, mennyire helyesen működik a rendszer rendkívüli bemenetek vagy nagy igénybevételt jelentő környezeti feltételek mellett [1]. A robusztussági teszt célja tehát, megfigyelni és ellenőrizni a rendszer működését ilyen helyzetekben, megvizsgálva a rendszer határait, stabilitását. Ezáltal olyan hibákat találhatunk meg amik jelentős problémát okozhatnának használat közben.

1.2.1 Robusztusági tesztelés a thyssenkrupp-nál

A thyssenkrupp-nál a robusztussági tesztek a rendkívüli bemenetek, vagyis különböző hibák, például járműsebesség vagy CRC (ciklikus redundancia-ellenőrzés) problémák, melletti működést vizsgálják. Az extrém, megterhelő környezeti feltételek vizsgálatára külön tesztek vannak. Jelenleg a thyssenkrupp által használt robusztussági tesztmanőverek két féle módszerrel készültek el. Egyik egy autós szimulátorban számítógépes kormány segítségével történő vezetés során rögzített járműsebesség és kormányszög, míg a másiknál valós autó vezetés során mérőberendezéssel mért adatok alapján. A robusztussági teszt futtatása közben a tesztmérnök által előre meghatározott hibákat, megadott valószínűséggel, véletlenszerűen aktiválja a tesztkörnyezet. Különböző vezetési módokhoz, helyzetekhez több mérés is van, de összesen limitált mennyiségű, így a tesztelhető tesztesetek száma is limitált.

1.3 Probléma definiálása

A tesztelés hatékonyságának növeléséhez a rendszert minél több valóságos munkapontban kell megvizsgálni. A tesztesetek véges számából adódik, hogy ezt csak adott munkapontokban tudjuk megtenni és minden futtatásnál ugyanazokat a munkapontokat vizsgáljuk. Több munkapont tesztelése esetén akár olyan hibát is találhatunk a rendszerben, amit a jelenlegi robusztussági teszttel nem, vagy csak a fejlesztési folyamat során később vettünk volna észre.

A robusztussági teszthez használt valós és szimulátorban végzett mérések számának növelése igencsak erőforrásigényes. Mindkét módszernél a mérések számának jelentős növelése indokolatlanul sok mérnöki munkaórába kerülne. A mérésekkel kapcsolatos esetleges költségektől (valós vezetésnél például üzemanyag) eltekintve is elmondható, hogy nehezen bővíthető a vizsgált tesztesetek száma. Továbbá tárolni is kell

minden ilyen mérést. A robusztussági tesztek szerveren vannak tárolva és ahhoz, hogy a tesztpadban lévő lokális számítógép mindig az aktuális legfrissebb referenciákat futtassa, azokat a szerverről a lokális számítógépre kell másolni. Egy-egy mérés nem foglal sok tárhelyet, viszont folyamatosan növelve a tesztesetek számát a másolás miatt időt is veszíthetünk.

1.4 Feladat célja

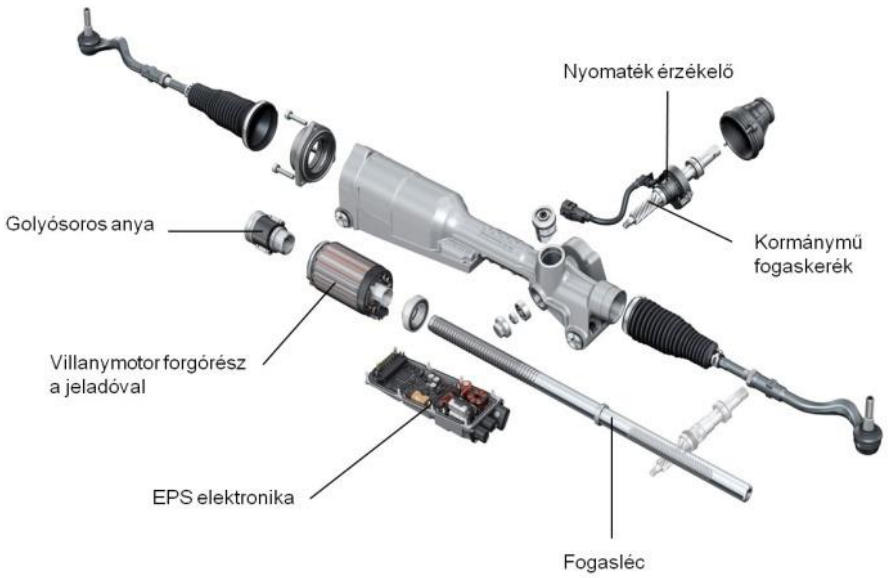
A jelenleg használt, teljes terjedelmében tárolt, állandó értékű referenciák helyett egy algoritmussal fogom elkészíteni a különböző tesztmanővereket. Az algoritmus a tesztmérnök által előre meghatározott munkapont tartományban, pszeudo-random módon, tetszőleges számú kormányszög és járműsebesség referenciát fog generálni. Az algoritmussal tetszőleges számú tesztesetet el lehet készíteni anélkül, hogy a mérnöknek azzal foglalkoznia, időt kelljen töltenie. Továbbá a tesztmanővereket nem teljes terjedelmében kell tárolni, mert a generálás pszeudo-random jellege miatt bármikor újból el lehet készíteni azokat. Az algoritmus biztosítja, hogy minden új tesztmanőver egyedi legyen, ezáltal növelve az adott tartományban vizsgált munkapontok számát és a robusztussági teszt hatékonyságát.

2 Kormányrendszer és tesztkörnyezet bemutatása

2.1 Elektromechanikus szervokormány

A jelenlegi gépjárművek biztonság szempontjából a legfontosabb alkatrészének mondható a kormányrendszer, hiszen ezáltal tudjuk irányítani a járművet. Kezdetekben a kormányzáshoz a vezetőnek kellett a teljes nyomatékot kifejtenie. Az autók súlya miatt felmerült az igény a kormányzás rásegítésére. Több megoldás terjedt el a gépjárművek fejlődése során, a biztonságosabb és kellemesebb vezetési élmény érdekében. Ezekből a thyssenkrupp által is fejlesztett elektromechanikus kormányrendszert fogom bemutatni.

Az elektromechanikus szervokormány esetében az elektromos motor biztosítja a rásegítéshez szükséges nyomatékot. A rásegítés mértékét a vezető által kormánytekeréssel kifejtett nyomaték határozza meg, ami közvetlenül függ többek között a sebesség és a környezeti tényezőktől, mint például az út felülete vagy az oldalszél. A kormányrendszerből elhagyott olajtartály és szivattyú a gépkocsi gyártásakor a hidraulikus kormányszervóhoz képest megtakarítást eredményez, és környezetvédelmi szempontból is jobbnak tekinthető. Az előnyeihez sorolható még az egyszerűbb szabályozás és a jelentős olaj megtakarítás, illetve jól együttműködik a modern ADAS (Advanced Driver-Assistance System) rendszerekkel. Ezek közé sorolható például a sávtartó és parkolóasszisztens [2].



1. ábra: Egy elektromechanikus szervokormány lehetséges felépítése [2]

2.2 Hardware-in-the-loop (HIL) tesztelés

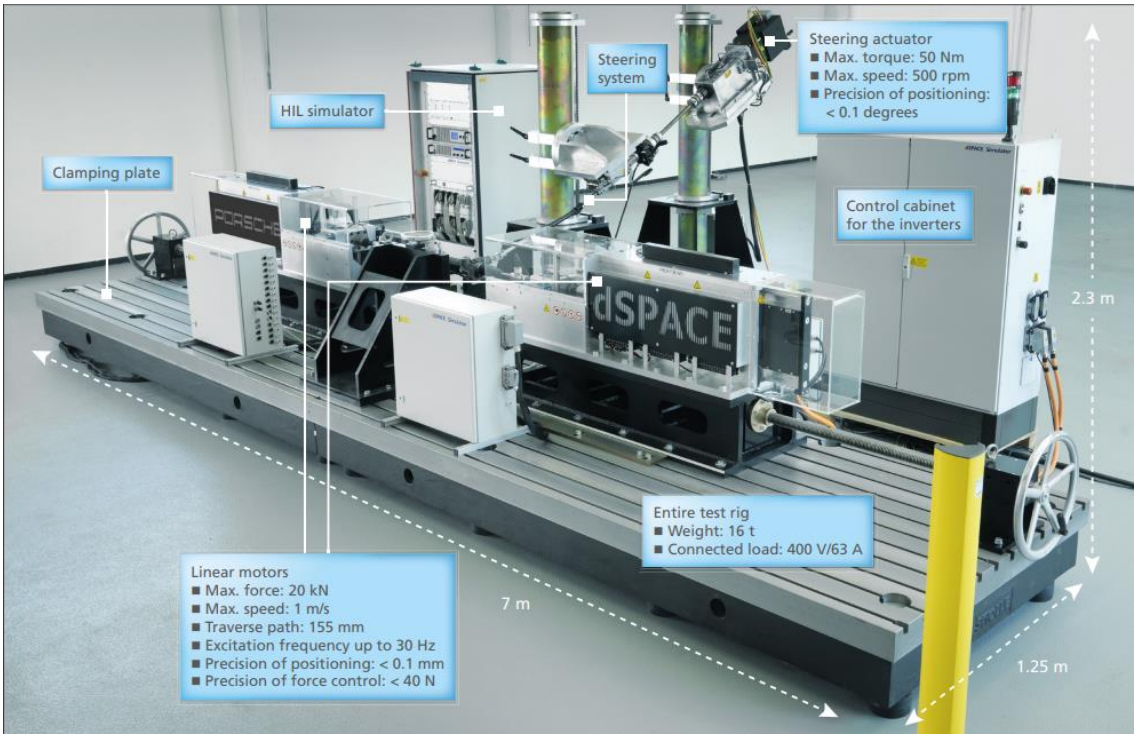
A kormányrendszer fejlesztése során a tesztelés költséges lenne, ha mindig egy teszt autóba beszerelve vizsgálnánk a működését, ezért tesztelés során többek között a Hardware-in-the-loop metódust használjuk. A Hardware-in-the-loop (továbbiakban HIL) egy olyan tesztelési metódus, ahol valós idejű szimulációval helyettesítjük azokat az eszközöket, amelyekkel a beágyazott rendszer interakcióba léphet, ezzel elérve azt, hogy egy adott környezetben úgy lehessen tesztelni, mint ha már integrálva lenne a beágyazott rendszer. Szimulált tesztkörnyezet használatával gyorsabban és költséghatékonyabban lehet tesztelni, illetve egyszerűbb reprodukálni egy-egy tesztesetet. A HIL bár nem váltja ki teljesen a gépjárműves teszteket, sok fejlesztési idejű hibát segít feltárni még a rendszer valós tesztautóba kerülése előtt, ezért a fejlesztési folyamat optimalizálása szempontjából is fontos módszer [3].

Kormányrendszer tesztelésénél a HIL szimuláció része a Rest Bus Simulation (továbbiakban RBS). Az RBS szimulálja a járműkommunikációs busz adatforgalmát, ebből következően pedig a kormányrendszer összes kommunikációs bemenetét is. Ez a szimuláció tartalmaz minden olyan üzenetet, amelyet a működése során fel kell tudnia dolgozni. Ilyen üzenet például a járműsebesség, amit egy külső eszköztől vár a kormányrendszer. Az üzenetek szimulálása lehetővé teszi az üzenetek tartalmának módosítását, amivel tesztelhetjük a kormányrendszer különböző bemenetekre való reakcióját. A korábban említett robusztussági tesztek közben injektált hibákat is ilyen üzenetek keretében küldjük el a rendszernek. Az RBS szimulációval tehát meg lehet vizsgálni és le lehet tesztelni a rendszer működését különböző üzenetekre.

2.3 Tesztpad

A thyssenkrupp-nál a tesztelés különböző szintjeihez egyedi tesztpadok lettek kialakítva. Az algoritmus a rendszerteszt osztály robusztussági tesztjeinek kiegészítéséhez készült, ezért a tesztpad alatt a rendszertesztéléshez használt tesztpadokat értem. A rendszer tesztpadok célja, hogy a rendszer minden bemenetét biztosítsuk (táp, kommunikáció, kerékterhelés, kormánynyomaték) és ezáltal rendszer szinten tudjunk tesztelni. A tesztpadok segítségével a kormányrendszer autóba beépítése nélkül tudjuk elvégezni a tesztelést, ezzel jelentős mennyiségű fejlesztési időt és költséget spórolva.

Egy ilyen tesztpad általános felépítését fogom bemutatni a dSPACE és Porsche által megépített tesztpad segítségével.



2. ábra: Rendszerteszteléshez használt tesztpad [4]

2.3.1 Kormány aktuátor

A 2. ábra egy rendszer tesztpad felépítését mutatja be. „Steering actuator” felirat jelzi a kormány aktuátort, amivel tetszőleges kormányzást tudunk megvalósítani automatizáltan. Elengedhetetlen alkatrésze a tesztpadnak, hiszen ezzel szimuláljuk egy autóvezető kormányzása által kibocsájtott nyomatékot és hozzájárul az automatizált tesztek futtatásának lehetővé tételéhez. Ahogyan az a felsorolt tulajdonságaiból is látható, nagyon pontos, segítve ezzel a tesztek mérését és reprodukálhatóságát.

2.3.2 Lineáris motor

A lineáris motor (képen „Linear motors”) egy olyan elektromos motor, amely egyenes vonal menti mozgást hajt végre. A lineáris motorok az autó kerekeire ható terhelés, például a gyorsításra, lassításra, kanyarodásra vagy az út felülete miatt keletkező rázkódás szimulálására használhatók. A lineáris motorok működését úgy, mint a kormány aktuátort automatizálni és adott teszthez igazítani is lehet.

2.3.3 HIL szimulátor

A 2. ábrán továbbá látható a „HIL simulator”. Ebben a szekrényben találhatóak az olyan eszközök, amelyek a tesztpad működtetéséért, a kormányrendszerrel való kommunikációért és a mérések rögzítéséért felelnek. Néhány fontos elemét fogom csak kiemelni. Ide tartozik többek között a komplex valós idejű HIL szimuláció megvalósítására alkalmazható eszköz, amely csatlakozik a kormányrendszerhez a járműkommunikációs buszon, illetve a szekrényben van még tápegység a kormányrendszer feszültségéének biztosításához és olyan eszközök, amik segítik a mérések végrehajtását. Nem utolsó sorban pedig egy számítógép is található, amin keresztül a tesztmérnök tudja irányítani a tesztpadot.

2.4 Járműkommunikáció

A gépjárműveknél elterjedt kommunikációs protokollok egyike a CAN (Controller Area Network), amely segítségével tudnak a különböző eszközök információt átadni egymásnak. A CAN hálózaton az eszközök egy központi számítógép irányítása nélkül tudnak kommunikálni egymással egy buszon keresztül, ezzel megspórolva több kábelt és megkönnyítve az eszközök diagnosztikáját. A korábban ismertetett RBS szimuláció az autó kommunikációs hálózatától függően például lehet a CAN protokollal küldött üzenetek szimulációja.

2.5 Tesztmanőver felépítése

Egy pszeudo-random generált tesztmanőver elkészítéséhez három jelet kell meghatározni:

- kormányszöget,
- járműsebességet,
- időtengelyt.

A jelek azonos elemszámmal kell, hogy rendelkezzenek. Ezeket egydimenziós tömb formájában adom át a tesztkörnyezetnek. A járműsebességhez és kormányszöghöz is hozzárendelve az időtengelyt, majd ezeket lookup-táblaként feldolgozza a tesztpad vezérlő modell, hogy a tesztpad elvégezze ezek alapján a szabályzást.

3 Perlin-zaj

Egy olyan tesztmanőverhez, ami nem autós mérések alapján készül, én egy zaj algoritmust választottam, mert gyorsan tudok vele tetszőleges számú mérést előállítani. Fontos, hogy a zaj egymást követő értékei között kapcsolat legyen, ez később látható lesz a tesztmanőver elkészítésénél. Ilyen algoritmusokat elsősorban textúrák készítéséhez használnak, mert természetszerű eredményeket lehet velük elérni, ezek közül én a „Perlin-noise" algoritmust választottam az egyszerűsége és az eredményeinek reprodukálhatósága miatt.

3.1 Bevezetés

A Perlin-zajt 1983-ban Ken Perlin találta fel filmek textúráinak javításához, mert zavarta a számítógépes grafikák mesterséges kinézete. Egy olyan véletlenszerű háromdimenziós zajt szeretett volna elkészíteni, aminek segítségével a textúrák természetesebbnek tűnnek. Ken Perlin az algoritmus felhasználási jellege miatt minden zaj elkészülését megismételhetővé tette. A Perlin-zaj algoritmus egyszerűsége és alkalmazhatósága folytán gyorsan elterjedt az iparágban. Később 2002-ben kiadott egy tanulmányt „Improving Noise" néven, amiben optimalizálta a korábbi algoritmus működését [5].

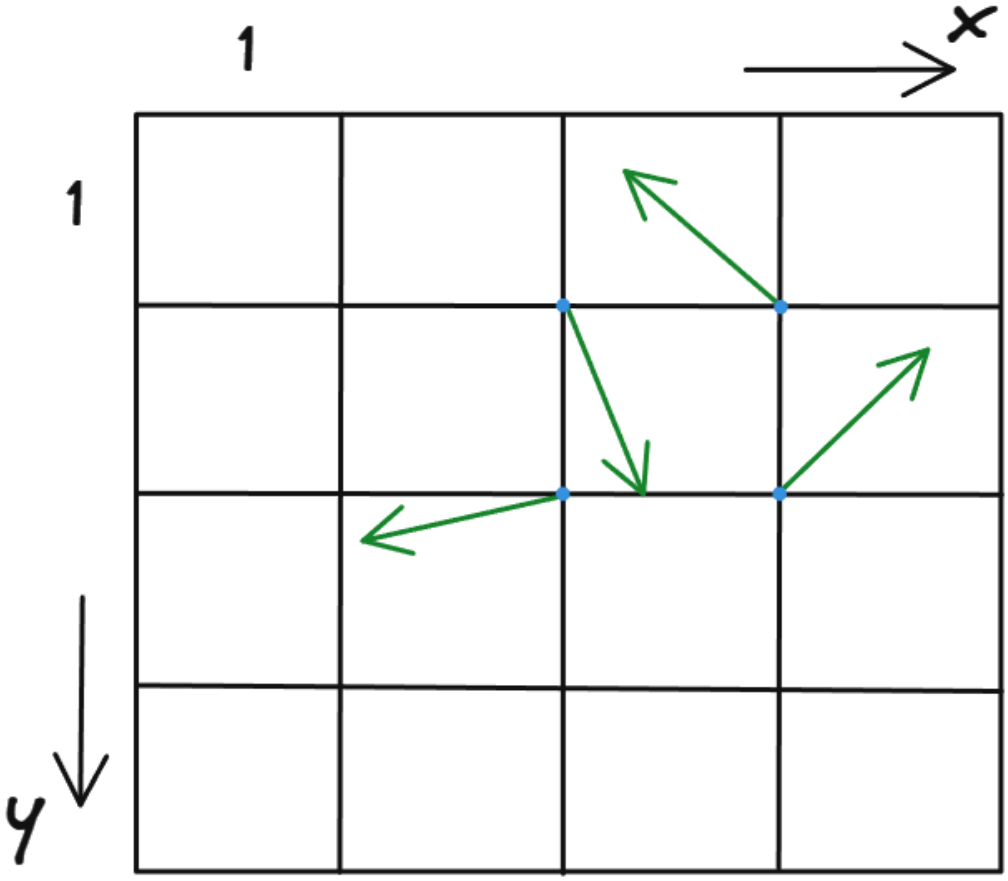
Az algoritmust el lehet készíteni különböző dimenziókban (például 2D, 3D, 4D). Én a kétdimenziós implementációt választottam. Az ennél több dimenziós megvalósítások használatának nem lett volna értelme, mert nagyobb a számítási kapacitásuk és nincsen a feladat szempontjából előnyük. Egydimenziós helyett pedig azért választottam, mert így egy zajból több tesztesetet is tudok készíteni és később könnyebb lesz megismételni ezeknek az újrafuttatását. Ezekért az okokért az algoritmus kétdimenziós megvalósítását fogom bemutatni.

3.2 Algoritmus működése

Képzeljünk el egy tetszőleges hosszúságú négyzetrácsot, ahol a négyzetek mindegyik csúcsához tartozik egy koordináta. Ezen a rácson végig haladva balról jobbra az *x*, míg fentről lefelé az *y* koordináta értéke növekszik. Minden ponthoz, azaz csúcshoz rendelünk véletlenszerűen egy gradiens vektort. A kétdimenziós megvalósításban tehát

minden négyzethez négy darab vektor fog tartozni. Ezeket a vektorokat aztán egységvektorokká kell alakítani. A vektorok iránya fogja meghatározni az elkészült zajt. Ezután végig menve a négyzetrácson, minden ponthoz négy távolságvektort is definiálunk. A távolságvektorok az adott pont és ahhoz tartozó négyzet csúcsainak a távolságát írják le. A távolságvektorok és gradiensvektorok skaláris szorzatából ki tudunk számolni négy értéket. A négy kapott értékből pedig bilineáris interpolációval (távolságsúlyozott átlaguk alapján) számoltam ki a jelenlegi ponthoz tartozó értéket.

Fontos megjegyezni, hogy az elkészült zaj elemszáma nem kell, hogy megegyezzen a négyzetrács csúcsainak, vagyis a gradiens vektorok számával. A Perlin-zaj rugalmassága annak köszönhető, hogy egy tetszőleges nagy négyzetrácsból tetszőlegesen sok elemű zajt lehet készíteni az algoritmus által bejárt és a zajhoz felhasznált gradiens vektorok számának módosításával.

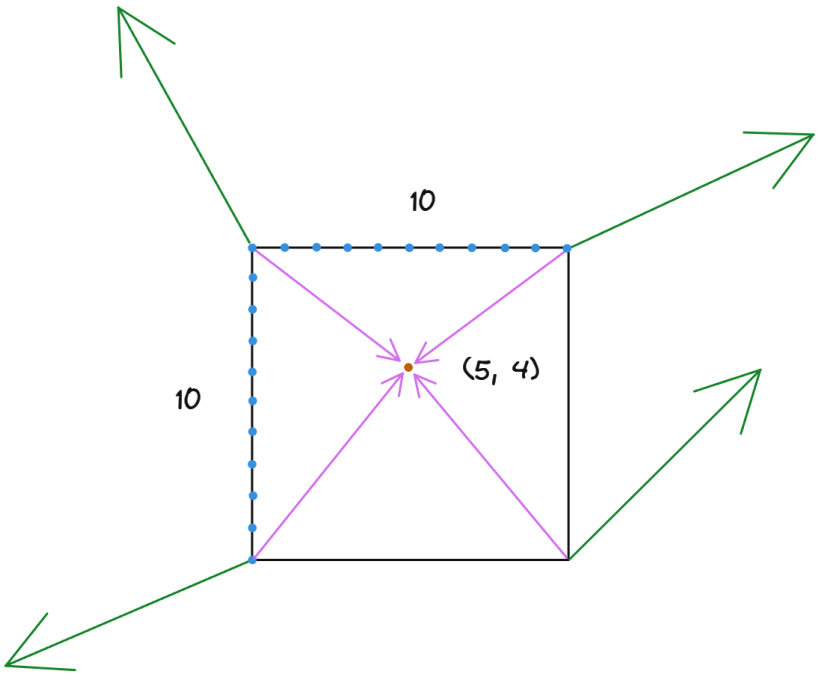


3. ábra: Egy lépésközű négyzetrács illusztrációja négy berajzolt gradiensvektorral

3.2.1 Lépésköz

A 3.2 fejezetben bemutattam a Perlin-zaj algoritmus működését. Látható, hogy az algoritmus egy koordinátapárhoz egy értéket rendel. Ha minden csúcsnak a négyzetrácsban kiszámoljuk az értékét, megjelenítve egy olyan kétdimenziós tömböt kapunk, aminek minden eleme nulla. Ez azért van, mert minden koordináta pontosan a hozzá tartozó négyzet csúcsánál van. Ezért nulla lesz a távolságvektorra és így a skaláris szorzata is.

Eddig egy egység oldal hosszúságú négyzetekről volt szó. De ha egy négyzetet több egységre osztunk fel, tehát minden oldala például 10 egység hosszú lesz, akkor egy száz elemű zajnál és négyzetrácsnál csak az első tíz gradiens vektort használjuk fel és a zaj csak minden tizedik eleme lesz nulla értékű. Ez azt is jelenti, hogy bizonyos időközönként mindig nulla értékes fog felvenni a zaj. A koordinátákat tekintve pedig például a tízes egységhossznál maradva, a (11, 5) már a második négyzethez tartozó első és ötödik elem lesz. Minél nagyobb egységet rendelünk egy négyzethez, egymáshoz annál közelebbi értékeket kapunk. Ebből következik, hogy az egy egységhez minél jobban közelítünk annál durvább, fehér zajhoz jobban hasonlító eredményt kapunk. A következőkben lépésköznek fogom nevezni a négyzetekhez rendelt egységhosszt.



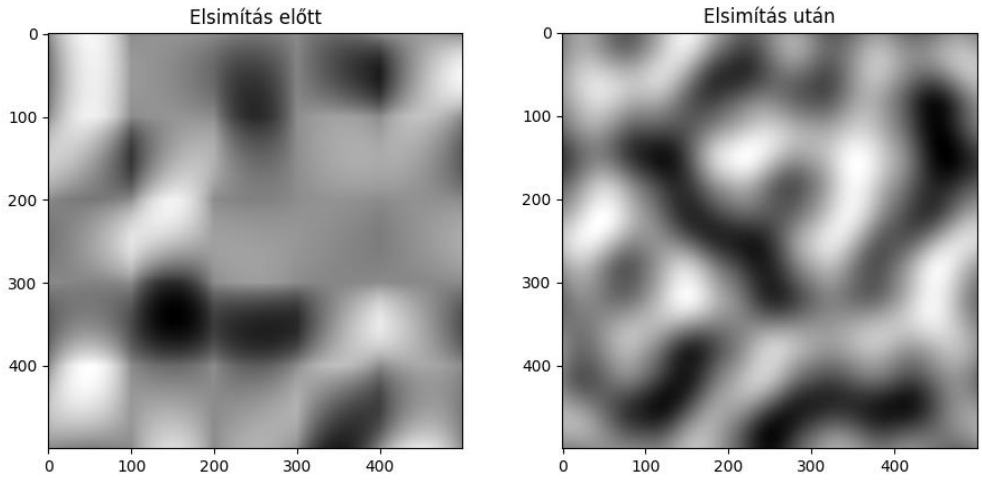
4. ábra: 10 lépésközű illusztráció, egy adott pontban berajzolt gradiens- (zöld) és távolságvektorokkal (lila)

3.2.2 Átmenetek elsimítása

A Perlin-zaj így már majdnem kész, csak egy dolog hiányzik. Jelenleg az élek mentén ugrás van és nem átmenet. Ezt a következő „fade” függvénnyel lehet orvosolni, amit Ken Perlin az 2002-es „Improving Noise” tanulmányában ismertetett [6]

$$6t^5 - 15t^4 + 10t^3$$

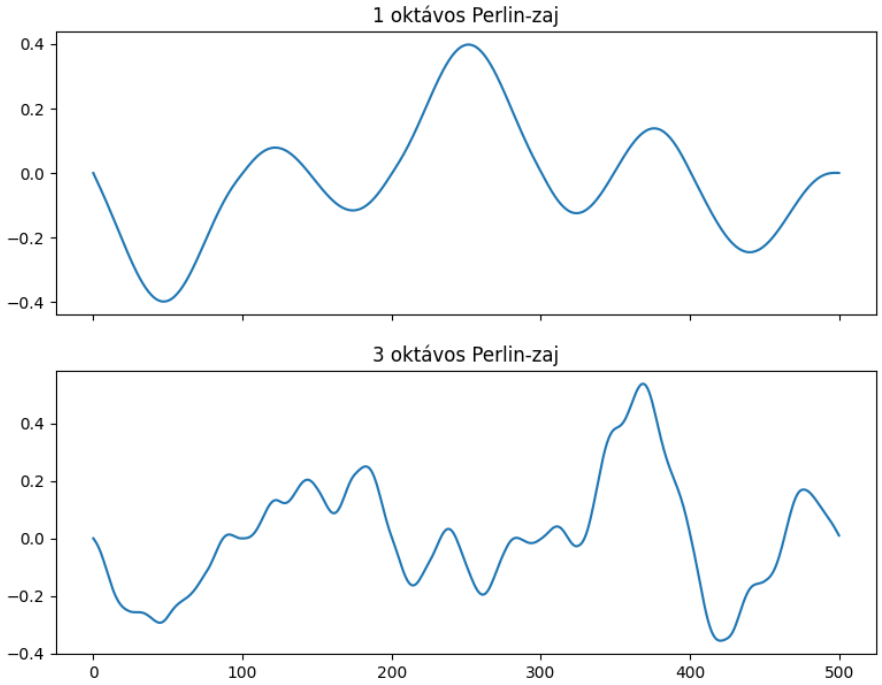
Interpoláció előtt a bal felső távolságvektor koordinátáit kell beilleszteni ebbe a képletbe és aztán az eredményükkel végrehajtani a bilineáris interpolációt. Ahogy az alábbi képen (5. ábra) is látható, használata előtt a négyzetrács négyzetei között nem igazán volt átmenet, utána viszont már nem láthatók a négyzetek határvonalai.



5. ábra: A zaj 2D-s megjelenítése átmenet elsimítása előtti és utáni

3.3 Oktávok

Az oktáv a zaj részletességét határozza meg. A Perlin-zaj algoritmus esetében ez több különböző frekvenciájú zaj összeadását jelenti. Leggyakoribb megoldás, hogy minden oktávnál a lépésközt és amplitúdót a felére csökkentjük, ezáltal növelve a zaj frekvenciáját. A különböző zajokat ezután elég összeadni az oktáv eléréséhez. Az oktávok segítségével el tudunk érni egy sokkal részletesebb zajt, tesztmanőver esetében pedig élethűbb kormány- és járműsebesség referenciát kaphatunk vele.



6. ábra: Egy és három oktávval generált zaj egy szelete

4 Tesztmanőver készítés megvalósítása

A tesztmanőver elkészítését a korábban ismertetett Perlin-zaj algoritmussal fogom megvalósítani. Ez az algoritmus fogja a kormányszög és járműsebesség referenciák alapját adni, amiket aztán módosítani kell a végleges tesztmanőver elérése érdekében.

4.1 Algoritmus igazítása a feladathoz

Annak érdekében, hogy olyan zajt kapjunk, melyből aztán kormányszöget és járműsebességet tudunk előállítani megfelelően fel kell paraméterezni a Perlin-zaj algoritmust. Ebben a fejezetben a zaj által generált elemek számáról, frekvencia jelentőségéről, reprodukálhatóságáról és az algoritmus futási idejéről fogok beszélni.

4.1.1 Gradiens vektorok generálása

A gradiens vektorokat pszeudo-random generálással fogom létrehozni, a reprodukálhatóság érdekében. A pszeudo-random szám generálás lényege, hogy statisztikailag véletlenszerűnek látszanak a kimenetei, de közben determinisztikus és megismételhető a működése. Az ilyen algoritmusoknak egy kezdőállapotot kell megadni, amit „seed-nek” nevezünk. Azonos kezdőállapotra sorrendben és értékben is megegyező véletlen számokat fogunk kapni. Tesztelésnél fontos, hogy a teszt procedúra megismételhető legyen. Például egy hiba előfordulásánál meg kell nézni, hogy reprodukálható-e, a kiváltó ok megértésénél is fontos, továbbá a hiba kijavítása után is érdemes ellenőrizni, hogy megismétlődik-e még a probléma.

A pszeudo-random számgeneráláshoz az egyik legelterjedtebb, tudományos számításokhoz használható Python könyvtárat a NumPy-t választottam. Ez a könyvtár rendelkezik pszeudo-random generálást megvalósító függvényekkel. A függvények közül én O’Neill permutációs kongruenciális generátorának a 128 bites implementációját, a PCG64-et használtam [7]. A Perlin-zaj algoritmus azonos gradiens vektorokra ugyanazt az eredményt adja, tehát ezzel generálva a gradiens vektorokat elég csak egy számot, a „seed-et” elmenteni az összes vektor helyett. Ezt a tesztmanőver készítés pillanatában az aktuális Unix idő (1970 január 1. óta eltelt másodpercek száma) másodpercre kerekítése alapján határozom meg, ezáltal garantálni tudom, hogy minden tesztmanőver egyedi

seed-et kapjon. Minden elkészült tesztesethez használt seed a tesztmanőverekkel együtt tárolva lesz, későbbi újra futtathatóság érdekében.

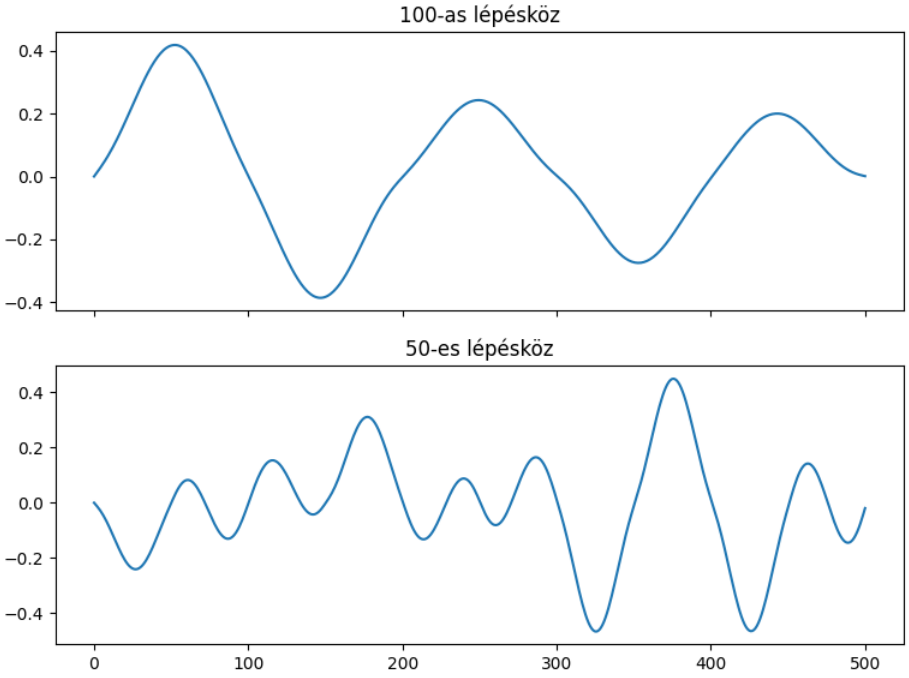
A feladathoz egy 512x512 hosszú négyzetrácsot és ahhoz tartozó gradiens vektorokat generáltam. Gyakorlatban ez két 512 széles és 512 hosszú tömböt jelent, ahol az egyik tömb a gradiens vektorok *x* míg a másik az *y* koordinátájukat reprezentálja. Ezt a két tömböt aztán feltöltöttem véletlenszerű számokkal -1 és 1 között. Bármilyen sok gradiens vektort lehet készíteni, viszont arra figyelni kell, hogy ha a lépésköz túl kicsi és sok elemű zajt szeretnénk kapni, akkor előfordulhat, hogy nem lesz elegendő gradiens vektor a kért elemszám megvalósításához. Ez túlindexelési hibához vezet, amit megoldhatunk a gradiens vektorok számának növelésével vagy maradékos osztás használatával a hossznál nagyobb indexekre. Utóbbi megoldásnál amint túllépjük a maximum hosszúságot a generált zaj is újra fog kezdődni, mert ugyanazokra a koordinátákra ugyanazokkal a gradiens vektorokkal lesz legenerálva. Azért ilyen hosszú gradiens vektor tömböket készítettem, mert a későbbi futtatások alapján ez volt a nagyság, ahol még nem kellett ismétlődő zajt használnom.

4.1.2 Lépésköz meghatározása

Tesztmanőver készítéshez fontos a lépésköz helyes megválasztása, mert nagy hatással van a zaj egymást követő elemei közötti változás mértékére, ezáltal meghatározza például kormányszög esetében, hogy a kormányszög elfordulásának sebességét. Ezért a lépésközt az alapján kell meghatározni, hogy milyen gyors tekerést szeretnénk.

Minden tesztmanőverhez társítok egy időtengelyt is, amivel megadom, hogy milyen időközönként kell a kormányszög és járműsebesség referenciának egy-egy értékét elvégeznie a tesztpadnak. Az időtengely meghatározásának segítségével a feladathoz tudunk igazítani egy túl nagy frekvenciájú zajt is. A teszt másodpercben mért hossza egyenesen arányos a lépésköz nagyságával. Azonos frekvencia elérése érdekében kétszer hosszabb tesztnél a lépésközt is a kétszeresére kell növelni.

A 7. ábrán látható különböző lépésközzel generált zajok egy-egy szelete azonos időtengellyel. Megfigyelhető a zaj frekvenciájának változása a lépésköz felezésénél.



7. ábra: 100-as és 50-es lépésközzel generált zaj egy szelete

4.1.3 Zaj elemeinek száma

A zaj elemeinek számát az határozza meg, hogy hány koordináta pontra számoljuk ki az algoritmus értékét. Mivel a gradiens vektorok számát és a lépésközt is tetszőlegesen állíthatjuk, ezért tetszőlegesen hosszú zajt készíthetünk. Természetesen ahogy az elemek száma, úgy a számítási kapacitás is exponenciálisan növekszik. Fontos megkülönböztetni a generált zaj elemeinek számát a teszt hosszától. Az ideális elemszámot itt a lépésköz és a gradiens vektorok száma alapján kell beállítani, figyelembe véve a korábban említett dolgokra.

4.1.4 Oktávok meghatározása

Az oktáv a negyedik olyan változó, amivel a kimenet frekvenciáját módosítani lehet. Az oktávok az én implementációmban a méréseim szerint, körülbelül kétharmaddal növelik a zaj frekvenciáját. Két oktávnál két különböző amplitúdójú és lépésközű zajt kell legenerálni, ezért a számítási kapacitás is nő. Minden egyes oktáv a kétszeresére növeli a zaj elkészülésének idejét. Ezért meg kell fontolni mennyi oktávot érdemes használni. Ha az oktávok növelésével párhuzamosan a lépésközt nem növeljük, akkor megtörténhet a korábban említett túlindexelés is, mivel mindegyik oktáv lépésköze az előzőének a fele, így ki lehet futni a gradiens vektorokból.

A Perlin-zaj különböző paramétereinek kipróbálása közben azt állapítottam meg, hogy nem érdemes öt oktávnál többet alkalmazni, mert öt oktáv felett marginális lesz a hozzáadott értéke tesztmanőver szempontjából és közben megnő a futási idő is. Nagyobb frekvenciájú zaj esetén kevésbé meghatározó a sok oktáv, így ott elég ötnél kevesebbet alkalmazni. Az oktávot tehát érdemes használni, a zaj frekvenciáját figyelembe véve. Különböző oktávok használhatóak továbbá különböző tesztesetek megvalósításához, ezért ezt a tesztmérnöknek van lehetősége állítani.

4.1.5 Futási idő csökkentése

A gradiens vektoroknál és az oktávoknál is említettem az algoritmus futási idejét. Egy oktáv esetén például egy 10 000 x 10 000 elemű zaj generálása körülbelül tíz percet vesz igénybe. Ahogyan azt az előző fejezetben is említettem, két oktávnál ugyanezt ennek a kétszerese legenerálni. De már egy 5 000 x 5 000 elemű zaj is két percig fog tartani, ami négy oktávnál nyolc percre növekedne.

Mint korábban említettem a Perlin-zaj algoritmust kétdimenzióban implementáltam, viszont tesztmanővert elég egydimenziós zajból készíteni. Ezért egy 5 000 x 5 000 elemű zajból elég csak egy darab tesztmanőverhez elegendő 5 000 elemű szeletet felhasználnom. Kiegészítettem tehát az algoritmust egy olyan paraméterrel, amivel tetszőleges számú sort lehet kihagyni a generálásból.

A bemutatott megoldással minden kiválasztott szelet után kihagyok tetszőlegesen sok egyébként felhasználható szeletet. Azért nem lesz jó, ha csak az első három szeletét használom fel, mert a Perlin-zaj értékei között mindkét irányban kapcsolat van. Lépésköztől függően bizonyos szeletig ugyanazokat a gradiens vektorokat használja az algoritmus minden szelet kiszámításához. Továbbá az adott pont koordinátája határozza meg a távolságvektorokat, ezért egy koordinátában közeli pontnak az értéke is hasonló lesz. Ha tehát egymás után lévő szeleteit vesszük ki a zajnak, akkor a szeletek közötti korreláció nagy lesz. Minél nagyobb a lépésköz annál közelebb lesznek egymáshoz a távolságvektorok és annál több szeletet kell kihagyni. Az egymásra hasonlító zajok pedig a járműsebesség referencia elkészítésénél problémát jelentenének.

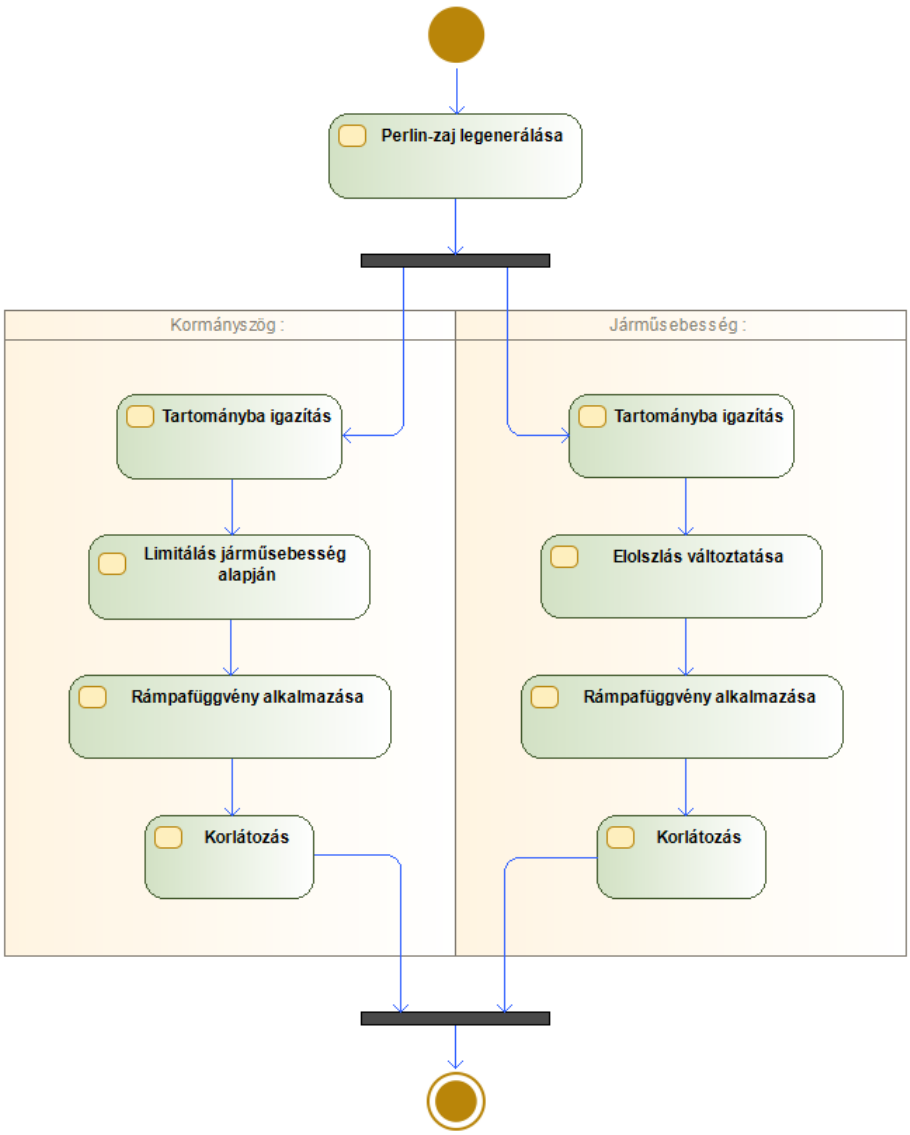
A robusztussági tesztek általában éjszaka kerülnek futtatásra. Több óra futtatás alatt különböző munkapontok tesztelésére is van idő. A robusztussági tesztek ezen tulajdonságát figyelembe véve egy tesztmanőverhez szükségesnél több szeletet fogok

felhasználni a legenerált zajból. Az elkészíthető tesztesetek száma tehát a zaj elemszámától és a kivett szeletek számától függ.

A zaj csak adott szeleteinek generálásával drasztikusan csökkent az algoritmus futási ideje. A korábbi egy oktávos 10 000x10 000-es példa így 10 perc helyett, csak minden 100. elemet generálva 5 másodperc alatt elkészül. A 100 darab 10 000 elemű zajból pedig akár 33 különböző tesztesetet is lehet készíteni.

4.2 Tesztmanőver készítése a generált zaj felhasználásával

A tesztmanőver készítő algoritmus működésének megértéséhez készítettem egy aktivitás diagrammot. A továbbiakban ezeket a műveleteket fogom bemutatni.



8. ábra: Aktivitás diagram a tesztmanőver készítő algoritmus működéséről

4.2.1 Zaj tartományba igazítása

A Perlin-zaj által generált értékek értékkészlete -1 és 1 között lesz. Ahhoz, hogy ebből kormányszög vagy járműsebesség legyen, előbb meg kell változtatni az értékkészletét. Megfelelő tartományba igazítani a referenciákat azért fontos, hogy ne fordulhasson elő olyan szituáció, ahol például nagyobb a kormányszög, mint a fizikai végpozíciója a tesztelendő kormányrendszernek. Ezek a határok a kormányrendszer mechanikai kialakításából adódnak, így kormányrendszerenként változnak.

Fontos tehát ezeknek a határoknak a betartása a végpozícióra felfeszülés és a valóságtól elrugaszkodó tesztmanőverek elkerülése érdekében. A végpozícióra felfeszülés akkor történik, amikor eléri a kormányrendszer a fogasléc egyik végét. Ilyenkor mechanikailag nem tud a rendszer tovább menni az adott irányba. Amikor végpozíciónál nagyobb kormányszöget kérünk ki a kormány aktuátortól, akkor elkezdjük felfeszíteni a végpozícióra a rendszert, mert a kormány aktuátor próbálná elérni a kívánt fokot, de a rendszer a fizikai határa miatt nem tudja. Nagy erővel próbálkozik a kormány aktuátor, ami akár kárt is okozhat a tesztpadban vagy a rendszerben. Ha a kormányszög tekerési tartományánál nagyobb értéket adunk meg, mint azt fizikailag képes elérni, akkor megtörténhet, hogy egy tesztmanőver során sokszor felfeszül a végpozícióra. Valós vezetés során viszont ez ritkán történik meg és ezt a szituációt a thyssenkrupp-nál külön teszt vizsgálja, tehát nem feladata a robusztussági tesztnek az ilyen működés vizsgálata.

A járműsebességet szimulációs modell álltja be a korábban bemutatott CAN protokoll segítségével. A szimulációs modell küld egy üzenetet (autónként változik, hogy ezt az üzenetet egyébként honnan kapná a rendszer), a kormányrendszer elfogadja azt járműsebességnek és annak függvényében számolja például a kiadható nyomatékot. A szimuláció ellenére így is van fizikai hatása a rendszerre. Ezt a később bemutatásra kerülő járműsebesség függő kerékterhelés szimulációnál lehet majd látni. Ezért mindkét jelnél fontos lesz a megfelelő tartományba igazítás. A két referenciára a tartományba igazítást különböző módon valósítottam meg.

A kormányszög igazítása egyszerűbb volt, mert negatív és pozitív kormányszög értékek is előfordulnak vezetés során. A tesztmérnöknek elég megadnia a kormányszögben értelmezett végpozíciót, amiből az algoritmus beállítja a kormányszög referencia értékkészletét. Ehhez elég csak felszorozni a meglévő zajt. A zaj értékei a Perlin algoritmus működése miatt csak ritkán érik el a határértékeiket. Tesztelendő

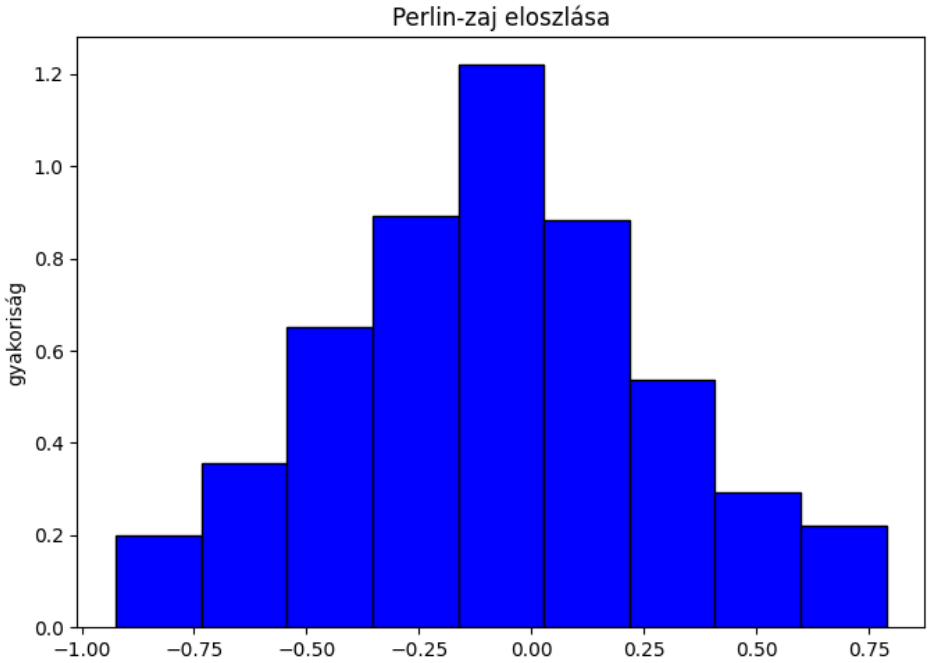
munkapont tartománytól függően nem célszerű mindig csak a maximális kormányszöghöz igazítani a referenciát. Például egy valódi autópályás vezetés során nagy járműsebesség mellett valószínűtlen, hogy a 200 fokot elérje a kormányszög. Nagy járműsebesség mellet túl nagy kormányszögnél, a jel későbbi korlátozása miatt teljesen elveszítené az eredeti formáját, ezáltal pedig egy valóságtól elrugaszkodó tesztmanővert kapnánk.

A járműsebességet a megadott munkapont tartománytól függően állítottam be. A tesztmérnök adhatja meg milyen járműsebesség tartományban szeretne tesztmanővert készíteni. Ennek a megvalósítása nehezebb volt, mert a negatív értékek értelmezhetetlenek a járműsebesség szempontjából, illetve itt konkrét értékek közé kell a zajt igazítani, hogy a tesztmérnök ténylegesen azt a munkapont tartományt tesztelje, amit megadott.

Korábban említettem, hogy a Perlin-zaj algoritmus ritkán éri csak el a határértékeit, viszont a járműsebesség tartomány beállításához szükséges, hogy minél közelebb legyen a zaj a határértékekhez. Ezt a következőképpen oldottam meg. Megnéztem a zaj minimumát és maximumát, és a határértékhez közelebb levő értékből kiszámoltam mennyivel kell megszorozni a zajt, hogy pontosan elérje azt. Ezáltal, a minimum vagy maximum értéke a zajnak pontosan eléri a Perlin-zaj algoritmus egyik határértékét, ami -1 vagy 1 és mindeközben a másik is közelebb lesz a határértékéhez. Ezután 0 és 1 közé igazítom a zaj értékkészletét majd kiszámolom a megadott járműsebesség tartomány értékei közötti különbséget. Ezzel felszorozva és a tartomány alsóhatárát hozzáadva igazítom a kért tartományba. Végül pedig úgy igazítottam a zaj szélsőértékeit, hogy azok egyenlő távolságra legyenek a megadott tartomány értékeitől. Ennek a procedúrának a segítségével a megadott értékek és a zaj értékei között csak minimális eltérés lesz.

4.2.2 Eloszlás változtatása

A Perlin-zaj eloszlása normális eloszláshoz hasonlít. Az értékkészletéből és a már korábban említett tulajdonságaiból következik, hogy az átlag, vagyis az eloszlás központi része mindig 0-hoz közelít.



9. ábra: Perlin-zaj eloszlása

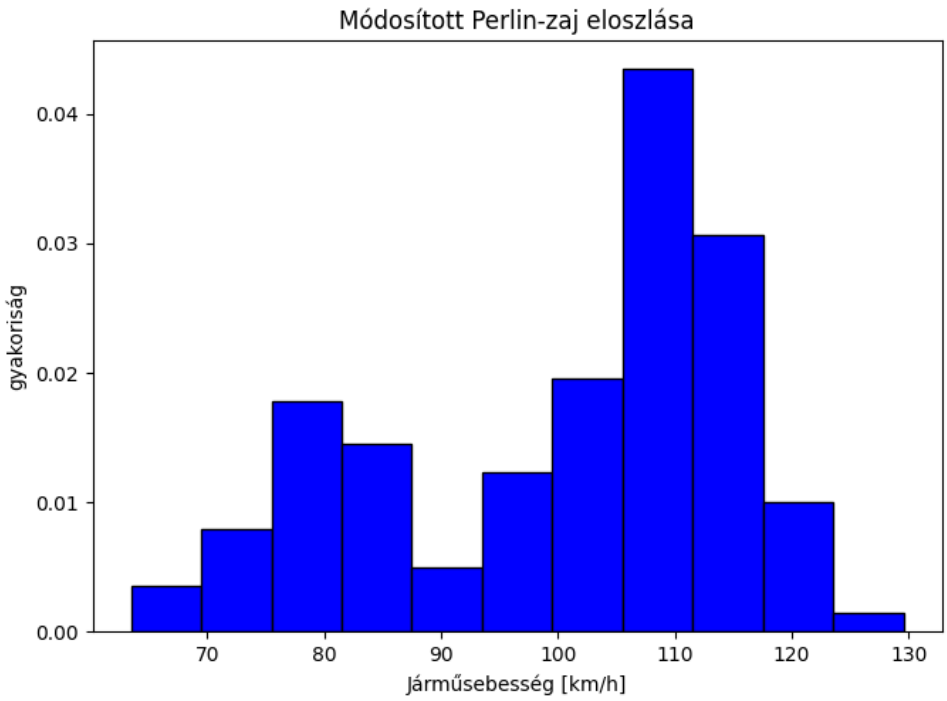
Ez ugyan a kormányszögnél egy jó tulajdonságnak mondható, mert a középpont ennél is nulla, tehát a normális eloszlás nulla körül ingadozó értéke vezetés szempontjából valóságos, de a járműsebességnél már nem ideális. A járműsebesség vezetési szituáció függő. Például a városi közlekedésnél más lesz egy autó átlagos járműsebessége, mint az autópályán. A különböző vezetési szituációkat a teszteléshez érdemes teljesen lefedni. Városi vezetésnél például a közlekedés nagyrészét egy adott járműsebesség tartományon belül tölti az autó, viszont piros lámpánál, zebránál, forgalmi dugónál mind le kell lassítani. Azért, hogy az ilyen a vezetés során ritkábban történő események is figyelembe vegyük meg kell változtatni a Perlin-zaj eloszlását.

A különböző vezetési módoknál normális eloszlással nem lehet teljesen lefedni egy-egy munkapont tartományt. A valósághoz közelebbi működés érdekében ezért a Perlin-zaj eloszlást a megadott munkapont tartomány szerint kell beállítani. A

tesztmérnök így meghatározhatja, hogy egy adott tesztesetnél milyen gyakran forduljanak elő a járműsebesség értékei.

Az eloszlás tetszőleges beállításához két olyan jelre van szükség, amelyek egymástól eltérő tartományban vannak. Ha két különböző értékkészlettel rendelkező jelet kombinálunk, akkor elérhetünk egy normális eloszlástól eltérő jelet. Például, ha van egy jel 60 és 100 között és egy másik 90 és 130 között, akkor az ezekből készült jel eloszlását úgy módosíthatjuk, hogy az elsőnek és a másodiknak is csak egy részét használjuk fel. A jelek felhasználásának aránya meghatározza, hogy milyen eloszlású lesz az elkészült jel.

Az eloszlás módosításának megvalósításához a tesztmérnöknek meg kell adnia a két járműsebesség tartomány határértékét és azt, hogy milyen arányban szerepeljenek az elkészült jelben. A két jel közötti átmenetet egy 0-tól 1-ig száz értéket felvevő erősítő függvény segítségével valósítom meg. Azért száz elemű, mert a két függvény között így lesz egyenletes az átmenet. A meghatározott aránynál az egyik jelet az erősítő függvénnyel, míg a másikat az inverzével szorzom, majd összeadom a kapott két értéket. Ezáltal a két jelet úgy tudom kombinálni, hogy ne legyen hirtelen értékváltozás az átmenet helyén. A végeredmény egy olyan járműsebesség referencia, aminek az eloszlását a tesztmérnök bemenetei alapján állítottam be.

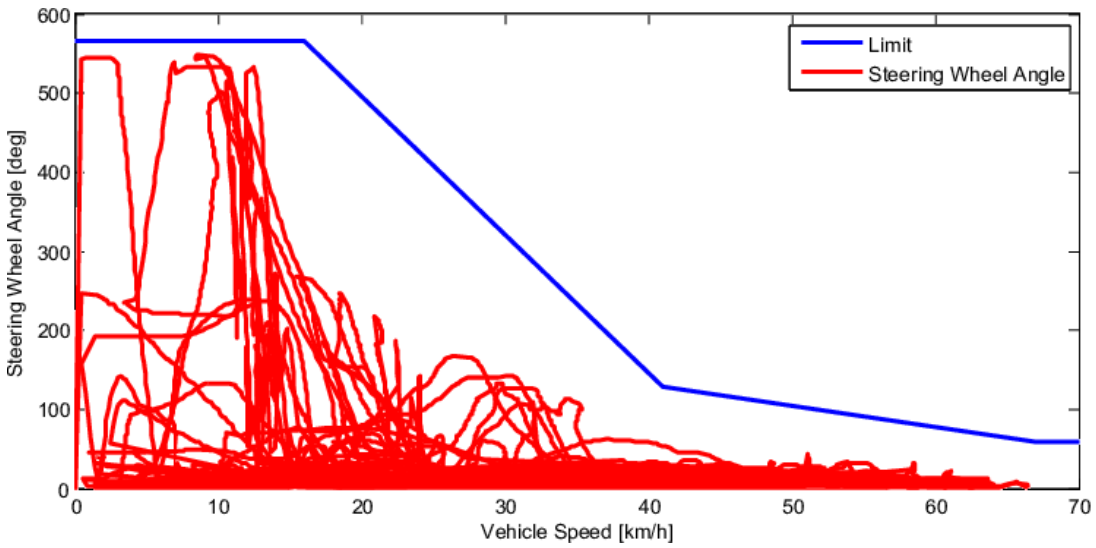


10. ábra: Módosított Perlin-zaj eloszlása (30% 60-100, 70% 90-130)

4.2.3 Kormányszög és járműsebesség kapcsolata

Ha a kormányszöget és járműsebességet végig külön kezelnénk akkor a végén olyan eredményeket kapnánk, amelyek bár külön-külön helyesek lennének, együtt nem felelnének meg egy élethű autóvezetésnek. A kormányszög egy adott pillanatban lehet például a végpozíció közelében, míg a járműsebesség referencia szerint éppen gyorsul az autó. Nagyon sok helyzet alakulhat ki, ahol a kettő referencia egy autó teljesen más működését sugallja. Egy tesztmanőver valóságtól eltérése mellett pedig fizikai kárt is okozhatna a járműsebesség függő kerékterhelés miatt. Ezért össze kell kapcsolni a kettőt, hogy legyen hatásuk egymásra és a tesztmanőver összeségében realisztikus legyen.

Az összekapcsoláshoz egy olyan tanulmányt vettem alapul, ahol a kormányszöget mérték adott járműsebességeknél városi közlekedés során. A mérések alapján egy limitet határoztak meg a kormányszögre, ami alatt az autóvezető még kontrollálni tudja az autót. Ezt a limitet aztán egy tesztpályán verifikálták [8].



11. ábra: A tanulmányban mért adatok, kormányszög járműsebesség függvényében [8]

A 11. ábrán piros színnel látható a kormányszög különböző járműsebességek függvényében, kékkel pedig a megállapított limit. A kormányszög elég gyorsan elkezd csökkeni járműsebesség növekedésnél.

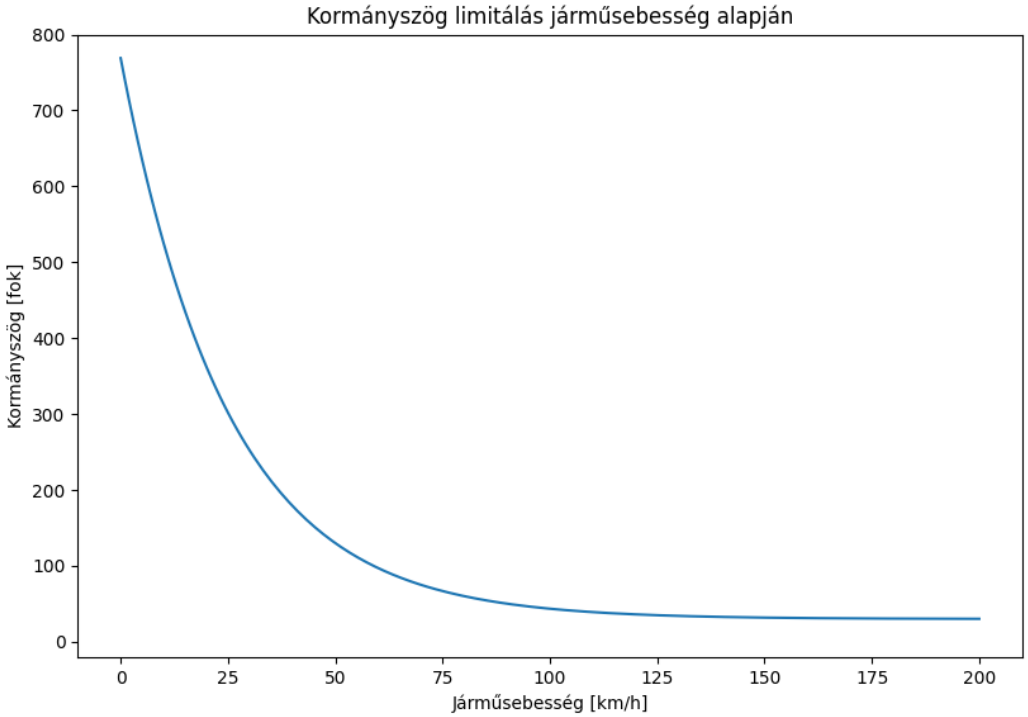
Bár a tanulmányban egyenes vonalak mentén határozták meg a kormányszög limitet, én egy exponenciális függvényt fogok használni. Ahogyan az a képen is látható egy exponenciális függvénnyel pontosabb limitet lehet meghatározni, mert pontosabban közelíti a mérésben megfigyelt működést. Az alábbi függvényt készítettem el, a

tanulmány mérései alapján a kormányszög limitálásához. A harmincat azért adtam hozzá a végén, hogy nagy járműsebességeknél se legyen túl kicsi a kormányszög az exponenciális függvény miatt.

$$f(x) = e^{\left(1-\frac{x}{50}\right)\times 2} \times 100 + 30$$

Az algoritmusban a már tartományba igazított kormányszöget a megváltoztatott eloszlású járműsebesség alapján limitálom. Mindkettő jel ugyanolyan hosszú, ezért csak azt kell megvizsgálni, hogy egy adott pillanatban mi éppen az értékük. Megnézem, hogy a kormányszög abszolút értéke nagyobb-e, mint a függvény értéke a járműsebességet beillesztve. Ahol nagyobb, ott a függvény eredményére limitálom a kormányszöget.

A függvény használatával a járműsebesség fogja meghatározni, hogy egy adott pontban maximum milyen értéket vegyen fel a kormányszög, összekapcsolva ezáltal a két jelet. A tanulmány mérési eredményei alapján meghatározott limit pedig biztosítja a kormányszög és járműsebesség referencia valósághű működését.



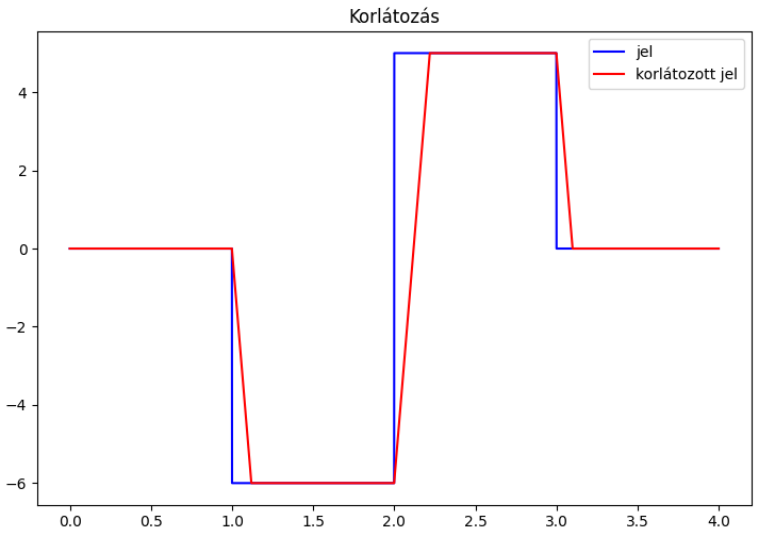
12. ábra: kormányszög limitálás járműsebesség függvényében

4.2.4 Kormányszög és járműsebesség korlátozása

A 4.2.3 fejezetben bemutattam a kormányszög és járműsebesség kapcsolatát, ami biztonságosabbá és járműdinamikai szempontból is valóságossá tette a tesztmanőver futtatását. Előfordulhat viszont még egy probléma a zaj generálása és módosítása közben. A kormányszög és járműsebesség tartományba igazítása után megeshet, hogy túl nagy lesz a gradiens. Túl nagy gradiens esetén pedig a tesztmanőver eltérne egy valósághű autóvezetéstől, illetve a legrosszabb esetben kárt is tehetne a tesztpadokban. Fontos ezért, hogy a gyorsulás és a kormány szögsebesség is korlátozva legyen. Ennek megoldásához szükség van egy olyan függvényre, amely tetszőleges szögsebességre és gyorsulásra tud korlátozni.

A korlátozó függvény megvizsgálja egy adott pillanatban a jel és az eggyel előtte lévő már korlátozott érték abszolút különbségét. Ha az előre meghatározott korlátnál magasabb ennek az eredménye, akkor a korlátot állítja be a jel értékének. Ellenkező esetben csak visszatér az adott pillanatban vizsgált jel értékével. A függvényt a jel második elemétől indítom és mindig a függvény legutolsó elemét használom a különbség megállapításához.

Kormányszög és járműsebesség referenciára is alkalmaztam a korlátozást. Járműsebességnél egy átlagos autó gyorsulása szerint határoztam meg, ez körülbelül 10 km/h másodpercenként. Kormányszögnél ezt másodpercenként 800 fok-ra állítottam. Ezek a korlátok természetesen egyszerűen változtathatók későbbi igények szerint.

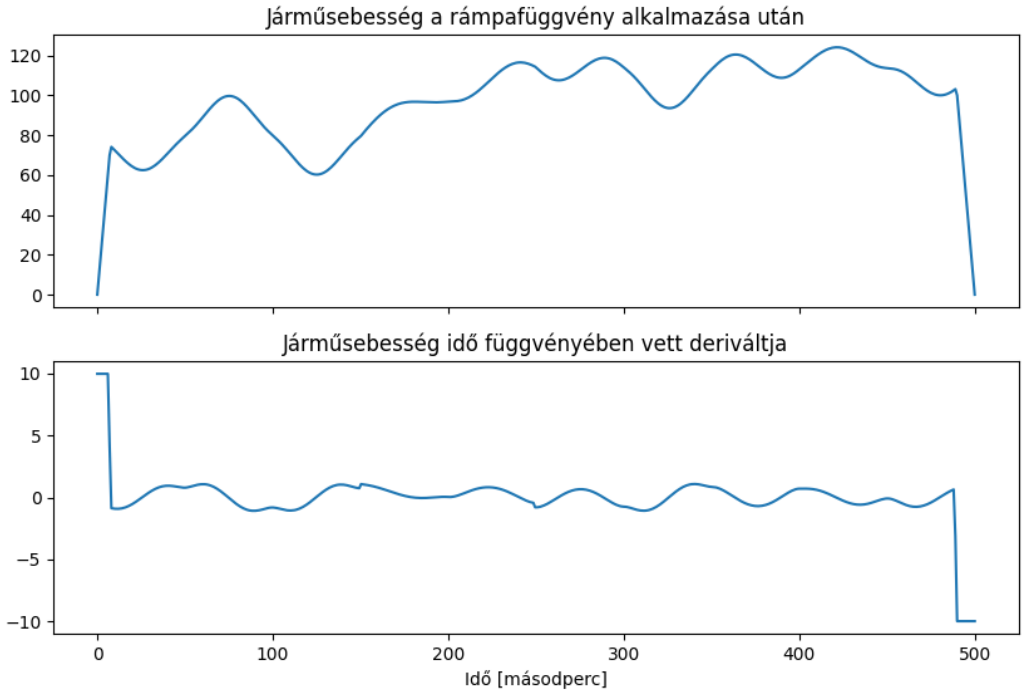


13. ábra: A korlátozás működését bemutató ábra

4.2.5 Rámpafüggvény alkalmazása

Egy jel tartományba igazítása után nem feltétlenül indul vagy fejeződik be nullával. A járműsebesség tetszőleges tartományba igazítása miatt az jel első és utolsó eleme általában nem lesz közel a nullához. Nem valósághű működés viszont például 50 km/h-ról indítani egy tesztmanővert. Állásból nem indulhat el egy autó 50 km/h-val, fokozatosan gyorsít, hogy elérje a kívánt sebességet. Hasonlóan nem realisztikus, hogy egy autó 50 km/h sebességnél hirtelen megálljon. A tesztmanőver referenciáin ezért egy rámpafüggvényt fogok alkalmazni.

A rámpafüggvény a rámpára hasonlító alakjáról kapta az elnevezést. A meredekségét a 4.2.4 fejezetben említett korlátok alapján határoztam meg. A referenciák első jelét átállítom nullára, majd a korlátozó függvényt alkalmazva megkapom az indulási sebesség eléréséhez szükséges rámpát. A jelet aztán megfordítva és ugyanezt végrehajtva megkapom az autó megállásához használhatót is.



14. ábra: járműsebesség a rámpafüggvény alkalmazása után

5 Eredmények kiértékelése

5.1 Kerékterhelés

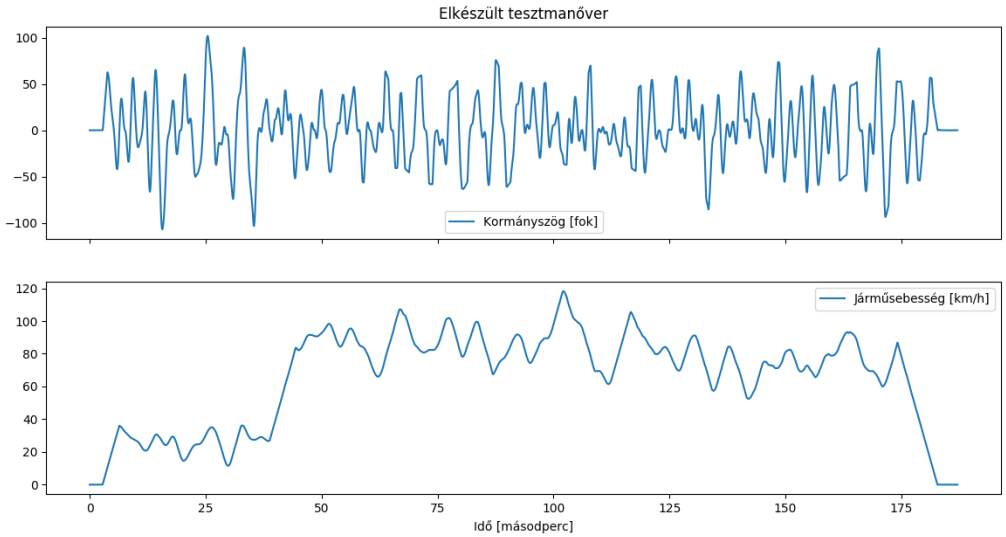
Az elkészült tesztmanővert kerékterhelés modellel együtt érdemes futtatni, hogy a valósághoz közeli terhelés alatt legyen a rendszer egy-egy munkapont tartomány megvizsgálása közben. A thyssenkrupp-nál a kerékterhelés modell valós autós mérések alapján készül a különböző tesztelendő rendszerekhez. Meghatározza, hogy adott járműsebességnél és fogasléc pozíciónál mekkora legyen a terhelés.

5.2 Elkészült tesztmanőver

Amennyiben a kormányszög és a járműsebesség jel minden hozzájuk rendelt módosításon átesetek (ez követhető a 4.2 fejezet aktivitás diagramján), akkor elkészült a végleges kormányszög és a végleges járműsebesség referencia. A futtatásához még egy jelet kell biztosítani a tesztkörnyezetnek, az időtengelyt. A járműsebességhez és kormányszöghöz is hozzárendeljük a meghatározott időtengelyt (azonosat, ezért közös lesz az időtengelyük), majd ezeket lookup-táblaként be kell tölteni a tesztpad vezérlő modellbe, hogy a tesztpad elvégezze ezek alapján a szabályzást.

A tesztmanőver a jelenlegi tesztkörnyezetbe integrálása nélküli egyszerű futtatáshoz egy mat kiterjesztésű fájlba csomagoltam az időtengelyt és a referenciákat, a „scipy” Python könyvtár segítségével. A mat kiterjesztést azért választottam, mert a meglévő robusztussági tesztek is ezt használják. A teljes integrációt követően a köztes mat fájl generálás elhagyhatóvá válik majd, mert az algoritmus közvetlenül be tudná tölteni a lookup-táblát a tesztpad vezérlő modellbe.

A 15. ábrán látható egy végleges három perces tesztmanőver, felül a kormányszög, alul pedig a járműsebesség referencia. A teszt első negyedének a 0 és 50 km/h közötti járműsebesség tartományt adtam meg, míg a maradéknak 50 és 120 km/h közöttit. Ez egy két oktávos 150 és 200 fok/másodperc kormány szögsebesség közötti tesztmanőver. Mind a két referencia egy 5000 elemű zajból lett kiválasztva, majd módosítva. A korlátokat és a limitálást pedig a már korábban ismertetett értékek alapján alkalmaztam.



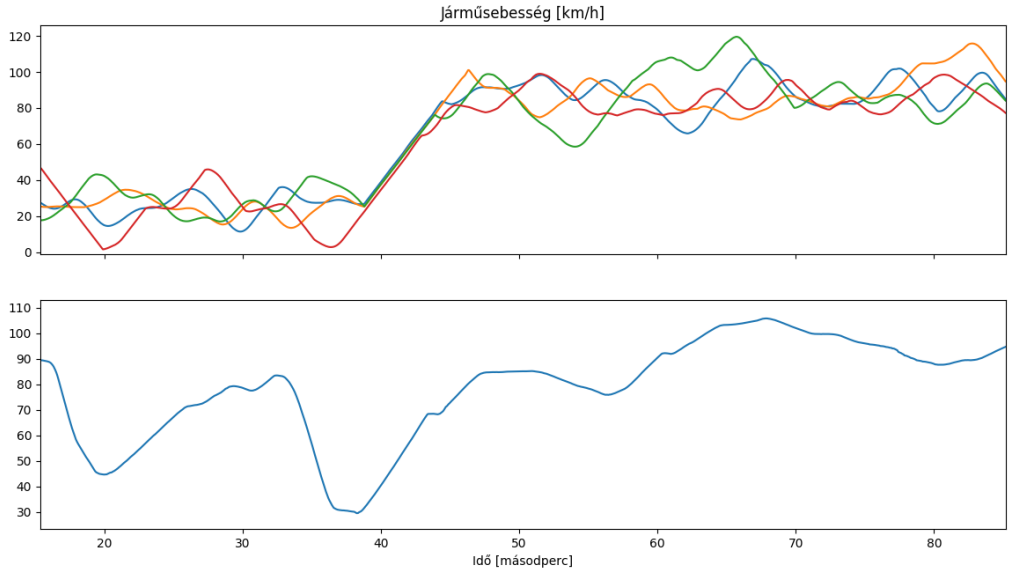
15. ábra: Elkészült tesztmanőver

5.3 Összehasonlítás egy jelenlegi robusztussági teszttel

Az összehasonlításhoz a jelenlegi robusztussági tesztekből kiválasztottam egyet, amely a városi vezetést teszteli. Ezután több két oktávos ennek a munkapont tartománynak megfelelő tesztmanővert generáltam le az általam készített algoritmussal. A szemléltethetőség érdekében az algoritmus bemeneteit úgy állítottam be, hogy a járműsebesség és a kormány szögsebesség is minél jobban hasonlítson a robusztussági tesztre. Az algoritmus paramétereit az 5.2 fejezetben ismertetettek szerint állítottam be, majd készítettem négy darab tesztmanővert ugyanazokkal a bemenetekkel, de más seed-el. A mérések bemutatása érdekében a teszteknek mindig csak egy részét fogom megjeleníteni.

5.3.1 Járműsebesség referenciák összehasonlítása

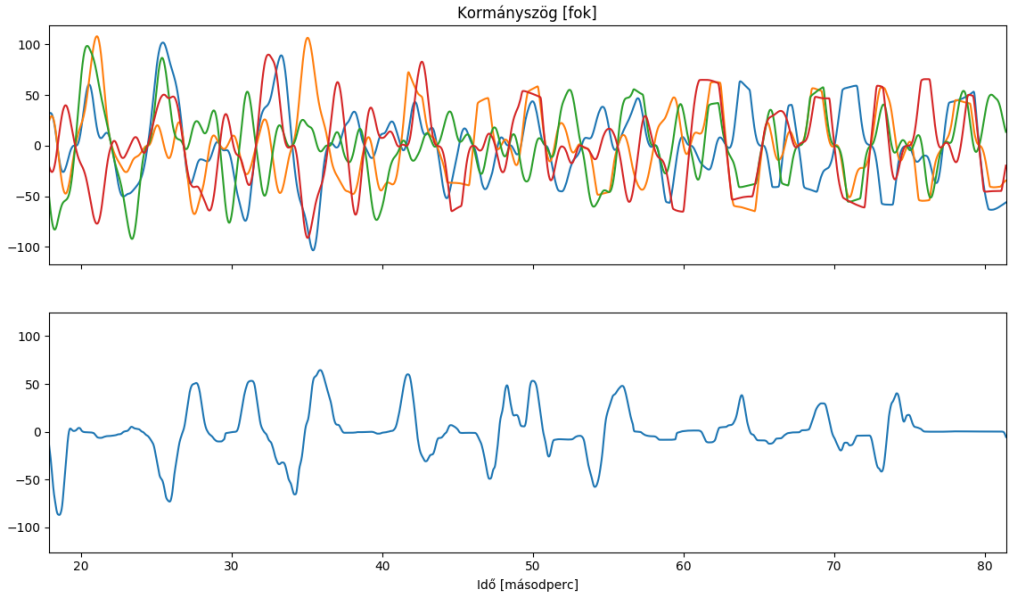
A járműsebességet a városi vezetés robusztussági teszt munkapont tartományába igazítottam. A 16. ábrán alul a jelenlegi robusztussági teszt, felül pedig a négy különböző tesztmanőver járműsebessége látható. Negyven másodperc környékén a megadott két járműsebesség tartomány közötti átmenet figyelhető meg. Az algoritmus által készített robusztussági tesztnek itt megfigyelhető az előnye a jelenlegivel szemben. A négy járműsebesség referencia az adott, városi vezetés munkapont tartományon belül sokkal több munkapontot tud megvizsgálni, mint a mostani teszt. Az algoritmus korábban bemutatott előnyei miatt pedig könnyen és gyorsan lehet még több tesztmanővert készíteni a kívánt tartományba, ezzel növelve a tesztelés hatékonyságát.



16. ábra: Járműsebesség (jelenlegi alul, generált tesztek felül)

5.3.2 Kormányszög referenciák összehasonlítása

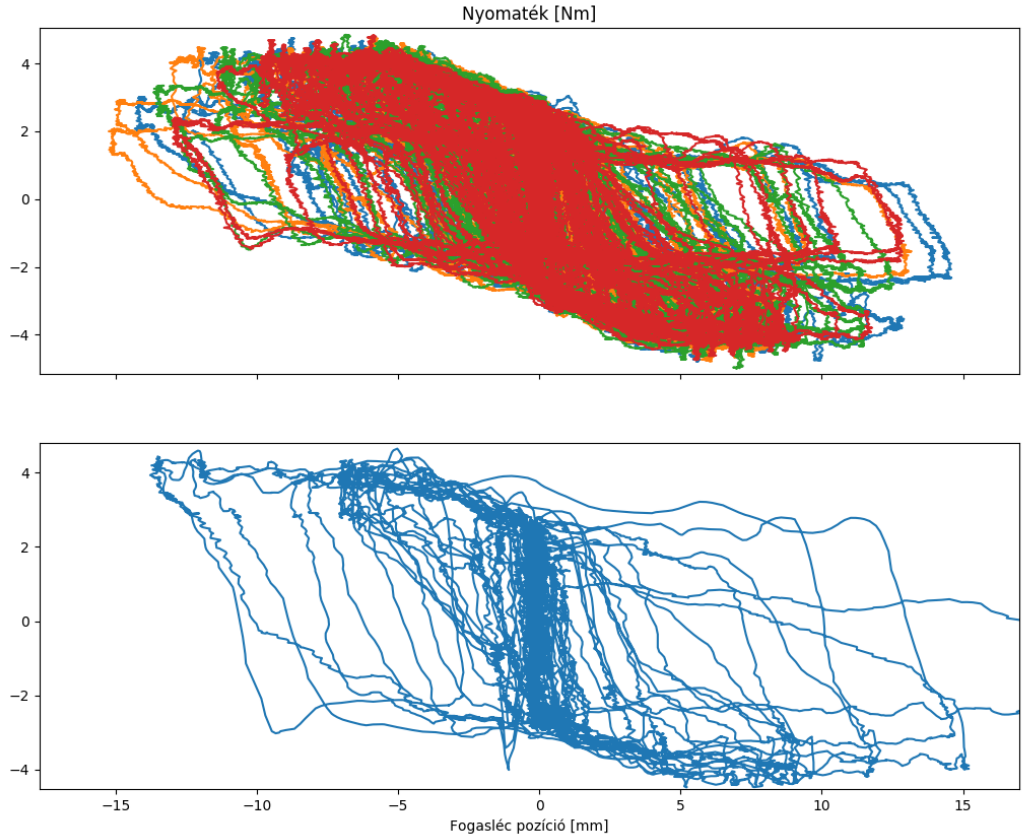
A kormányszöget és a kormány szögsebességét is a meglévő robusztussági teszt alapján készítettem el. A kormányszög maximumát száz fokban határoztam meg, mert ennek a munkapont tartománynak a kormányszög értékei a járműsebesség alapján limitáló függvény szerint körülbelül ezen a tartományon belül mozognak. A 17. ábrán 40 másodpercnél megfigyelhető, ahogy az előző ábra ugyanezen pontján növekvő járműsebesség miatt, a járműsebesség alapján limitáló függvény csökkenti a kormányszöget, ezzel közelítve a valós értékeket. Ahogyan az előző ábrán a járműsebesség esetén, az 17. ábrán is jól látható, hogy az eredeti robusztussági teszthez képest a négy különböző referencia segítségével sokkal több munkapontot tudunk tesztelni. A generált tesztmanőverek számának növelésével pedig egyre részletesebben tudjuk lefedni a munkapont tartományt és minden különböző tesztmanőver futtatásnál kicsit más munkapontokat érint a rendszer.



17. ábra: Kormányszög (jelenlegi alul, generált tesztek felül)

5.3.3 Nyomatékszenzor jelének kiértékelése

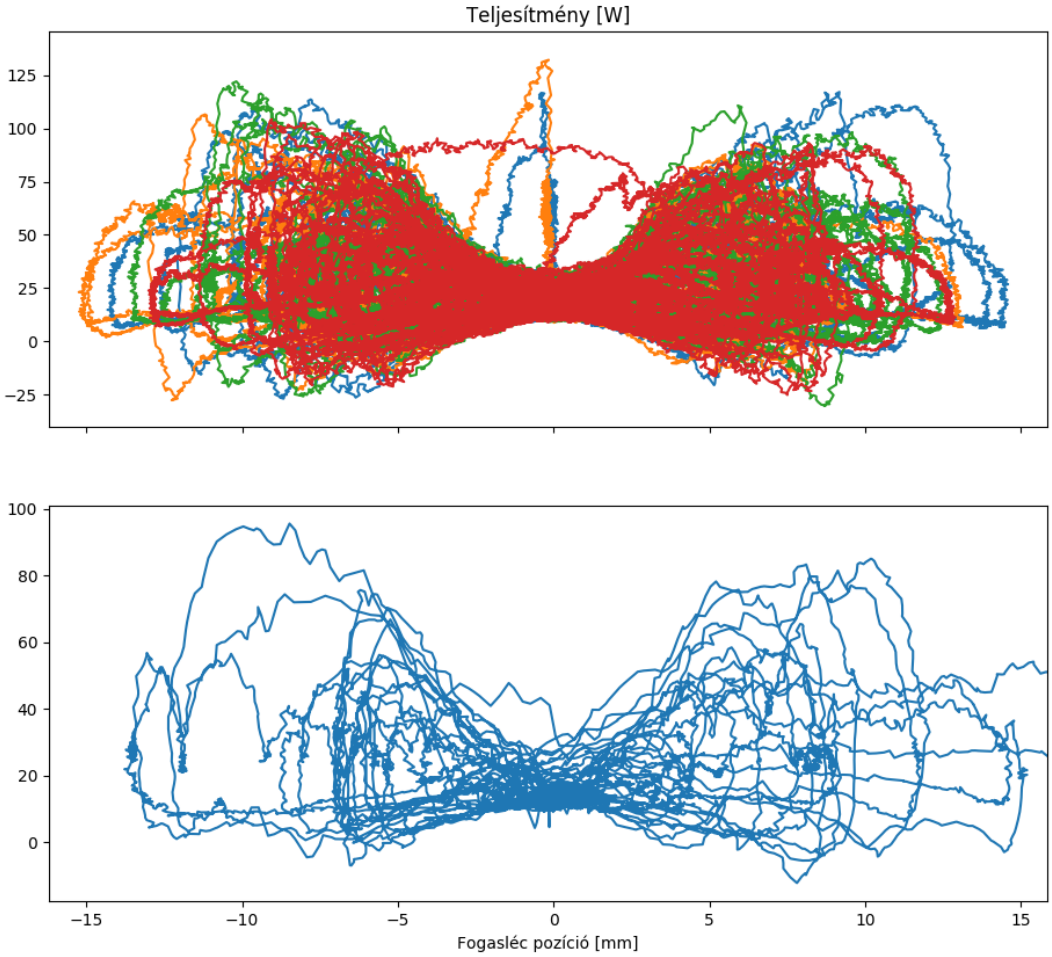
A kormányrendszer legfontosabb bemenete a vezető által kifejtett kormány nyomaték. A 18. ábrán a kormány nyomaték látható a fogasléc pozíciójának függvényében. A kormány nyomatékot a kormány aktuátor állítja elő, és egy külön valós, kalibrált szenzor méri. A fogasléc pozíció függvényében felrajzolt kormány nyomaték ábra segítségével még szemléletesebben lehet bemutatni a rendszer által bejárt munkapontokat. Minél sűrűbb az ábra, annál több munkapontot vizsgáltunk meg és annál jobban van lefedve a tesztelt munkapont tartomány. A ábra felső részén láthatóak a generált tesztek nyomaték tekintetében, a megegyező paraméterezés miatt hasonlítanak egymásra, de így is van közöttük annyi eltérés, hogy minden új seed-nél kicsit más munkapontokat érintsenek. Ahogyan az az ábrán is megfigyelhető a jelenlegi és a generált tesztek is a 4 és -4 Nm tartományban mozognak, de a generált tesztek mennyisége miatt sokkal több különböző eset vizsgálható meg, míg a jelenlegi teszttel minden futtatáskor pontosan ugyanazon munkapontokat tudjuk ellenőrizni.



18. ábra: Nyomaték fogasléc pozíció függvényében (jelenlegi alul, generált tesztek felül)

5.3.4 Rendszerteljesítmény vizsgálata

A 19. ábrán a rendszer teljesítménye figyelhető meg a fogasléc pozíció függvényében. A nyomatékhoz hasonlóan, a generált tesztmanővereknél itt is nulla milliméter környékén van a tesztmanőverek között a legnagyobb átfedés és távolabb már egyre inkább eltérnek egymástól. Az ábra alsó részén megfigyelhető, hogy nagyobb teljesítményt a mérés során kevesebbszer ad le a rendszer, főleg a kisebb teljesítményekben járatja a rendszert a jelenlegi robusztussági teszt. Az ábra felső részén viszont a generált tesztek jellegre teljesen hasonlítanak, de sokkal jobban lefedik a legnagyobb és legkisebb teljesítmény értékek közötti tartományt. Ebből következően ahogyan az eddigi ábráknál is látható volt a jelenlegi robusztussági teszthez képest a négy különböző tesztmanőver sokkal több munkapontban tudja megvizsgálni a rendszer működését.



19. ábra: Teljesítmény fogasléc pozíció függvényében (jelenlegi alul, generált tesztek felül)

5.3.5 Következtetések levonása

A generált tesztmanőverek egyenként a legtöbb helyzetben megegyeznek a városi vezetést vizsgáló robusztussági teszttel. Több azonos bemenet alapján készült tesztmanőverrel viszont egy adott munkapont tartományon belül a jelenlegi robusztussági tesztnél jelentősen több különböző munkapontot lehet megvizsgálni. Az algoritmus egyszerű és gyors tesztmanőver készítésének köszönhetően, pedig ezáltal sokkal átfogóbban és hatékonyabban tudjuk tesztelni a rendszert egy adott munkapont tartományban, ha mindig új tesztmanővert generálunk.

6 Összefoglalás

A szakdolgozatom során bemutattam az autóipari szoftvertesztelést és azon belül a robusztussági tesztek működését és fontosságát. Ezután röviden ismertettem a kormányrendszer rendszerteszteléséhez szükséges tesztpadot, tesztelési metódust. A thyssenkrupp-nál használt robusztussági tesztek helyett a Perlin-zaj algoritmus segítségével egy pszeudo-random tesztmanőver készítő algoritmust terveztem, a tesztelés hatékonyságának növelésére. Bemutattam a Perlin-zaj algoritmus működését, kétdimenziós megvalósítását és a feladathoz való igazítását. A szakdolgozatban részletesen bemutattam azokat a problémákat és azokhoz tartozó megoldásokat, amikkel az algoritmus tervezése és megvalósítása közben találkoztam. Az elkészült algoritmussal tetszőleges számú valós autóvezetést megközelítő tesztmanővert lehet generálni, amelyek futtathatóak a tesztpadon. Végül az elkészült algoritmussal generáltam néhány egy jelenlegi robusztussági teszt munkapont tartományával megegyező tesztmanővert és összehasonlítottam a mért eredményeket. A feladat célját, a robusztussági tesztelés során vizsgált munkapontok számának, ezáltal hatékonyságának növelését elértem.

6.1 Továbbfejlesztési lehetőségek

Az algoritmussal jelenleg egy valós autóvezetést megközelítő tesztmanővert lehet elkészíteni, azonban vannak olyan szituációk, amelyeknél kevésbé életszerű eredményeket kaphatunk, továbbá a tesztmanőver megfelelő beállításának egy része sem működik jelenleg automatikusan. Ezeket fogom kifejteni ebben a fejezetben.

6.1.1 Kormányszög frekvencia automatikus beállítása

A generált referencia frekvenciája meghatározza a kormányszög sebességet. A jelenlegi megvalósításban a kormányszög referencia frekvenciáját manuálisan kell beállítani. Ahhoz, hogy egy tesztmérnöknek csak egy munkapont tartományt, a rendszer végső pozícióját fokban és a teszt hosszát kelljen beállítania, meg kell oldani, hogy a kívánt teszt időtartamtól és munkapont tartománytól függően automatikusan történjen a kormányszög frekvencia modulációja.

A zaj frekvenciája, mint már korábban kifejtettem függ a generált zajt elemszámától, lépésközétől és oktáv számától, a teszt hosszától és a kormányszög

tartományba igazításától is. Ha tesztmanőver készítés során ezeket mind figyelembe vesszük és aszerint módosítjuk például a lépésközt vagy a teszt hosszát, akkor automatikusan tudunk tetszőleges kormány szögsebességtartományt beállítani a teszthez.

6.1.2 Kormányszög és járműsebesség kapcsolatának kibővítése

A jelenleg járműsebesség alapján kormányszög limitálására használt függvényt egy városi vezetés során készült mérésből határoztam meg. Ennek a mérésnek a felhasználása lehetővé tette, hogy valósághű kormányszög referenciát kapjuk a járműsebesség referencia függvényében. Egy autót viszont nem csak városi vezetésre használnak és a jelenlegi mérés alapján beállított limit ezt nem veszi figyelembe. Például, ha egy versenyzést szimuláló tesztmanővert szeretnénk elkészíteni, ahol nagyobb járműsebességeknél nagyobb kormányszögek mérhetők, mint a városi vezetés során, akkor a jelenlegi megoldás túlságosan lekorlátozná a kormányszög referenciát.

A megoldás erre a problémára, hogy több különböző vezetési szituáció mérései alapján több különböző függvényt is készítünk, majd vizsgált munkapont tartománytól és vezetési szituációtól függően alkalmazzuk azok egyikét. Ezzel tovább növelhetjük a vizsgált munkapontok mennyiségét és növelhetjük a robusztussági teszt hatékonyságát.

6.1.3 Járműsebesség függő gyorsulás korlátozás

A járműsebesség korlátozás mértékét én egy átlagos autó 0-100 km/h-ra való gyorsulásából határoztam meg. Ez ugyan legtöbb esetben egy realisztikus járműsebességhez közeli referenciát fog eredményezni, de a valóságban egy autó nem feltétlen gyorsul lineárisan. Álló helyzetből indulva általában kevesebb idő alatt lehet elérni a 100 km/h, mint aztán arról a 200 km/h-t.

A gyorsulást ezért meg lehetne határozni valós mérések alapján. Különböző mérésekből el lehetne készíteni egy a kormányszög limitációhoz hasonló módon egy függvényt, amivel a tesztmanőver generálás jobban megközelíteni egy autó gyorsulását, ezáltal segítve a tesztmanőver életszerűségét.

7 Irodalomjegyzék

[1] I. Majzik és Z. Micskei, „Robusztusság tesztelés,” 2011. [Online]. Available: https://inf.mit.bme.hu/sites/default/files/materials/category/kateg%C3%B3ria/oktat%C3%A1s/msc-t%C3%A1rgyak/szoftverellen%C5%91rz%C3%A9si-technik%C3%A1k/11/SZET-2011-EA12a_robusztussag_teszteles.pdf.

[2] P. Kőfalusi, „Futómű rendszerek mechatronikája,” 2014. [Online]. Available: https://mogi.bme.hu/TAMOP/futomu_rendszerek_mechatronikaja/ch07.html.

[3] Liu, Chuanliangzi, "DEVELOPMENT OF HARDWARE-IN-THE-LOOP SIMULATION SYSTEM FOR ELECTRIC POWER STEERING CONTROLLER TESTING", Master's report, Michigan Technological University, 2015.

[4] dSPACE Magazin, „A Test of Character for Steering Systems,” 2016. [Online]. Available: https://www.dspace.com/shared/data/pdf/2016/24-31_A_Test_of_Character_for_Steering_Systems_en.pdf.

[5] K. Perlin, „Making Noise,” 1999. [Online]. Available: https://web.archive.org/web/20160310194211/http://www.noisemachine.com/talk1/index.html.

[6] K. Perlin, „Improving Noise,” 2002. [Online]. Available: https://mrl.cs.nyu.edu/~perlin/paper445.pdf.

[7] „Permuted Congruential Generator,” [Online]. Available: https://numpy.org/doc/stable/reference/random/bit_generators/pcg64.html#numpy.random.PCG64.

[8] N. Tobias, H. Peter, O. Sebastian, S. Falko, M. Markus, R. Andreas és R. B. Jurger, T. Nothdurft et al., "Stadtpilot: First fully autonomous test drives in urban traffic," 2011 14th International IEEE Conference on Intelligent Transportation

Systems (ITSC), Washington, DC, USA, 2011, pp. 919-924, doi: 10.1109/ITSC.2011.6082883.

Köszönetnyilvánítás

Ezúton is szeretnék köszönetet mondani Horváth Patriknak a rengeteg segítségért, amelyet a feladat megvalósítása során kaptam. Emellett köszönettel tartozom a thyssenkrupp Components Technology Hungary Kft.-nek, aki biztosította a megvalósításához szükséges technikai hátteret.