

RRT és Megerősítéses Tanulás kombinálása trajektória követés esetére

Szerző:

Angyal Bálint Gergő

BME

Közlekedésmérnöki és Járműmérnöki Kar

Konzulens:

Kővári Bálint

TDK dolgozat

2022

Contents

1	Bevezető	2
2	Algoritmusok	4
2.1	Véletlenszerű pálya generátor	4
2.2	Kinematikai jármű modell	6
2.3	Dinamikai jármű modell	8
2.4	Környezet vizualizációja	9
2.5	Rapidly-exploring Random Tree	9
2.6	Az RRT és a jármű modell összekapcsolása	10
2.7	Megerősítéses tanulás	11
2.8	RRT és DQN ötvözése	14
2.9	Tanítás folyamata	17
2.10	Továbbfejlesztési lehetőségek	24
3	Eredmények	28
3.1	Összegzés	31
3.2	Források	31

Chapter 1

Bevezető

Napjainkban a járműtechnika egyik legdinamikusabban fejlődő ágazata az önvezetés, rengeteg kiaknázatlan lehetőség, kiforratlan koncepció lelhető fel a témában. Mindezek mellett, a növekvő forgalomból adódóan egyre inkább nő a minél magasabb fokú vezetést segítő rendszerek, vagy akár a teljesen autonóm járművek igénye.

Ezen gondolatok mentén választottam feladatom témájaként a gépi tanulás alkalmazását a járműirányításban. A járműirányítás, mint automatizálási feladat egy szerteágazó problémakört foglal magába. Alapvetően automatizálásról akkor beszélünk, ha a rendszer bizonyos feladatokat részben vagy egészben önállóan képes végrehajtani. Fontos azonban megjegyezni, hogy a vezetést segítő figyelmeztető, vagy esetleg beavatkozó rendszerek még nem minősülnek önvezetésnek, példászerűen beszélhetünk itt sávelhagyásra való figyelmeztetésről, automatikus vészfékezésről.

Az önvezetés témakörébe sokminden besorolható, ezeket hat szintre oszthatjuk. Ezek közül az első három során a vezetés még ember által zajlik, de a számítógép már egy nagyfokú felügyeletet nyújt. Erről bővebben az 1.1 ábra ad tájékoztatást az SEA J3016 szabvány szerint.

A járműirányítás nehézsége talán leginkább annak komplexitásában rejlik, így az elmúlt években egyre népszerűbbek a különböző gépi tanulási módszerek a témakörben. Én a megerősítéses tanulással foglalkoztam. Ez annak köszönhető, hogy sok előnnyel rendelkezik - például korlátosság tekintetében - ismert társával szemben, a felügyelt tanulással szemben. De természetesen ennek a módszernek is vannak hátrányai, dolgozatom során ezen hátrányok javítására törekedtem. Az egyik legnagyobb problémát a tanuláshoz szükséges adathalmaz mérete jelenti. Az adathalmaz csökkentéséhez a témában ismeretes RRT algoritmust használtam fel. Az algoritmus részletes ismertetésére később térek ki, de segítségével egy általa definiált útvonal mentén gyűjthette a program a szükséges mintákat, így gyorsítva a tanulást.

Munkám kezdetét egy irodalom kutatás jelentette, hiszen szükségem volt az alapvető is-



A VEZETÉSAUTOMATIZÁLÁS SZINTJEI A SAE J3016 SZERINT

Bővebb információ: sae.org/standards/content/j3016_202104

Copyright © 2021 SAE International. A táblázat angol eredetijét a SAE International készítette. Fordítás: Villanyautosok.hu

	SAE 0. SZINT	SAE 1. SZINT	SAE 2. SZINT	SAE 3. SZINT	SAE 4. SZINT	SAE 5. SZINT
Mit csinál a humán sofőr a vezetőülésben?	Minden esetben a sofőr vezet, amikor ezek a vezetőtámogató szolgáltatások aktívak, még akkor is, ha a sofőr lába nincs a pedálokon, illetve nem kell kormányoznia.			A sofőr nem vezet, amikor ezek az automatizált vezetés szolgáltatások aktívak, még akkor sem, ha az illető a sofőr ülésében ül.		
	A sofőrnek folyamatosan felügyelnie kell ezeket a vezetőtámogató szolgáltatásokat. A biztonság érdekében szükség esetén a sofőrnek kormányoznia, fékeznie vagy gyorsítania kell.			Amikor a szolgáltatás kéri át kell vevie a vezetést.	Ezek az automatizált vezetés szolgáltatások egyáltalán nem igénylik, hogy a sofőr átvégye az irányítást.	
EZEK VEZETŐTÁMOGATÓ SZOLGÁLTATÁSOK EZEK AUTOMATIZÁLT VEZETÉS SZOLGÁLTATÁSOK						
Mit csinálnak ezek a funkciók?	Ezek a funkciók csupán figyelmeztetést és pillanatnyi asszisztenciát nyújtanak a sofőrnek.	Ezek a funkciók kormányzás VAGY fékezés/gyorsítás támogatását biztosítanak a sofőrnek.	Ezek a funkciók kormányzás ES fékezés/gyorsítás támogatását biztosítanak a sofőrnek.	Ezek a funkciók korlátozottan, előre meghatározott helyzetekben képesek vezetni az autót, és nem működnek, amennyiben nem teljesül minden feltétel.		Ez a funkció minden helyzetben képes vezetni az autót.
Példák ezekre a funkciókra	• Automatikus vészfékezés • holtlér figyelmeztetés • sávellhagyás figyelmeztetés	• sávközépen tartás • adaptív tempomat vagy	• sávközépen tartás • adaptív tempomat együtt és	• automatizált vezetés forgalmi dugóban		• vezető nélküli helyi taxi • a pedálok és a kormány nem feltétlenül szükségesek
				azonos a 4. szinttel, de ez a funkció mindenhol képes vezetni minden körülmény esetén		

Figure 1.1: SEA J3016 szabvány

meretek elsajátítására. A kutatást főként a megerősítéses tanulással kapcsolatban végeztem. Így mások munkái alapján szerezhettem tapasztalatot a gépi tanulás hangolásában, különböző algoritmusváltozatok eredményeiben és korlátaiban. Ötleteket meríthettem, hogy az algoritmusok, technikák milyen irányban fejleszthetők; gondolatébresztőnek is kíváló volt: ötletek kezdtek körvonalazódni a fejemben, melyekről úgy éreztem, az eredményei tanulságosak, vagy akár közvetlenül hasznosak lehetnek. Számos cikket olvastam még a véletlenszerű pályagenerálásról, szerettem volna egy jól paraméterezhető pályagenerátort, hogy azt könnyedén összhangba tudjam hozni az aktuális jármű modell adottságaival.

Irodalomkutatásom során a következő cikkek jelentettek segítséget:

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12]

Chapter 2

Algoritmusok

2.1 Véletlenszerű pálya generátor

Elsőként a véletlenszerű pálya generátort szeretném bemutatni, hiszen ez jelenti a szimulált környezet alapját. A fejlesztés során fontos szempontnak tartottam, hogy a generátor jól paraméterezhető legyen, ezzel megkönnyítve a későbbi tanításokat és teszteléseket.

A generálás első lépéseként hiperparaméterek által megadott tartományban generáltunk hiperparaméterek által megszabott darabszámú pontokat. A generált pontok egymástól vett minimum távolsága szintén hiperparaméterrel állítható.

A meglévő ponthalmazból kiválasztjuk azokat, amelyek egy, a többi pontot körülölelő konvex sokszöget alkotnak, majd őket egy adott körüljárási iránynak megfelelően sorbatéve egy listában tároljuk.

A szomszédos pontpárokat sorban ellenőrizzük és ha azok távolsága meghalad egy hiperparaméter által definiált értéket, akkor vektor műveletekkel kiszámoljuk az általuk alkotott szakasz felezőpontját, majd egy 0 és 360 között véletlenszerűen választott számértéknek megfelelő állásszögű, hiperparaméternek megfelelő hosszúságú vektort adunk, így az eredeti két pont közé egy további kanyart iktattunk. Ezt addig folytatjuk, amíg van olyan egymást követő pontpár, melyek túl messze vannak egymástól.

Következő lépésként a fent említett ponthalmaz első elemére állítjuk jármű modellünket, és addig haladunk vele a következő pontig amíg, az egy adott közelségbe nem ér, ekkor a következő pontot kezdjük közelíteni. Így eljutva az utolsó pontba is. A haladás során lépésenként tároljuk el a jármű modell x-y koordinátáit és a heading-et, ezek lesznek az út középpontjának pontjai.

A pályavégék összezárása során a modellünket az előző lépés befejező állapotában hagyjuk. Hasonlóképpen az eddigiekhez, most az első pont felé vesszük az irányt és egy adott, de mostmár jóval kisebb távolságra közelítjük meg. A megközelítés után ellenőrizzük a jármű és a pont heading-ét, ha ezek között maximum 1 fok eltérés van akkor az utat lezártnak tekintjük. Végezetül a becsatlakozási pont előtti pontokat töröljük a listából.

Ezzel a zárt pályánk előállt. Már csak az út szélességének kalkulálására van szükség. Ezt úgy teszük, hogy az út középpontjának minden egyes pontjára egy menetirányra merőleges vektort állítunk (mindkét irányba), melynek hossza az útszélesség felének felel meg. Így tehát a véletlenszerűen generált zárt pályánk előállt.

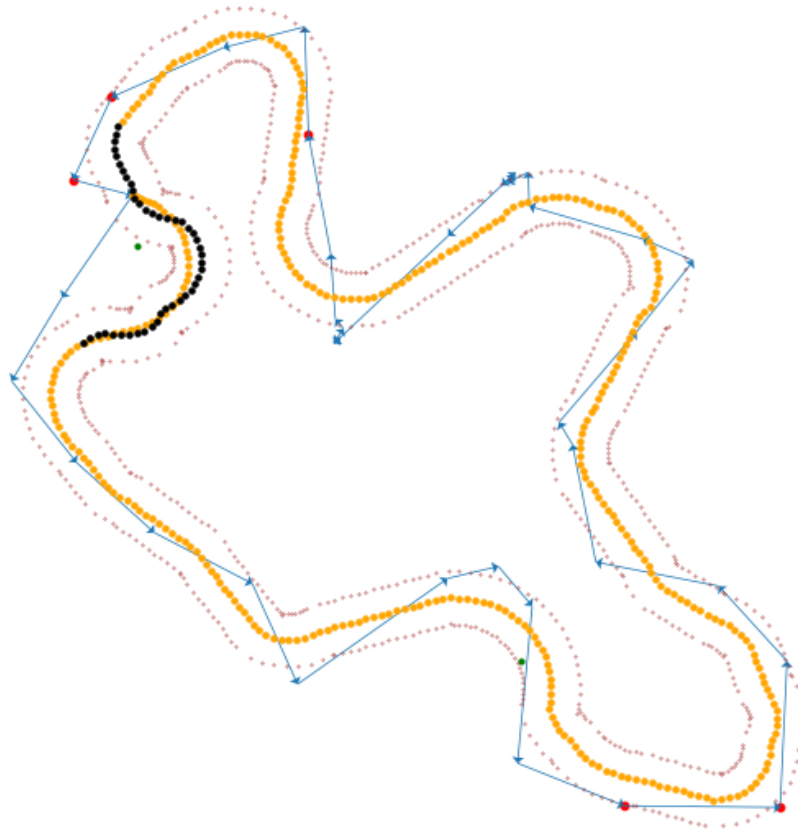


Figure 2.1: A pályagenerálás folyamata

A 2.1 ábra mutatja a generálás összes állapotát:

- Zöld pont: kezdeti ponthalmaz
- Piros pont: konvex sőkszög csúcsai
- Kék nyilak: a sorrendbe rakott, további kanyarokkal ellátott pontok sora
- Sárga pontsor: az út középpontja az utolsó pontig
- Fekete pontsor: az út középpontja az utolsó ponttól a lezárásig
- Kicsi barna pontsor: útszélek

Fontos lépése a pályagenerálásnak a már elkészített pályák tesztelése, hisz a kész pályának a következő kritériumoknak kell megfelelnie, ahhoz hogy a tanítást és tesztelést megfelelően el tudjuk végezni:

- A pálya trajektória nem keresztezheti önmagát
- A sávszélei nem érinthetik egymást
- A pályának zártnak kell lennie

A pályagenerálást követő tesztelés esetén, ha kiderül, hogy a generált pálya nem felel meg a kritériumoknak, akkor új pálya generálása történik. A sikeresen generált pálya fájlba való kiírását, és beolvasását is támogatja a generátor, ezzel könnyítve és gyorsítva a teszteléseket.

2.2 Kinematikai jármű modell

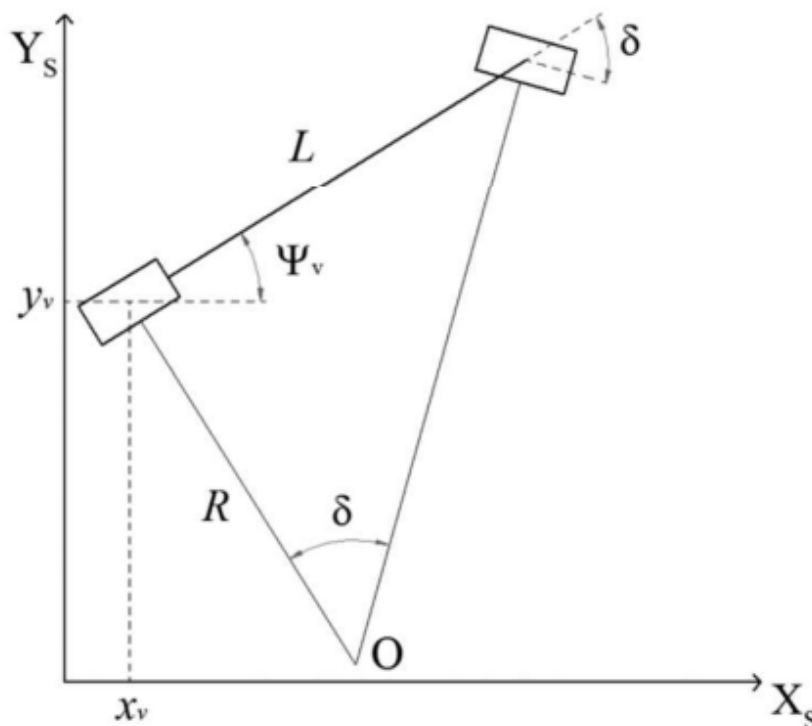


Figure 2.2: Kinematikai jármű modell

Mind a tanítás és mind a tesztelés elengedhetetlen része egy jármű modell, amely a bemenő beavatkozási adatok függvényében adja meg saját pozícióját és orientációját, ezt kezdetben egy kinematikai modell valósította meg.

A kinematikai jármű modellelmet a 2.2 ábrán látható adatok definiálják. Melyek következőket jelentik:

- L : tengelytáv ("jármű hossza")
- Ψ : orientáció
- δ : kormányszög

A feladat szempontjából a járművet pontszerűnek tekintettem a jármű tömegközéppontja alapján.

Működését tekintve a modell gyorsulás és kormányszög bemenő paramétereként és az időbeli diszkretizáció szerint kiszámítja az új állapotát, amely a következő adatok összessége:

- X-Y koordináta párok, melyek a tömegközéppontot határozzák meg
- Orientáció, mely a vízszintes tengelytől vett szögeltérés előjelesen
- Pillanatnyi sebesség, mint skalár érték

A modell kinematikai jellegénél fogva elhanyagolja az erőtan hatásokat, így a kerék-talaj surlódásból eredő szlipet is.

Jelenlegi munkáim során a modell sebessége állandó volt, így a gyorsulás konstans nulla volt. A kormányszögek pedig diszkrét értékeket vehettek fel. Az egyik érték minden esetben a 0 fok volt, ezen kívül a maximális kormányszöget vehette fel mindkét irányba.

2.3 Dinamikai jármű modell

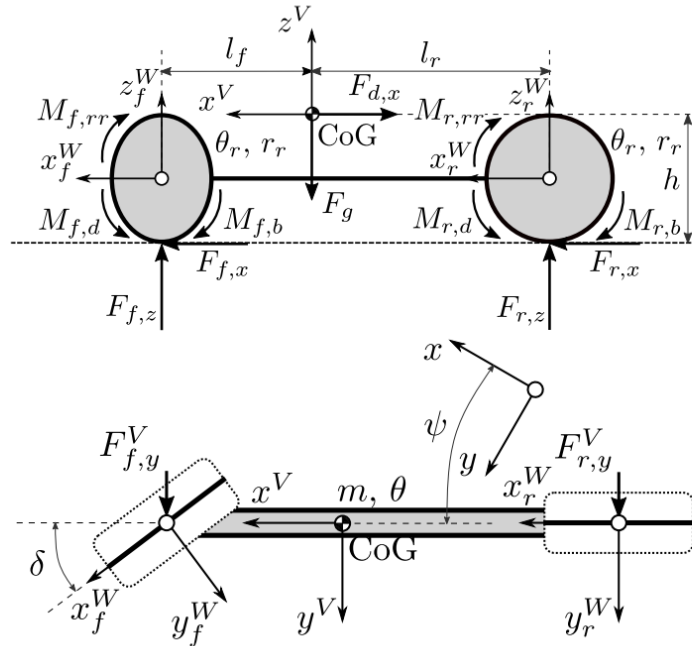


Figure 2.3: Dinamikai jármű modell

A modellt a 2.3 ábra szerint a következő adatok definiálják:

- m : tömeg
- θ : A kocsiszekrény inerciája
- l_f : Az első tengely és a tömegközéppont vízszintes távolsága
- l_r : Az hátsó tengely és a tömegközéppont vízszintes távolsága
- h : A tömegközéppont magassága
- θ_f/r : A kerekek inerciája
- r_f/r : A kerekek rádiusa

Fejlesztés során dinamikai jármű modellt is használtam, így közelítve jobban a valóságot. Ebben az esetben a beavatkozásokat a gyorsítónyomaték és a kormányszögek jelentették, melyek a kinematikai modellhez hasonlóan alakultak. A kinematikai modellel szemben problémát jelentett a megcsúszás lehetősége, ennek kiküszöbölésére minden lépés során a kívánt (állandó érték) és a pillanatnyi sebesség különbségével arányos hajtónyomatékot kapott beavatkozásként, így tartva a sebességet a kerék kipörgése nélkül.

2.4 Környezet vizualizációja

Az eredmények szemrevételezésére és a fejlesztés koordinálásához szükségünk volt valamilyen vizualizációra. A legkézenfekvőbb megoldást választottam, a jármű modell pozícióját jelenítettem meg pontként minden lépése során. A 2.4 ábra mutatja, hogy a folyamat nagyon jól nyomonkövethető, mert nem animációszerűen látom a jármű pillanatnyi állapotát, hanem a már bejárt trajektóriával is tisztában vagyok. Ezeket szükség esetén a döntések, vagy bejövő állapot reprezentációk terminálban való megjelenítésével egészítettem ki.

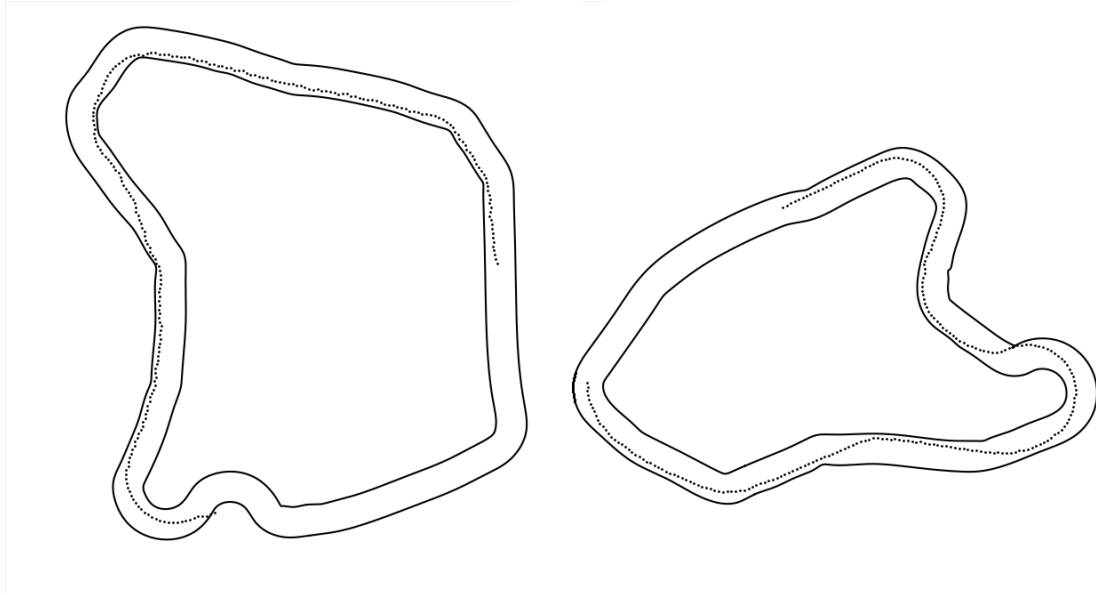


Figure 2.4: A környezet vizualizációja

2.5 Rapidly-exploring Random Tree

Az RRT algoritmus egyike a témában közkedvelt algoritmusoknak. Funkcióját tekintve egy mintavételezésen alapuló útvonaltervező algoritmus. Az ilyen módszerek fő feladata, hogy akár akadályokkal ellátott terekben is találjanak egy útvonalat a kezdőponttól a célpontig.

Algorithm 1 Rapidly-exploring Random

```

 $G.init(q_{init})$ 
 $q_{new} \leftarrow q_s$ 
for  $k = 1$  to  $K$  do
     $q_{rand} = RANDOMNODE()$ 
     $q_{nearest} = T.NEARESTNEIGHBOR(q_{rand}, G)$ 
     $q_{new} = NEW(q_{near}, q_{target}, dt)$ 
     $G.addnode(q_{new})$ 
     $G.addedge(q_{near}, q_{new})$ 
return  $G$ 

```

Az RRT esetében ez a következőképpen valósul meg. Kezdetben a ponthalmazunk csak a kezdőpontot tartalmazza, első lépésként pedig egy véletlenszerűen választott ponthoz legközelebbi pontot keresi meg, a már meglévő ponthalmazból. Amikor ez megvan, a két pontot összekapcsolja. Mivel kezdetben csak a kiinduló pont található a ponthalmazban, így az első véletlenszerűen generált pont biztosan hozzá kapcsolódik. A másodikként generált pont már a kettő meglévő pont közül a hozzá közelebb esőhöz kapcsolódik. Ezeket a lépéseket végtelenszer iterálva a tér minden pontjába eljutunk, így a cél pontba is.

Az eredeti RRT algoritmus megismerése után, különböző megszorításokkal próbálkoztam a trajektória területe tekintetében. Korlátoztam a fa csomópontjainak távolságait, és az egymást követő vektorok szögeltéréseit. Ezzel az volt a célom, hogy a trajektória valamivel közelebb legyen egy jármű mozgásához. Az algoritmust egy akadályokkal, és egy célterülettel rendelkező környezetbe helyeztem, ennek eredményeit a 2.5 ábra mutatja.

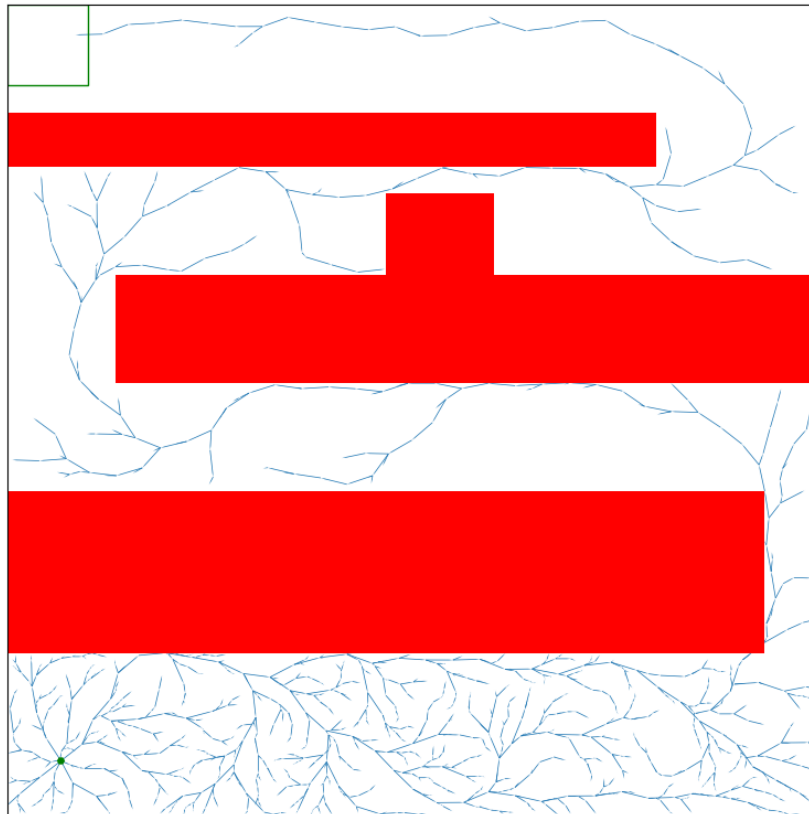


Figure 2.5: Eredmény

2.6 Az RRT és a jármű modell összekapcsolása

Az RRT járműmodellel való üzemelését úgy oldottam meg, hogy a véletlenszerűen kiválasztott ponthoz legközelebbi pontban eltárolt heading-et beállítottam a járműnek (az adott pontot mindig úgy tároltam el, hogy hozzá társítottam a jármű aktuális állapotát), majd ebből a helyzetből indulva, meghívtam a Step függvényt, olyan kormánysszöggel, hogy a jármű a

véletlenszerűen kiválasztott pont felé haladjon. Ennek eredményeit a 2.6 ábra mutatja, ahol a jármű egy egyenes sávon belül halad.

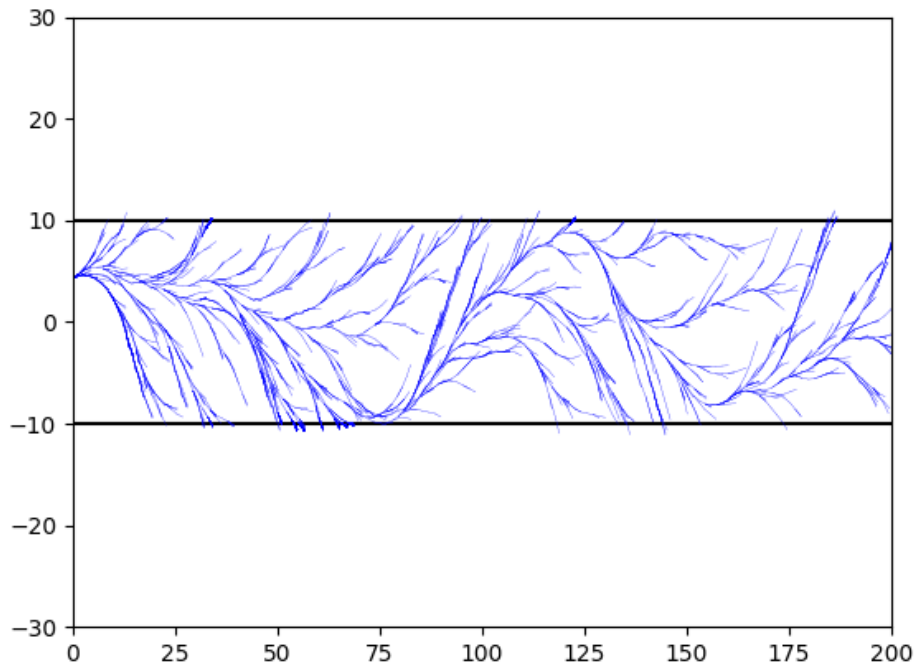


Figure 2.6: Haladás egyenes sávon

Az ábrán tökéletesen kivehető a korábban szemléltetett fához hasonló szerkezet, de már teljes mértékben járművel lekövethető, megvalósítható formában.

2.7 Megerősítéses tanulás

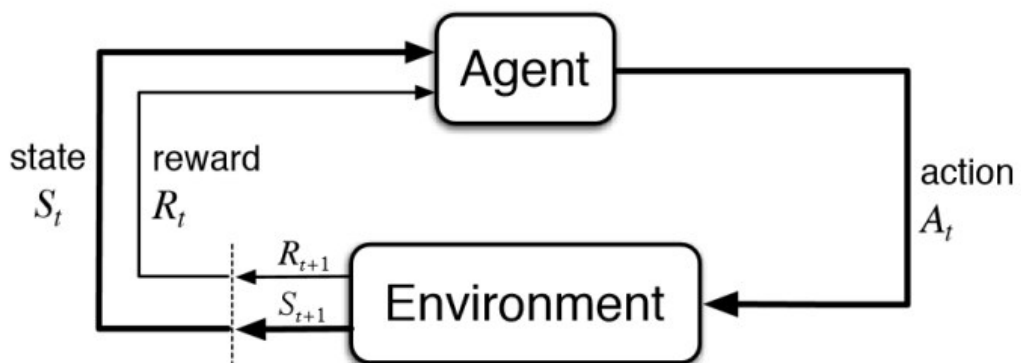


Figure 2.7: A Mély Q-Tanulás elvi alapja

A megerősítéses tanulás egy, a többi gépi tanulási módszertől elkülönülő rendszer. Működési

alapját a 2.7 ábra mutatja. Ahogy az ábra is mutatja az ügynökünk egy beavatkozással hat a környezetünkre, erre a környezet egy jutalom/büntetésként értelmezhető skalár visszacsatolással karakterizálja az adott beavatkozás értékét, illetve az aktuális állapot reprezentációt is átadja. Az ügynök az állapot reprezentáció tudatában választja meg a következő beavatkozást. A továbbiakban kicsit részletesebben mutatom be a Deep Q-Learning algoritmust.

Első fontosabb fogalomként a Q-táblát említeném. Ez a táblázat két kiterjedése irányában tartalmazza az összes lehetséges beavatkozás és állapot reprezentáció értéket, mindegyik beavatkozás-állapot reprezentáció párhoz tartoztatunk egy számértéket. A számértékek az adott állapot esetén végrehajtott adott beavatkozás esetén a "játék" (továbbiakban epizód) végéig maximálisan szerezhető jutalmak mennyiségét jelöli. Könnyen belátható, hogy ha a táblázatunk teljesen ki van töltve, akkor minden állapot reprezentációban tudjuk, hogy milyen beavatkozást kell végrehajtanunk, azt amelyiknek a legnagyobb a Q értéke.

Ahogy láthatjuk a Q-táblázat nagy segítség lehet bizonyos feladatok esetén, de vannak korlátai sajnos. Abban az esetben ha az állapot reprezentáció terünk túl nagy, a Q-tábla is óriási méreteket ölt és ennek feltöltése rengeteg időt és munkát venne igénybe. Ha kicsit távolabbról szemléljük a dolgokat akkor láthatjuk, hogy a Q-táblázat gyakorlatilag egy állapot reprezentációtól és beavatkozástól függő függvény: $Q(a, s)$. Ebből kifolyólag a Q-táblázat tetszőleges pontossággal közelíthető egy függvény approximátorral, mi egy neurális hálózatot fogunk erre a célra használni.

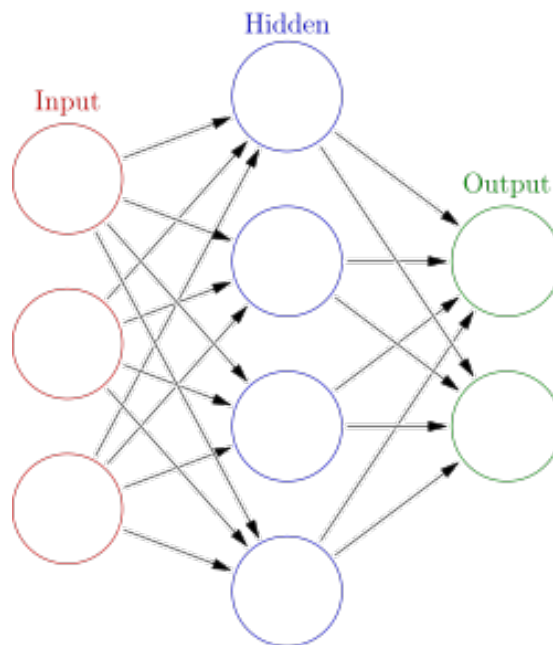


Figure 2.8: A Neurális Hálózat

A neurális hálózatok alapelvét a 2.8 ábra mutatja. Látható, hogy van egy n dimenziós vektorunk mint bemenet, illetve egy m dimenziós vektorunk mint kimenet. A neuronok közti

kapcsolat során az értékekkel különböző mátrix műveletek végződnek, és ezek összesített eredménye adja az m dimenziós vektor kimenetet. A hálózatok betanítása klasszikus esetben a következőképpen zajlik. A bemenetként beadott vektorunkat átadjuk a hálózatnak, az pedig egy kimenet vektorral válaszol, majd az elvárt kimenettel összehasonlítjuk, az eltérések függvényében változtatunk a hálózat súlyain. Hálózat súlyain azokat az értékeket értjük, amelyektől függ, hogy a bemenet vektor skalárjai milyen változáson mennek keresztül. Az eltérés felméréséhez egy úgynevezett veszteség függvényt használunk, melynek milyensége feladatfüggő. Ezt a módszert felügyelt tanításnak nevezzük.

Alapvető probléma sok esetben, hogy nincs információnk arról, hogy az adott állapot reprezentáció esetén mi a "helyes megoldás", az elvárt kimenet nem ismert. Erre kínál megoldást a DQN algoritmus.

Ha belegondolunk a Q -táblázat működésébe, észrevehetjük, hogy egy adott állapot reprezentációhoz és beavatkozáshoz tartozó Q érték megegyezik az erre a "lépésre" kapott jutalom és a következő állapot reprezentáció és a hozzátartozó legkedvezőbb beavatkozás Q értékének összegével. Ezt az egyenletet Bellman-egyenletnek hívjuk.

$$Q(s, a) = r + \gamma * Q_{max}(s', a) \quad (1)$$

Így már kicsit közelebb kerültünk a válaszhoz, hisz ha a neurális hálózatunkba betápláljuk az állapot reprezentációt és a beavatkozást, illetve a következő állapot reprezentációt amibe kerültünk, akkor a Bellman egyenlet szerinti reláció alapján már ki tudjuk számolni a veszteség értéket. Ez úgy történik, hogy

$$r + \gamma * Q_{max}(s', a) \quad (2)$$

kifejezésből adódó értéket tekintjük pontos értéknek, a neurális hálózat által prediktált pedig $Q(s, a)$ érték lesz. Ha kicsit jobban megvizsgáljuk a neurális hálózatot mint függvény approximátort, akkor láthatjuk, hogy valójában a hálózat súlyaitól is függ,

$$Q(s, a, \theta).$$

Így viszont a Bellman egyenlet a mi esetünkben így módosul:

$$Q(s, a, \theta) = r + \gamma * Q_{max}(s', a, \theta) \quad (3)$$

Ebből láthatjuk, hogy ha a fent említett módszerrel számolunk egy veszteség értéket, majd ennek alapján változtatunk a hálózat súlyain, akkor mind a két prediktált érték változik. Ebben az esetben pedig a két érték nem tud egymáshoz a relációnak megfelelően konvergálni, mert egyik változása a másik változását vonja magával. Itt leledzik a DQN algoritmus másik zsenilitása: a DQN algoritmus kettő neurális hálózatot használ, melyek kezdetben

azonosak. Az egyiket (fő) arra használja, hogy a $Q(s,a)$ értéket prediktálja, a másikat (cél) arra, hogy a $Q(s',a)$ értéket. A fő hálózat minden tanítási lépés után frissül a veszteség értéknek megfelelően, de a cél hálózat csak egy előre meghatározott k tanításonként másolódik a fő hálózatból. Ezzel azt érjük el, hogy k tanításig a fő hálózat megfelelően konvergálhat, és csak ez után változik a cél hálózat értéke. A Bellman-egyenletünk ez esetben így fest:

$$Q(s, a, \theta_n) = r + \gamma * Q_{max}(s', a, \theta_t) \quad (4)$$

Mostmár tudjuk, hogy tanítjuk be hálózatunkat, de maradt még egy fontos dolog hátra. A tanításhoz szükséges mintákat nem a műveletvégzés pillanatában bejövő adatokból nyerjük, hanem egy pufferben tároljuk el azokat, és ezek közül véletlenszerűen választunk a tanulás során. A tanuló minták a következő formát öltik:

$$(s, a, r, s', t) \quad (5)$$

ahol s : állapot reprezentáció, a : beavatkozás, s' : következő állapot reprezentáció, r : jutalom, t : terminál (véget ért-e az epizód) jelölések érvényesek.

A tanulás folyamatának fontos témája a beavatkozás választás, amely klasszikus esetben az ε -greedy módszernek megfelelően történik. Ennél a módszernél kezdetben 1 az ε értéke, és minden tanuló lépésnél megszorozzuk egy ε -decay értékkel, ami egy 1-nél kisebb szám, így az folyamatosan csökkenni fog. Minden lépés előtt az algoritmus véletlenszerűen előállít egy számot nulla és egy között, amennyiben ez a szám az ε aktuális értékénél nagyobb, az ügynök a hálózat alapján prediktált legjobb beavatkozást választja, ellenkező esetben véletlenszerűen dönt. Ezzel szimulálni tudjuk a hálózat tanulását, és magasabb szintű betanultság esetén nagyobb valószínűséggel választ beavatkozást a hálózat döntése alapján.

2.8 RRT és DQN ötvözése

Munkám során viszont nem az ε -greedy módszert alkalmaztam a beavatkozások megválasztására, hanem a korábban ismertetett RRT algoritmus segítségével állítottam elő a tanuló mintákat. A későbbiekben ismertetem az RRT algoritmus alkalmazásának mikéntjét. A kész fa, amely a 2.9 ábrán látható, minden egyes iteráció során tárolja az aktuális állapot reprezentációt, beavatkozást, az új állapot reprezentációt, amibe került és a hozzá tartozó jutalmat. Olyan okból kifolyólag, hogy esetünkben a "játéknak" nincs jól meghatározott vége, az epizód végét egy adott pont terjedelem jelentette; természetesen a terminal értéket is tároltuk a bufferban.

A bal oldali ábrán szemléltetés céljából egy, kis méretű ponthalmazból álló fát jelenítettem meg, így jól láthatjuk a tanulóminták pontjait. A jobb oldali ábra pedig egy magasabb iterációs szám eredményeként született trajektórián keresztül mutatja be a fa terjedését, folytonos formában szemléltetve.

A kombinált algoritmus folyamatábrája alább látható. A módszer 2 fő műveletre bon-

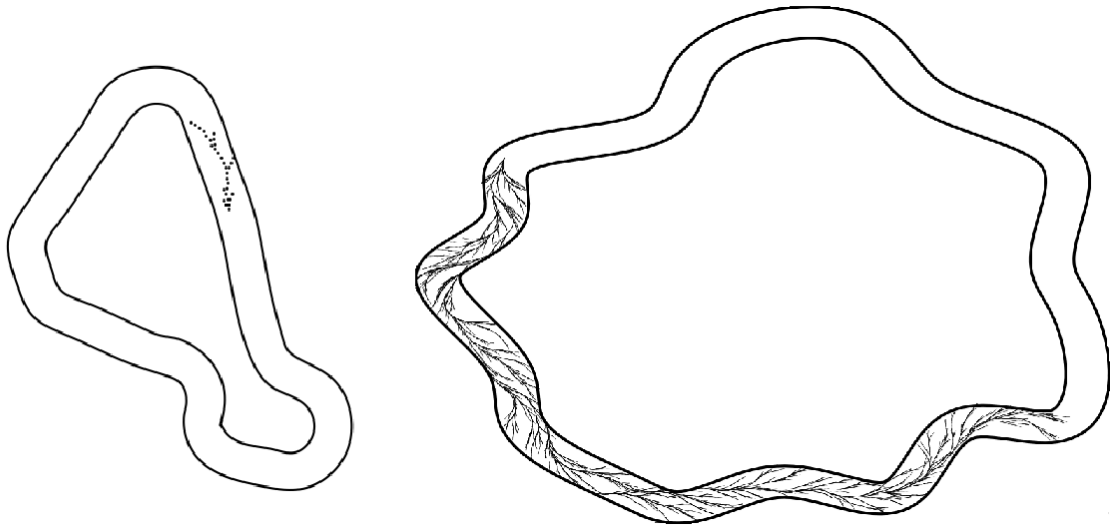
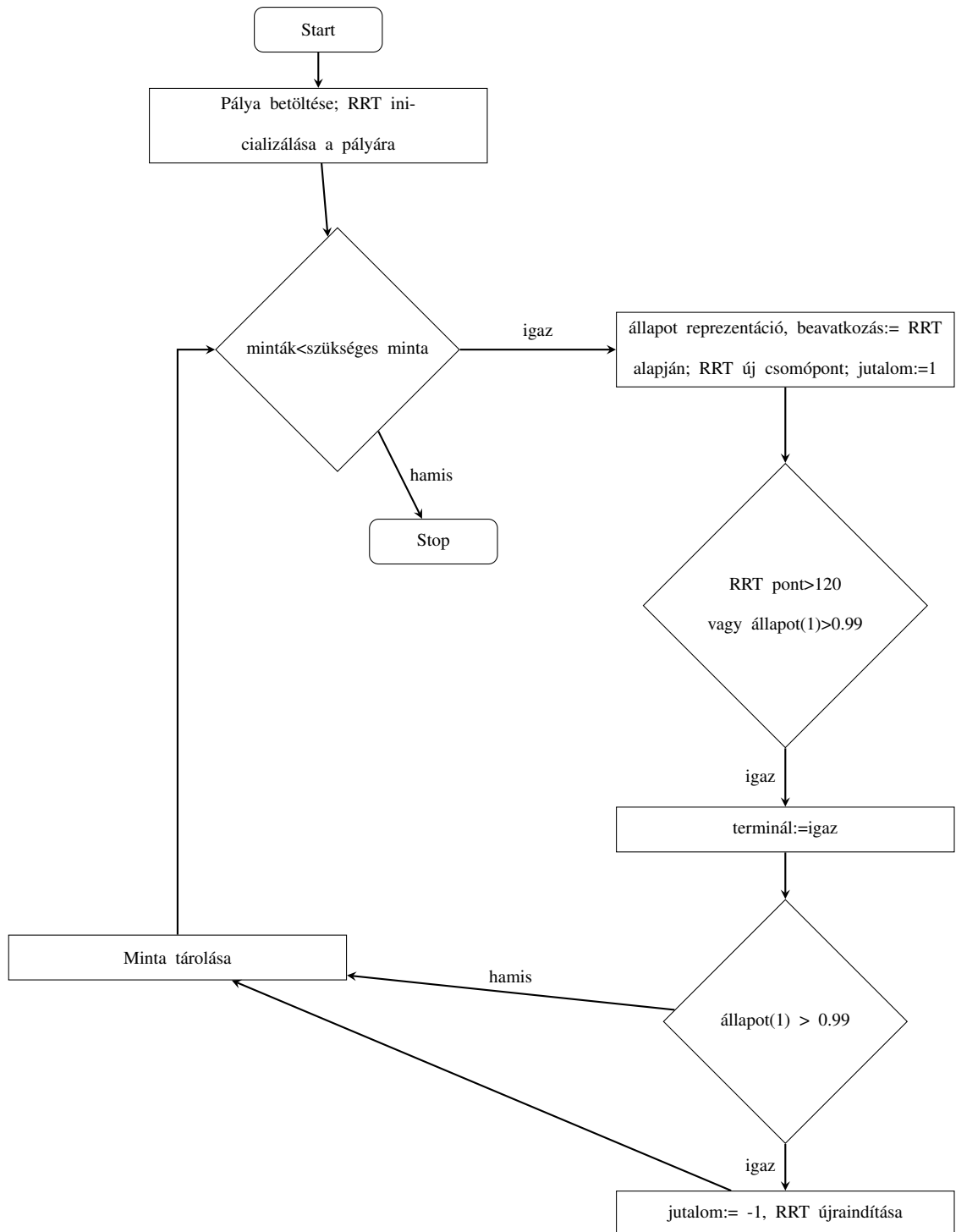
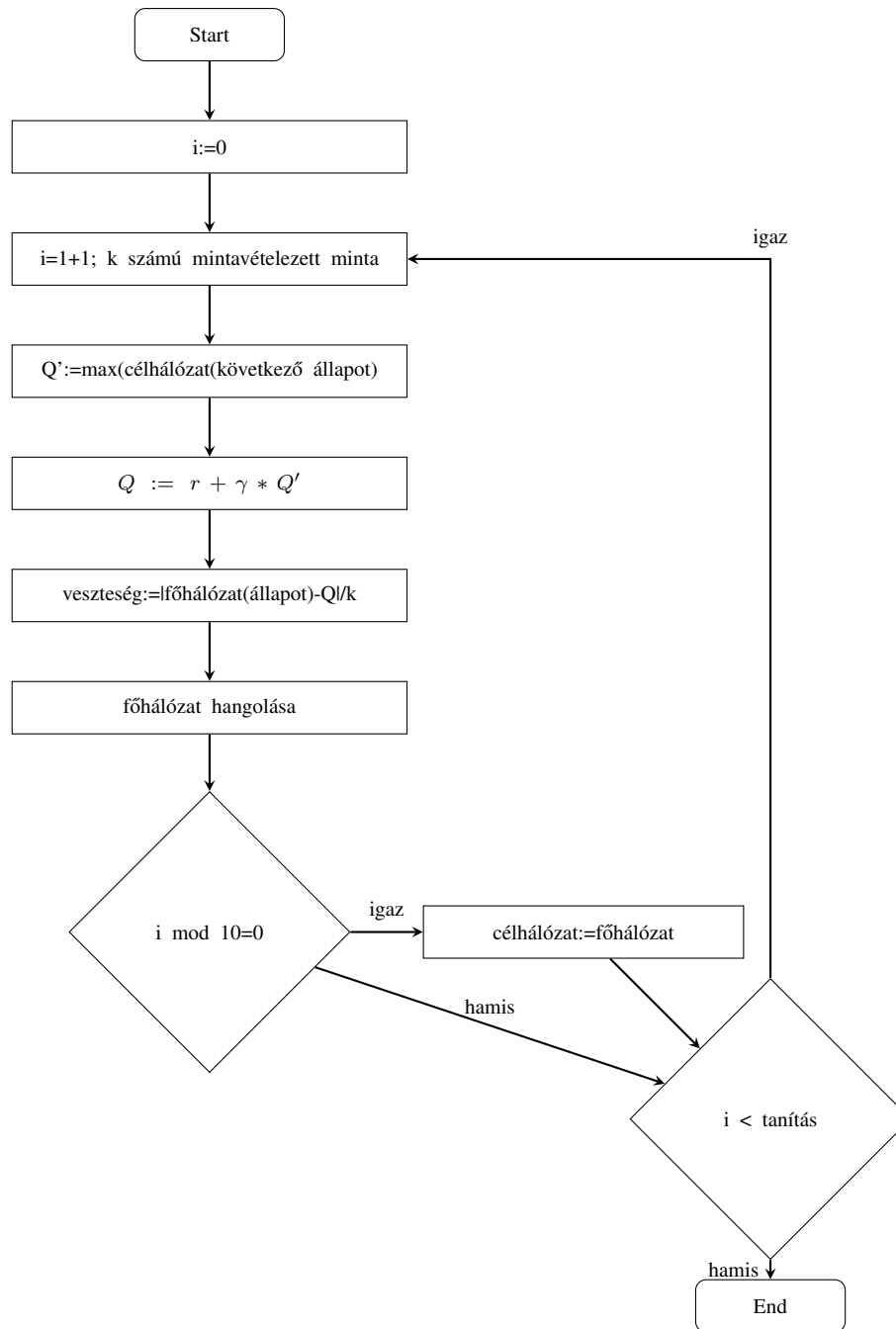


Figure 2.9: Az RRT által generált trajektória, egy és több "fa" esetén

tható, a tanuló minták készítésére, és a tanításra. A tanuló minták gyűjtése és a tanulás:





2.9 Tanítás folyamata

A problémával való ismerkedést a kinematikai jármű modellel kezdtem. Ez valamivel könnyebb szituációt eredményezett, mert a jármű képtelen a megcsúszásra és tetszőleges kormánysszöggel irányítható, így a nagy görbületű kanyarokban is könnyen befordítható. Ebben az esetben az állapot reprezentáció egy 2 elemű vektort jelentett, amely az útközépvonalától való távolságot és az út középvonalához viszonyított relatív szöget jelentette (mindkettő előjeles érték, -1 és 1 közé skálázva).

A jutalmak meghatározására 3 függvénnyel próbálkoztam. Ezekben azonos volt a közép-

től vett távolság feldolgozása: $-|d|$, ahol d a relatív távolság. Ehhez adtam hozzá a relatív szöget feldolgozó tagot, amelyek a következők voltak:

1. $\cos(\alpha * (\pi/2))$
2. $1 - |\sin(\alpha * (\pi/2))|$
3. $1 - |\alpha|$

A 2.10 ábrán a három függvény diagrammja látható.

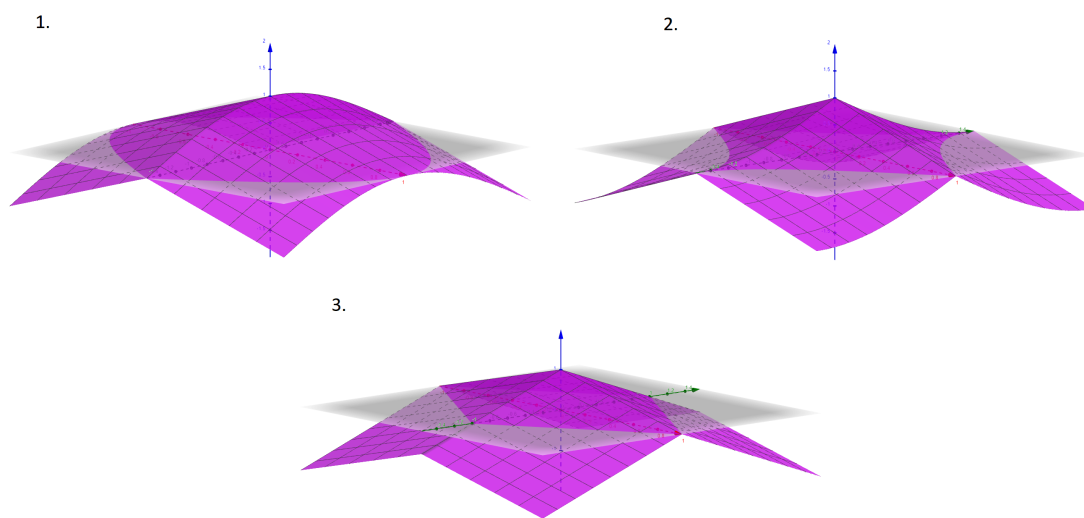


Figure 2.10: A jutalom függvények

A módszer eredményeibe a 2.11 ábra ad betekintést, ahol jól látható, hogy a modell igyekszik a sáv közepén tartózkodni.

A korábban mutatott 2.9 ábrán jól látható, hogy sok minta begyűjtése esetén a tanulóminták a pálya egészét behálózzák. Ez a vizualizáció viszont nagy adathalmaz esetén kevésbé követhető, így szükségem volt valami másra is, hogy betekintést nyerhessek a tanulómintáimba.

A két dimenziós állapottérnek köszönhetően egy gyakorisági diagramm segítségével ellenőrizni tudtam a tanuló minták alakulását, vagyis a "fa" trajektória területét az állapottérben. Az ábra vízszintes tengelyén a relatív szögeltérés, a függőleges tengelyén pedig a relatív távolság látható.

Ugyanezt a diagrammot felhasználva megjelenítettem a tanulás során felhasznált minták gyakoriságát, illetve a tesztelés során jelentkező állapot reprezentációkat is.

Másik hasznos segítségem az a hő térkép volt, amely azt mutatta meg, hogy a hálózat betanulása szerint az egyes állapotok mennyire előnyösek. A diagramm vízszintes tengelyén a jármű relatív szögeltérése helyezkedett el, a függőleges tengelyén pedig a relatív távolsága

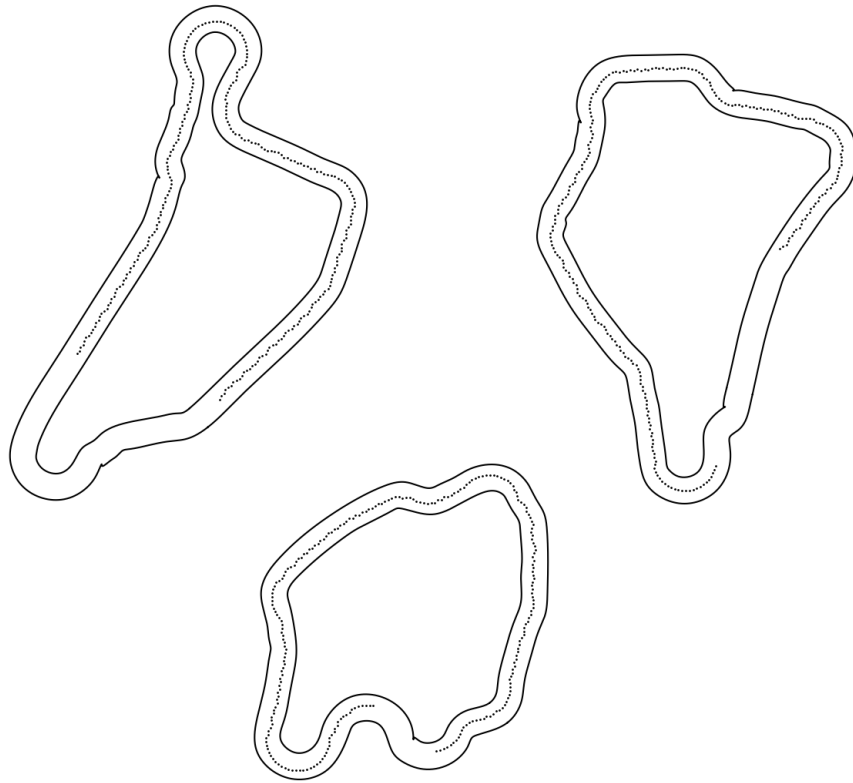


Figure 2.11: A kinematikai modell eredményei

az út középvonalától. Az egyes pontok színe annál világosabb, minél nagyobb Q értéket prediktál hozzá a hálózat, a legelőnyösebb beavatkozás esetén. Használtam még egy ehhez hasonlatos diagrammot, amely a Q érték helyett a legmagasabb Q értékű beavatkozás indexét társította egy adott ponthoz. Így láthatjuk, hogy az állapottér egyes eseteiben milyen döntést hoz a hálózat.

A 2.12 ábrán egy teljes vizualizációt mutatok be, ahogy balról jobbra haladva láthatjuk:

1. A bufferban tárolt minták.
2. A tanulás során használt minták
3. A tesztelés során feldolgozott minták
4. A betanult hálózat Q érték hő térképe
5. A betanult hálózat beavatkozás szerinti hő térképe

A 2.12 ábrát vizsgálva tanulságos következtetéseket vonhatunk le a folyamat tekintetében. Az első diagrammról leolvashatjuk, hogy a modell a tanulás során a sáv szélét érintette, de a szögelfordulás meg sem közelíti a 90 fokot. Ezek jó gondolatébresztők lehetnek az állapot

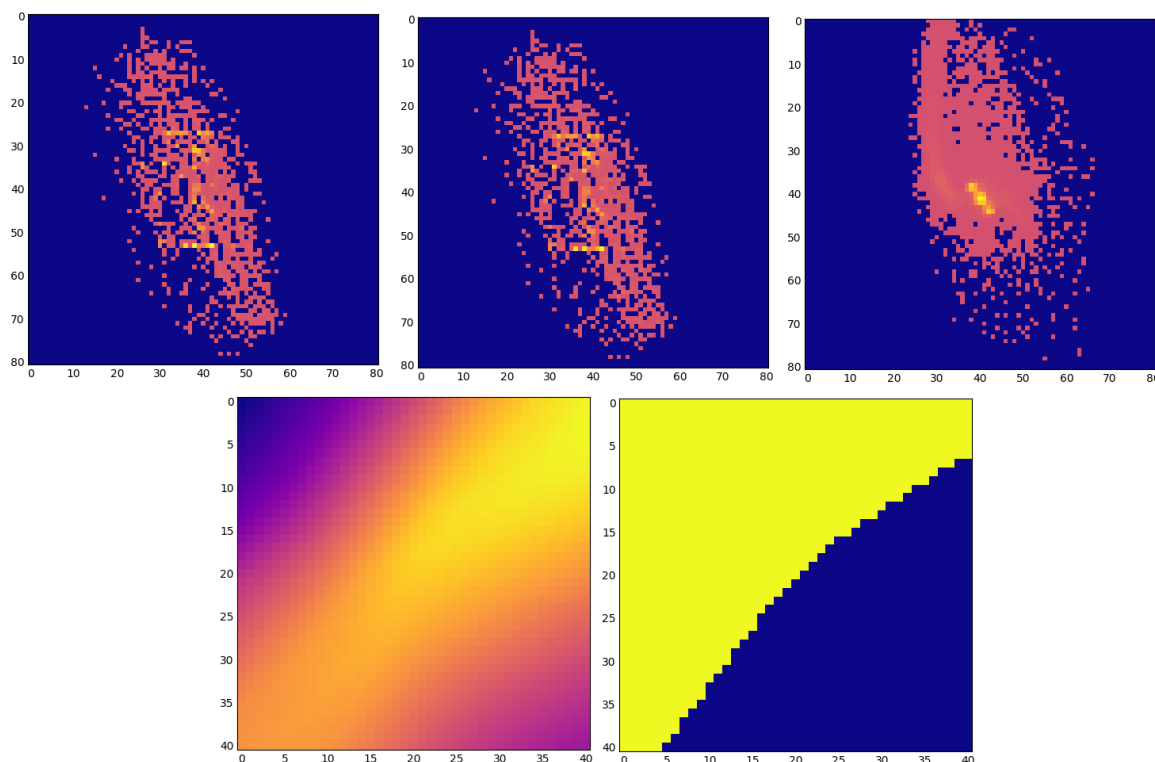


Figure 2.12: A minták és a velük tanított hálózat

reprezentáció, és a jutalom meghatározó függvények átgondolására, hisz valamely esetekben ezek a függvények a $[-90, 90]$ fokos tartományra lettek hangolva.

A második ábra egy gyakorisági diagramm, ahol a gyakoriság tekintetében logaritmikus skálát használtam. Az ábra még így is mutatja, hogy bizonyos minta nagyságrendekkel több alkalommal lett kiválasztva, láthatjuk, hogy ennek oka, hogy már a mintagyűjtés során nagyobb arányban voltak ezek a minták. Ennek hangolására is érdemes lehet energiát fordítani, így gyorsítva a tanulást.

A tesztelés során jelentkező állapotokat vizsgálva szembetűnő jelenség, hogy az egyik irányba való kitérések száma jelentősen több mint a másik irányba; de egyben azt is láthatjuk, hogy a szögelfordulás esete hasonló.

A jobb oldali, alsó ábrarész mutatja, hogy az állapot vektor függvényében, hogyan avatkozik be a modell; jól mutatja az ábra, hogy ez a betanulás egy kezdeti stádiuma, a beavatkozások elrendeződése még nem tökéletes. A bal oldali, alsó diagramm az állapot vektor függvényében mutatja meg a Q értékek maximumát; jól látható a szimmetria, miszerint a bal és jobb irányba való kitérés és elfordulás azonos lehetőségeket nyújt.

A hálózat állapotát jelző rajzok alapján láthatjuk, hogy a hálózat a valóságnak megfelelően minősíti az egyes állapotokat, és az ábra a 45° -os egyenesre szimmetrikus. A beavatkozásokot mutató ábrára tekintve láthatjuk, hogy az csupán 2 beavatkozást tartalmaz: az ügynök semmilyen esetben sem tartja előnyösnek az egyenesen haladást. Ezzel többször találkoztam a munkám során, ezek alapján átgondolandó, hogy szükséges-e a beavatkozás vektor

módosítása.

A dinamikai jármű modell megcsúszásából eredő problémák közelítéséhez továbbra is a kinematikai modellal dolgoztam, azzal a változtatással, hogy csökkentettem a maximális kormánysszöget, így egy élesebb kanyar beviteléhez az ideális ív megtalálása is szükséges volt.

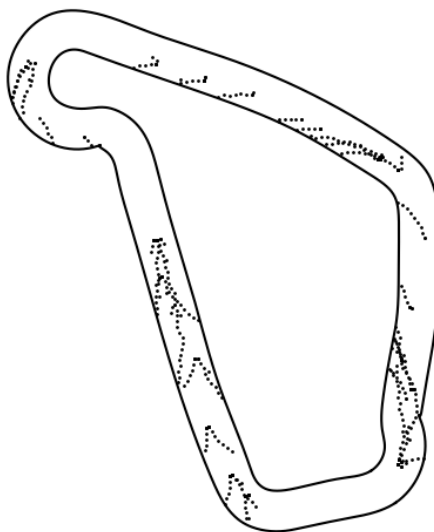


Figure 2.13: Az új minták gyűjtése

Az új esetet vizsgálvabeláthatjuk, hogy a korábban sikeres jutalom meghatározó algoritmusaink itt nem fognak sikert nyújtani, hisz azok a pálya középvonalaiba kényszerítik a járművet. Ebben az esetben az epizódonkénti jutalom osztással dolgoztam; az ügynök minden lépés során 0 jutalmat kapott, kivéve, ha az az utolsó lépés volt az epizód során, ekkor -1-et kapott, ha a pályán kívül volt és 1-et ha a pályán belül. Az epizódok hosszát úgy határoztam meg, hogy egy adott lépés teljesítése, vagy a pályán kívülre kerülés jelentette az epizód végét. Igyekeztem az epizód hosszát úgy megválasztani, hogy a pozitív és negatív jutalmak egyensúlyban legyenek.

Az új jutalom meghatározó módszer már lehetővé teszi a megfelelő betanulást, de az állapot reprezentációnk nem elégséges a pálya elkövetkező íveinek felmérésére. Ehhez további tagokkal bővítettem a vektort, ezek a tagok egy egymástól egyenlő távolságban elhelyezkedő pontokban számítják a pálya relatív szögeltérését a pálya jelenlegi szakaszához képest, így tájékoztatva az ügynököt a várható ívekről. Ezzel a két változtatással már betanítható volt a hálózat, a tanuló minták gyűjtését a 2.13 ábra mutatja. Jól látható, hogy a fa terjedése megáll amint valamely pontja eléri a pálya szélét.

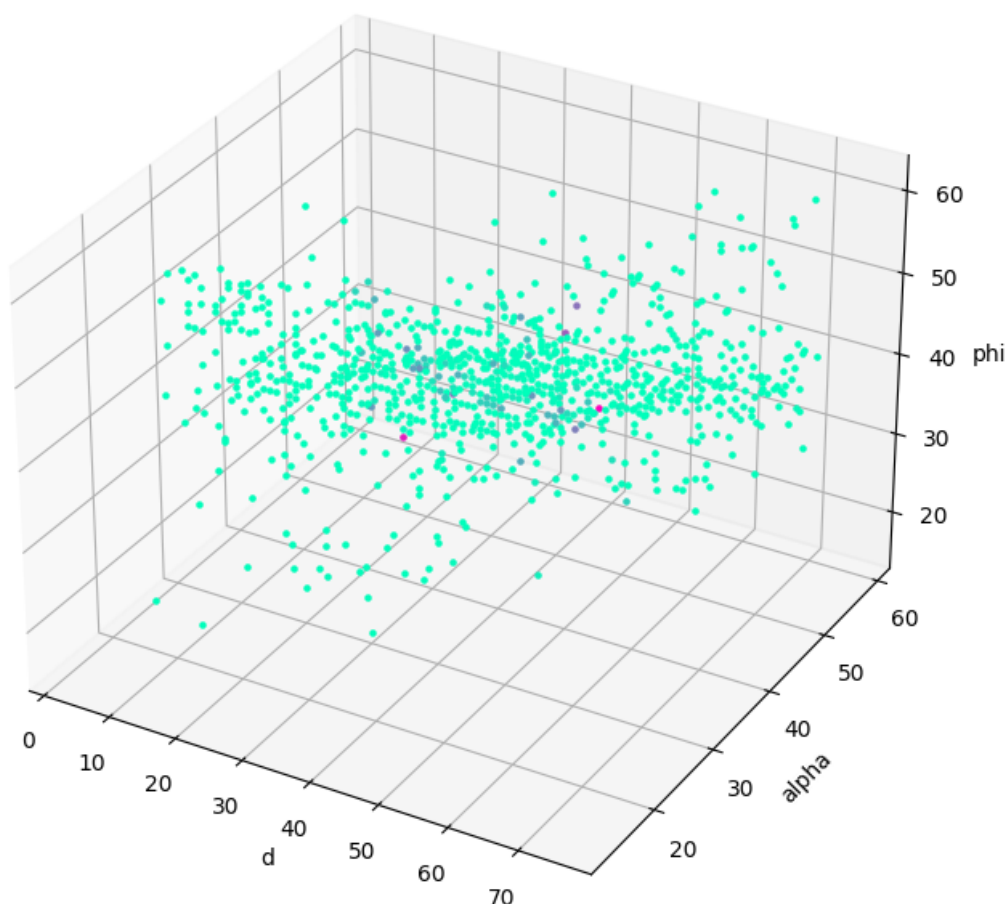


Figure 2.14: A 3 dimenziós állapot reprezentáció esete

Abban az esetben, ha egy előrejelző szögünk van, egy 3 dimenziós állapotterről beszélünk. Ebben az esetben a korábbi diagrammokhoz hasonlóan készíthetünk hő térképeket. Bár a fejlesztéseim során jobbra tíz darab előrejelző szöggel végeztem a tanításokat, az egy darab előrejelző szöggel gyűjtött minták 3 dimenziós megjelenítése tanulságos lehet, az állapottérben való terület mértékébe adhat betekintést.

Az 2.14 ábra jól mutatja, hogy az RRT-vel való üzemelésnek köszönhetően a minták megfelelően terülnek a kettőnél több dimenziós állapotterek esetén is, már kevés minta során is (1000).

A különböző vizualizációk nem csak a tanulóminták felügyelése során voltak hasznosak, hanem a tanulás eredményének és folyamatának vizsgálata során is nagy segítségemre voltak, különösen a magas dimenzió számú állapottér esetén.

Az egyik ilyen fontos kérdéskör, hogy az állapot reprezentációs vektor, mely tagjai milyen mértékben befolyásolták a hálózat eredményét, hiszen ennek tudatában hangolni tudjuk, hogy milyen információkat használunk a tanítás során. Ennek kiértékelésére egy oszlop diagrammot készítettem, ahol minden egyes állapotleíró skálához összegeztem, hogy egy 1000 pályán való tesztelés során az egyes beavatkozások meghozásához milyen mértékben használta a hálózat az adott adatot.

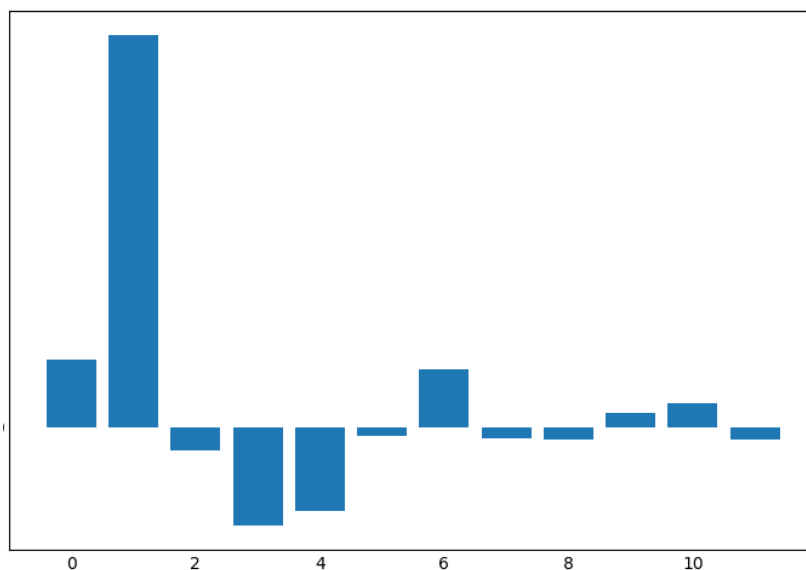


Figure 2.15: Az állapot vektor befolyása

A 2.15 ábrán jól látható hogy a döntéseket legjobban meghatározó érték a második, a relatív szögeltérés volt.

Később a jutalmazási módszeren annyit módosítottam, hogy minden egyes lépés után kapott 1 pontot, ha azzal a pályán belül maradt, és -1-et ha kilépett, az epizód hosszán nem változtattam; ezzel hasonló eredményeket értem el, de a tanulást meggyorsítottam, ugyanazt a minőséget kevesebb mintával és tanítással tudtam biztosítani, a tanulás eredményeit a 2.16 ábrán láthatjuk. A betanulás minősége egyértelműen leolvasható, a kanyarok előtt a külső ívre húzódik a jármű, hogy onnan kezdhesse meg a fordulást.

A dinamikai jármű modellhez szükséges ismereteim terjesztése, és tapasztalat szerzés után a valódi dinamikai modellel dolgoztam. A dinamikai modell nagy hátránya a kinematikai modellhez képest, hogy képes a megcsúszásra. Hamar megtapasztaltam, hogy a kinematikai modellnél használt módszerek önmagukban kevesek a dinamikai modell betanításához.

A legfőbb problémát az jelentette, hogy a megcsúszások miatt egy adott beavatkozás eredménye kevésbé volt egyértelmű, mint korábban. Ennek orvosolására az állapot vektorát egy további taggal, a szlip értékkel bővítettem. Ennek köszönhetően az ügynök tudta, hogy mikor csúszik meg a jármű és összekapcsolhatta ezt a beavatkozások kedvezőtlen eredményeivel. Ugyan a szlip mint adat, egy vektor mennyiség és mind a négy kerék, vagy mind a két tengely rendelkezik ezzel a mennyiséggel, kezdetekben egyetlen előjel nélküli skalárként kezeltem.

A dinamikai modell esetén a beavatkozások lehetőségei is bővültek, ezt az indokolta, hogy az ügynök ugyan kap információt a szlipről, de ahhoz, hogy a megcsúszást el is tudja

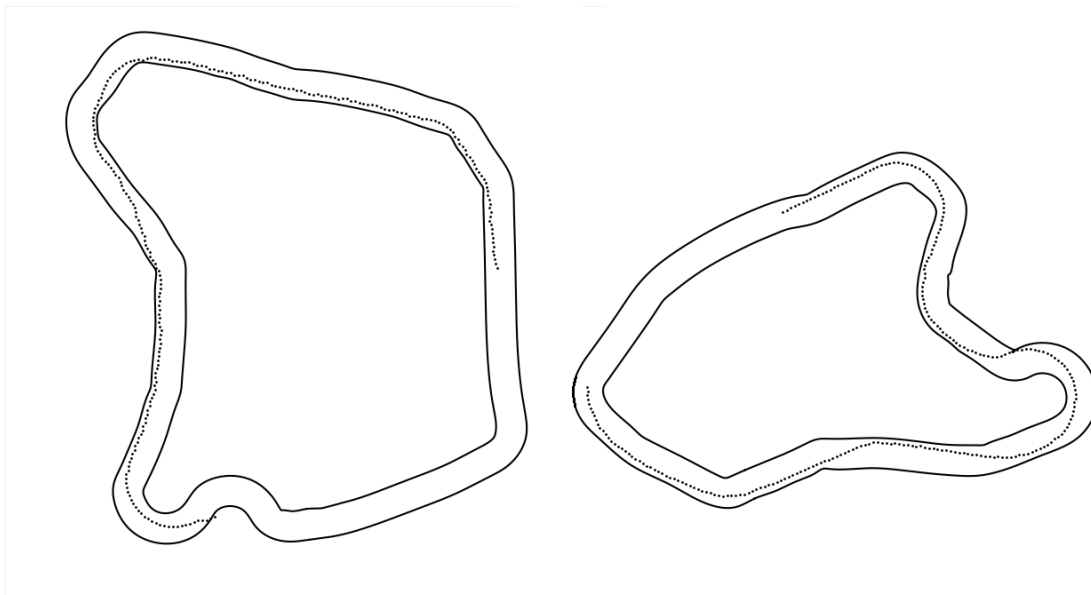


Figure 2.16: A csökkentett kormányszöggel elért eredmények

kerülni, ahhoz szükséges, hogy elegendő kormányszögek közül válogathasson. A jutalom meghatározó módszerem nem változtattam.

Az állapot reprezentáció és beavatkozás tér módosításával kedvezőbb eredményeket értem el, de ezek koránt sem voltak elégségesek.

Az állapot reprezentációt tovább bővítettem, így az mind a két tengely hossz- és keresztirányú szlipjét tartalmazta előjelhelyesen. Ezzel a módosítással az eredmények ugrászerűen javultak. Erről informálódhatunk a 2.17 ábrán.

Természetesen a tanulás menetét is igyekeztem vizuális megoldásokkal ellenőrizni, ehhez a veszteség függvény értékeinek logaritmusát ábrázoltam a tanítási iterációk függvényében. A 2.18 ábrán jól látható, hogy a tanulás során a veszteségértékek csökkennek, ahogy a hálózat súlyainak hangolása egyre tökéletesedik.

2.10 Továbbfejlesztési lehetőségek

Munkám során sokszor kerültem válaszút elé, ahol több irány közül kellett választanom, hogy merre megyek tovább. Természetesen voltak olyan esetek is, amikor több esetet is kipróbáltam így megkönnyítve választásomat. Ezek a kipróbálatlan ötletek jó gondolatébresztők voltak, továbbfejlesztési lehetőségeket vetettek fel előttem. A következőkben ezekről írok pár szót.

Ahogy a felhasznált minták diagrammján láthattuk az előző fejezetben, a hálózat nem egyenletesen tanul a mintákból, sokszor kap olyan állapotot tartalmazó mintát, amelyet már korábban is kapott, és ahhoz hasonló esetekre már kellően be is tanult. Ennek a problémának a javítására több módszer is eszembe jutott. Mindkettő a minták csoportosításán alapszik.

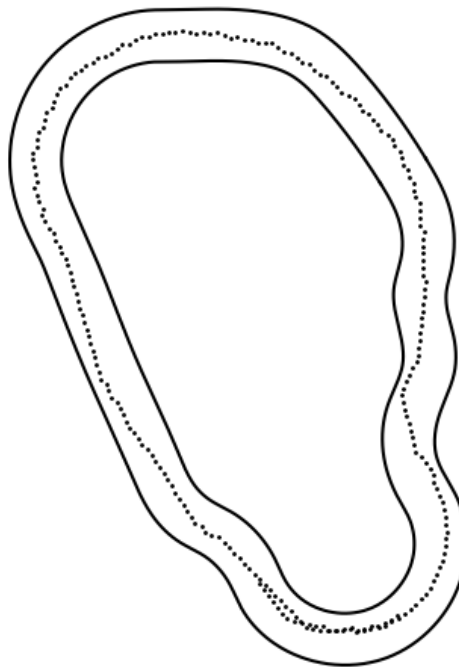


Figure 2.17: A dinamikai modellel elért eredmények

Az egyik módszer esetében a mintákat előre definiált szempontoknak megfelelően, adott számú csoportra bontanám, és mindegyik csoporthoz hozzárendelnék egy számértéket, amely az elfogadott maximális veszteség értéket mutatja. A tanítás során a mintakötegeket mindig egy adott csoportból választanám és amikor az adott csoport mintáival való tanítás veszteség értéke kisebb, mint a csoporthoz rendelt számérték, akkor ebből a csoportból már nem választanék több mintát, ezt a folyamatot addig iterálnám, amíg mindegyik csoport mintáira be nem tanul megfelelően a hálózat.

Ezzel a módszerrel nem csak a tanítást gyorsítanám meg, hanem súlyozni tudnám, hogy az egyes állapot csoportok tekintetében milyen betanulást várok el.

A másik módszernél a minták szétválasztására egy autoencodert használnék. Az autoencoderek segítségével n dimenziós adatokat m dimenzióba tömöríthetünk. Ezt úgy használnám fel, hogy az n dimenziós mintát 2 dimenzióba tömöríteném, majd a kapott két dimenziós halmazt egy csoportosító algoritmus szerint csoportosítanám így létrehozva a mintacsoportokat.

A fenti, 2.19 ábra egy ilyen szétválasztási folyamatot mutat. Az első ábra a nyers adathalmazt mutatja (az adathalmaz ugyan kettőnél több dimenziós volt, de a szemléletesség miatt az első két tagjuk szerint ábrázoltam). A második ábrán az autoencoderrel két dimenzióra csökkentett adatok láthatók, kékkel az egyik és pirossal a másik csoport (k-közép klaszterezést használtam). A következő kép a kicsomagolt, újra több dimenziós adatokat mutatja, de már látjuk melyik minta melyik csoportba tartozik.

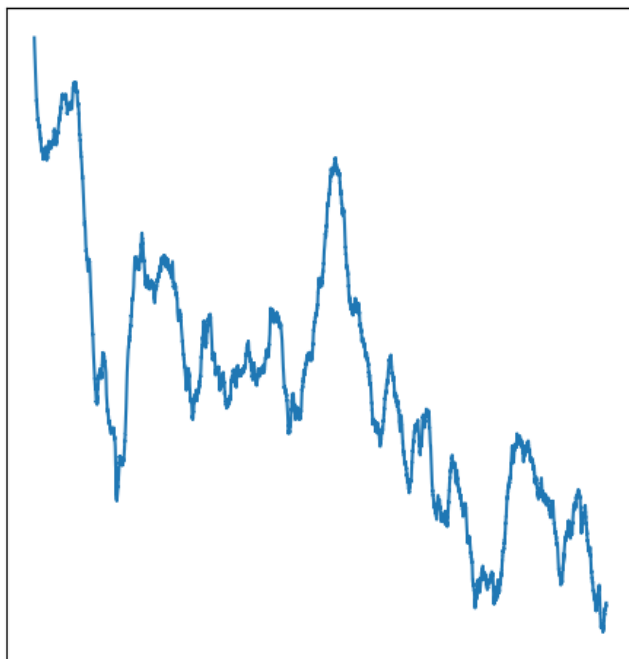


Figure 2.18: A veszteség függvény alakulása

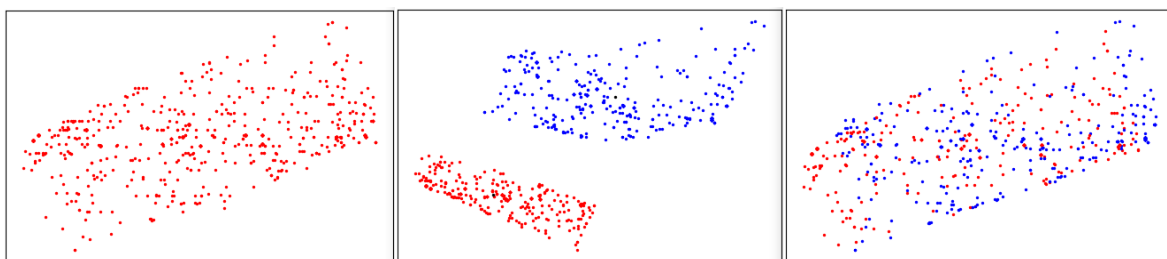


Figure 2.19: Mintaszétválasztás autoencoderrel

Ahogy az ábra is szemlélteti, az autoencoderrel szétválasztott, eredeti minták nem különülnek el egymástól jól meghatározható módon, illetve az elkülönülést sem mi határozzuk meg, nem tudjuk súlyozni az egyes szituációkhoz tartozó minták betanítási mértékét, úgy mint az előző módszerrel. Az esetszétválasztást teljes mértékben az autoencoder neurális hálójára bízunk.

Ezek a különbségek nyilvánvaló adottságai az egyes módszereknek, de az, hogy ezek milyen előnyökkel és hátrányokkal járnak csak tesztelés útján mondhatók meg biztosan.

A tanítás gyorsításának és specifikusságának javítására egy másik ötlet is felmerült bennem. Ez nem a kész mintákat kívánja szétválasztani, hanem a minták elkészítése során igyekszik a megfelelő minták előállítására. Ezt oly módon képzeltem el, hogy továbbra is az RRT felhasználásával választjuk meg, hogy a fát melyik csomópontban folytatjuk tovább. A beavatkozás meghatározásához viszont a hálózatot használjuk fel. Ezzel azt érhetjük el, hogy kívánt Q értékű minták arányát tudjuk növelni, a módszer hasonló a korábban szemléltetett ϵ -greedy módszerhez, de itt nem a legmagasabb Q értékű mintákat gyűjtjük, hanem a feladatra specifikusan, előre meghatározott Q értékűeket.

A módszerrel jó eséllyel növelhetjük a tanulás sebességét, irodalom kutatásom, és saját munkám során is megfigyeltem, hogy a hálózat gyorsabban tanul, ha negatív jutalmakat kap (sávtartás esetén).

A fent bemutatott módszerekkel a tanítás sebességét és minőségét is javíthatjuk, ezeket ötvözve, tovább gondolva pedig mégjobb eredményeket érhetünk el.

Chapter 3

Eredmények

A klasszikus Mély Q-Tanuló algoritmus RRT-vel való ötvözése után egy átfogó összehasonlító tesztet végeztem, melyben azonos körülmények között hasonlítottam össze a klasszikus algoritmust az általunk készített kombinált algoritmussal. A tanulás állandó paraméterei a következők voltak:

- Tanulási ráta: 0.01
- γ : 0.9
- Epizód: Ha eléri a 120-as lépéshosszt, vagy $|d|$ eléri a sáv szélesség 99%-át
- Azonos pályán való tanulás
- 10 db előretekintő szög, 6 pont távolságra egymástól
- Pályaívек kormány szöge: $30-31^\circ$
- Jármű modell kormány szöge: 25°
- Jutalom: minden epizód végén, -1 vagy 1 a korábban ismertetett meghatározás szerint

A tanulás és a tesztelés is azonos pályákon zajlott. Ezt a tényt az garantálja, hogy előre legenerált, majd fájlba mentett pályák beolvasásával zajlott mind a tanítás, mind a tesztelés.

A tesztelés során három mennyiséget mértem. A sikerességi ráta azt mutatta meg, hogy a pályák hány százalékán értük el a maximális, 250-es lépéshosszt.

Bár nagyon fontos kérdés, hogy sikerül-e teljesíteni a teljes epizódot, ezek mellett fontos azt is figyelembe venni, hogy a nem teljesített epizódokon mennyi lépést tett, hisz nem mindegy, hogy például 10 vagy 240 lépés után hibázott a modell. Ezen mennyiség méréséhez az epizódok alatt megtett lépések átlagát vettem figyelembe.

Az utolsó mért mennyiség a korábbiakban bemutatott, $-|d| + \cos(\alpha * (\pi/2))$ alakot öltő függvényvel meghatározott jutalmak epizódonkénti átlaga. Ez a mennyiség azt mutatja meg, hogy milyen mértékben tartózkodott a jármű modell a sávközepén az úttal 0° -os szöget

bezárva. Ez azért lényeges, mert egyenes útszakaszok esetén kedvezőbb, ha a jármű a sáv közepén halad.

A tanításokat különböző minta és tanulási iterációs számmal végeztem. A minták és a velük végzett tanulási iterációk száma megegyezett. Ezek a számok:

- 1000
- 5000
- 10 000
- 50 000
- 100 000

voltak.

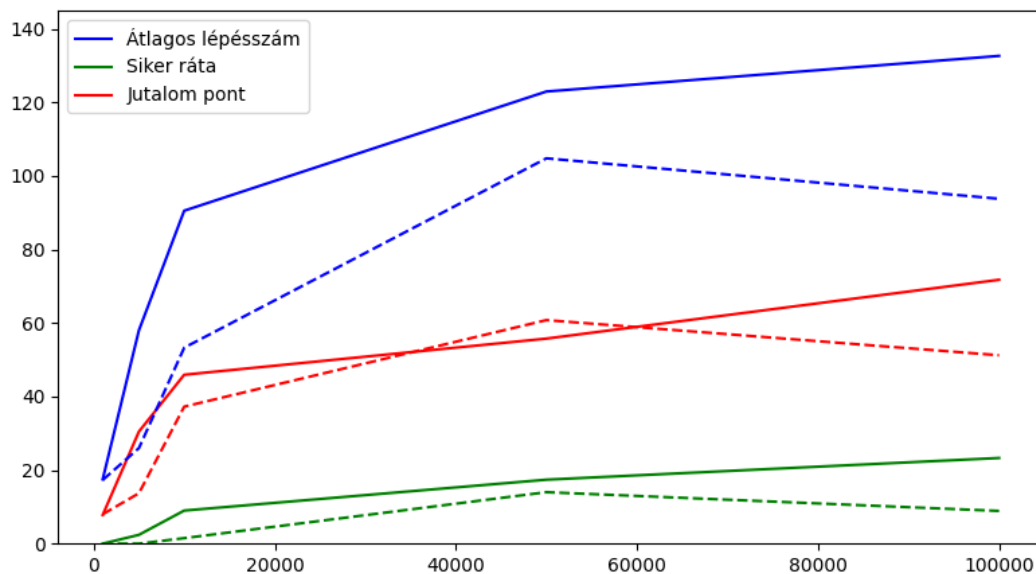


Figure 3.1: Módszerünk összehasonlítása a klasszikus DQN-nel

Minták száma	Átlag lépés	Siker ráta	Jutalom
1000	17.316	0.0%	8.49
5000	25.99	0.0%	13.71
10000	53.29	15%	37.26
50000	104.8	14%	60.8
100000	93.8	8.9%	51.24

Table 3.1: A klasszikus algoritmus eredményei

A 3.1 diagramm címkézése mutatja melyik szín melyik mennyiséget jelöli; a folytonos vonalak az RRT-vel kombinált algoritmust mutatják, a szaggatott pedig a klasszikus algoritmus eredményeit jelenítik meg.

Minták száma	Átlag lépés	Siker ráta	Jutalom
1000	17.5	0.0%	7.8
5000	58.03	2.4%	30.48
10000	90.55	9%	45.94
50000	122.972	17.4%	55.77
100000	132.66	23.3%	71.77

Table 3.2: Az RRT-s algoritmus eredményei

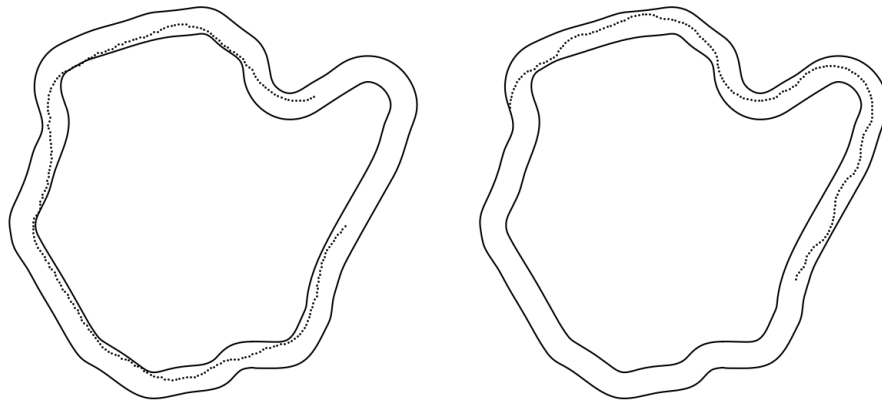


Figure 3.2: Módszerünk összehasonlítása a klasszikus DQN-nel

A diagramm és 3.1, 3.2 táblázatok jól mutatják az RRT-s ötvözés sikerességét, szinte minden körülmények között jobban teljesített. Láthatjuk, hogy 1000 mintával való tanítás során az eredmények között nincs szignifikáns eltérés, azonban 5000 és annál több minta esetén már nagyobb eltéréseket diagnosztizálhatunk.

Megállapíthatjuk, hogy az RRT-s módszer a mintaszám növekedésének függvényében a korai dinamikus növekedés után egy alacsonyabb, de megközelítőleg konstans növekedést mutat.

A klasszikus algoritmus a közepes progresszióját már közepesen magas mintaszám esetén elveszti és értékei csökkenni kezdenek, a túltanulás tünetei mutatkoznak meg.

A teszteket vizuálisan is dokumentáltam, a 3.2 ábra mutatja ezt, a bal oldalon az RRT-s algoritmussal, a jobb oldalon a klasszikus algoritmussal elért eredmény látható.

Feltűnő lehet a siker ráta alacsony szintje, ez egy szándékosan előidézett jelenség. Annak köszönhető, hogy a generált pályákat szándékosan nehézre állítottam, hogy könnyebben megmutatkozzanak az esetleges különbségek. Olykor az adott maximális kormánysszöggel teljesíthetetlen, 180° -nál nagyobb ívben fordító kanyarok voltak.

Végeztem teszteket reálisan teljesíthető pálya köteggel is. Azonos tanítóminta esetén az RRT-s megoldás 80%-os siker rátával teljesített, míg a klasszikus algoritmus csupán 46%-ot mutatott.

3.1 Összegzés

Dolgozatom során rengeteg új tapasztalattal gazdagodhattam. Ebbe beletartozik a Python nyelvvel való megismerkedés, ugyan a nyelvet korábbi tanulmányaim révén ismertem, de ez volt az első nagyobb hangvételi implementáció.

Legnagyobb újdonságot azonban a gépi tanulás jelentette, hisz előtanulmányaim során csak érintőlegesen találkozhattam vele, érdekesség jeleggel. Érdeklődésemet már ekkor felkeltette a téma, hisz külső szemmel nagyon nehéz volt elképzelni, hogy a gépi tanuláson kívüli módszerek, algoritmusok "statikusságához" képest hogyan lehet elérni ekkora univerzalitást és "dinamikusságot".

Így nagy örömet okozott, hogy végre betekintést nyerhettem a gépitánuulás világába és ilyen tapasztalattal gazdagodva folytathatom szakmai fejlődésem.

3.2 Források

- <https://villanyautosok.hu/2021/05/11/ezek-az-onvezetes-szintjei>
- <https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html>
- https://upload.wikimedia.org/wikipedia/commons/thumb/4/46/Colored_neural_network.svg/250px-Colored_neural_network.svg.png
- https://www.mdpi.com/applsci/applsci-10-07171/article_deploy/html/images/applsci-10-07171-g003.png
- https://mmaws.bme.hu/2019/pages/program/papers/Paper_17_Kovari_B_Becsi_T_Szalay_Zs_IFFK_2019.pdf

List of Figures

1.1	SEA J3016 szabvány	3
2.1	A pályagenerálás folyamata	5
2.2	Kinematikai jármű modell	6
2.3	Dinamikai jármű modell	8
2.4	A környezet vizualizációja	9
2.5	Eredmény	10
2.6	Haladás egyenes sávon	11
2.7	A Mély Q-Tanulás elvi alapja	11
2.8	A Neurális Hálózat	12
2.9	Az RRT által generált trajektória, egy és több "fa" esetén	15
2.10	A jutalom függvények	18
2.11	A kinematikai modell eredményei	19
2.12	A minták és a velük tanított hálózat	20
2.13	Az új minták gyűjtése	21
2.14	A 3 dimenziós állapot reprezentáció esete	22
2.15	Az állapot vektor befolyása	23
2.16	A csökkentett kormánysszöggel elért eredmények	24
2.17	A dinamikai modellel elért eredmények	25
2.18	A veszteség függvény alakulása	26
2.19	Mintaszétválasztás autoencoderrel	26
3.1	Módszerünk összehasonlítása a klasszikus DQN-nel	29
3.2	Módszerünk összehasonlítása a klasszikus DQN-nel	30

List of Tables

3.1	A klasszikus algoritmus eredményei	29
3.2	Az RRT-s algoritmus eredményei	30

Bibliography

- [1] Amit Ailon, Nadav Berman, and Shai Arogeti. On controllability and trajectory tracking of a kinematic vehicle model. *Automatica*, 41(5):889–896, 2005.
- [2] Arden Dertat. Applied deep learning - part 3: Autoencoders. *towardsdatascience*, Oct 2017.
- [3] Arpad Feher, Szilard Aradi, and Tamas Becsi. Q-learning based reinforcement learning approach for lane keeping. In *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*, pages 000031–000036, 2018.
- [4] Bálint Kővári, Ferenc Hegedüs, and Tamás Bécsi. Design of a reinforcement learning-based lane keeping planning agent for automated vehicles. *Applied Sciences*, 10(20), 2020.
- [5] Gustavo Maciel. Generating procedural racetracks, Dec 2013.
- [6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharmashan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015.
- [7] Ahmad El Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. End-to-end deep reinforcement learning for lane keeping assist, 2016.
- [8] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, Oct 2017.
- [9] Qun Wang, Weichao Zhuang, Liangmo Wang, and Fei Ju. Lane keeping assist for an autonomous vehicle based on deep reinforcement learning. Technical report, SAE Technical Paper, 2020.

- [10] Wikipedia. Rapidly-exploring random tree — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Rapidly-exploring%20random%20tree&oldid=1117495688>, 2022. [Online; accessed 06-November-2022].
- [11] Qingyang Xu, Caixia Zhang, Li Zhang, and Yong Song. The learning effect of different hidden layers stacked autoencoder. In *2016 8th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, volume 02, pages 148–151, 2016.
- [12] Qichao Zhang, Rui Luo, Dongbin Zhao, Chaomin Luo, and Dianwei Qian. Model-free reinforcement learning based lateral control for lane keeping. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, 2019.