Pszeudo-random tesztmanőver generálás robusztussági tesztekhez

Mikes Marcell

Tartalomjegyzék

[Összefoglaló 5](#_Toc136814538)

[Abstract 6](#_Toc136814539)

[1 Robusztussági teszt 7](#_Toc136814540)

[1.1 Tesztelés 7](#_Toc136814541)

[1.1.1 Tesztelés az autóiparban 7](#_Toc136814542)

[1.2 Robusztussági teszt 7](#_Toc136814543)

[1.2.1 Robusztusági teszt a thyssenkruppnál 8](#_Toc136814544)

[1.3 Probléma definiálása 8](#_Toc136814545)

[1.4 Feladat célja 9](#_Toc136814546)

[2 Tesztpad, kormányrendszer 10](#_Toc136814547)

[2.1 Elektromechanikus szervokormány 10](#_Toc136814548)

[2.2 Hardware in loop 11](#_Toc136814549)

[2.3 Tesztpad 11](#_Toc136814550)

[2.3.1 Kormánykerék Aktuátor (Kormányrobot?) 12](#_Toc136814551)

[2.3.2 Linear motor 12](#_Toc136814552)

[2.3.3 HIL simulator 13](#_Toc136814553)

[2.3.4 Tesztpad kommunikáció? 13](#_Toc136814554)

[3 Perlin-zaj 14](#_Toc136814555)

[3.1 Bevezetés 14](#_Toc136814556)

[3.2 Algoritmus működése 14](#_Toc136814557)

[3.2.1 Lépésköz 15](#_Toc136814558)

[3.2.2 Átmenetek elsimítása 17](#_Toc136814559)

[3.3 Oktávok 17](#_Toc136814560)

[3.4 Algoritmus igazítása a feladathoz 18](#_Toc136814561)

[3.4.1 Gradiens vektorok generálása 18](#_Toc136814562)

[3.4.2 Lépésköz meghatározása 19](#_Toc136814563)

[3.4.3 Zaj elemeinek száma 20](#_Toc136814564)

[3.4.4 Oktávok meghatározása 21](#_Toc136814565)

[3.4.5 Futási idő csökkentése 21](#_Toc136814566)

[4 Tesztmanőver készítés 23](#_Toc136814567)

[4.1 Aktivitás diagram 23](#_Toc136814568)

[4.2 Zaj tartományba igazítása 24](#_Toc136814569)

[4.3 Eloszlás változtatása 25](#_Toc136814570)

[4.4 Kormányszög és járműsebesség kapcsolata 27](#_Toc136814571)

[4.5 Kormányszög és járműsebesség korlátozása 29](#_Toc136814572)

[4.6 Rámpafüggvény alkalmazása 31](#_Toc136814573)

[5 Kiértékelés 32](#_Toc136814574)

[5.1 Elkészült tesztmanőver 32](#_Toc136814575)

[5.1.1 Kerékterhelés 32](#_Toc136814576)

[5.1.2 Tesztmanőver futtatása (nem biztos, hogy ez kell) 32](#_Toc136814577)

[5.2 Futtatott tesztesetek? 32](#_Toc136814578)

[5.3 Összehasonlítás jelenlegi robusztussági tesztekkel 32](#_Toc136814579)

[5.4 Következtetések levonása 32](#_Toc136814580)

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: ?

……………………………………………………

Mikes Marcell

Összefoglaló

TODO

Abstract

TODO

# Robusztussági teszt

## 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.

### 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 idővesztesség. Az autóiparban a helytelen működés sérülést vagy akár halált is okozhat.

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 elvárásának megfelelő termék készül-e el. Kormányrendszer esetében ez például lehet kényelmi funkció, a kormány középre igazítja magát az autó elindításakor vagy például a vezetés érzete. A nem biztonsággal kapcsolatos funkcionalitások működésére is figyelmet kell fordítani.

## Robusztussági teszt

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

### Robusztusági teszt a thyssenkruppnál

A thyssenkrupp-nál a robusztussági tesztek a rendkívüli bemenetek, vagyis különböző hibák melletti működést vizsgálják. Az extrém, megterhelő környezeti feltételek vizsgálatára külön tesztek vannak. Jelenleg a thyssenkrupp által használt robusztussági tesztmanőverek két féle mérési módszerrel készültek el. Egyik egy autós szimulátorban számítógépes kormány segítségével történő vezetés, míg a másiknál valós autó vezetés mérésével. A robusztussági teszt futtatása közben pedig a tesztmérnök által előre meghatározott hibákat, megadott valószínűséggel, véletlenszerűen injektálja be a tesztkörnyezet. A különböző vezetési módokhoz, helyzetekhez több mérés is van, de összesen limitált mennyiségű, így a tesztesetek száma is limitált.

## 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 pedig 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 vezetett mérések számának növelése igencsak erőforrásigényes. A teszteset kívánt hosszától függően több tíz perc is lehet egy ilyen mérés levezetése. 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. Ezeket a jelenlegi tesztkörnyezet és tesztfolyamatok miatt, minden új szoftver verziónál újra le kell tölteni a tesztgépekre. Egy-egy mérés nem foglal sok tárhelyet, viszont folyamatosan növelve a tesztesetek számát, tárhelyet és a letöltés miatt időt is veszíthetünk.

## Feladat célja

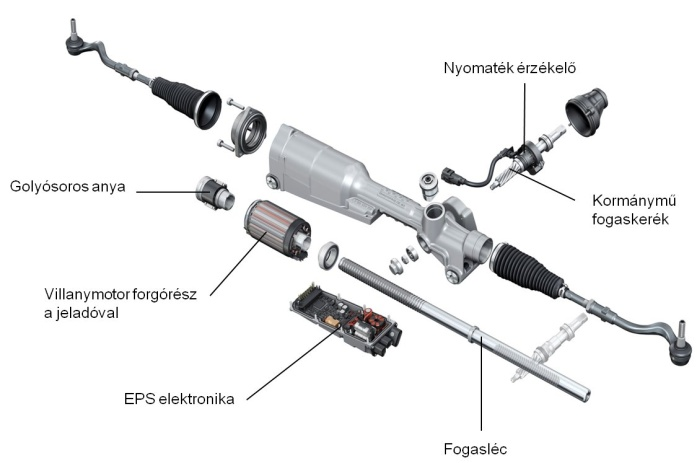
A jelenleg használt mérési módszerek helyett egy algoritmussal fogom elkészíteni a különböző tesztmanővereket. A tesztmérnök által előre meghatározott tartományban, pszeudo-random módon, tetszőleges számú kormányszög és járműsebesség referenciát generálok. Az algoritmussal bármennyi tesztesetet el lehet készíteni anélkül, hogy valakinek azzal foglalkoznia, időt kelljen töltenie. Továbbá a tesztmanővereket nem 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.

# Tesztpad, kormányrendszer

## Elektromechanikus szervokormány

A jelenlegi gépjárművek talán 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 erőt. A kibocsájtott erőt többek között meghatározza a sebesség és a környezeti tényezők, 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, a jelentős olaj megtakarítás, illetve jól együttműködik a modern ADAS (Advanced Driver-Assistance System) rendszerekkel. [hivatkozás]



. ábra: Egy elektromechanikus szervokormány lehetséges felépítése [hivatkozás]

## Hardware in loop

A kormányrendszer fejlesztése során a tesztelés költséges lenne, ha mindig egy teszt autóba beszerelve vizsgálnánk a működését. Ezért tesztelés során a HIL metódust használjuk. A Hardware-in-the-loop (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.

Kormányrendszer tesztelésénél például a HIL szimuláció része az RBS (Rest Bus Simulation). Az RBS a kormányrendszer egy valós autóban más eszközöktől kapott üzeneteit szimulálja. Ez a szimuláció tartalmaz minden olyan üzenetet, amelyet a működése során fel kell tudnia dolgozni. Az RBS szimulációval tehát meg lehet vizsgálni és le lehet tesztelni a rendszer működését különböző üzenetekre.

## Tesztpad

A thyssenkrupp-nál a tesztelés különböző szintjeihez egyedi tesztpadok lettek kialakítva. Ezeknek a célja, hogy valósághűen tudjuk tesztelni azokat a szituációkat, amikkel a termék az átadása után találkozhat. A tesztpadok segítségével a kormányrendszer autóba beépítése nélkül tudjuk elvégezni a tesztelést, ezzel jelentős mennyiségű időt és pénzt spórolva. Az algoritmus a rendszerteszt osztály robusztussági tesztjeinek leváltásához készült, ezért a továbbiakban tesztpad alatt a rendszerteszteléshez használt tesztpadokat értem.

A thyssenkrupp-nál használt tesztpadokat egy dSPACE és Porsche által megépített tesztpad által fogom bemutatni. A legtöbb helyen felépítésében és komponenseiben is hasonlítanak egymáshoz.

A képen gép, mérnöki tudomány, fedett pályás látható

Automatikusan generált leírás

. ábra: Rendszerteszteléshez használt tesztpad [hivatkozás]

### Kormánykerék Aktuátor (Kormányrobot?)

A kép tetején „Steering actuator” felirattal látható az aktuátor, ami a kormányzásért felel. Ennek a thyssenkrupp-nál STWA (Steering Wheel Actuator) a neve. Elengedhetetlen alkatrésze a tesztpadnak, hiszen ezzel szimuláljuk egy autóvezető kormányzását és hozzájárul az automatizált tesztek futtatásának lehetővé tételéhez. Ahogyan az a felsorolt tulajdonágaiból is látható nagyon pontos, segítve ezzel a tesztek mérését és reprodukálhatóságát.

### Linear motor

A lineáris motor (képen „Linear motors”) egy olyan elektromos motor, amely a forgó motorokkal ellentétben, nem forgó hanem egyenes vonal menti mozgást hajt végre. A lineáris motorokat az autó kerekeire ható terhelés megvalósítására használják. A kerékterhelés szimulálása fontos a kormányrendszer valósághű teszteléséhez. A lineáris motorok működését úgy, mint a kormánykerék aktuátort lehet automatizálni és adott teszthez lehet igazítani.

### HIL simulator

A képen „HIL simulator-nak” feltüntetett szekrényben találhatóak az olyan eszközök, amelyek a tesztpad működéséért felelnek. Néhány fontos elemét fogom csak kiemelni. Ide tartozik többek között a „dSPACE SCALEXIO”, amely komplex valós idejű HIL szimuláció megvalósítására alkalmazható. Ebben a szekrényben van még a tápegység is, amely a kormányrendszer feszültségét biztosítja. Nem utolsó sorban pedig a számítógép, amin keresztül tudja a tesztmérnök irányítani a tesztpadot.

### Tesztpad kommunikáció?

A gépjárműveknél elterjedt kommunikációs protokollok egyike a CAN (Controller Area Network), amely segítségével tudnak a különböző eszközök információt átadni egymásnak. A CAN segítségével különböző eszközök egy központi számítógép nélkül tudnak kommunikálni egymással. A korábban ismertetett RBS szimuláció az autó kommunikációs hálózatától függően például lehet a CAN protokollal küldött üzenetek szimulációja.

# Perlin-zaj

Egy olyan tesztmanőverhez, ami nem autós mérések alapján készül, én egy zaj algoritmust választottam, mert költséghatékonyan tudok vele tetszőleges számú mérést előállítani. Fontos, hogy a zaj egymást követő értékei között kapcsolat legyen. 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.

## Bevezetés

A Perlin-zajt 1983-ban Ken Perlin találta fel filmek kinézetének 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. Az algoritmus mellett készített egy programozási nyelvet is tetszőleges textúrák és azok árnyékainak megvalósításához. A Perlin-zaj algoritmus egyszerűsége és alkalmazhatósága miatt gyorsan elterjedt az iparágban. Később 2002-ben kiadott egy tanulmányt „Improving Noise” néven, amiben optimalizálta a korábbi algoritmus működését. [hivatkozás]

Az algoritmust el lehet készíteni különböző dimenziókban (például 2D, 3D, 4D). Én a kétdimenziós implementációt választottam. Ennél több dimenziónak nem lett volna értelme, mert megnő a számítási kapacitás és nem is tudnám felhasználni az egész zajt. 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.

## 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 rendelni kell egy véletlenszerű 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. Aztán a távolságvektorok és gradiensvektorok skaláris szorzatából ki tudunk számolni négy értéket. Ezután a négy kapott értékből bilineáris interpolációval (távolságsúlyozott átlaguk alapján) számoltam ki a jelenlegi pont értékét.

A képen képernyőkép, sor látható

Automatikusan generált leírás

. ábra: 1 lépésközű négyzetrács illusztrációja négy berajzolt gradiensvektorral

### 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 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 tizes 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.

A képen sor, lézer látható

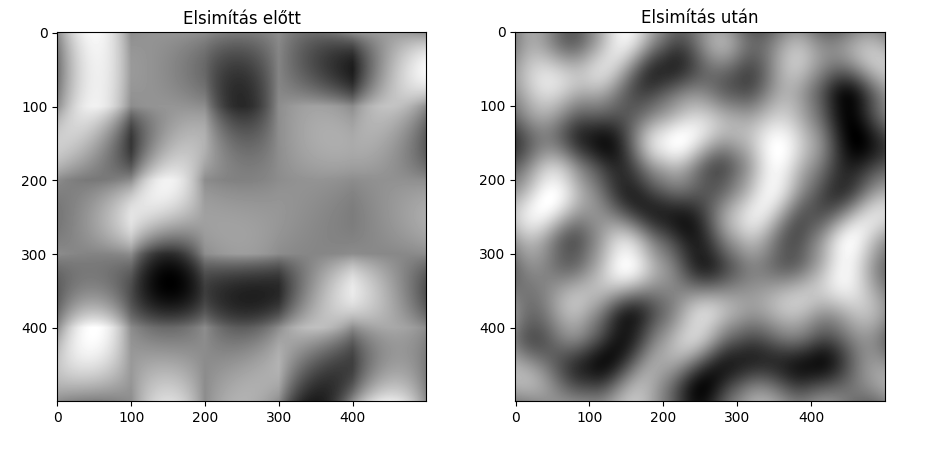
Automatikusan generált leírás

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

### Átmenetek elsimítása

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

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



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

## 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 három különböző zajt ezután elég összeadni a három oktáv eléréséhez. Az oktávok segítségével el tudunk érni egy sokkal részletesebb zajt. Tesztmanőver esetében pedig élethűbb kormány- és járműsebesség referenciát kaphatunk vele.

A képen szöveg, Diagram, diagram, sor látható

Automatikusan generált leírás

. ábra: 1 és 3 oktávval generált zaj egy szelete

## Algoritmus igazítása a feladathoz

Annak érdekében, hogy olyan zajt kapjunk, melyből aztán kormányszöget és járműsebességet tudunk csinálni megfelelően fel kell paraméterezni az algoritmust. A zaj által generált elemek száma, frekvenciája és az algoritmus futási ideje is fog számítani tesztmanőver készítésnél. Továbbá fontos az eredmények reprodukálhatósága is.

### 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, illetve ez a kiváltó ok megértésénél is fontos lehet. Továbbá a hiba kijavításánál is érdemes ellenőrizni, hogy fennáll-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 [hivatkozás]. 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) percre 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, én 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.

### 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 segítségével egy túl nagy frekvenciájú zajt is el lehet nyújtani. A teszt percben mért hossza egyenesen arányos a lépésköz nagyságával. Ugyanaz a frekvencia elérése érdekében kétszer hosszabb tesztnél a lépésközt is a kétszeresére kell növelni.

A képen szöveg, Diagram, sor, diagram látható

Automatikusan generált leírás

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

### Zaj elemeinek száma

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

### 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 nő. Minden egyes oktáv a kétszeresére növeli a zaj elkészülésének idejét. Ezért meg kell megfontolni mennyi oktávot érdemes használni. Ha az oktávok növelésével párhuzamosan a lépésközt nem növeljük, akkor előléphet 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 oktáv fölé menni, mert öt oktáv felett marginális lesz a hozzáadott értéke tesztmanőver szempontjából és közben lassabb is lesz a futási idő. 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.

### 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. A zajból három egydimenziós szeletet kell kivenni, egy kormányszöget és kettő járműsebességet. A két járműsebesség szükségét később a 4.3 fejezetben fogom kifejteni. Ezért egy 5 000x5 000 zajból egy darab tesztmanőverhez csak három 5 000 elemű szeletet kell felhasználnom. Kiegészítettem 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 első három szeletét használom fel a zajnak, mert a Perlin-zaj értékei között 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 három hasonló zajt fogunk kapni. 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 ezt a tulajdonságát figyelembe véve több, mint három szeletet fogok felhasználni a legenerált zajból. Az elkészíthető tesztesetek száma tehát a zaj elemszámától és a kivett szeletek számától függ.

A zaj csak adott elemeinek 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.

# Tesztmanőver készítés

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.

## Aktivitás diagram

Az algoritmus működésének egyszerűbb megértéséhez készítettem egy aktivitás diagrammot. Ebben a fejezetben ezeket a műveleteket fogom bemutatni.

A képen szöveg, képernyőkép, diagram, Párhuzamos látható

Automatikusan generált leírás

. ábra: Aktivitás diagram az algoritmus működéséről

## 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 kormányrobottól, akkor elkezdjük felfeszíteni a végpozícióra rendszert, mert a kormányrobot próbálná elérni a kívánt fokot, de a rendszer a fizikai határa miatt nem tudja. Nagy erővel próbálkozik a kormányrobot, ami akár kárt is okozhat a tesztpadban vagy a rendszerben. Ha a kormányszög tekerési tartományának nagyobb értéket adunk meg, mint azt fizikailag képes elérni, akkor a 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 kerékterhelésnél szimulációnál lehet majd látni. Ezért mindkét jelnél fontos lesz a megfelelő tartományba igazítás. Ezt erre a két referenciára 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 nem fogja a 200 fokot elérni a kormányszög. Bár a végső algoritmus nem engedné, hogy nagy járműsebesség mellet túl nagy legyen a kormányszög (ezt később a 4.4 fejezetben kifejtem), a jel viszont 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 határétékekhez. 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.

## 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.

A képen diagram, szöveg, képernyőkép, Diagram látható

Automatikusan generált leírás

. 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 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 más tartományban vannak. Ha két különböző értékkészlettel rendelkező jel kombinációját állítjuk elő, 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, így a két függvény között egyenletes lesz 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.

A képen szöveg, diagram, képernyőkép, Diagram látható

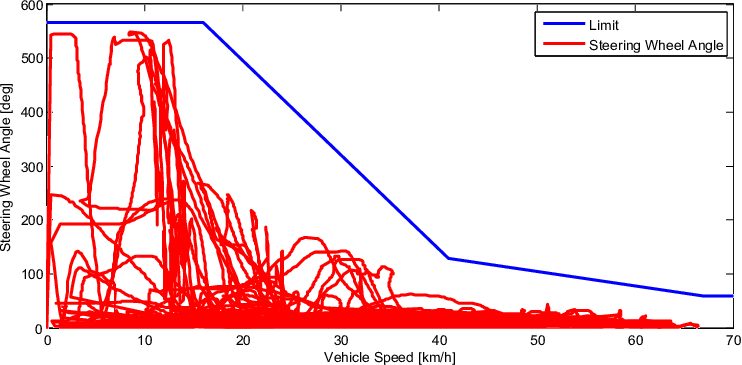
Automatikusan generált leírás

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

## 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 fednének le egy élethű autóvezetést. 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 állítottak be a kormányszögre, ami alatt az autóvezető még kontrollálni tudja az autót. Ezt a limitet aztán egy tesztpályán verifikálták. [hivatkozás]



. ábra: A tanulmányban mért adatok, kormányszög járműsebesség függvényében [hivatkozás?]

A fenti képen piros színnel látható a kormányszög különböző járműsebességek függvényében, kékkel pedig a megállapított limit. A kormányszög elég gyorsan el kezd 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 jobb limitet lehet meghatározni, mert jobban illeszkedik a mérésből megfigyelt működéshez. Az alábbi függvényt készítettem el, a tanulmány mérései alapján a kormányszög limitálásához. A tizenötöt azért adtam hozzá a végén, hogy nagy járműsebességeknél se legyen túl kicsi a kormányszög, mert egy keveset ott is kell kormányozni.

Az algoritmusban a már tartományba igazított kormányszöget, a megváltoztatott eloszlású járműsebesség alapján limitálom. Mindkettő jel ugyanolyan hosszú, ezért csak azt kell megnéznem, hogy egy adott pillanatban mi éppen az értékük. Megnézem a kormányszög abszolút értékére, hogy 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 méréseinek követése pedig biztosítja a kormányszög és járműsebesség referencia valósághű működését.

A képen szöveg, képernyőkép, diagram, sor látható

Automatikusan generált leírás

. ábra: kormányszög limitálás járműsebesség függvényében

## Kormányszög és járműsebesség korlátozása

A 4.4 fejezetben bemutattam a kormányszög és járműsebesség kapcsolatát, ami biztonságosabbá 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 jelenleg körülbelül 10 km/h másodpercenként. Kormányszögnél ezt 800 fok/másodperc-re állítottam. Ezek a korlátok természetesen egyszerűen változtathatók későbbi igények szerint.

A képen szöveg, diagram, sor, Diagram látható

Automatikusan generált leírás

. ábra: A korlátozás működését bemutató ábra

## 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. Ezért egy rámpafüggvényt fogok alkalmazni.

A rámpafüggvény a rámpára hasonlító alakjáról kapta az elnevezést. A meredekségét a 4.5 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.

Kormányszögnél hasonlóan egy +100 fokban induló kormánytekerés nem valósághű és a tesztpad esetében nagyon gyors kormányszög elfordulási sebességet idézne elő. (a kormányszög is eltolódik nulláról, de nem volt időm megnézni pontosan miért ezért ez valszeg nem lesz benne, csak itt hagytam ha esetleg mégis lenne)

A képen szöveg, Diagram, sor, képernyőkép látható

Automatikusan generált leírás

. ábra: járműsebesség a rámpafüggvény alkalmazása után

# Kiértékelés

## Elkészült tesztmanőver

Amennyiben a kormányszög és a járműsebesség jel minden hozzájuk rendelt módosításon átesetek (ez követhető a 4.1 fejezet aktivitás diagramján), akkor elkészült a végleges kormányszög és a végleges járműsebesség referencia. A futtatásához még egy jelet kell biztosítani a tesztkörnyezetnek, az időtengelyt. Az időtengely határozza meg, hogy a referenciák értékeit mennyi időnként kell letekernie a kormányrobotnak vagy beállítania a szimulációs modellnek. Én az idő hiánya miatt egy fix 5 perces időtengellyel dolgoztam.

### Kerékterhelés

### Tesztmanőver futtatása (nem biztos, hogy ez kell)

A tesztmanőver a jelenlegi tesztkörnyezetbe integrálása nélküli egyszerű futtatáshoz egy mat kiterjesztésű fájlba csomagoltam az időtengelyt és a referenciákat. …lehet ezt felesleges belerakni

## Futtatott tesztesetek?

## Összehasonlítás jelenlegi robusztussági tesztekkel

## Következtetések levonása