[[1]](#footnote-1)

Kormánymű geometria optimalizáció megerősítéses tanulással

Dóka Tamás, *Phd hallgató, BME*

*Absztrakt*—Ebben a beszámolóban egy négykerekű jármű kormánymű geometriájának optimalizálását tűztem ki célul megerősítéses tanulás segítségével. A problémát négy paraméterre és kilenc lehetséges akcióra – hatásra redukáltam, és ehhez készítettem egy egyszerű ügynök implementációt, amellyel megpróbáltam elfogadható eredményeket generálni, majd összefoglaltam az eredményeket.

# Bevezetés

A

Mérnöki feladatok nagy részében valamilyen paraméterhalmaztól függő problémát kell optimalizálnunk, nincs ez máshogy a gépészeti tervezésben sem. A tervezés során általában nem állnak rendelkezésre a pontos számításokhoz szükséges adatok, így a tervezés korai fázisában ezeket becsléssel határozzuk meg, majd iteratívan közelítünk az optimális megoldáshoz. Megerősítéses tanulással is egy problémát optimalizálunk, legyen szó a go játék megnyeréséről vagy egy komplex tervezés során felmerülő döntések meghozataláról. A házi feladat során egy négykerekű jármű kormányművének egyszerűsített, kétdimenziós (felülnézeti) reprezentációjának geometriai optimalizációját terveztem megvalósítani megerősítéses tanulás segítségével. A beszámolóban először röviden bemutatom a megerősítéses tanuláshoz használt OpenAI Gym környezet felépítését, az optimalizációs problémát, a tanítás során használt környezetet, az általam implementált tanító algoritmust, végül az eredmények összefoglalását.

# OpenAI Gym megerősítéses tanuláshoz

A megerősítéses tanulás alapja egy környezetből (environment) és egy a környezettel interakciót létesítő ügynökből (agent) áll (1. ábra).

A környezet „tartalmazza” a problémát, amelyet az ügynök igyekszik megoldani a környezeten végzett hatásokkal (actions), amelyek megváltoztatják a környezet belső állapotát (state). Az ügynök által elvégzett hatásra bekövetkezett új állapot mellett az ügynök visszakap egy jutalom (reward) értéket is, ami a hatás „jóságát” jellemzi a környezet kezdeti állapota mellett. Az ügynök célja, hogy minden állapot mellett olyan hatást gyakoroljon a környezetre, ami után a legnagyobb jutalmat kapja meg.

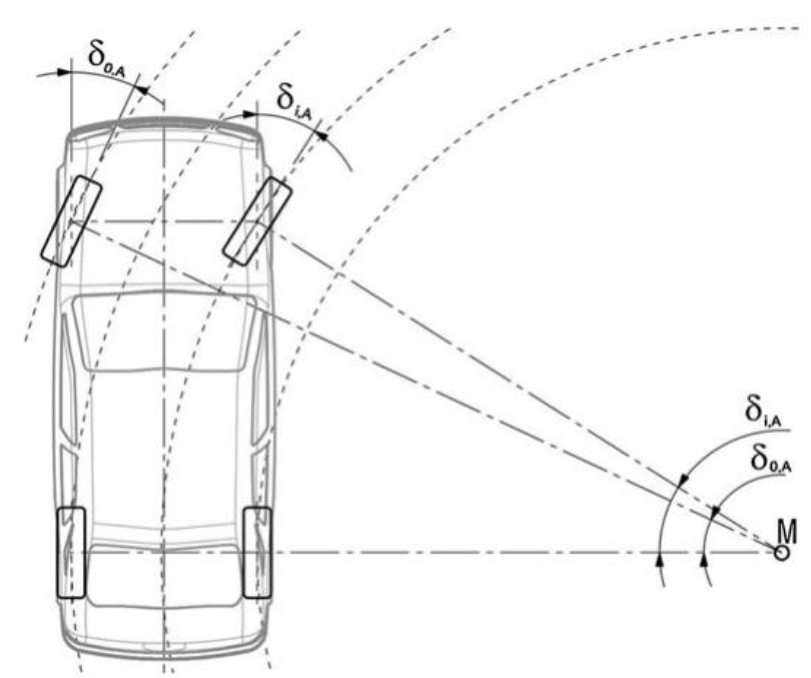
Az ügynök – környezet interakció megvalósítására megalkottak egy „szabványt”, ami lehetővé teszi, hogy egy sztenderd interfészen keresztül kommunikáljon egymással a környezet és az ügynök implementáció, ezt nevezik OpenAI Gymnek. OpenAI Gym környezetekből több szabadon hozzáférhető, így saját ügynök implementációkat lehet rajtuk tesztelni, illetve saját készítésű környezeteket is lehet általános ügynök algoritmusokkal (például stable baselines) tesztelni, ha rendelkeznek gym környezet interfészekkel. Ezek az interfészek többek között a reset(), step(), és render() függvények.



1. Ábra. A környezet és az ügynök kapcsolata

# Az optimalizációs probléma

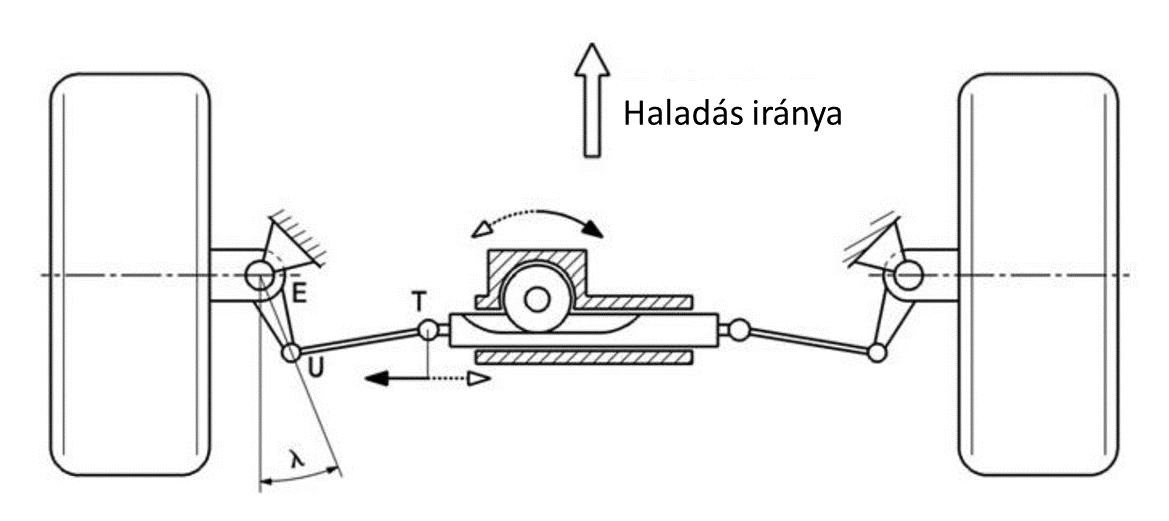
Az optimalizációs probléma egy négykerekű jármű ideális kanyarodásának megvalósítása fix elrendezésű kormánymű geometriával. Ideális, csúszásmentes kanyarodás akkor jön létre, ha a jármű kerekei egy közös középpontból húzott köríveken haladnak kanyarodás során (2. ábra – M pont). Ha csak elsőkerék-kormányzású a jármű, úgy a két hátsó kerék közös tengelye megszab egy egyenest, amin ez a közös középpont elhelyezkedhet. Ha meghatározunk egy fordulókör sugarat, a külső köríven haladó kerék talppontjából ezzel a távolsággal húzott ív, és a hátsó tengely egyenese kiadja a közös középpontot. Ahhoz, hogy az ideális kanyarodás megvalósuljon, a belső keréknek is e középpontból húzott köríven kell haladnia, így minden fordulókör sugárhoz, adott lesz a külső , és a belső kerék , egyeneshez képesti elfordulási szöge (2. ábra).



2. ábra. Egy négykerekű jármű kerekeinek és a fordulókör középpontjának viszonya.

## Kormánymű geometria

A közúti járművek legelterjettebb kormánymű kialakítása az úgynevezett fogaskerék – fogasléc (rack-pinion) elrendezés (3. ábra)



3. ábra. Fogaskerék – fogasléc kormánymű kialakítás

Ebben az egyszerű kialakításban a fogasléc oldalirányban való lineáris mozgatása a külső és belső kerék eltérő mértékű elfordulását eredményezi, így megfelelő méretek megválasztásával létrejöhet az ideális kanyarodáshoz közeli állapot minden fordulókör nagyság mellett. Az optimalizálás során tehát egy adott tengelytáv (wheelbase), nyomtáv (track width), valamint a kerék talppontja és az elfordulási tengelye (3. ábra – E pont) közötti távolság (kingpin distance) mellett kell meghatározni, hogy a forgatókar (control arm) (3. ábra E-U) és kormányösszekötő-rúd (tie-rod) (3. ábra E-T) végpontjai hogyan helyezkedjenek el. További kritérium még egy mindennapi használathoz feltétlenül szükséges minimális fordulókör előírása is.

## Változtatható paraméterek

Az optimalizálás során tehát a változtatható paraméterek a 3. ábrán jelölt T és U pontok x és y irányú koordinátái lesznek, összesen négy paraméter. A derékszögű koordináta-rendszer középpontját az első tengely közepére helyezzük és y irány felel meg a menetiránynak.

Ebben az esetben, amennyiben az U pont y koordinátája az első tengely mögött helyezkedik el, negatív lesz, a fogasléc pozitív x irányba való mozgatásakor a kerekek néhány speciális konfigurációtól eltekintve, a pozitív iránynak megfelelő forgást végeznek (a jármű balra fordul). Amennyiben U pont y koordinátája pozitív, a fogasléc negatív x irányba való mozgatásával érhető el ugyanez a működés.

A paraméterek beállítása után kiszámítható a fogasléc mozgatásából következő kerék elfordulások mértéke, így a külső köríven haladó kerék elfordulásának függvényében felírható lesz a belső kerék elfordulása. Ez a függvényszerűség az ideális kanyarodásra is felírható, így a kormánymű hibája az ideális görbe és az adott paraméterek mellett létrejött görbe közötti különbség négyzetösszegeként definiálható. Az optimalizálási feladat megoldott, ha ez a hibaérték eléri a nullát, vagy egy általunk előírt határ alá csökken, vagyis a tényleges görbe megközelíti az ideálisat és a kormánymű biztosítja a minimális fordulókört.

# A tanítás során használt környezet

A megerősítéses tanítás megkezdéséhez először az interakcióban résztvevő környezet felépítését kezdtem meg. A környezetnek az ügynök által kiváltott hatásra vissza kell adnia a hatásra kialakult belső állapotát, valamint egy jutalom értéket. A környezetnek továbbá követnie kell a sztenderd gym felépítést is, amelynek a kötelező elemei a következők:

**def** \_\_init\_\_(self):

Itt történik meg a fő paraméterek beállítsa, inicializálása, amelyek a tanítás és kiértékelés során nagyrészt változatlanok maradnak (például tengelytáv (wheelbase)). Ezen kívül itt kell definiálnunk a környezetre kifejthető hatások halmazát:

self.action\_space

Valamint a koordináták értékeit tartalmazó visszatérési állapot (state) lehetséges értékeinek kereteit:

self.observation\_space

A környezet inicializálása után, a kezdeti állapot meghatározásához, illetve tanítási epizódok között is meghívásra kerül a visszaállító (reset) függvény.

**def** reset(self):

...

**return** self.state

A következő nagyon fontos függvény az ügynök tanítása során használt lépés (step) függvény, ami az ügynök által átadott hatás (action) alapján kiszámítja a környezet új belső állapotát, valamint a jutalom (reward) értékét.

**def** step(self, action):

...

**return** np.array(self.state), reward,done, {}

Ez a step függvény adja a környezet gerincét, működésére a következő fejezetben részletesebben is kitérek.

További kötelező elem a környezet grafikus megjelenítésére szolgáló render függvény, azonban ennek funkcionális megvalósítása a projektből kimaradt.

## Step függvény

A step függvény feladata az ügynök hatása miatt kialakult új belső állapot, illetve az ehhez a hatáshoz tartozó jutalom kiszámítása.

Első lépésként az ügynök által kiadott action dekódolását végezzük el, amely az előző fejezetben említett változtatható paraméterek egyikének, egy előre definiált mértékkel való növelését vagy csökkentését jelentik, vagy az összes érték változatlanul hagyását eredményezi. A változtatást egy dedikált függvény végzi el, amely a módosított állapottal tér vissza.

**def** \_take\_action(self, action, state):

**return** modified state

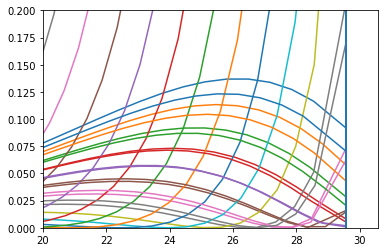
A módosított állapot ellenőrzése következik, mint például, hogy az előre definiált geometriai tartományon kívül esik-e a módosított állapot. Ezt követően a módosított állapotban található koordináta értékek alapján meghatározzuk a geometriát: a kormányösszekötő rúd és a forgatókar hosszát, az egyenes haladási irányhoz tartozó kezdeti szögek értékét. Továbbá ezekből meghatározható a fogasléc maximális x irányú elmozdulása is. Ezt az értéket geometriai és fizikai megfontolások határozzák meg. A fogasléc mozgatása során korlátozást jelent, ha a kormányösszekötő rúd és a forgatókar valamelyik oldalon kollineárissá válik, hiszen egyik esetben nem lehet tovább távolítani E és U pontokat, másik esetben (ha az E és U pontok közeledésével jön létre ez az állapot) az egytengelyűség állapotából való továbbmozdítás nem determinisztikus, így nem garantálható, hogy kerék megfelelő irányban fordul tovább. További korlátozást jelent, hogy ezeknek a kollineáris állapotoknak a közelében a rendszer rendkívül merevvé válik, a kormányzáshoz szükséges forgatónyomaték biztosításához nagyon nagy erőkre lenne szükség, amely többek között túlméretezett alkatrészekhez vezetne, valamint az esetek többségében egyszerűen kivitelezhetetlen. Ezeknek az elveknek megfelelően határozzuk meg a fogasléc maximális elmozdulását.

Az állapot paraméterek geometriája alapján meghatározzuk a fogasléc mozgatási irányát, hogy az iterációs ciklusban pozitív irányú kerékelfordulást eredményezzen. Ez a mozgási irány a valóságban sem jelent problémát, hiszen egy megfelelő irányváltó áttétellel megvalósítható mindkét irányban való mozgás.

A következő szakasz a fogasléc mozgatása miatt kialakuló kerékelfordulások számítása. Itt a teljes mozgástartományt osztjuk egy előre meghatározott érték szerint egyenlő darabokra, ami azt eredményezi, hogy minden konfigurációban egyenlő számú pontban ismerjük majd a szögértékeket. Tehát ha az osztás értéke 10, akkor 50 milliméteres maximális fogasléc mozgatáshoz 5 milliméteres, 150 millimétereshez 15 mm-es elmozdulásonként számítjuk ki a szögértékeket.

A kanyarodás szempontjából nem a fogasléc pozíciók mérvadóak, hanem a külső íven forduló kerék elfordulási szögének függvényében a belső kerék elfordulási karakterisztikája, vagyis ennek ideálistól való eltérése. Így a ciklus kimenete az osztásonkénti külső elfordulási szög, valamint az ezekhez az értékekhez tartozó belső elfordulási szög hibája.

Amennyiben a geometria alkalmas a kívántnál kisebb fordulókör biztosítására, vagyis a külső kerék nagyobb szögben való kormányzására az elvártnál, a hibaértékek vizsgálata csak a kívánt tartományban szükséges. Ennek megfelelően a határszög (border angle) feletti tartomány értékeit elhagyva, és a határszög esetén adódó hibaértéket lineárisan interpolálva kapjuk a hibagörbét. A hibagörbe értékeit négyzetre emelve kapjuk a hiba karakterisztikát.



4. ábra. Kormánymű hiba karakteriszikák

A kormánymű hibája tehát a hiba karakterisztika integrálja az egyenes (0 fok) és a minimális fordulókörhöz tartozó határszög között.

## Reward függvény és epizód vége

A megerősítéses tanulás lényege, hogy az ügynök által végzett hatást az adott, ismert állapotban a „jóságának” megfelelően jutalmazzuk vagy büntetjük, annak érdekében, hogy az általunk kívánt állapotba juttassa a környezetet. Esetünkben a cél az ideális kormánymű geometria elérése, vagyis a hibaérték minimalizálása.

Ennek megfelelően, ha az adott hatásra a hiba értéke csökken, a jutalom értékét pozitívra, ha stagnál vagy nő, akkor a jutalom értékét negatívra állítom.

Az ügynök a tanítás során epizódokat játszik, amelyben vagy egy előre meghatározott lépésszám után, vagy ha végleges állapotba jut, megáll. Végleges állapot, ha nem megvalósítható geometria adódik, a kormánymű hibája az előre meghatározott érték alá csökken, vagy ha numerikus hibák miatt több azonos szögérték szerepel a kimeneti tömbben, így nem érvényesül a függvényszerűség a külső kerék szögelfordulása és a hibaértékek között. Ha ilyen végleges állapotba jutunk a környezet a (done) változó igazra állításával jelzi az ügynöknek az epizód végét.

# Az ügynök

A környezet felépítése és tesztelése után az ügynök implementációját kezdtem elkészíteni. Természetesen a környezet forráskódját folyamatosan változtattam az ügynökkel való kísérletezés során, mivel a tanítás során számos problémába futottam, amelyek rejtve maradtak a kezdetleges tesztek során.

Az ügynök megvalósítására alapvetően két lehetőség közül lehet választani: saját ügynök írása, vagy előre megírt és tesztelt ügynökök alkalmazása. Mivel nem volt korábbi tapasztalatom megerősítéses tanulással, úgy gondoltam, hogy saját ügynök használatával sokkal több információt szerezhetek a környezet hibáiról, illetve az ügynök működéséről is. A kezdeti sikeres összeállítás után sajnos nem sikerült megfelelő tanítást végeznem, így előre megírt ügynök alkalmazását is megpróbáltam, ezek tapasztalatait foglaltam össze a következő alfejezetekben.

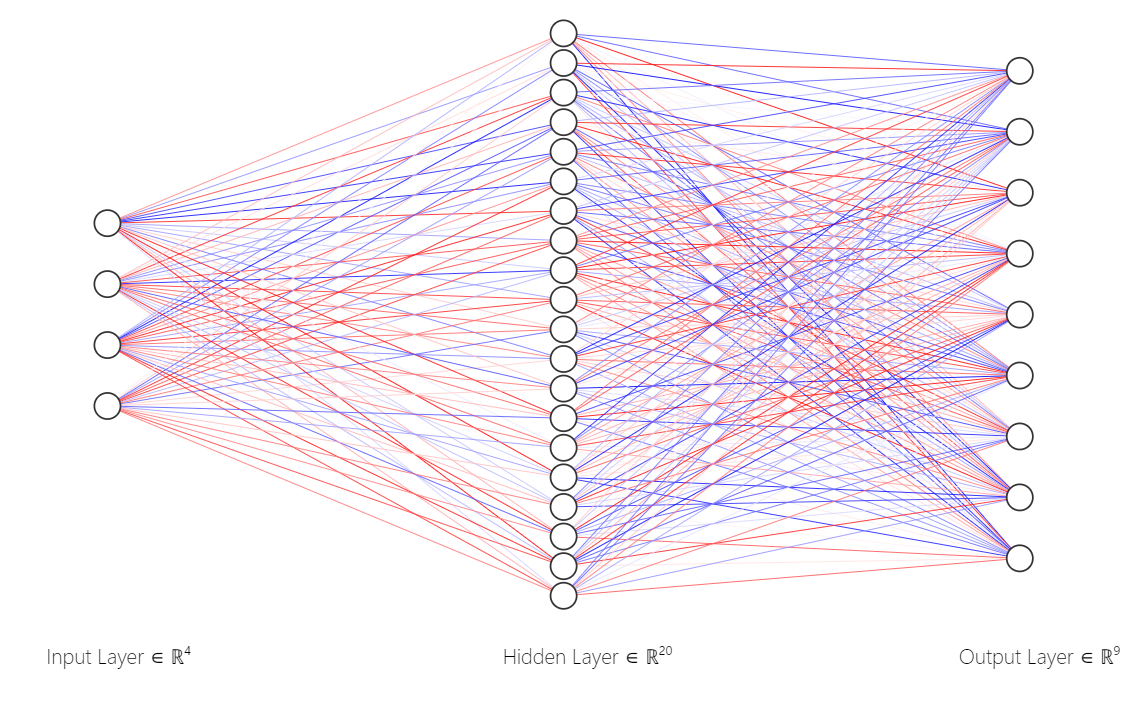
## Saját ügynök implementáció

Az ügynök felépítését egy példát bemutató blogposzt alapján kezdtem el. A példában [1] a Lunar Lander gym környezethez készít a szerző egy pytorch alapú egyszerű neurális hálót jól követhető módon, kódot is mellékelve, ez alapján könnyen el tudtam készíteni a saját ügynök implementációmat is a környezethez.

Az ügynök egy egyszerű neurális hálózat, amelynek a bemenete a környezet által átadott megfigyelt állapot (state), kimenete pedig a lehetséges hatások (actions) közüli választás valószínűsége az adott állapot mellett. A hálózatban egy rejtett réteg található (5. ábra). A hálózat tanításához szükségünk van tanító példákra, majd a tanítást követően teszt és validáló példákra is. Mivel esetünkben tanító példák nem álnak rendelkezésre, így generálnunk kell őket. Erre szolgál a hálózatunk generate\_batch(env, batch\_size, t\_max) függvénye, ami a környezet állapota alapján hatásokat generál, követi a hatások után járó jutalom értékét, majd amennyiben az epizód véget ér, az állapotokat, hatásokat és jutalmakat eltárolja, és ezt addig ismétli, amíg el nem érjük az epizódok számának határát (batch\_size). Ezt követően azokat az epizódokat kiválasztjuk, amelyeknek a legnagyobb volt az összegzett jutalom a végén. Ezt végzi el a hálózatunk filter\_batch nevű függvénye.

A tanítás során a hálózat belső paramétereit optimalizáljuk úgy, hogy a legjobb kiválasztott tanító példáknak megfelelően válasszanak az állapotnak megfelelő hatást. Ezt, a példát követve CrossEntropyLoss hibafüggvénnyel és Adam optimalizáló eljárással végeztem.

A hálózat különböző paramétereit beállíthatjuk, mint például a tanulási rátát (learning rate), a rejtett réteg méretét, azért, hogy megfelelően jó eredményt érhessünk el a tanítás után. A saját ügynökkel nem sikerült sajnos a kezdeti próbák során értékelhető tanítást végeznem, így egy tesztelt, előre megírt ügynök implementációt is megpróbáltam betanítani a környezeten.



5. ábra. Az ügynök hálózat felépítése

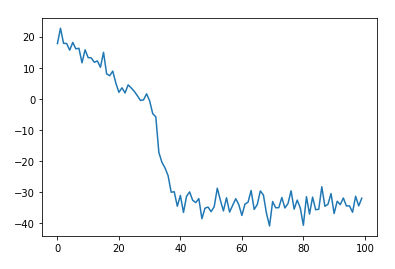
## Stable Baselines ügynök implementáció

OpenAI gym környezetek tesztelésére és azokon való tanításra létrehoztak környezetfüggetlenül megírt ügynököket, amelyek különböző algoritmusokkal próbálják megtanulni a környezetben való helytállást. Erre egy tesztelt, jól dokumentált könyvtár a stable baselines. Az ügynök algoritmusok közül a DQN algoritmust választottam, ez megfelelő volt a környezetemhez, mivel diszkét a hatás halmaza, és folytonos a megfigyelés halmaza. A DQN ügynökkel futtatva a környezetet sajnos további hibák jöttek elő a környezetről, ami szélsőséges esetekben nem működik megfelelően, így a tanítás sok esetben egy belső errorral megszakad, aminek legtöbbször valószínűleg numerikus kerekítésekből származó geometriailag nem helyes konfigurációk az okai.

# Eredmények

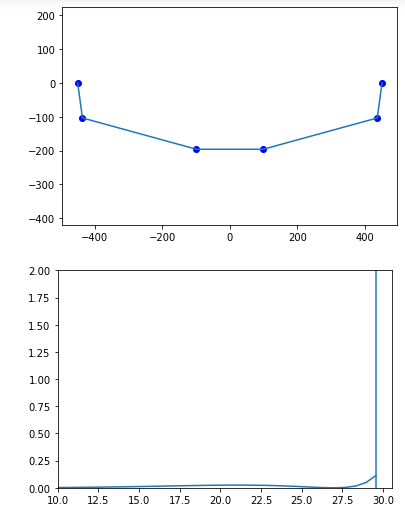
A környezetet futtatva sajnos működőképes tanítást nem sikerült megvalósítanom. Tapasztalatlanságom miatt a stable baselines ügynökkel való tanítás valószínűleg több okból is meghiúsulhatott, amelyeket az előre implementált ügynökök alaposabb vizsgálatával fel lehet tárni a jövőben.

A saját ügynökkel való tanításnak a tapasztalatai alapján továbbra is gondok vannak a környezettel, nincs minden felmerülő hiba megfelelően kezelve, így sok epizód végződik hibás konfigurációval, amelyek így a helyes alkalmazott hatás (action) után is rossz jutalmat eredményeznek. Továbbá a tanítás során a hálózat rossz megoldásra tanult rá, amikor nem történt környezeten belüli hiba, jellemzően egy rossz jutalomérték mellett ugyanazt az akciót hajtotta végre újra és újra (6. ábra). A problémát próbáltam szűkíteni, így 9 lehetséges akció helyett csak 5 vagy 3 akciót engedélyezni, azonban ezek a próbálkozások sem vezettek eredményre eddig.



6. ábra. Az ügynök kumulált jutalma a tanítás előrehaladásával

A környezet tesztelésére egy klasszikus optimalizáló eljárást is kipróbáltam, amivel sikerült egy véletlenszerű állapotból megtalálni optimális megoldást, ahol a kormánymű hibája minimális, tehát, feltételezhető a környezet helyes működése, a szélsőséges hibáktól eltekintve. A 7. ábrán látható a klasszikus módszerrel megtalált optimális kormánymű geometria.



7. ábra. Klasszikus optimalizáló eljárással talált megoldás a problémára

Források

1. Donal Byrne, “Landing A Rocket With Simple Reinforcement Learning,” in <https://medium.com/coinmonks/landing-a-rocket-with-simple-reinforcement-learning-3a0265f8b58c>

1. . [↑](#footnote-ref-1)